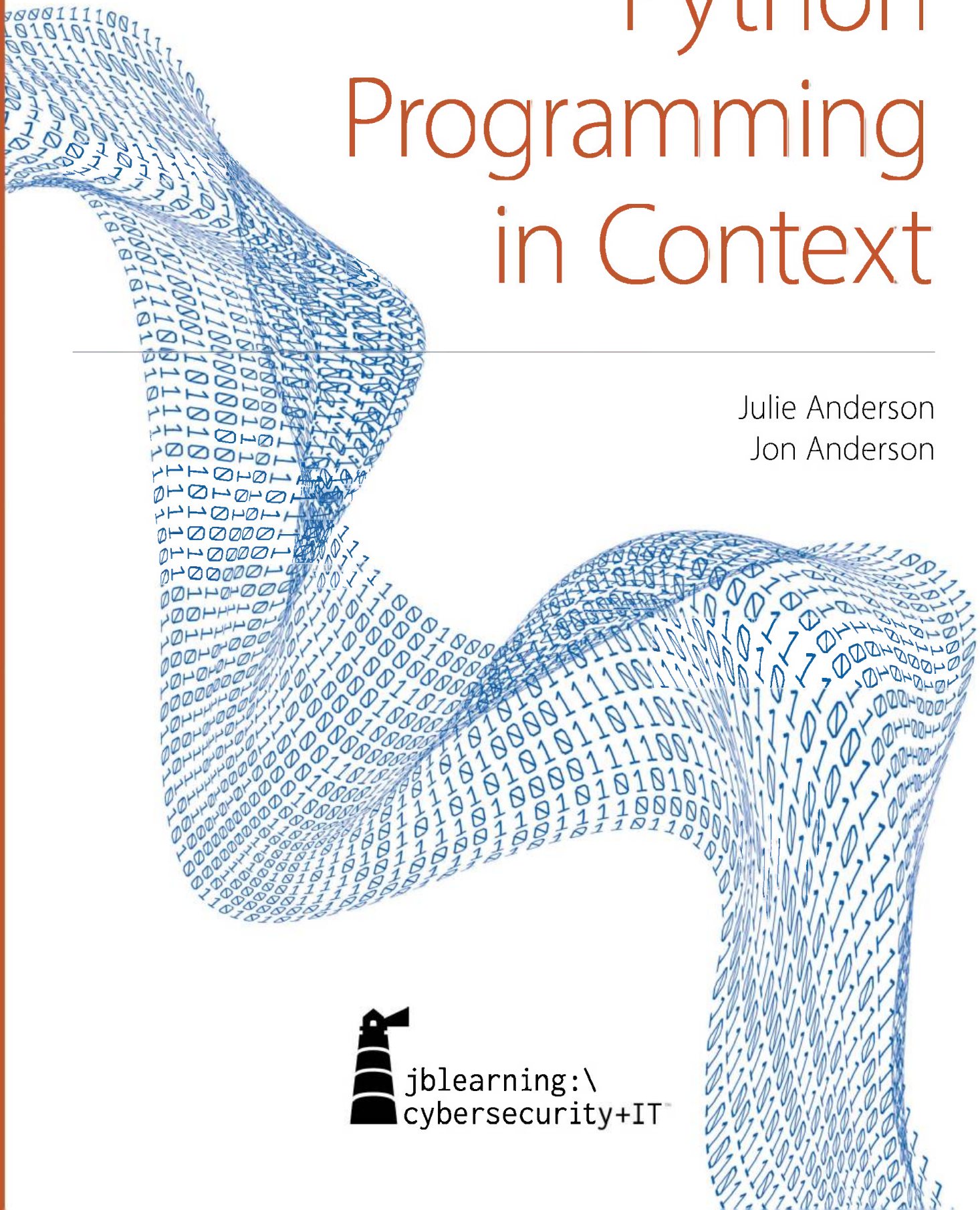
jblearning:\cybersecurity+IT™

FOURTH EDITION

Python Programming in Context

Julie Anderson
Jon Anderson



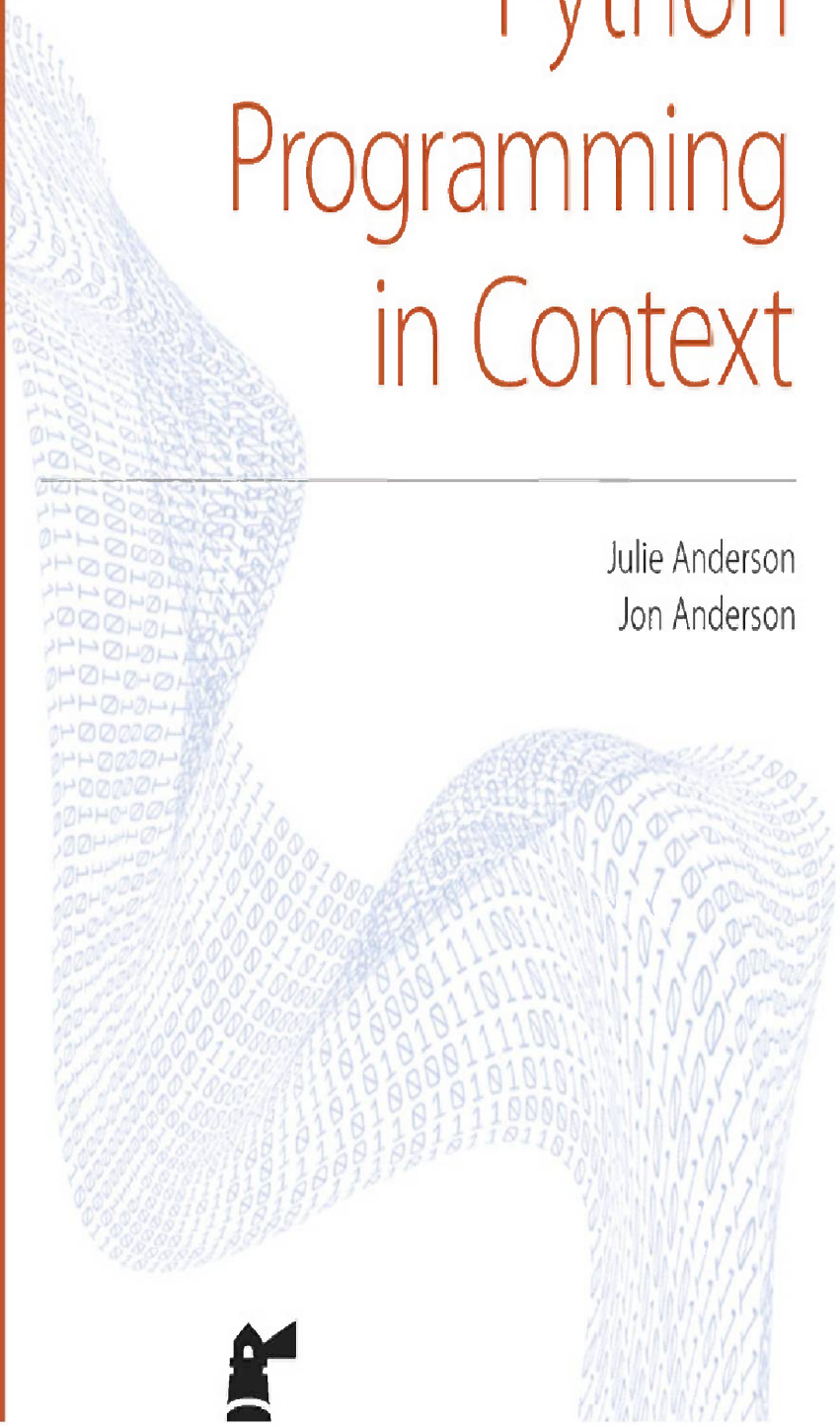
jblearning:\
cybersecurity+IT™

jblearning:\cybersecurity+IT™

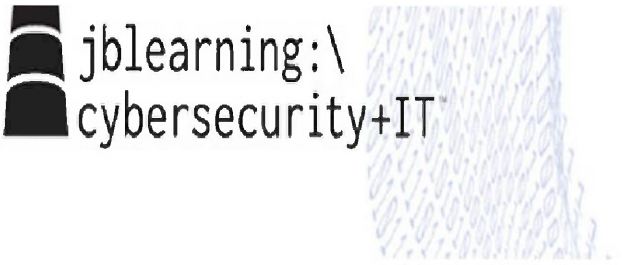
FOURTH EDITION

Python Programming in Context

Julie Anderson
Jon Anderson






jblearning:\
cybersecurity+IT



World Headquarters
Jones & Bartlett Learning
25 Mall Road
Burlington, MA 01803
978-443-5000
info@jblearning.com
www.jblearning.com

Jones & Bartlett Learning books and products are available through most bookstores and online booksellers. To contact Jones & Bartlett Learning directly, call 800-832-0034, fax 978-443-8000, or visit our website, www.jblearning.com.

Substantial discounts on bulk quantities of Jones & Bartlett Learning publications are available to corporations, professional associations, and other qualified organizations. For details and specific discount information, contact the special sales department at Jones & Bartlett Learning via the above contact information or send an email to specialsales@jblearning.com.

Copyright © 2025 by Jones & Bartlett Learning, LLC, an Ascend Learning Company

All rights reserved. No part of the material protected by this copyright may be reproduced or utilized in any form, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the copyright owner.

The content, statements, views, and opinions herein are the sole expression of the respective authors and not that of Jones & Bartlett Learning, LLC. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not constitute or imply its endorsement or recommendation by Jones & Bartlett Learning, LLC and such reference shall not be used for advertising or product endorsement purposes. All trademarks displayed are the trademarks of the parties noted herein. *Python Programming in Context, Fourth Edition* is an independent publication and has not been authorized, sponsored, or otherwise approved by the owners of the trademarks or service marks referenced in this product.

There may be images in this book that feature models; these models do not necessarily endorse, represent, or participate in the activities represented in the images. Any screenshots in this product are for educational and instructive purposes only. Any individuals and scenarios featured in the case studies throughout this product may be real or fictitious but are used for instructional purposes only.

28325-9

Production Credits
Vice President, Product Management: Marisa R. Urbano
Vice President, Content Strategy and Implementation: Christine Emerton
Director, Product Management: Matthew Kane
Director, Content Management: Donna Gridley
Manager, Content Strategy: Orsolya Gall
Content Strategist: Sara Holloway

Content Coordinator: Samantha Gillespie
Director, Project Management and Content Services: Karen Scott
Manager, Project Management: Jackie Reynen
Project Manager: Jennifer Risden
Senior Digital Project Specialist: Angela Dooley
Product Marketing Manager: Mark Adamiak
Content Services Manager: Colleen Lamy
Senior Director of Supply Chain: Ed Schneider
Manufacturing Buyer: Bob Valentine
Composition: Straive
Cover Design: Briana Yates
Media Development Editor: Faith Brosnan
Rights & Permissions Manager: John Rusk
Rights Specialist: James Fortney
Cover Image (Title Page, Part Opener, Chapter Opener): © Yuri Hoyda/Shutterstock
Printing and Binding: Gasch Printing, LLC

Library of Congress Cataloging-in-Publication Data
Names: Anderson, Julie, 1947- author. | Anderson, Jon. Python programming in context.
Title: Python programming in context / Julie Anderson, Jon Anderson.
Description: Fourth edition. | Burlington, Massachusetts : Jones & Bartlett Learning, [2025] | Includes index. |
Identifiers: LCCN 2023054476 | ISBN 9781284283211 (paperback)
Subjects: LCSH: Python (Computer program language)
Classification: LCC QA76.73.P98 M544 2025 | DDC 005.13/3–dc23/eng/20231213
LC record available at https://lccn.loc.gov/2023054476

6048

Printed in the United States of America
28 27 26 25 24 10 9 8 7 6 5 4 3 2 1

For my husband, Tom; my sons, Brian and Jon; my daughters-in-law, Silvia and Laurel; and my grandchildren, Wren, Ben, and Reyna

– JAA

For my wife, Laurel, and my kids, Wren and Reyna

– JTA



© Yuri Hoyda/Shutterstock.

Brief Contents

Preface
Acknowledgments

CHAPTER 1 Introduction to Python

CHAPTER 2 Playing by the Rules

CHAPTER 3 Codes and Other Secrets

CHAPTER 4 Introducing the Python Collections

CHAPTER 5 Bigger Data: File I/O

CHAPTER 6 Image Processing

CHAPTER 7 Data Mining: Cluster Analysis

CHAPTER 8 Cryptanalysis

CHAPTER 9 Fractals: The Geometry of Nature

CHAPTER 10 Planet Objects

CHAPTER 11 Simulation

CHAPTER 12 A Hierarchy of Shapes

CHAPTER 13 Interactive Games

Appendix A: Installing the Required Software

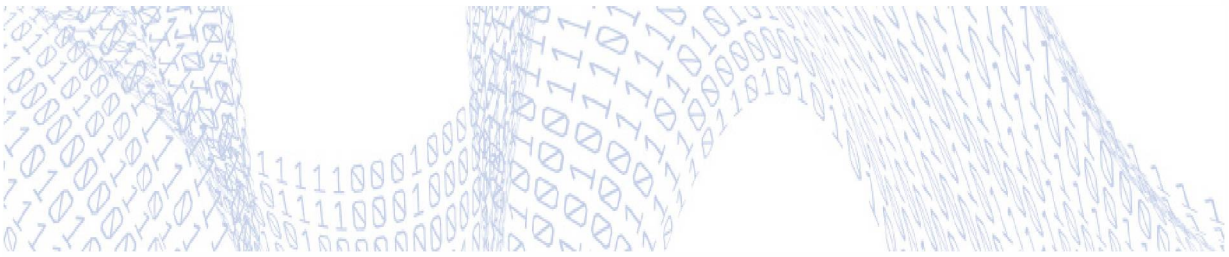
Appendix B: Python Quick Reference

Appendix C: `turtle` Reference

Appendix D: Answers to Selected Try It Out Exercises

Glossary

Index



© Yuri Hoyda/Shutterstock.

Contents

Preface
Acknowledgments

CHAPTER 1 Introduction to Python
1.1 Objectives
1.2 What Is Computer Science?
1.3 Why Study Computer Science?
1.3.1 Everyday Applications of Computer Science
1.3.2 Why Computer Science Is Important
1.4 Problem-Solving Strategies
1.5 Python Overview
1.5.1 Primitive Elements
1.5.2 Naming Objects
1.5.3 Abstraction
1.5.4 Repetition
1.6 Summary
Key Terms
Programming Exercises

CHAPTER 2 Playing by the Rules
2.1 Objectives
2.2 The Chatbot

2.3 The String Data Type
2.4 Making Decisions
2.4.1 Boolean Expressions
2.4.2 Compound Boolean Expressions and Logical Operators
2.4.3 Selection Statements
2.5 Implementing the Chatbot
2.6 Playing Games
2.6.1 Guess a Number
2.6.2 The `while` Loop
2.6.3 Printing Numbers in Strings
2.6.4 Asking for Input
2.7 Completing Guess a Number
2.8 Validating User Input
2.9 Counting the Guesses
2.9.1 The Accumulator Pattern
2.10 The Game of Nim
2.10.1 Random Numbers
2.10.2 Returning a Value from a Function
2.11 Summary
Key Terms
Programming Exercises

CHAPTER 3 Codes and Other Secrets
3.1 Objectives
3.2 The String Data Type
3.2.1 Repetition
3.2.2 Indexing
3.2.3 String Slicing
3.2.4 String Searching
3.2.5 String Methods
3.2.6 Character Functions
3.3 Encrypting and Decrypting Messages
3.4 Transposition Cipher
3.4.1 Encrypting Using Transposition
3.4.2 Decrypting a Transposed Message
3.5 Substitution Cipher
3.6 Creating a Key
3.7 The Vigenère Cipher
3.8 Summary

Key Terms
Programming Exercises

CHAPTER 4 Introducing the Python Collections

4.1 Objectives
4.2 What Is Data?
4.3 Storing Data for Processing
4.3.1 Strings Revisited
4.3.2 Lists
4.4 Calculating Statistics on Data
4.4.1 Simple Dispersion
4.5 Central Tendency
4.5.1 Mean
4.5.2 Median
4.5.3 Mode
4.6 Frequency Distribution
4.6.1 Using a Dictionary to Compute a Frequency Table
4.6.2 Computing a Frequency Table Without a Dictionary
4.6.3 Visualizing a Frequency Distribution
4.7 Dispersion: Standard Deviation
4.8 Summary
Key Terms
Programming Exercises

CHAPTER 5 Bigger Data: File I/O

5.1 Objectives
5.2 Using Files for Large Data Sets
5.2.1 Text Files
5.2.2 Iterating over Lines in a File
5.2.3 Writing a File
5.2.4 String Formatting
5.2.5 Alternative File-Reading Methods
5.3 Reading Data from the Internet
5.3.1 Using CSV Files
5.3.2 Using a `while` Loop to Read a File
5.3.3 List Comprehension
5.4 Reading and Manipulating Data with `pandas`
5.4.1 `pandas` Data Structures
5.4.2 Reading CSV Data with `pandas`

5.4.3 Data Manipulation with `pandas`
5.5 Reading JSON Data from the Internet
5.6 Correlating Data
5.7 Summary
Key Terms
Programming Exercise

CHAPTER 6 Image Processing
6.1 Objectives
6.2 What Is Digital Image Processing?
6.2.1 The RGB Color Model
6.2.2 The `cImage` Module
6.3 Basic Image Processing
6.3.1 Negative Images
6.3.2 Grayscale
6.3.3 A General Solution: The Pixel Mapper
6.4 Parameters, Parameter Passing, and Scope
6.4.1 Call by Assignment Parameter Passing
6.4.2 Namespaces
6.4.3 Calling Functions and Finding Names
6.4.4 Modules and Namespaces
6.5 Advanced Image Processing
6.5.1 Resizing
6.5.2 Stretching: A Different Perspective
6.5.3 Flipping an Image
6.5.4 Edge Detection
6.6 Summary
Key Terms
Programming Exercises

CHAPTER 7 Data Mining: Cluster Analysis
7.1 Objectives
7.2 What Is Data Mining?
7.3 Cluster Analysis: A Simple Example
7.4 Implementing Cluster Analysis on Simple Data
7.4.1 Distance Between Two Points
7.4.2 Clusters and Centroids
7.4.3 The K-Means Cluster Analysis Algorithm
7.4.4 Implementation of K-Means

7.4.5 Implementation of K-Means, Continued
7.5 Implementing Cluster Analysis: Earthquakes
7.5.1 File Processing
7.5.2 Visualization
7.6 Cluster Analysis Shortcomings and Solutions
7.7 Summary
Key Terms
Programming Exercises

CHAPTER 8 Cryptanalysis
8.1 Objectives
8.2 Introduction
8.3 Cracking the Rail Fence
8.3.1 Checking Our Work with a Dictionary
8.3.2 A Brute-Force Solution
8.3.3 A Rail Fence Decryption Algorithm
8.4 Cracking the Substitution Cipher
8.4.1 Letter Frequency
8.4.2 Ciphertext Frequency Analysis
8.4.3 Letter Pair Analysis
8.4.4 Word Frequency Analysis
8.4.5 Pattern Matching with Partial Words
8.4.6 Regular Expression Summary
8.5 Summary
Key Terms
Programming Exercises

CHAPTER 9 Fractals: The Geometry of Nature
9.1 Objectives
9.2 Introduction
9.3 Recursive Programs
9.3.1 Recursive Squares
9.3.2 Classic Recursive Functions
9.3.3 Drawing a Recursive Tree
9.3.4 The Sierpinski Triangle
9.3.5 Call Tree for a Sierpinski Triangle
9.4 Snowflakes, Lindenmayer, and Grammars
9.4.1 L-Systems and the `match` Statement
9.4.2 Automatically Expanding Production Rules

9.4.3 More Advanced L-Systems
9.5 Summary
Key Terms
Programming Exercises

CHAPTER 10 Planet Objects
10.1 Objectives
10.2 Introduction
10.2.1 Focusing on Data
10.2.2 Object-Oriented Programming
10.2.3 Python Classes
10.3 Designing and Implementing a `Planet` Class
10.3.1 Constructor Method
10.3.2 Accessor Methods
10.3.3 Mutator Methods
10.3.4 Special Methods
10.3.5 Methods and `self`
10.3.6 Details of Method Storage and Lookup
10.4 Designing and Implementing a `Sun` Class
10.5 Designing and Implementing a Solar System
10.6 Animating the Solar System
10.6.1 Using Turtles
10.6.2 Planetary Orbits
10.6.3 Implementing the Simulation
10.7 Summary
Key Terms
Programming Exercises

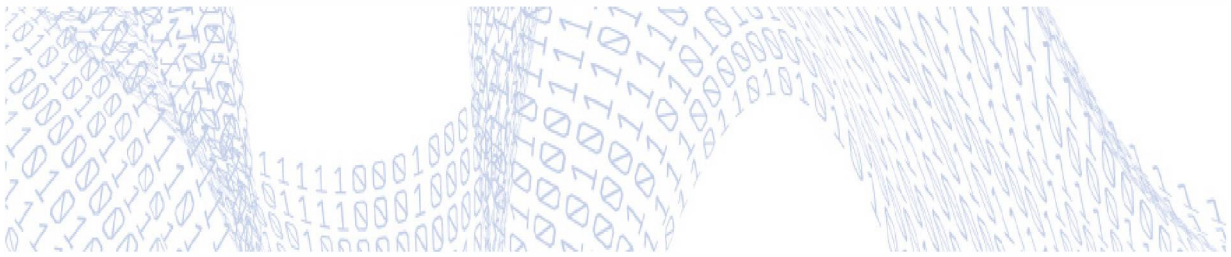
CHAPTER 11 Simulation
11.1 Objectives
11.2 Bears and Fish
11.3 What Is a Simulation?
11.4 Rules of the Game
11.5 Design
11.6 Implementation
11.6.1 The `World` Class
11.6.2 The `Fish` Class
11.6.3 The `Bear` Class
11.6.4 Main Simulation

11.7 Growing Plants
11.8 A Note on Inheritance
11.9 Summary
Key Terms
Programming Exercises

CHAPTER 12 A Hierarchy of Shapes
12.1 Objectives
12.2 Introduction
12.3 First Design
12.4 Basic Implementation
12.4.1 The `Canvas` Class
12.4.2 The `GeometricObject` Class
12.4.3 The `Point` Class
12.4.4 The `Line` Class
12.4.5 Testing Our Implementation
12.5 Understanding Inheritance
12.6 Limitations
12.7 An Improved Implementation
12.8 Implementing Polygons
12.9 Summary
Key Terms
Programming Exercises

CHAPTER 13 Interactive Games
13.1 Objectives
13.2 Introduction
13.2.1 Event-Driven Programming
13.2.2 Simulating an Event Loop
13.2.3 A Multithreaded Event Loop
13.3 Event-Driven Programming with the Turtle
13.3.1 A Simple Drawing Using Key Presses
13.3.2 Placing Turtles Using Mouse Clicks
13.3.3 Bouncing Turtles
13.4 Creating Your Own Video Game
13.4.1 The `LaserCannon` Class
13.4.2 The `BoundedTurtle` Class
13.4.3 The `Drone` Class
13.4.4 The `Bomb` Class

13.4.5 Putting All the Pieces Together
13.5 Summary
Key Terms
Programming Exercises
Appendix A: Installing the Required Software
Appendix B: Python Quick Reference
Appendix C: `turtle` Reference
Appendix D: Answers to Selected Try It Out Exercises
Glossary
Index



© Yuri Hoyda/Shutterstock.

Preface

Introduction

Computer science deals with people who have problems to solve. Algorithms help find the solutions to those problems. To be a computer scientist means first and foremost that you are a problem solver, capable of constructing algorithms either from scratch or by applying patterns from past experience.

The only way to be successful in learning computer science is through deliberate and incremental exposure to the fundamental ideas of the discipline. A beginning student of computer science needs practice to establish a thorough understanding of concepts before continuing on to the more complex parts of the curriculum. In addition, a beginner needs the opportunity to be successful and to gain confidence. As students progress through the introductory computer science sequence, we want them to focus on aspects of problem-solving, algorithm development, and algorithm understanding.

In this text, we use Python as the programming language because it has a clean, simple syntax and an intuitive user environment. The basic collections are powerful yet easy to use. The interactive nature of the language creates a place to test ideas without the need for a lot of coding. Finally, Python

provides a textbook-like notation for representing algorithms, alleviating the need for an additional layer of pseudocode. This allows the illustration of many relevant, modern, and interesting problems that make use of algorithms.

Key Features

This text is designed to be a first course in computer science that focuses on problem-solving, with language features being introduced as needed to solve the problem at hand. We have structured the text around problems of general interest, rather than a traditional language-element structure. Thus, you will not see chapter titles in this text like "Loops" and "Conditionals"—but you will see chapter titles such as "Planet Objects" and "Playing by the Rules."

Throughout the text, concepts are introduced using a spiral model. Because the syntax of Python is easy to learn, we can quickly introduce the basics of standard programming constructs. As students progress through the text, more of the details and background for these constructs are added to their toolbox. In this way, students are exposed to important computer science concepts at the point when those concepts are needed to solve a problem.

An illustration of how this spiral approach touches on a specific topic in different ways can be seen in our presentation of functions. Students begin to write functions with parameters in Chapter 1. Chapter 2 then introduces functions with return values. In Chapter 6, students learn about passing functions as parameters to other functions and the details of Python's scoping rules. In Chapter 8, they encounter Python's keyword and optional parameters. Chapter 9 covers recursive functions. In Chapter 10, students learn about writing functions that are methods of a class. In Chapter 12, students learn to write abstract methods. Finally, in Chapter 13, students are introduced to writing callback functions for event-driven programming.

Pedagogical Aids

Throughout the text, pedagogical aids emphasize Python programming constructs and good programming practices as they are introduced to solve

problems. In this way, the text can be approached both from a problem-solving view and as a Python reference.

These pedagogical aids include:

- Tables that summarize each newly introduced function or method.

TABLE 2.8 The `input` Function

Function	Description
`input(prompt)`	Outputs the prompt, then returns a string containing any characters typed by the user when the user presses enter or return. The enter or return key is not part of the returned string.

- Recap boxes that summarize important concepts.

RECAP:

A list is a sequential collection of heterogenous objects. Lists are written as commadelimited values enclosed in square brackets.

- Heads Up boxes that alert students to possible pitfalls, and provide techniques for avoiding errors.

HEADS UP:

Be careful to update the `while` loop condition in the loop body to avoid an infinite loop.

- Good Programming Practice boxes that provide problem-solving strategies as well as tips for writing readable and maintainable code.

GOOD PROGRAMMING PRACTICE:

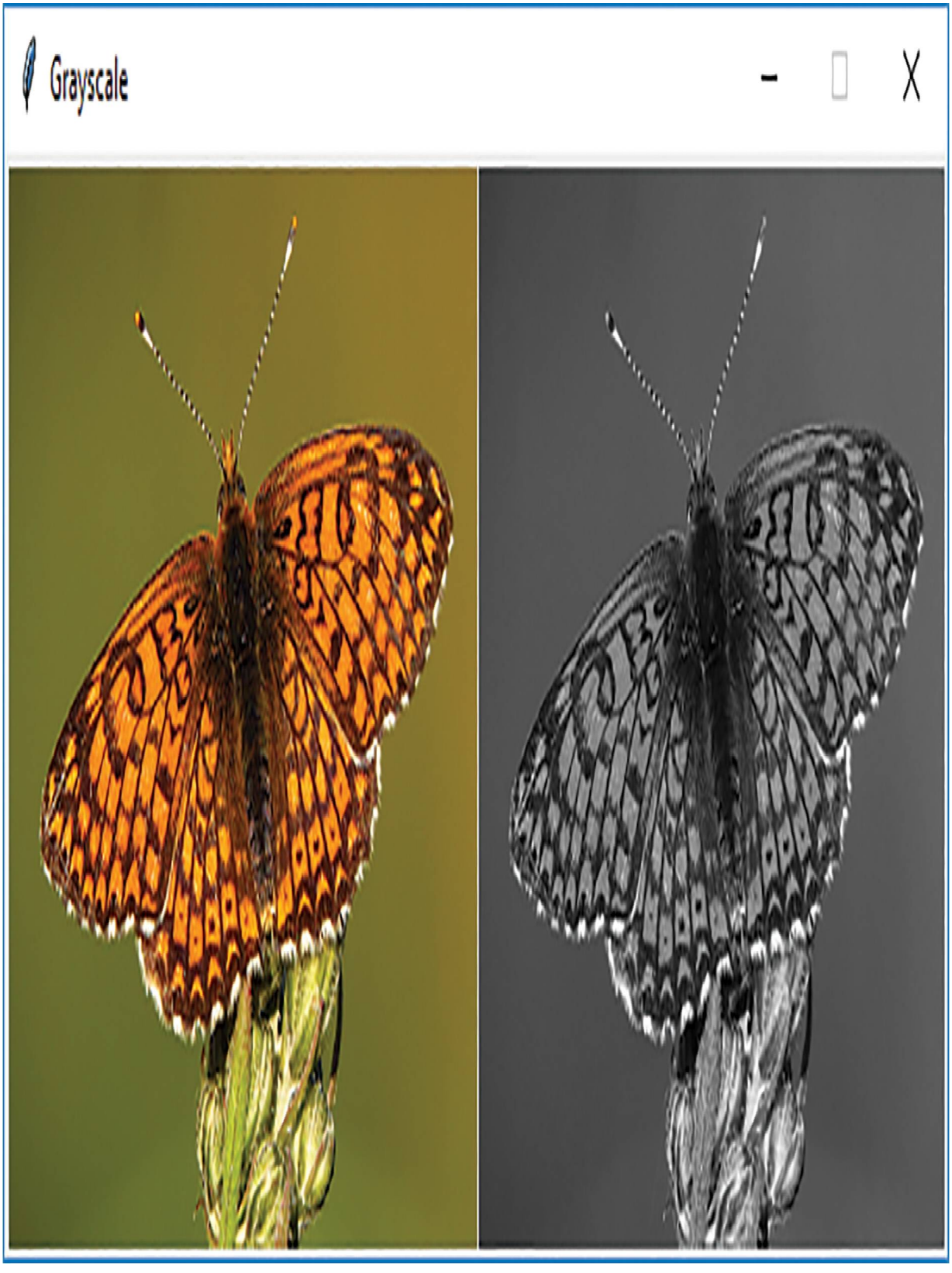
Wherever possible, verify that input from the user is valid. If not valid, allow the user to reenter the input.

- Python code that is color-coded so the student can better see syntax elements.

```
1 def getMaxSoFar(aList):
2     maxSoFar = aList[0]
3     for pos  in range(1, len(aList)):
4         if aList[pos] > maxSoFar:
5             maxSoFar = aList[pos]
6
7     return maxSoFar
```

Description

- Figures that are presented in color, which is important for several turtle applications and image-processing programs in which color is used to highlight variations in data.



© Luka Hercigonja/Shutterstock.

What's New in the *Fourth Edition*

The *Fourth Edition* contains many updates and new features found in Python version 3.10.

- In response to instructor and student feedback, we replaced Chapter 2. The new chapter, *Playing by the Rules*, covers the same programming topics as the previous Chapter 2 while solving problems that are more engaging to students. Using rules-based programming, the student develops a chatbot and teaches the computer strategies to play games. We also introduce the `while` loop here so that students can write event-controlled loops in addition to the `for` loops introduced in Chapter 1. In previous editions, `while` loop construction was spread throughout multiple chapters. Similarly, we introduce the `input` function here (previously introduced in Chapter 3) for ease in playing the games.
- For the development system, we now use the popular, freely available Jupyter Notebook. All sessions and listings in the book have been written using Jupyter Notebook.
- In Chapter 4, *Introducing the Python Collections,* we introduce the `matplotlib` library as a means to graph data.
- In Chapter 5, *Bigger Data: File I/O*, we introduce the `pandas` library for reading and manipulating online data.
- In Chapter 9, *Fractals: The Geometry of Nature*, we use the new Python `match` command to determine which production rule to apply for L-systems.
- We have also added a glossary of terms at the end of the text.

How to Use This Text

Chapter Organization

This text is organized into three parts. Chapters 1–5 introduce all of the key control structures, emphasizing straightforward imperative programming constructs such as variables, loops, and conditionals. By the end of the first

five chapters, all of the major Python data types have been covered—including integers, floats, strings, lists, dictionaries, tuples, and files.

In the first chapter, we introduce the idea of an object as something you can use. In a sense, programming in Python is an objects-always approach. Students begin by using common programming concepts while employing the modules that Python provides. These modules allow us to address more interesting problems without introducing unnecessary complexity. For example, we cover simple graphics in Chapter 1 by using a turtle graphics module. In Chapter 6, we cover image processing through the use of a simple image object that automatically loads an image from a file but allows the students to get and set the values of pixels.

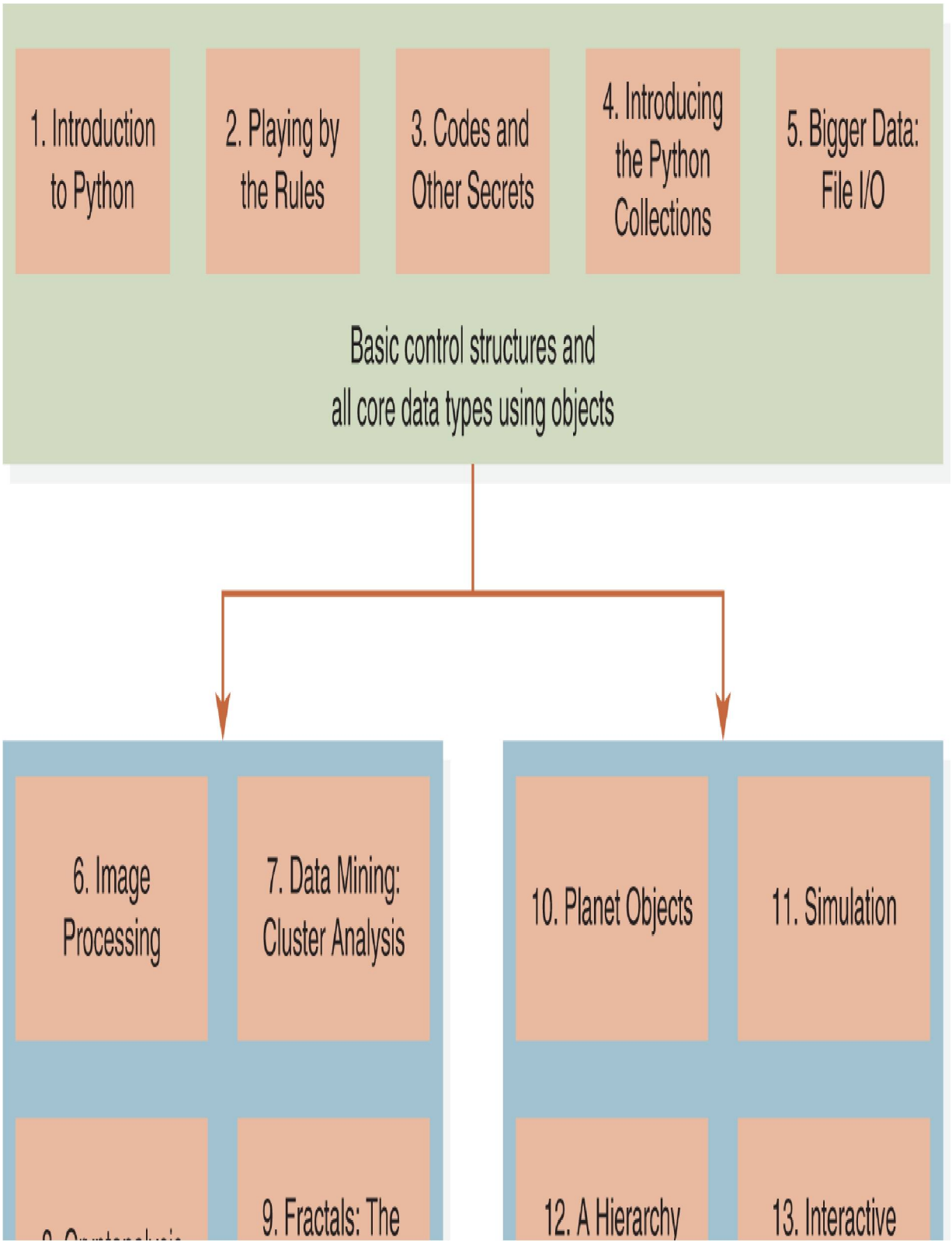
The next chapters contain more details about the concepts introduced in Chapters 1–5. Chapters 6–9 provide students with an opportunity to get more comfortable with basic programming concepts while presenting additional problem-solving patterns. Students also learn more about the internal mechanisms of Python.

The last chapters emphasize object-oriented programming and introduce the concepts needed to design and build classes. When these topics are introduced, students are comfortable with the idea of using an object; in turn, building their own objects is a natural next step. Our first examples emphasize the importance of interactions between multiple real-world objects. After implementing simple classes, we introduce inheritance in a natural way by creating a graphics library and by implementing interactive games as an extension of the turtle module.

Using the Text

This text can be used in several ways. **Figure P.1** provides an overview of some possibilities. One path is to move through the text sequentially. Instructors could cover the entire text in a single semester, or because there is enough exploratory material to last an entire year, instructors could cover the material in two semesters. A second path would be to cover Chapters 1–5 and then move to Chapters 10–13. This approach provides a balance between imperative and object-oriented programming that fits easily into a single semester. If instructors have additional time, they can pick and choose from Chapters 6–9 before continuing to Chapters 10–13. Another suggestion

would be to teach Chapters 1–9 in an introductory course that covers only imperative programming and the use of objects.



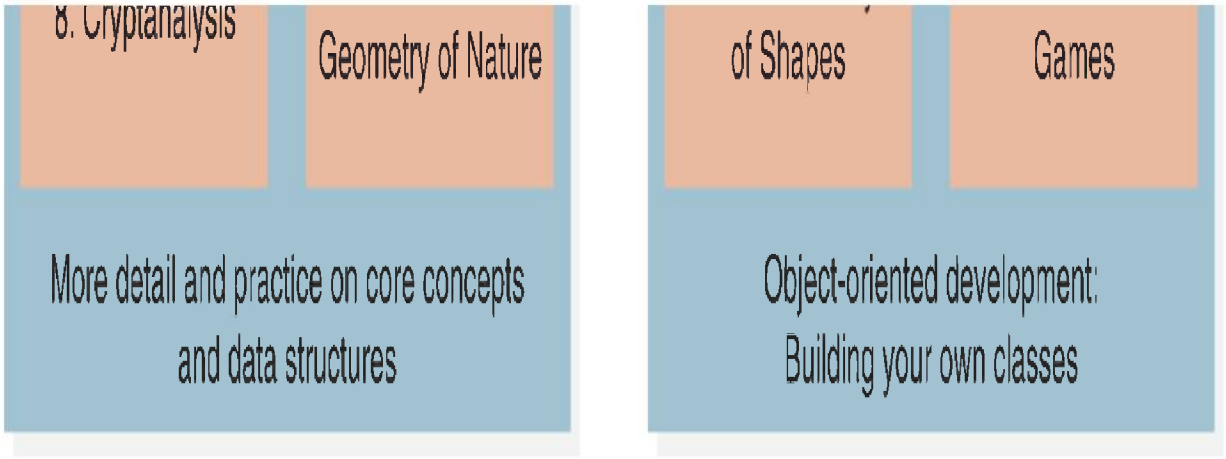


Figure P.1 Road map.

Description

Using the Exercises

This text includes three kinds of exercises. First, some Try It Out exercises ask students to run the code that has been presented as part of the section. With these exercises, students use the code to learn or explore on their own. Second, other Try It Out exercises have students modify or extend the code provided in the section. Many times, a simple version of the code is provided, and students are asked to modify the code to improve it or add additional features. Third, programming exercises at the end of each chapter describe complete projects that are related to the materials covered in the chapter.

Throughout the text, we have included Try It Out exercises within each section. These exercises are structured so that they can be used in several ways:

- If you are reading this text as a self-study, the Try It Out exercises provide breaking points where you have the opportunity to stop reading and practice the concepts.
- Instructors may find that many of the exercises contain material to cover in a lecture that complements the material the students have read.
- In a traditional lecture course, the exercises can be used as homework problems. We have had good luck in the introductory computer science

course using many small programming assignments. Often the students are able to do more than 30 small programming assignments during a single semester.

- This text can also be used as part of a "hands-on" introduction to computer science. A small amount of lecture can be used to introduce the main points of a chapter, but the majority of the class time is used to work on the exercises that complement the reading. This approach follows a growing trend in active learning computer science education and was well received by our students.

Key Topics by Chapter

Table P.1 maps the problems that are introduced in each chapter with the key computer science concepts that are established in the chapter.

TABLE P.1 Problems and Key Concepts

Chapter	Problem Key	Computer Science Concepts
1. Introduction to Python	Drawing polygons and approximating a circle Common problem-solving strategies	Problem-solving strategies, numeric data types, assignment statements, using `turtle` objects, simple `for` loops, simple functions
2. Playing by the Rules	Creating a chatbot Guessing a Secret Number Playing the game of Nim	Simple string definition and searches, Boolean expressions and simple selection statements, the accumulator patterns, `while` loops, functions that return values, user input, `random` module
3. Codes and Other Secrets	Cipher algorithms—transposition, substitution algorithms	More on the string data type and methods, character functions, modulo arithmetic, keyword parameters
4. Introducing the Python Collections	Computing simple statistics	Python list, tuple, and dictionary data types; `math` module; `statistics` module; `matplotlib` library
5. Bigger Data: File I/O	Working with data	File input and output, string formatting, reading online data in CSV and JSON formats, reading and manipulating data using the `pandas` library, list comprehension

Chapter	Problem Key	Computer Science Concepts
6. Image Processing	Digital image processing, pixel manipulations, enlarging and reducing images, edge detection algorithms	Nested loops, passing functions as parameters, scoping rules
7. Data Mining: Cluster Analysis	Cluster analysis and visualization of large data sets	More on Python list and dictionary data types, more on reading from files
8. Cryptanalysis	Methods for cracking transposition and substitution ciphers, frequency analysis	Storing nontrivial data in lists and dictionaries, string processing, regular expression module
9. Fractals: The Geometry of Nature	What fractals are; how we can use them to simulate trees, shrubs, snowflakes, and other natural objects	Recursion, grammars, and production rules; the Python `match` command
10. Planet Objects	Calculating and plotting the interaction of large celestial bodies	Introduce object-oriented design and construction of classes, instance variables, methods
11. Simulation	Simulating predator–prey relationships	Using objects in a simulation; many objects with simple behaviors can lead to interesting complex results
12. A Hierarchy of Shapes	Designing an object-oriented graphics library with points, lines, polygons, and other shapes	Inheritance, abstract classes and abstract methods, polymorphism
13. Interactive Games	Designing and implementing simple interactive games	Event-driven programming, inheritance, static variables, static methods

Supplements

Instructor supplements, including answers to chapter exercises, a test bank, and PowerPoint lecture outlines, and a sample syllabus, are available for instructor download. Source code from listings presented in the chapters is also provided for students and instructors. For more information and to request access, contact your account representative at go.jblearning.com/python4e.

Contact the Authors

If you want to ask a question or if you discover a technical error in the text, please contact Julie Anderson at JulieAustinAnderson@gmail.com. We will post any corrections on the text's website.



© Yuri Hoyda/Shutterstock.

Acknowledgments

We would like to acknowledge the contributions to this text from many partners, colleagues, and family members.

First and foremost, thank you to our publisher, Jones & Bartlett Learning, for its help and guidance through the publishing process: Ray Chew, Director of Product Management; Jennifer Risden, Project Manager; and Sara Holloway, Content Strategist. We also want to thank the compositor, Straive; the copy editor, Ellen Newman; and Briana Yates, who designed the cover.

Additional thanks go to the following reviewers for their constructive comments and suggestions:

- Nathan Backman, Buena Vista University
- Jerry Cooley, South Dakota State University
- Erica Eddy, University of Wisconsin–Parkside
- Susan Eileen Fox, Macalester College
- Harry J. Foxwell, George Mason University
- Fredrick Glenn, Gwinnett Technical College
- Carolyn Granda, Indiana University Southeast
- Zack Hubbard, Rowan-Cabarrus Community College

- Xiangdong Li, New York City College of Technology, City University of New York
- Kevin Lillis, St. Ambrose University
- Renita Murimi, Oklahoma Baptist University
- David R. Musicant, Carleton College
- Jason Myers, Wayne State University
- Michel Paquette, Vanier College
- Kate Pulling, College of Southern Nevada
- Yenumula B. Reddy, Grambling State University
- Dennis Roebuck, Delta College
- Jay Shaffstall, Muskingum University
- Christine Shannon, Centre College
- Ivan Temesvari, Oakton Community College



© Yuri Hoyda/Shutterstock.

CHAPTER 1

Introduction to Python

Introduction to Numeric Types, Turtle Graphics, Simple `for` Loops, and Functions

1.1 OBJECTIVES

- To provide examples of computer science in the real world
- To provide an overview of common problem-solving strategies
- To introduce Python's numeric data types
- To show examples of simple programs
- To introduce turtle graphics
- To introduce simple functions
- To introduce loops

1.2 What Is Computer Science?

You probably rely on computers and networks every day. Yet even though computers have become part of the backdrop of our lives, you may not have explored the world of programming.

The important question in this chapter is "What is computer science?" The problem with answering this question is that the term is misleading right from the start. You might think computer science is the exploration and discovery of how a computer works. That makes little sense given that computers are manufactured by humans. According to a quote often attributed to the early computer scientist Edsger Dijkstra, "Computer science is no more about computers than astronomy is about telescopes."

Computer science is not like biology or physics, in which we are trying to understand how things work. We know how they work; what we study is ways to make them do more and different tasks. So, what is computer science? **Computer science** is the study of **algorithms**.

To put it another way, computer science is primarily about problem-solving and computational processes. For beginning computer scientists, finding the solution to a problem is often the easiest part. Defining an algorithm, which involves turning the solution into a set of step-by-step instructions that can be performed by a computer (creating a computational process), can be difficult. Computer scientists often call this set of instructions a **program**. You may think of it as something like a recipe for a beginning cook. First, bring the water to a boil, then add the macaroni, and so on.

You are likely familiar with the use of advanced programs, such as Apple's Siri or the Google Assistant, that are designed to make the computer look intelligent. It is important to dispel this idea right away: Computers are not intelligent. Here are six important things to remember about computers as you learn to program:

1. Computers are limited.
2. Computers can follow instructions that we give them.

3. Computers do what you tell them to do really fast, so they appear smart (but they are not).
4. Computers don't remember anything unless you tell them how and what to remember.
5. Computers take your instructions literally. If you tell them to do something dumb, they will do it.
6. Computers do only what they are told and in exactly the order you tell them.

1.3 Why Study Computer Science?

Now that you have a better idea of what computer science is, you may be wondering why you should learn any more about it. Our belief is that computer science is for everyone. Some of the biggest computer success stories involve people who were not computer scientists by training, but rather came to computer science because they had an interesting problem to solve. For example, a physicist at the European Council for Nuclear Research (CERN) needed a better way for physicists around the world to share information. The solution to this information-sharing problem became what is called the World Wide Web. He wrote the computer programs for the first web server and browser. In 2016, that physicist, Sir Tim Berners-Lee, was awarded the Association of Computing Machinery's A. M. (Alan M.) Turing Prize, often referred to as the computing equivalent of the Nobel Prize.

1.3.1 Everyday Applications of Computer Science

Few computer scientists work on problems limited to computers, such as building a better operating system or improving a local area network. Instead, most computer scientists work with people, writing programs that solve problems in areas such as biology, chemistry, business, economics, publishing, automotive design, or entertainment. Look around you and think about the applications you use many times each day—from your browser to an instant messaging program and emails, from word processing to your fitness program, and from your personal calendar to games.

In addition to our laptop computers, tablets, and cell phones, we use computers daily that we may not even realize are computers. For example, the computers in your car check your gas mileage several times every second, examine the wear on your brake pads, monitor your speed and emissions, and update your Global Positioning System (GPS) display to keep track of your exact position and plot it on a map on the dashboard display. More and more appliances have computer-aided functions.

If you go to the doctor and need medical imaging such as a computed axial tomography (CAT) scan or magnetic resonance imaging (MRI), you are relying on sophisticated computer programs to create and interpret the images of your body. If you peek into the cockpit of a newer airplane, you will see panels of displays with virtual switches. In addition, the pilot of the airplane has spent many hours training in a simulated environment controlled by a computer.

Some computer programs run behind the scenes to make the world a more orderly place. As a student, you probably know that a computer keeps track of your credit at the cafeteria, your grade-point average, and the amount of money you owe the college this month. The computers in the library keep track of which books the library owns, who has books checked out, and when they are due back in the library.

An important branch of computer science is **data science**, or the use of computational algorithms and scientific processes to gain understanding of or find patterns in large data sets. Nearly all big businesses, university researchers, and even government agencies use data science in some way to gain insights. Within data science is an emerging area called **machine learning**, where a computer program is fed lots of data showing both correct and incorrect results, and the program uses sophisticated algorithms to "learn" to distinguish the correct results from the incorrect results. For example, some companies, including Intel and Verily Life Sciences, are using machine learning to help diagnose diseases.

In fact, you would be hard-pressed to name an area of your life that is not assisted by computers.

1.3.2 Why Computer Science Is Important

In one sense, studying computer science is important because it gives you a better understanding of how the technological world around you works. After reading this text, you may have a new appreciation for the applications that you use every day. We hope you will even aspire to write your own new application or an improved version of some existing application.

In another sense, computer science helps to prepare you for your future. Many of the most interesting jobs in the near future will be at the intersection of computer science and some other domain. A manager at a successful

software company said, “I sometimes turn away applicants who have perfect scores on the standardized tests.” These applicants may be technically perfect, he explained, but they lack the ability to communicate with people and solve real-world problems.

Most importantly, a first course in computer science will help you in the following ways:

- You will be able to apply new problem-solving skills.
- You will learn to apply logic.
- You will learn about process (a series of actions or steps taken to achieve a specific outcome).
- You will understand and apply abstraction.
- You will learn to think and communicate more clearly.

1.4 Problem-Solving Strategies

Problem-solving happens on three levels:

- *Strategy:* A high-level idea for finding a solution.
- *Tactics:* Methods or patterns that work in many different settings.
- *Tools:* Tricks and techniques that are used in specific situations.

As you progress through this text, you will encounter several different problem-solving strategies. In addition, you will see that computer science uses many different problem-solving tactics. In particular, you will learn to recognize patterns in the problems you solve that lead to patterns in the programs you write. Finally, as you use the Python programming language, you will learn about the tools that Python provides to help you write your solution as a program.

A simple example will illustrate some of these concepts. Consider this question: "A class has 12 students. At the beginning of class, each student shakes hands with each of the other students. How many handshakes take place?"

Your first instinct might be to simply say that since each person must shake hands with 11 other people, the answer is $12 \times 11 = 132$ handshakes. But you would be wrong. To help you make progress toward the correct answer, you can employ a strategy called simplification. **Simplification** is a strategy that reduces a problem to a trivial size.

Let's assume that instead of 12 people, only one person is in the classroom. With only a single person, no handshakes will take place. But what happens when a second person enters the classroom? Upon entering the room, the second person must shake hands with the first (and only other) person in the room for one handshake. Now suppose a third person enters the classroom. The third person must shake hands with the first two, making a total of $2 + 1 + 0 = 3$ handshakes. The fourth person who enters the room must shake hands with the three people already in the room, so our total handshake count is now $3 + 2 + 1 + 0 = 6$. By this time, you should see a

pattern of **generalization**—a technique that enables you to go from some specific examples to a solution that can be implemented as a program.

In our handshaking problem, the pattern tells us that the *N*th person to enter the classroom shakes hands with $N - 1$ other people, and the total number of handshakes is the sum $1 + 2 + 3 + \ldots N - 1$. At this point, we might simply write a computer program that adds up the numbers from 1 to $N - 1$. Adding numbers is something that computers are particularly good at. But we will also point out that a general mathematical solution exists for this problem of adding a sequence of numbers:

$$sum = \frac{n \cdot (n+1)}{2}$$

For our handshake problem, we need to add up the numbers from 1 to 1 less than the number of students, because as we discovered earlier, there are no handshakes with only one student. Given that there are 12 students, $n = 11$. Plugging 11 into the formula gives us:

$$\frac{11 \cdot 12}{2} = 66$$

You can verify this result yourself by simply adding the numbers from 1 to 11.

In fact, we can *prove* that the formula gives us the correct answer by using **representation**, another important strategy that will solve our problem. Proving that

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$

is true for all values of *n* using mathematical induction would be a daunting task for most people. However, let's visualize the problem of adding up the numbers from 1 to *N* as shown in **Figure 1.1**.

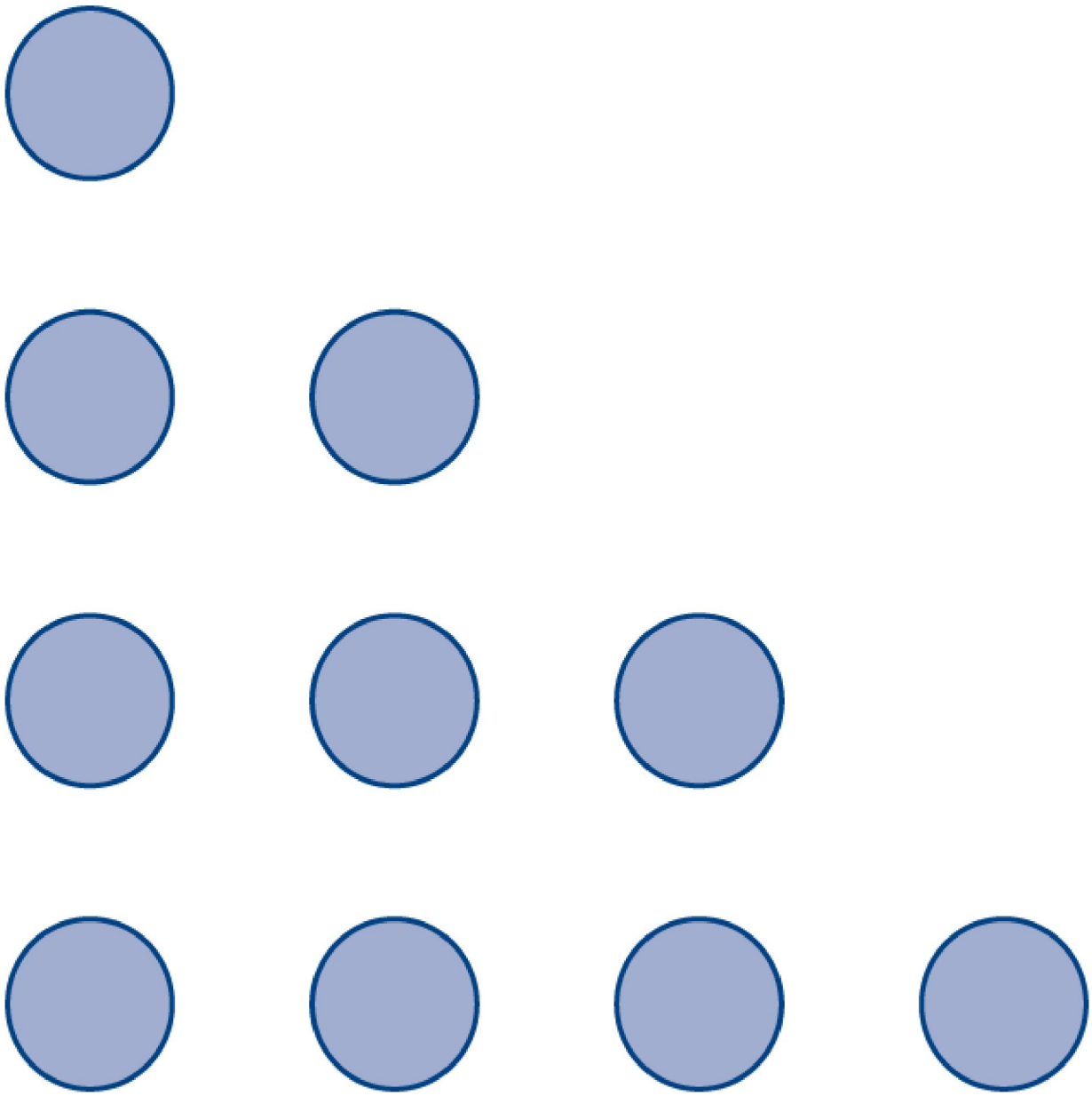


Figure 1.1 Representing the sum of the numbers from 1 to *N* graphically.

Description

In this representation of this slightly simplified problem, we show each of the numbers we want to add as a row of circles, thus representing the addition of 1 + 2 + 3 + 4. Now we could just count the circles to get our answer, but that exercise is not very interesting and does not prove anything. The interesting part comes in **Figure 1.2**, where we have taken all four rows of dots, duplicated them, and flipped them diagonally. The dots now form a

rectangle that is 4 rows high and 5 columns wide. It is now easy to see that the total number of dots is just $4 \times 5 = 20$. But we have twice as many dots as we started with, so the number of dots we started with is $20 \div 2 = 10$.

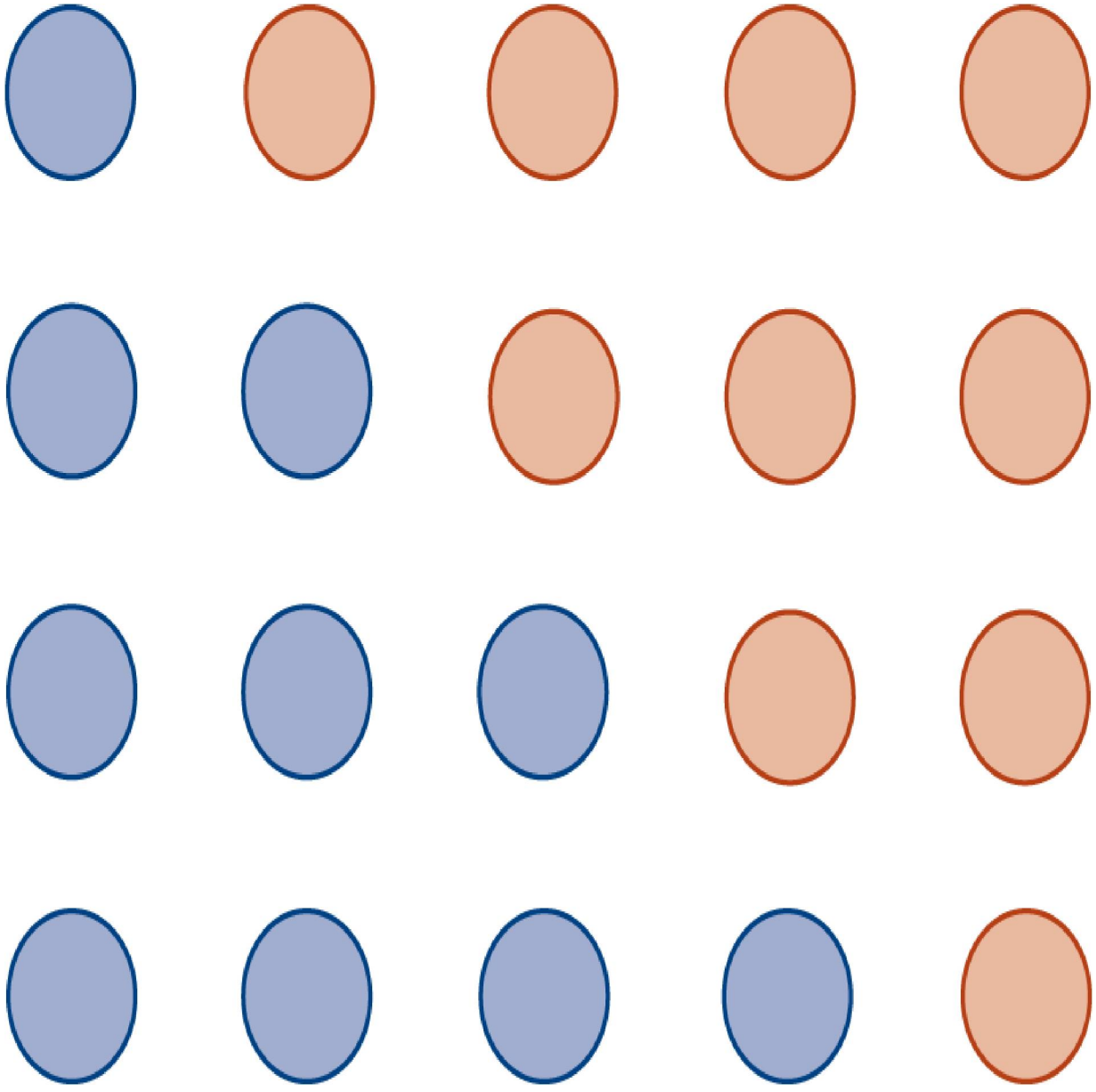


Figure 1.2 The sum of the numbers 1 to N is $\frac{n \cdot (n+1)}{2}$.

Description

If you generalize our example a little bit, it is easy to see that this graphical trick works no matter how many rows of dots we use. Therefore, we have shown a proof for an interesting mathematical sequence, but because we chose a good representation for the problem, we have not had to do anything more complicated than simple multiplication and division.

1.5 Python Overview

In this text, the language you will use to write your computer programs is called **Python**. Why did we choose Python instead of a language like C++ or Java? The answer is simple: We want you to focus on learning the problem-solving strategies and techniques that a computer scientist uses. **Programming languages** are tools, and Python is a good beginning tool. Languages like Java and C++ are fine tools as well, but they require you to keep track of many more details than Python does.

The best way to learn Python is to try it out—so let's get started. Python is free to download and install. You will find detailed instructions on installing Python on various operating systems at www.python.org. In fact, python.org is a good resource for learning more about Python and its many **libraries**, which are prewritten Python code we can incorporate into our programs.

After installing Python, the first thing you do is start the Python interpreter. Depending on your operating system, there are any number of ways to do this. For example, you might start a program that is installed with Python called IDLE—named after Eric Idle of Monty Python fame—that also stands for Python's *I*ntegrated *D*evelopment and *L*earning *E*nvironment. In this text, we will be using Jupyter Notebook, a popular development platform for writing and running Python programs from a browser window. Instructions for downloading and getting started with Jupyter Notebook are in Appendix A. **Figure 1.3** shows Jupyter Notebook after we have pressed the *new* button and then chosen Python 3. Notebook is now ready to accept Python commands.

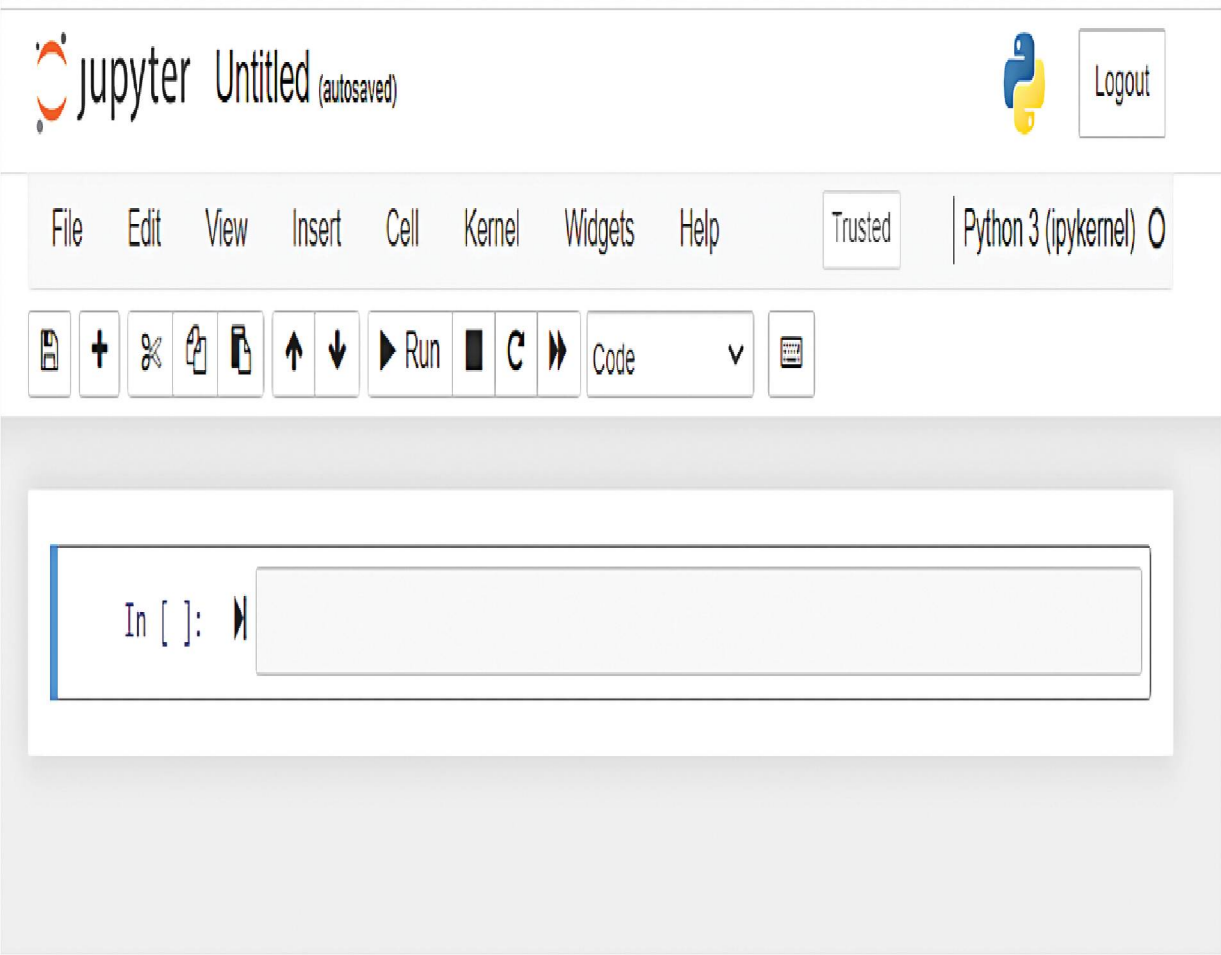


Figure 1.3 Jupyter Notebook.

© Jupyter.

As you progress through this chapter, you will see that example programs appear in boxes called *listings*, and commands that you can type interactively into Jupyter Notebook or whichever development platform you are using appear in boxes called *sessions*. Whenever you see a session box, we strongly encourage you to try the session for yourself. Also, once you have typed in the example shown in the session, feel free to try some variations to find out for yourself what works and what does not.

As we begin to explore Python, we will answer three important questions you should ask about any programming language:

- What are the primitive elements?
- How can we combine the primitive elements?

- How can we create our own abstractions?

1.5.1 Primitive Elements

At the deepest level, the one primitive element in Python is the **object**. Like you, Python thinks of the things in its world as objects. If you look around your world, you will see many objects: pencils, pens, your chair, a computer. Python even considers numbers to be objects—an idea that may be a bit confusing to you, because you probably do not think of numbers as objects. But Python does, and we will see why this is important shortly.

Python classifies the different kinds of objects in its world into **types**. Some of the easiest types to work with are numbers. Python knows about several different types of numbers:

- Integers
- Floating-point numbers
- Complex numbers

Integers

Integers are the whole numbers that you learned about in math class. We can do a lot with Python just using integers. For starters, we can use Python as a calculator. Notice that Jupyter Notebook provides a box where you can type an expression. Let's try a few mathematical expressions. Type the commands shown in **Session 1.1** using Jupyter Notebook. After you have typed in an expression, press the *Run* button. The output will be printed under the box.

```
In [1]: 2 + 2
```

```
Out[1]: 4
```

```
In [2]: 100 - 75
```

```
Out[2]: 25
```

```
In [3]: 7 * 9
```

```
Out[3]: 63
```

```
In [4]: 14 // 2
```

```
Out[4]: 7
```

```
In [5]: 15 // 2  #integer division truncates the remainder
```

```
Out[5]: 7
```

```
In [6]: 15 % 2   #to get remainder, use modulo operator
```

```
Out[6]: 1
```

Session 1.1 Simple integer math

Description

The examples in Session 1.1 illustrate some very important Python concepts that you should become familiar with as soon as possible. The first concept is Python's *read–eval–print* loop.

At a high level, the Python interpreter is really very simple. It does three things over and over again: (1) read, (2) evaluate, and (3) print. These are illustrated in **Figure 1.4**.

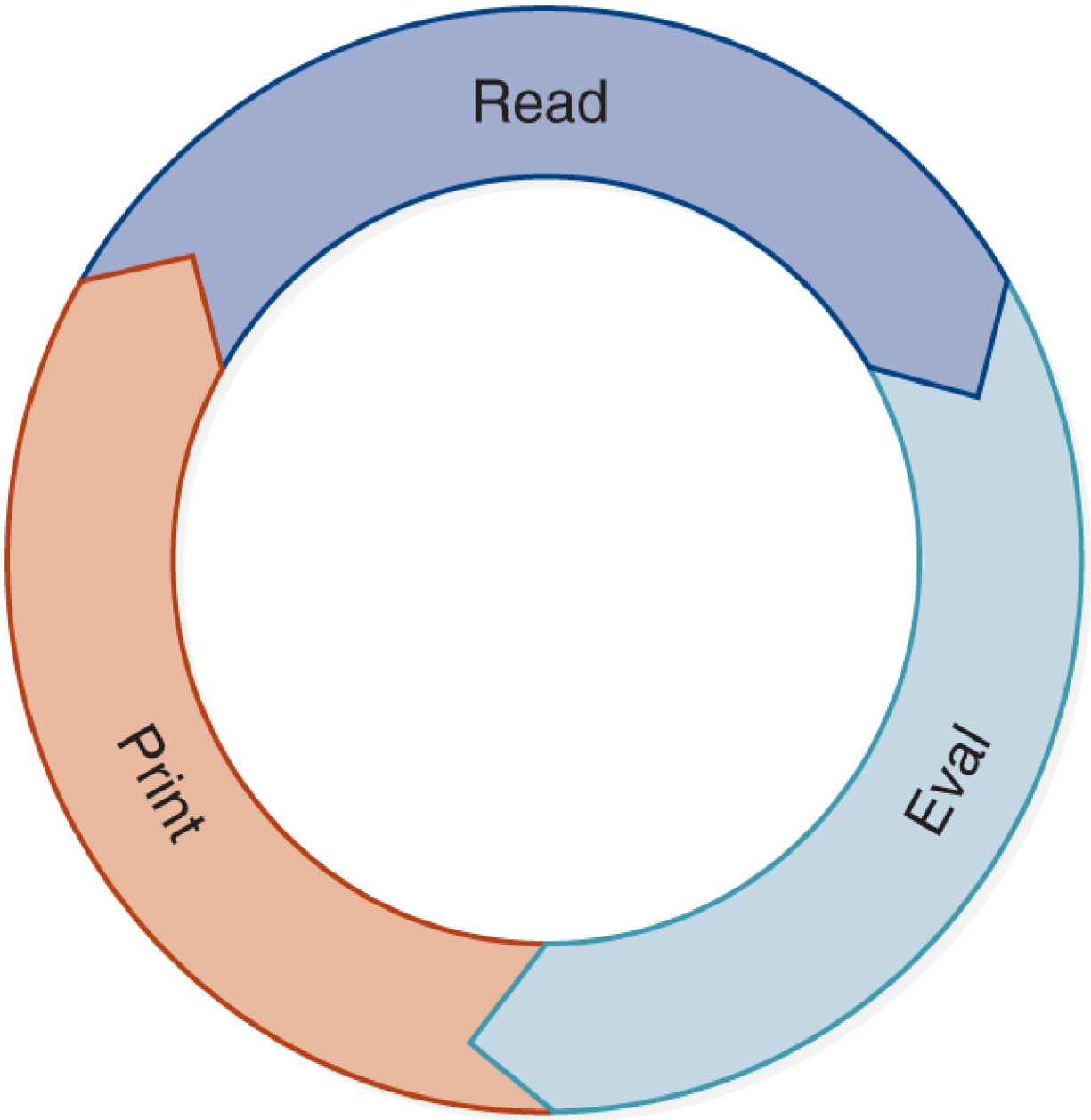


Figure 1.4 The read–eval–print loop in Python.

First, Python *reads* one line of input. In the first example, Python reads 2 + 2, then it *evaluates* the expression 2 + 2 and determines that the answer is 4. Finally, Python *prints* the resulting value of 4. After displaying the result, Python displays another box to show you that it is waiting to read another expression.

In general, a Python **expression** is a combination of operators and operands. **Table 1.1** shows Python's arithmetic operators.

TABLE 1.1 Python's Arithmetic Operators

Operator	Operation
+	Addition
−	Subtraction
*	Multiplication
//	Integer division
/	Floating-point division
%	Modulo (remainder after division)
**	Exponentiation

One element to note in Session 1.1 is the hash character (#) that appears on two lines. This hash character starts a **comment**. The Python interpreter ignores any text following the hash character until the end of the line. Thus, comments allow the programmer to place descriptive documentation into Python code that will not impact the execution of the program. You will see comments in many of our sessions.

When we evaluate an expression, we get a result. In the examples in our Python session, some of the operators are familiar mathematical operators: + and −. Although you may be more familiar with × and ÷ for multiplication and division, you will not find those symbols on a standard keyboard, so Python, like almost all programming languages, uses "*" for multiplication and "/" or "//" for division.

One thing that may surprise you in the example is the result of the expression 15 // 2. Of course, we all know that 15 divided by 2 is really 7.5.

However, because both operands are integer objects and // is the integer division operator, Python produces an integer object as a result. Integer division works like the division you learned when you were young: 15 divided by 2 equals 7, remainder 1. You can find the remainder part of the result by using the modulo operator (%). For example, 15 % 2 evaluates to the remainder value of 1.

HEADS UP:
When using the integer division operator (//), the quotient will not include the remainder. To get the remainder, use the modulo operator (%).

Let's try some more calculations. Create a new Notebook session and type the expressions in **Session 1.2**.

```
In [1]: 12 * 6 + 2      #multiply, then add

Out[1]: 74

In [2]: 2 + 12 * 6      #multiply, then add

Out[2]: 74

In [3]: (2 + 12) * 6    #add, then multiply
```

```
Out[3]: 84

In [4]: 12 * (6 + 2)    #add, then multiply

Out[4]: 96

In [5]: 2 ** 3          #2 to the 3rd power

Out[5]: 8
```

Session 1.2 More numeric calculations

Description

Here we are mixing multiplication and addition. As shown in the first two calculations, the order in which we type the expressions does not make a difference in the result. The reason is that Python performs all multiplications and divisions before performing any additions and subtractions. We can override those rules by putting parentheses around any calculation we want to perform first, as shown in the third and fourth expressions.

Finally, we introduce the exponentiation operator (**) for raising a number to a power. In this case, we raise 2 to the third power, yielding 8.

Table 1.2 summarizes the order in which Python performs calculations. Operators are evaluated in order from top to bottom; operators at the same level are evaluated from left to right as they occur in the expression, except for exponentiation, which is evaluated from right to left.

TABLE 1.2 Python's Order of Operations

Operators	Operations	Evaluation Order
( )	Parentheses	Left to right
**	Exponentiation	Right to left
* // /	Multiplication and division	Left to right
+ −	Addition and subtraction	Left to right

Try It Out

1.1 Find the sum of the numbers 8, 9, and 10.

1.2 Find the product of the numbers 8, 9, and 10.

1.3 Compute the number of seconds in a non-leap year.

1.4 Compute the number of inches in 1 mile.

1.5 Compute the number of 2-ft^2 tiles to cover the floor of a 10- by 12-ft room.

1.6 Compute the number of handshakes required for each person in your class to shake hands with every other person exactly one time.

1.7 Find the average age of five people around you using integer division. Double-check your answer.

1.8 Compute the factorial of 13.

1.9 Compute 2 to the 120th power.

Floating-Point Numbers

Integer division is useful in some cases, but it can also trip you up. What if you want to divide 15 by 2 and get 7.5 as the answer? To get the result as a floating-point number, you must use the floating-point division operator (/).

In Python, you can tell the difference between a floating-point number and an integer because a floating-point number has a decimal point. **Floating-point numbers** are Python's approximation of what you called real numbers in math class. We say that floating-point numbers are an approximation because, unlike real numbers, floating-point numbers cannot have an infinite

number of digits following the decimal point. This is because of the way floating-point numbers are stored in the computer's memory. **Session 1.3** presents some examples of calculations using floating-point numbers.

```
In [1]: 2.0 + 2.0

Out[1]: 4.0

In [2]: 15.3 / 2.5

Out[2]: 6.12

In [3]: 6.0 * 1.5

Out[3]: 9.0

In [4]: 2.5 ** 25.0

Out[4]: 8881784197.001253

In [5]: 2.0 ** 500.0
```

```
In [5]: 2.0 ** 500.0

Out[5]: 3.2733906078961418e+150

In [6]: 1.33e+5 + 1.0

Out[6]: 133001.0
```

Session 1.3 Floating-point math

Description

Notice that when the operands are floating-point numbers, the result is a floating-point number, including the remainder as a fractional part. You should also notice that when the result of a floating-point operation gets really big, Python uses scientific notation to express the results. Finally, notice that you can use scientific notation for a floating-point number in a Python expression.

Try It Out

1.10 Find the average age of five people around you using floating-point division. Double-check your answer.

1.11 Find the volume of a sphere with a radius of 1 using the formula $4/3\pi r^3$.

1.12 Compute 1/3 of 15. Did you get the right answer?

1.13 The Andromeda galaxy is 2.9 million light years away. There are 5.878×10^{12} miles per light year. How many miles away is the Andromeda galaxy?

1.14 How many years would it take to reach the Andromeda galaxy if you traveled at a speed of 65 miles per hour?

Although 3.273e + 150 is a good approximation, we know that there are not really 147 zeros in the result. One of the disadvantages of using scientific notation is that you lose some *precision* in your result. If you want to get very exact results, integers allow us to perform calculations to virtually unlimited precision. **Session 1.4** shows the real value of 2 **500 using integers.

```
In [1]: 1 2 ** 500

Out[1]: 327339060789614187001318969682759915221664204604306478948329136809613379640467455488327009232590415715088668412756007100921725654588539305332852758937 6

In [2]: 1 100000000 * 500  #100 million * 500

Out[2]: 50000000000

In [3]: 1 100_000_000 * 500  #_ can be a separator for large numbers

Out[3]: 50000000000
```

Session 1.4 The use of integers to obtain very precise answers for large numbers

Description

When typing large integers, you might be tempted to insert commas between groups of 3 numbers—for example, 100,000,000. Unfortunately, Python does not allow commas to be inserted into integers. Session 1.4 shows that Python does allow an underscore (_) to act as a visual separator for typing large numbers. The underscore is not considered part of the integer.

Complex Numbers

The final primitive numeric type in Python is the **complex number**. As you may remember, complex numbers have two parts: a real part and an imaginary part. In Python, a complex number is displayed as *real* + *imaginary* j. For example, 5.0 + 3 *j* has a real part of 5.0 and an imaginary part of 3. Although we mention complex numbers here to give you a complete list of the numeric primitives, we will concentrate only on integers and floating-point numbers.

Mixing Numeric Types

What happens when we mix numbers of different types in one calculation? Let's look at the examples shown in **Session 1.5** to find out.

```
In [1]: 1 100 * 3.4  #mix of integers and float -> float result
```

```
Out[1]: 340.0
```

```
In [2]: 1 100 + 3.4
```

```
Out[2]: 103.4
```

```
In [3]: 1 100 - 3.4

Out[3]: 96.6

In [4]: 1 100 ** 3.4

Out[4]: 6309573.44480193

In [5]: 1 1000 / 10.2    #division is a special case

Out[5]: 98.03921568627452

In [6]: 1 1000 // 10.2

Out[6]: 98.0
```

Session 1.5 Mixing integers and floating-point numbers

Description

As you can see from Session 1.5, when you mix floating-point numbers with integers, the result will be a floating-point number.

Note that for division, the result depends on which division operator you use. When you use the floating-point division operator (/), regardless of whether the operands are integers or floating-point numbers, the result will be a floating-point number. When you use one or more floating-point numbers with the integer division operator (//), the result is also a floating-point number, but with the fractional part truncated.

HEADS UP:
When you use the floating-point division operator (/), the result will always be a floating-point number. When you use the integer division operator (//) with one or both operands being floating-point numbers, the result will be a floating-point number with the fractional part shown as 0.

As shown in **Session 1.6**, you can also tell Python to explicitly convert a number to either an integer or a floating-point number by using the **int** or **float** **functions**. These functions are built-in Python code that operate on a value you put into parentheses after the function name. For example, `float(5)` will convert the integer 5 to the floating-point number 5.0. Conversely, `int(3.99999)` will convert the floating-point number 3.99999 to the integer 3. When converting floating-point numbers to integers, Python always truncates the fractional part of the number. If you want to convert a floating-point number to the nearest integer, you can use the built-in **round** function. For example, `round(3.99999)` will round the floating-point number 3.9999 to 4.

```
In [1]: float(5)

Out[1]: 5.0

In [2]: int(3.99999)

Out[2]: 3

In [3]: round(3.99999)

Out[3]: 4
```

Session 1.6 Converting between integers and floating-point numbers

Description

These numeric-type conversion functions are summarized in **Table 1.3**.

TABLE 1.3 Python's Functions for Converting Between Numeric Types

Function	Description
`float(i)`	Converts the integer `i` to a floating-point number.
`int(f)`	Converts the floating-point number `f` to an integer. Any fractional part is truncated.
`round(f)`	Converts the floating-point number `f` to the closest integer.

In summary, we have seen that Python supports several types of primitive objects in the number family: integers, for ordinary simple math, when precision is required, or when dealing with very large numbers, and floating-point numbers, for working with scientific applications. At this point, however, Python is nothing more than a calculator. In the next section, we add some Python primitives that will give us a lot more power.

1.5.2 Naming Objects

Very often we have an object that we would like to remember. Python allows us to name objects so that we can refer to them later. For example, we might want to use the name *pi* rather than the value 3.14159 in a mathematical expression. We might also want to give a name to a value that we will use multiple times rather than recalculate it each time.

In Python, we can name objects using an **assignment statement**. A **statement** is like an expression, except that it does not produce a value for the read–eval–print loop to print. An assignment statement has three parts:

1. The left-hand side
2. The right-hand side
3. The assignment operator (=)

The right-hand side can be any Python expression, and the left-hand side is the name to which we want to assign the result of evaluating the expression.

```
name = python expression
```

When the Python interpreter evaluates an assignment statement, it first evaluates the expression that it finds on the right-hand side of the equals sign. Once the right-hand side expression has been evaluated, the resulting object is referred to using the name found on the left side of the equals sign. In computer science, we call these names **variables**. More formally, we define a variable to be a named **reference** to a data object. In other words, a variable is simply a name that allows us to locate a Python object.

Suppose we want to calculate the volume of a cylinder for which the radius of the base is 8 cm and the height is 16 cm. We will use the formula *volume = area of base * height*. Rather than calculate everything in one big expression, we will divide the work into several assignment statements.

First, we will name the numeric objects “pi,” “radius,” and “height.” Next, we will use the named objects to calculate the area of the base and, finally, the volume of the cylinder. **Session 1.7** shows how we use this sequence of assignment statements and Python arithmetic to solve our problem.

```
In [1]: pi = 3.14159

In [2]: radius = 8.0

In [3]: height = 16

In [4]: baseArea = pi * radius ** 2

In [5]: baseArea

Out[5]: 201.06176

In [6]: cylinderVolume = baseArea * height

In [7]: cylinderVolume

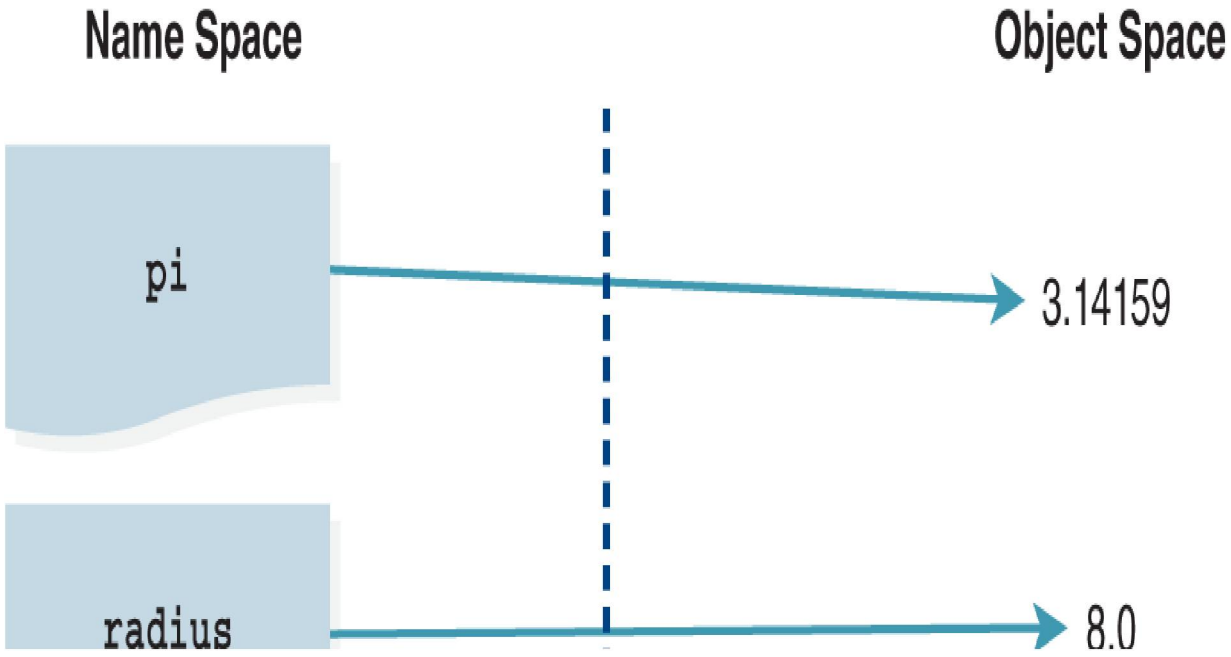
Out[7]: 3216.98816
```

Description

After studying Session 1.7, you may have some questions:

- How is the use of the equals sign in Python different from what you learned in math class?
- If you change the value for `baseArea`, will `cylinderVolume` automatically change?
- Why doesn't Python print out the value of `pi` after the first assignment statement?
- Which names are legal in Python?

Let's look at these questions one at a time. The equals sign in Python is very different from the symbol that you learned in math class. In fact, you should think of it not in terms of equality, but rather as the assignment operator, which has the job of associating a name with an object. **Figure 1.5** illustrates how names are associated with objects in Python. All the names and objects in this figure come from Session 1.7. The relationships between names and the objects they reference are indicated by the arrows between them.



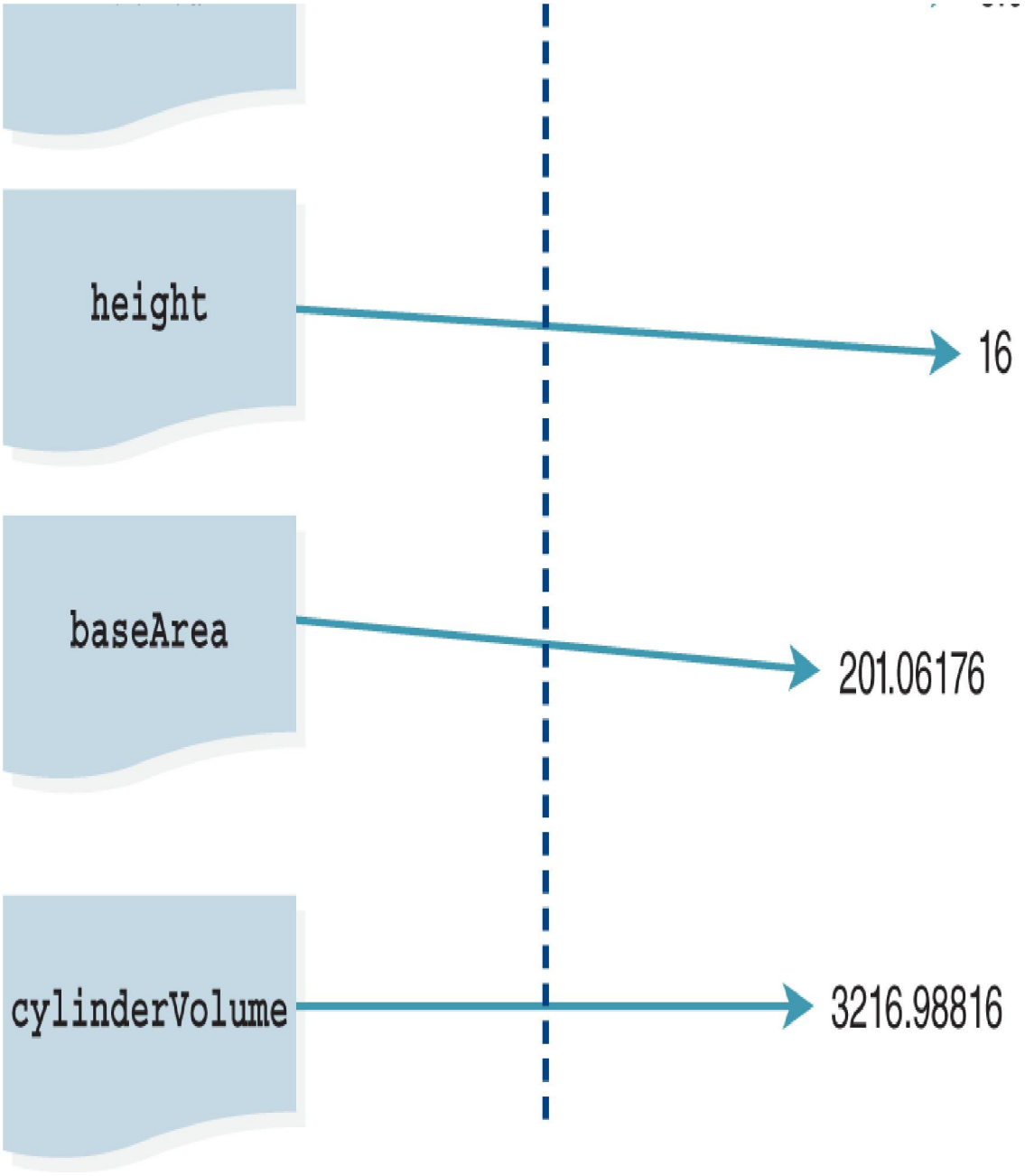
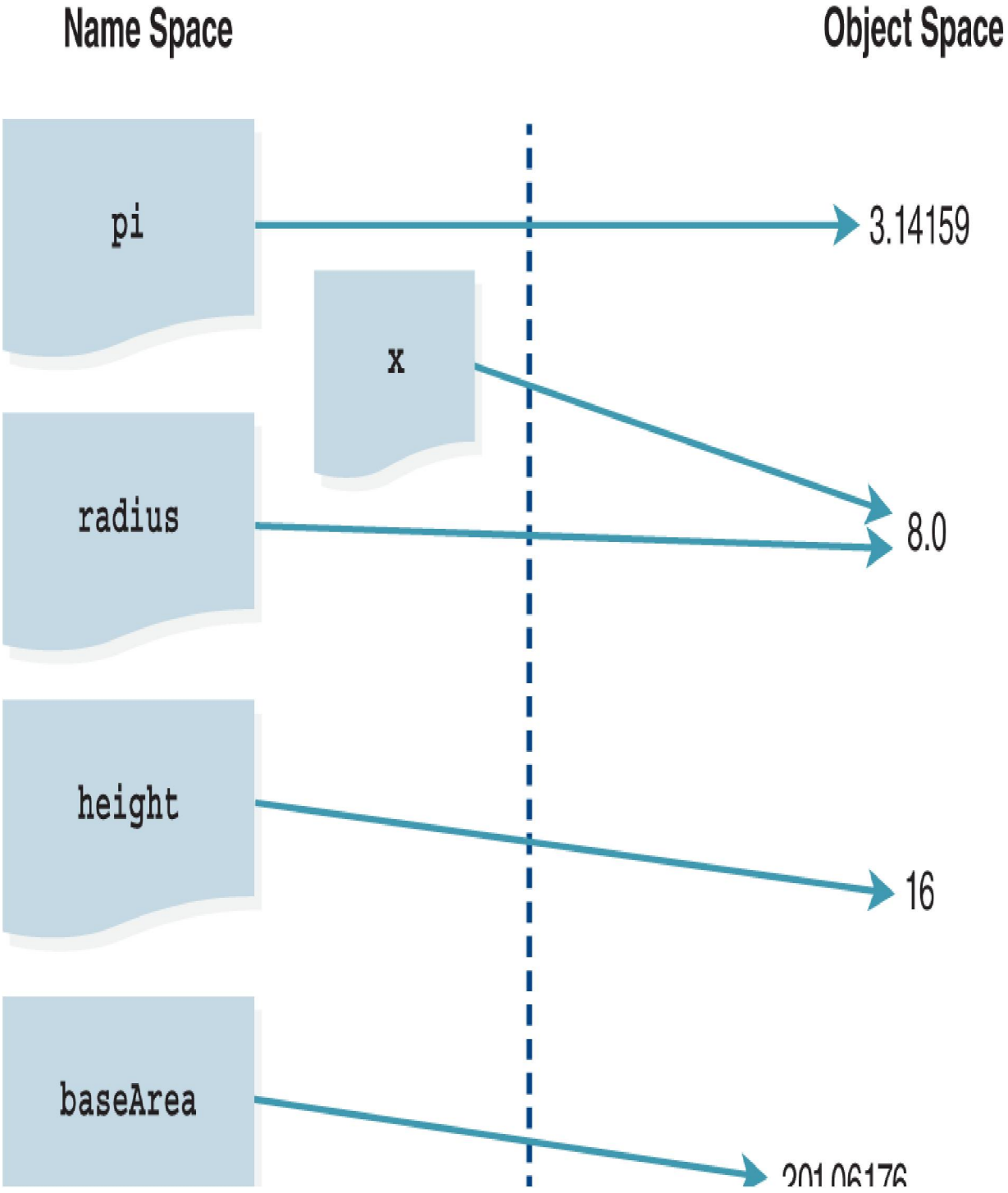


Figure 1.5 A reference diagram illustrating simple assignment in Python.

Description

Another way of thinking about assignment is to imagine that an assignment statement is like taking a sticky label with a name written on it and attaching it to an object. You know that you can put more than one sticky label on an object in the real world, and the analogy holds true in the Python world as

well: A Python object can have more than one name. For example, suppose you made the following additional assignment: `x = 8.0`. After executing that statement, you added another label called radius with another arrow pointing at the object 8.0, as shown in **Figure 1.6**. Now the object 8.0 has two names: radius and x.



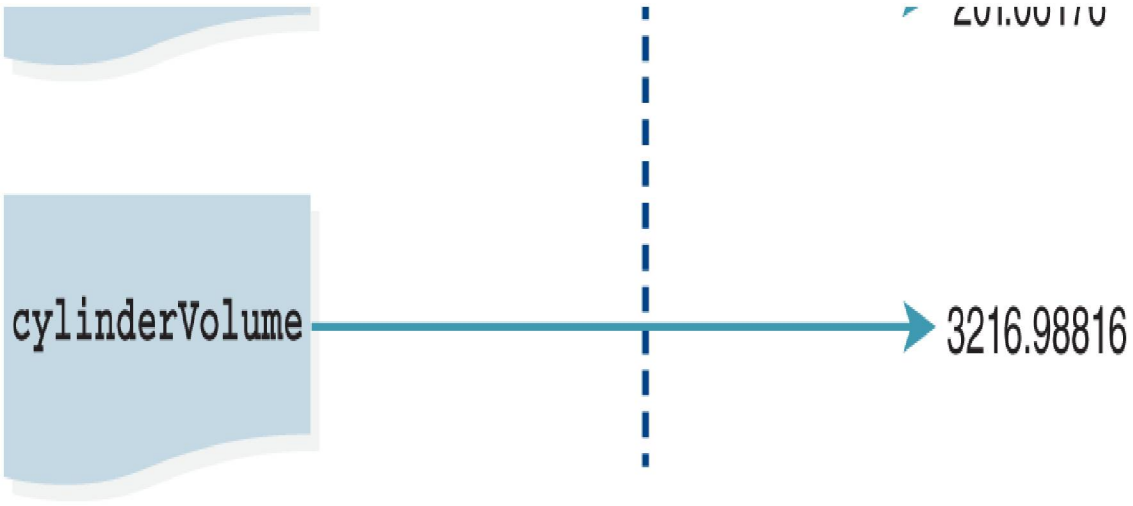


Figure 1.6 Reference diagram after `x = 8.0`.

Description

Variables can take the place of the actual object in a Python expression. When Python evaluates the expression `pi * radius ** 2`, it first looks up `pi` and `radius` to see which objects they refer to; it then substitutes their values into the expression. The expression thus becomes `3.14159 * 8.0 ** 2`. Next, Python evaluates `8.0 ** 2` to get an intermediate result of `64.0`. Python then evaluates `3.14159 * 64.0` to get the value `201.06176`. After the right-hand side of the expression is evaluated, Python assigns the name `baseArea` to `201.06176`. Similarly, Python evaluates the expression `baseArea * height` by substituting the value `201.06176` for `baseArea` and multiplying by `16`. Then Python assigns the name `cylinderVolume` to the value `3216.98816`.

Let's look at one more example using assignment. Consider the Python statements in **Session 1.8**.

```
In [2]:  1  a = 10

In [3]:  1  b = 20

In [4]:  1  a = b

In [5]:  1  a

Out[5]: 20

In [6]:  1  b

Out[6]: 20
```

Session 1.8 Python evaluates assignment statements in sequence

Description

After these three assignment statements, what object does `a` refer to? To answer this question, it is important to recognize that Python evaluates the statements from top to bottom, one after another. Let's rephrase what is going on in the order that Python performs its work.

1. Assign the name `a` to the integer object `10`.
2. Assign the name `b` to the integer object `20`.
3. Find the object named `b`, then assign the name `a` to that object.

The answer to our question is that `a` now refers to the object `20`. This is shown in **Figure 1.7**. In addition, because we have not changed what `b` refers to since the original assignment, `b` continues to refer to the object `20`. If you are confused by this example, try to draw the reference diagram yourself one step at a time.

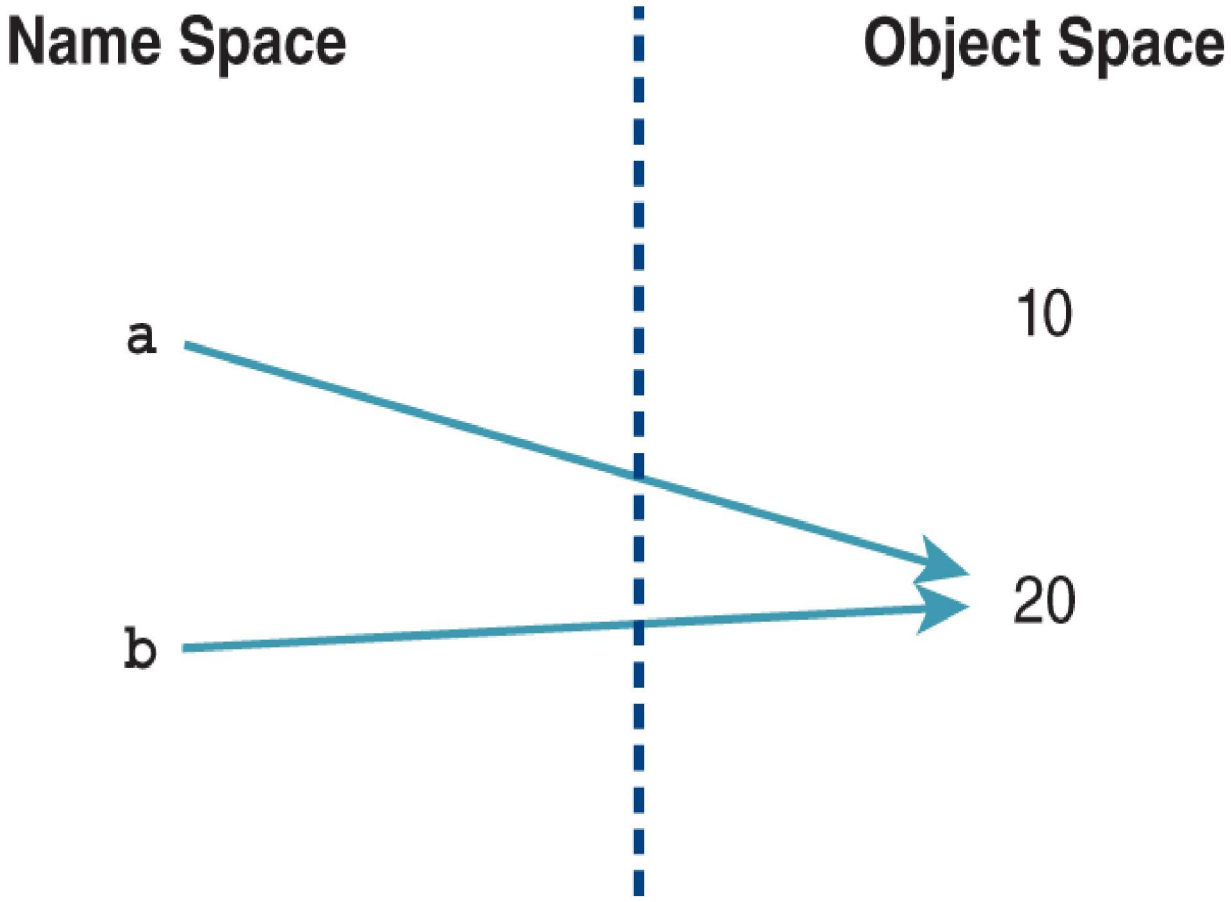


Figure 1.7 Result of assignment `a = b`.

Description

Now that you understand more about the assignment operator, you will appreciate that attaching the name `baseArea` to a different object will have no impact on the name `cylinderVolume`. If you want the new value for

`baseArea` to affect the value of `cylinderVolume`, you will need to ask Python to re-execute the statement `cylinderVolume = baseArea * height` with the new value for `baseArea`.

Since assignment is a statement rather than an expression, it does not return a value. As a consequence, the read–eval–print loop has nothing to print out. That is why you do not see any output following the assignment statements in Sessions 1.7 and 1.8.

A name on a line all by itself is a very simple Python expression. Notice that just typing a name causes Python to find the value for the object and print it as the result of the expression. You can see this in Session 1.7 when we type `baseArea` and `cylinderVolume`, and at the end of Session 1.8 when we type `a` and `b`. Python responds by printing their values.

There are several important rules to remember about naming things in Python. Names must start with either a letter or an underscore (_). Names can include any letter or number or an underscore, but cannot contain a space. Python is case-sensitive, which means that the names `baseArea`, `basearea`, and `BaseArea` are all different.

Some names, called **keywords**, are reserved by Python for its own use. These keywords correspond to important Python capabilities that you will learn about. **Table 1.4** lists all of Python's reserved keywords. Again, these reserved words are case-sensitive. For example, `True` is a reserved word, but `true` is not. To avoid confusion, it is a good programming practice not to choose names that are the same as the reserved words, except for case.

TABLE 1.4 Python's Reserved Keywords

and	continue	finally	is	raise
as	def	for	lambda	return
assert	del	from	None	True
async	elif	global	nonlocal	try
await	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield

Description

RECAP:
Python names must follow these rules:

- Names must start with a letter (uppercase or lowercase) or an underscore (_).
- Names can contain letters (uppercase or lowercase), underscores (_), or digits, but no spaces.
- Names cannot be Python's keywords (shown in Table 1.4). Names are case-sensitive.

GOOD PROGRAMMING PRACTICE:
Notice that we use **camelCase** when creating our names; that is, we start a name with a lowercase letter and capitalize internal words. This makes the name easier to read and easier to remember how it is spelled. It is also good practice to avoid choosing object names that differ from Python's reserved keywords only in case.

Try It Out

1.15 Given the following Python statements:

```
a = 79
b = a
a = 89
```

a. Draw a reference diagram to show the labels and objects after the first two statements.

b. Draw a reference diagram to show the labels and objects after the last statement.

1.16 Which of the following are legal variable names:

a. `_abc123`

b. `123abc`

c. `abc123_`

d. `_123`

e. `Abc 123`

1.17 Consider the following statements:

```
a = 10
b = 20
c = a * b
d = a + b
```

Draw a reference diagram to show all the objects and names after evaluating these statements.

1.18 What are the values of *a* and *b* after Python evaluates each of the following four statements?

```
a = 10
b = 20
a = b
b = 15
```

1.19 Consider the following statements:

```
idx = 0
idx = idx + 1
```

```
idx = idx + 2
```

What is the value of `idx` after Python evaluates these three statements?

1.5.3 Abstraction

Abstraction is defined as a concept or idea not associated with any specific instance. For example, you can think of mathematical functions on your calculator such as square root, sine, and cosine as abstractions. These functions can calculate a value for *any* number, but the method of calculating a square root is independent of the particular number. The square root function works equally well for all numbers because it is a general function. We do not have a special square root function for each possible number, just one function that works for all numbers.

One way of thinking about a function is as a "black box." You send information into the black box on one side and new information comes out on the other side. You don't know exactly what goes on inside the box, but you do know the behavior that the box should exhibit. **Figure 1.8** illustrates this concept for the square root function.

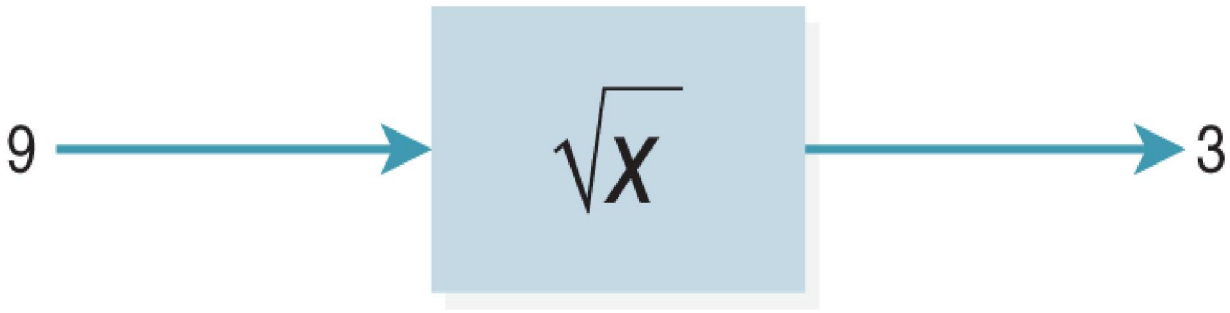


Figure 1.8 A black box view of the square root function.

The Python language contains many such abstractions. Many of the new things we will see in Python are, in fact, abstractions built using the Python primitives we have already talked about or will talk about later. In other words, much of Python is written in Python.

The `turtle` Module

Many of the additional parts of Python functionality are found in **modules**—prewritten Python code that implements an abstraction designed to make programming easier. To apply the power of a module, you have to tell Python to load the module you want. The statement you need to use to load a module is:

```
import modulename
```

When you import a module, an object is created inside Python. That object has the type *module* and has a name attached to it that matches the name you used on the import line. Every object in Python has three important characteristics: (1) an identity, (2) a type, and (3) a value. In addition, some Python objects have special values called **attributes**, and some Python objects also have **methods**, which are bits of Python code that allow us to ask the object to do something interesting. Let's look at a simple example before we go any further.

The example we will use is the `turtle` module, which provides us with a simple graphics programming tool known as a turtle. Very simply, a turtle is a drawing object that we can control. A turtle can move forward or backward, and it can turn in any direction. When a turtle moves, it draws a line if its tail is down. A turtle is a Python object that has both attributes and methods. Some of the attributes of a turtle are shown in **Table 1.5**.

TABLE 1.5 Turtle Attributes

Position	The coordinates of the turtle on the screen
Heading	The direction the turtle is facing
Color	The color of the turtle
Tail position	The turtle's tail can be up or down

Some methods of the turtle are summarized in **Table 1.6**. The second column is titled "Parameter(s)." Functions and methods are abstractions of generalized behaviors. Just as mathematical functions like cos(20) or $\sqrt{20}$ take parameters, Python functions and methods may also accept parameters. **Parameters** are the way we tell the function *specifically* what it should do.

If you see the Python keyword `None` in the Parameter column, that means that the method does not need any parameters to do its job.

TABLE 1.6 Summary of Simple Turtle Methods

Name	Parameter(s)	Description
`Turtle`	None	Creates and returns a new turtle object.
`forward`	distance	Moves the turtle forward by the specified `distance`.
`backward`	distance	Moves the turtle backward by the specified `distance`.
`right`	angle	Turns the turtle clockwise by `angle` degrees.
`left`	angle	Turns the turtle counterclockwise by `angle` degrees.
`up`	None	Picks up the turtle's tail.
`down`	None	Puts down the turtle's tail.
`pencolor`	Color name	Changes the color of the turtle's tail.
`fillcolor`	Color name	Changes the color that the turtle will use to fill a polygon.
`color`	Color name	Changes the color of the turtle's tail and the color the turtle will use to fill a polygon.
`heading`	None	Returns the current heading.
`position`	None	Returns the current position.
`goto`	x, y	Moves the turtle to position `x`, `y`.
`begin_fill`	None	Remembers the starting point for a filled polygon.
`end_fill`	None	Closes the polygon and fills it with the current fill color.
`dot`	None	Draws a dot at the current position.

To start, we will concern ourselves with just a few of the turtle methods. For example, we can tell the turtle to go forward, go backward, turn left, turn right, or tell us its position. The turtle has a tail that can be up or down. If the turtle moves when the tail is down, it draws a line. If the turtle moves when the tail is up, nothing is drawn. **Session 1.9** shows a Python session in which we create a `turtle` object and try out some of the turtle's capabilities.

```
In [1]: import turtle
```

```
In [2]: t = turtle.
```

In [2]:
```
turtle
```

Out[2]:
```
<module 'turtle' from 'C:\\Users\\julie\\AppData\\Local\\Programs\\Python
\\Python310\\lib\\turtle.py'>
```

In [3]:
```
gertrude = turtle.Turtle()
```

In [4]:
```
gertrude
```

Out[4]:
```
<turtle.Turtle at 0x17dbf297730>
```

In [5]:
```
gertrude.forward(100)
```

In [6]:
```
gertrude.right(90)
```

In [7]:
```
gertrude.forward(50)
```

In [8]:
```
gertrude.position()
```

```
Out[8]: (100.00,-50.00)

In [9]: gertrude.heading()

Out[9]: 270.0
```

Session 1.9 Using the `turtle` module

Description

Let's look at this example line by line. In line 1, we use an `import` statement to load the `turtle` module. In line 2, we ask Python to evaluate the name `turtle`. Python rewards us by telling us the identity of the object to which `turtle` is assigned. That identity indicates the location of the Python code for the module. This location will vary according to the kind of computer you are using and where you installed Python. If we were really adventurous, we could go there and look at the turtle methods that are stored in the `turtle.py` file.

Once we have the `turtle` module loaded, we can start to use the methods in the module to do something interesting. In the third statement we have an assignment statement, through which we make a new `Turtle` object and give it the name `gertrude`.

If you are typing this session interactively as you are reading, you will see that when a turtle is created, a new window, shown in **Figure 1.9a**, appears on the screen. The arrow in the middle of the screen represents the turtle. When a turtle is first created, it is at position (0.0, 0.0) in the middle of the window. The turtle's initial heading is 0.0 degrees, or straight to the right. The color attribute for the new turtle is black, and the tail is down. As you move the turtle around, it remembers its current position, the direction in which it is facing, and the status of its tail (up or down).

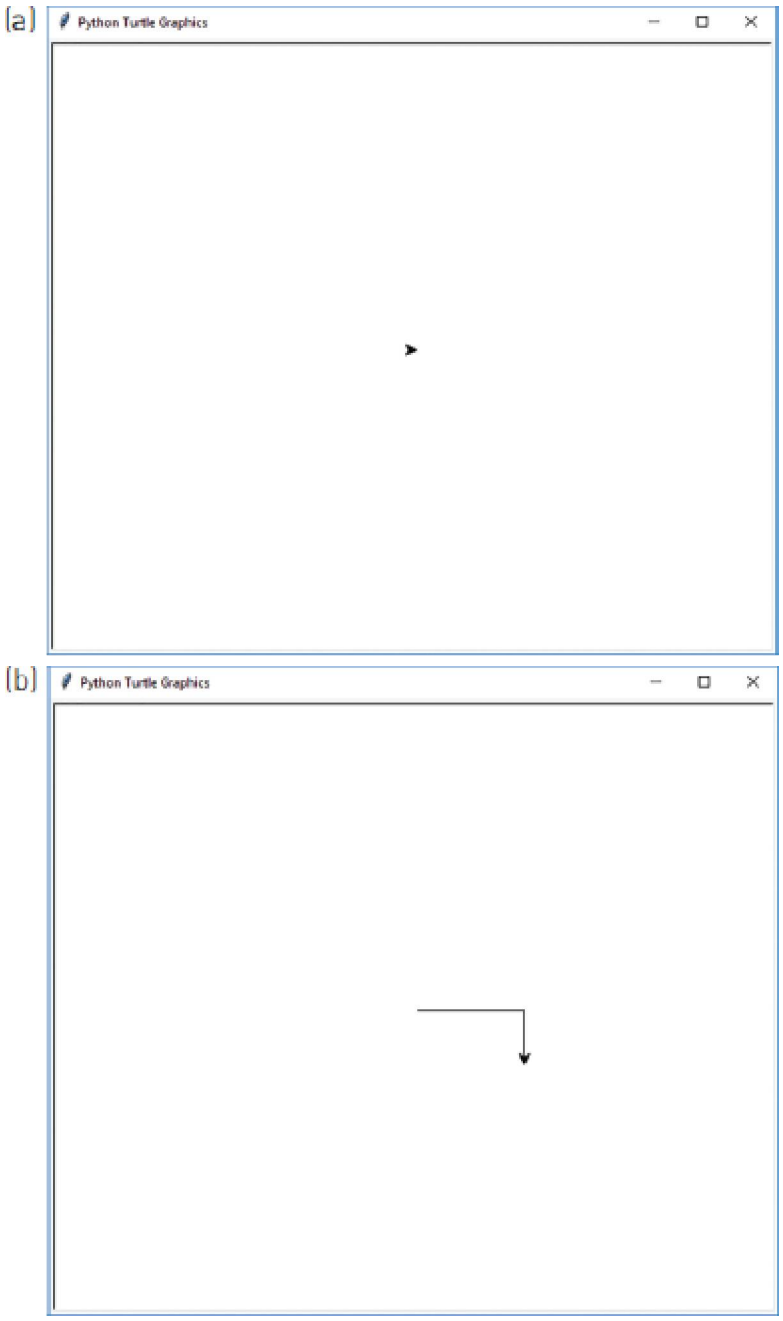


Figure 1.9 Using turtle graphics. (a) The turtle when it is first created. (b) The turtle after some movements.

© 2001–2025 Python Software Foundation; All Rights Reserved.

Before we go any further with this example, we need to explain the third statement in more detail. Notice that the expression `turtle.Turtle()` contains a new operator—the dot (.). The **dot operator** tells Python to look up the name right in front of the dot and return the object it names, a process

referred to as **dereferencing**. In this case, the dereferencing operation allows Python to find the `turtle` module. Once we have the module, Python looks within the module for the name to the right of the dot. In this case, Python looks for the `Turtle` method. A good way to think of this is that `Turtle` is Python code "inside" the `turtle` module. So, the dot operator gets us to the `turtle` module, and then inside the `turtle` module we find the object named `Turtle`. Remember that Python names are case sensitive, so `turtle` refers to the module itself and `Turtle` refers to a method within the `turtle` module.

As we can see, `turtle.Turtle()` is a method that creates a new object. Note that the `Turtle` method does not take any parameters; nevertheless, we need to include the parentheses when calling a method, even if they are empty. The type of the new object created is `Turtle`. A method that creates a new object is called a **constructor**. New `turtle` objects that we construct are called **instances** of the type `Turtle`.

RECAP:

To call an object's method, use the dot operator `(.)` with the name of the object, following this format:

```
objectName.methodName(parameters)
```

HEADS UP:

When calling a method, the parentheses after the method name are always required, even if the method does not take any parameters.

The fourth statement simply demonstrates that when you evaluate a name that corresponds to a more complicated object, Python returns its representation of that object. Unlike a number, where the representation is self-evident, a turtle's representation gives you a unique identification for the object. In this case, the result tells us that the type of `gertrude` is `turtle.Turtle`. Furthermore, our turtle can be found at location

`0x17dbf297730` in the computer's memory. (When you make your own turtle, it will be at a different location.)

Now that we have our new turtle object named `gertrude`, lines 5, 6, and 7 instruct `gertrude` to do some drawing. As you might guess, the line `gertrude.forward(100)` causes the turtle to move forward 100 units. Once again, the dot notation is very important to understanding how Python interprets the statement. First, the dot tells Python to dereference the name `gertrude`. When Python finds the object, it evaluates the method `forward` that is "inside" the turtle. The `forward` method knows that it needs to move forward 100 units because we pass it a parameter of 100. Similarly, the next two statements instruct the turtle to turn right 90 degrees and then move forward again, but this time by only 50 units. If you run this example on your own, your window should look like **Figure 1.9b**.

The last two statements (lines 8 and 9) of Session 1.9 show that we can also use methods to ask the turtle for information about itself. First, we ask `gertrude` to tell us its location with the method call `gertrude.position()`. In response, the `position()` method returns the value (100.0, −50.0), and the print part of the read–eval–print loop prints out that value. Methods and functions can return values, and those values can then be included in Python expressions. The fact that the `position` method returns a value is no different than the cosine function returning a value. Similarly, the return value of the `heading()` method tells us that `gertrude` is currently facing 270.0 degrees.

In the world of our turtle, the coordinates (0.0, 0.0) are in the center of the window. The *x* coordinate grows in a positive direction as the turtle moves toward the right. If the turtle moves toward the left side of the window, the *x* coordinate gets smaller and becomes negative to the left of the middle of the window. Similarly, the *y* coordinate grows as the turtle moves toward the top of the window, but gets smaller as the turtle moves toward the bottom and becomes negative below the middle of the window. One pixel on your computer screen corresponds to 1 unit of turtle movement. The turtle's heading varies from 0 to 359, moving counterclockwise around the window. If the turtle's heading is 0.0 degrees, it is facing to the right; 90.0 degrees is up, 180.0 degrees is to the left, and 270.0 degrees is down.

Try It Out

1.20 Create a turtle called `sven`. Now tell `sven` to go forward 50 units. What is `sven`'s position now?

1.21 Create a turtle called `ole`, and tell `ole` to turn right 45 degrees and go forward 50 units. Notice that you now have two turtles in the same window.

1.22 On a sheet of graph paper, sketch out a simple line drawing of something. Using the turtle methods listed in Table 1.6, recreate your line drawing.

Getting Help

The `turtle` module has many other methods. Table 1.6 shows just a few of them. If you are curious about other `turtle` methods, you can use Python's built-in help subsystem that allows you to see the methods and attributes that are defined in a module. To use this help, type the following command into the Python interpreter:

```
help("modulename")
```

So, to see the functions and methods for the `turtle` module, type

```
help("turtle")
```

Be warned, however, that the `turtle` module has a large number of attributes and methods. Instead of displaying all that information, the interpreter will display a message saying that it has squeezed the text. One option is to double-click the message to expand the squeezed lines. In this case, however, the large number of lines (more than 8,000!) could be overwhelming. Instead, we recommend right-clicking and choosing `view`. This will display a pop-up window for easily scrolling through the information.

If you know the name of a method, but want to check how to call it, you can use the `help` command with the name of the method. For example, to get help on the `turtle`'s `forward` method, type

```
help("turtle.forward")
```

You may also have noticed as you typed the statements in Session 1.9 that Jupyter Notebook responded by displaying the parameters that a method accepts. These helpful aids are called **calltips**.

We'll introduce additional `turtle` methods as needed in future examples.

Try It Out

1.23 Run the `help` command on the `turtle` module. Be sure to `import turtle` before running the command.

1.24 Run the `help` command on the `turtle backward` method.

1.25 Run the `help` command on the `round` built-in function.

Writing Your Own Functions

We are not limited to the functions that the authors of Python have given us. We, too, can write our own functions to add our own abstractions to the Python language. In fact, functions are just another kind of Python object with several special capabilities. We **define** a function in Python using the `def` statement. Here, we show a general template for using the `def` statement to define a function.

```
def functionName(param1, param2, ...):
    statement1
    statement2
    ...
```

Description

The `def` statement begins by giving the function a name. Next, we specify any parameters we want our function to accept. In Python, we can have zero or more parameters for any function we write. All the parameters must be listed inside the parentheses that follow the function name. Next, we have a colon character, which tells the Python interpreter that the sequence of indented statements that follow is the body of the function. The **function body** contains the executable statements. Python knows to stop reading lines for the `def` statement when it encounters a line that is not indented. A group of Python statements that are indented at the same level is called a **block**.

HEADS UP:
When defining a function, you can indent all the statements in the function block using tabs or spaces, but you must maintain a consistent practice throughout the function. In other words, if you indent with a tab, all lines in the function need to be indented using a tab. If you indent using spaces, then all lines in the function need to be indented the same number of spaces.

Now let's try to write a function. Suppose that we are working on a graphics program that requires us to draw a lot of squares. Telling the turtle how to draw a square each time we need one is tiresome and repetitious. In fact, a square is an example of an abstraction. We know that a square is a geometric shape that has four sides of equal length and four 90-degree

corners. What we don't know is how long the sides are for any particular square.

Our goal is to write a function that can use any turtle to draw a square of any size. To solve this problem, then, we need two pieces of information—namely, the turtle that will draw the square and the size of the square. These pieces of information will become the parameters to our function.

The next step is to use the parameters along with the methods of a turtle. We can draw a square by moving the turtle forward and turning right by 90 degrees four times in a row. **Listing 1.1** shows a complete Python function for drawing a square with a turtle.

```
1 def drawSquare(myTurtle, sideLength):
2     myTurtle.forward(sideLength)   #side 1
3     myTurtle.right(90)
4     myTurtle.forward(sideLength)   #side 2
5     myTurtle.right(90)
6     myTurtle.forward(sideLength)   #side 3
7     myTurtle.right(90)
8     myTurtle.forward(sideLength)   #side 4
9     myTurtle.right(90)
```

Listing 1.1 A function to draw a square using a turtle

Description

Notice that the statements in the function are the same turtle commands that we typed interactively when we first started using the turtle. Placing the commands inside the function groups them together so that we can run the

whole set of commands as if we had typed them one after another. This is one of the great powers of writing a function. It is also important to remember that Python will execute the commands in exactly the order they are typed in the function.

Note the comments that appear on lines 2, 4, 6, and 8. These comments identify which side of the square is being drawn.

GOOD PROGRAMMING PRACTICE:
Using comments can make your code easier to read and understand.

Type the `drawSquare` function into a text file using Notepad or TextEdit exactly as shown in Listing 1.1, including the indentation of the function block. After you type in the function, save it to a file called ds.py. Be sure to save the ds.py file in the same folder as you are working in Jupyter Notebook so you will be able to call the function later to draw a square. You have just created your first module! In the same way that `turtle` is a module created by other Python programmers, you have created a module called `ds`.

HEADS UP:
Be careful to save the `drawSquare` function in a file with the extension `.py`, rather than `.txt`. The `.py` extension indicates to the Python interpreter that the file contains Python code.

We can now call the `drawSquare` function, as shown in **Session 1.10**. After running the commands in this session, you should have an image that looks like **Figure 1.10**.

```
In [1]: from ds import *

In [2]: import turtle

In [3]: t = turtle.Turtle()

In [4]: drawSquare(t, 150)
```

Session 1.10 A Python session that demonstrates calling a function

Description

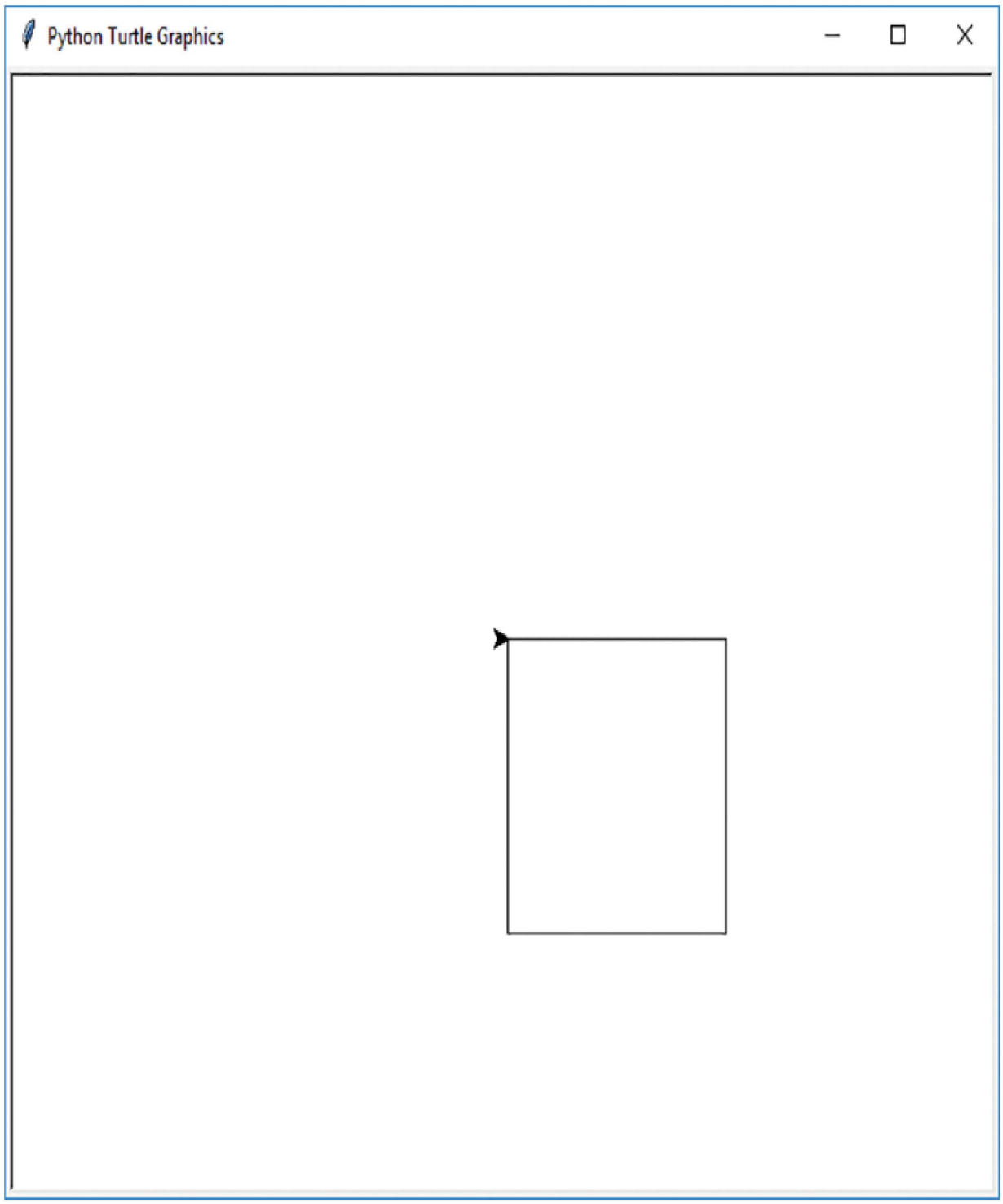


Figure 1.10 The result of the statements in Session 1.10.

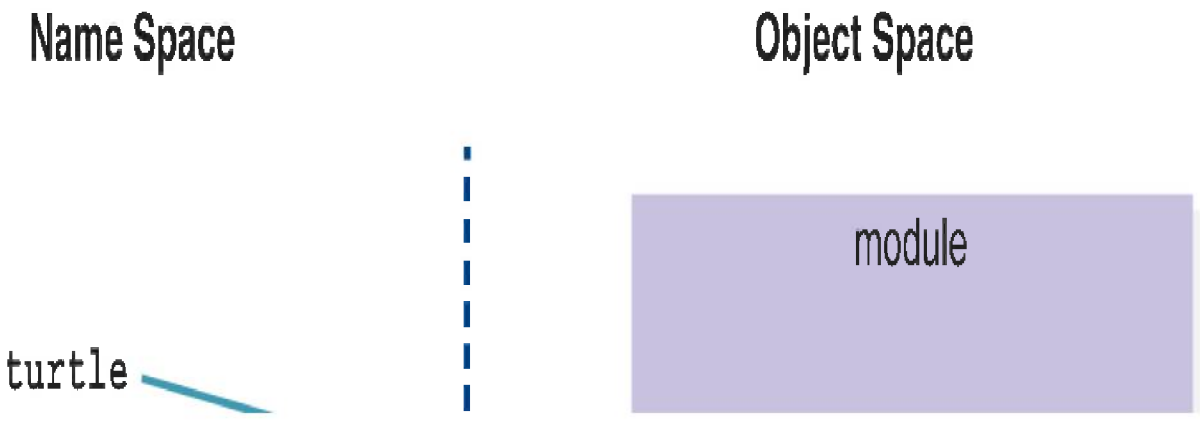
© 2001–2025 Python Software Foundation; All Rights Reserved.

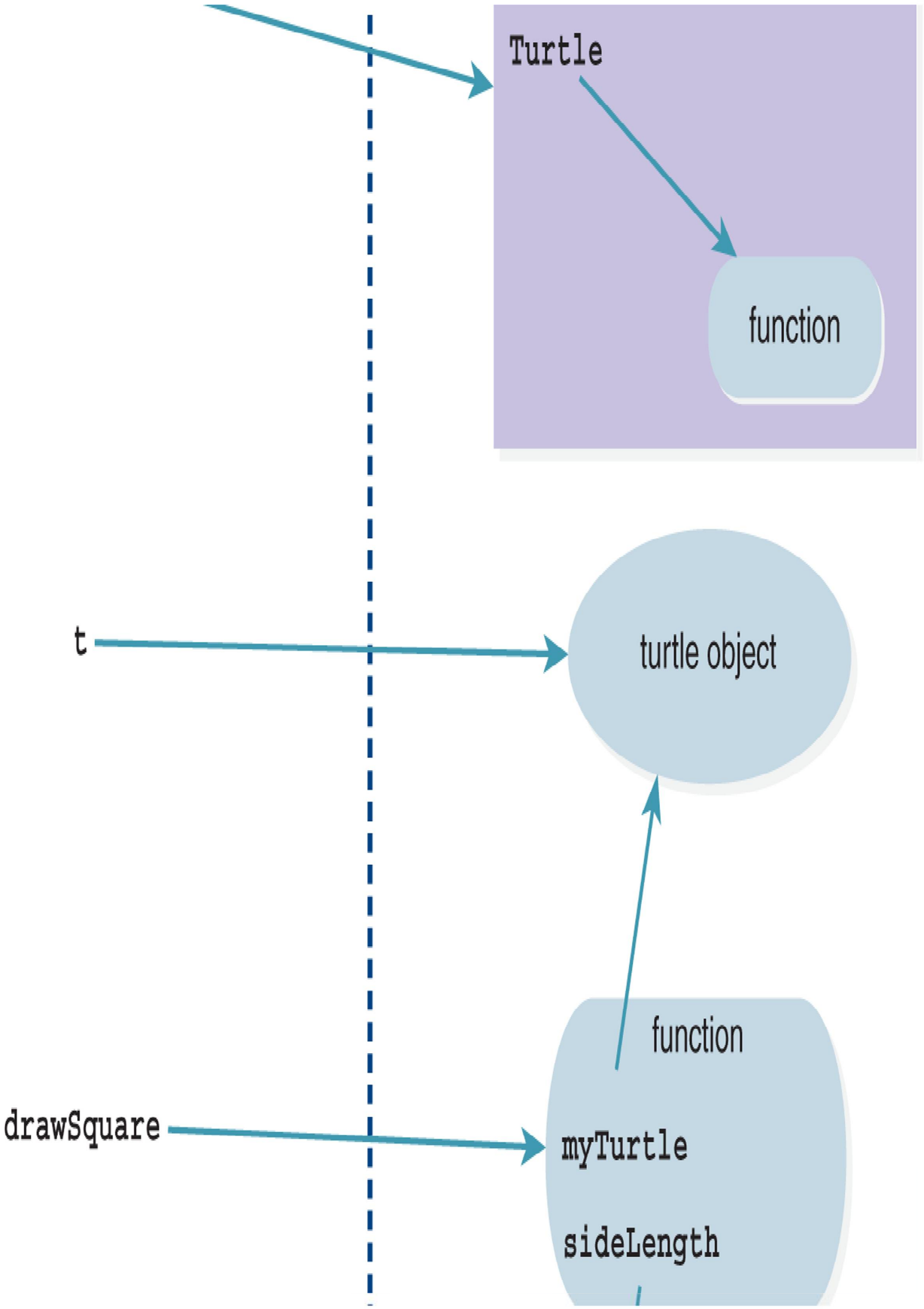
There is a lot going on in this simple example, but there are only a few new things that you have not seen before. First, we import the `turtle` and `ds` modules. When we execute the statement `import turtle`, any functions inside the turtle module are "hidden" from us unless we use the `turtle` prefix with the dot operator. However, if we use the statement `from ds import *`, all the functions defined in the file `ds.py` are visible to us without the `ds` prefix.

We then make a new `turtle` object named `t`, and next we call `drawSquare`. Python knows that it should treat the object `drawSquare` like a function call because of the parentheses that follow its name. You can even prove to yourself that `drawSquare` is a plain old object by typing `drawSquare` without the parentheses after the name. If you do so, Python will tell you something like `<function drawSquare at 0x000001C3863E01E0>`.

When we call the function `drawSquare`, we pass it two objects, `t` and `150`. When a function is called, the objects are matched up with the parameter names they were given when we defined the function. The first parameter in the list names the first object, the second parameter in the list names the second object, and so on. In this case, `t` goes by the name `myTurtle` inside the `drawSquare` function. The `turtle` object now has two names. Similarly, inside the `drawSquare` function, `150` now has the name `sideLength`.

The reference diagram in **Figure 1.11** indicates how all the names and objects are matched up. This diagram is simplified somewhat, but we will add more details to such diagrams in later chapters. The important thing to notice is the relationship between the names of the parameters of a function and the objects that are passed to the function when it is called.




Turtle
function
t
turtle object
function
drawSquare
myTurtle
sideLength

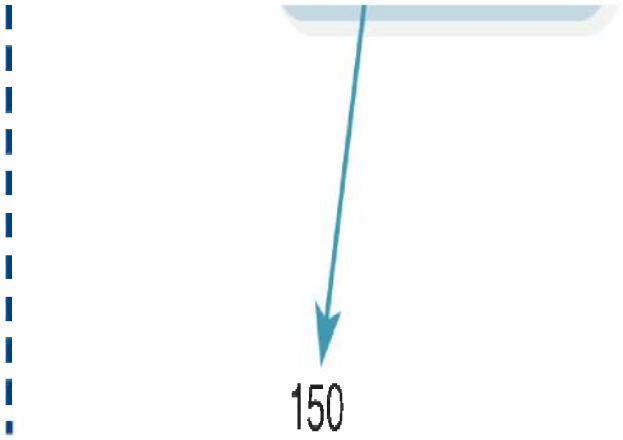


Figure 1.11 A reference diagram for the function `drawSquare`.

Description

Figure 1.11 shows three names: (1) `turtle`, which references the `turtle` module we imported previously; (2) `t`, which references the turtle object we created with the call to the turtle constructor (`turtle.Turtle()`), shown inside the `turtle` module; and (3) `drawSquare`, which references the function object we defined for drawing squares. In the function object named `drawSquare`, you can see two additional names: (1) `myTurtle`, which references the turtle object named `t`, and (2) `sideLength`, which references the integer object `150` that has no other name.

Try It Out

1.26 Modify the `drawSquare` function to draw a rectangle whose width is twice the `sideLength`.

1.27 Create a new function called `drawRectangle` that takes three parameters: `myTurtle`, `width`, and `height`.

1.28 Suppose that in Session 1.10 we had used the statement `import ds` instead of `from ds import *`. Continue the rest of the session. Draw a reference diagram that illustrates this session.

1.29 Call your function defined in Exercise 1.27 as follows: `drawRectangle(t,50,300)`.

1.30 Draw a reference diagram for the previous exercise.

1.5.4 Repetition

Although the `drawSquare` function we discussed earlier worked correctly, our solution is unsatisfying in one respect: We had to duplicate the move forward and turn to the right statements four times. We can eliminate this duplication by using a statement that Python provides for repeating a block of code multiple times—a **`for` loop**. The `for` loop is an example of using **repetition** in our program.

Before we rewrite our `drawSquare` function, let's look at the structure of a `for` loop to get a better idea of how this statement works:

```
for i in range(n):
    statement1
    statement2
    ...
```

Description

Notice that the `for` loop template has some similarity to the `def` template. There is a beginning line ending with a colon, followed by an indented block of code called the **loop body**. All you need to know about a `for` loop at this point is that the indented block of statements will be evaluated `n` times, where `n` is the parameter to the `range` function. If the first line was `for i in range(10):`, then the block of statements would be evaluated 10 times. Each time the block of statements is evaluated is called an **iteration** of the loop. We will talk more about the `range` function as well as the `i` and `in` parts of the statement shortly.

Now let's see how we can use this idea of simple repetition to make our `drawSquare` function more elegant. In fact, all we need to do is wrap the two

lines that tell the turtle to move forward and turn right inside a loop that will be evaluated four times. You can see the new and improved `drawSquare` function in **Listing 1.2**.

```
1 def drawSquare(myTurtle, sideLength):
2     for i in range(4):
3         myTurtle.forward(sideLength)
4         myTurtle.right(90)
```

Listing 1.2 A better version of the `drawSquare` function

Description

HEADS UP:
By now you may have noticed colors in the Python code. Like most coding environments, Jupyter Notebook colors the code so we can more easily distinguish the parts of our programs. By default, keywords are dark green, our function names are blue, built-in function names are light green, and comments are cyan and italicized. As we introduce additional Python elements, you will see those elements colored as well.

Drawing a Spiral

Our goal for this section is to understand how we can use the `for` loop to have the turtle create a more complicated square spiral pattern. To create a spiral pattern, the sides of our square need to grow each time the turtle moves forward. We can make that happen easily when we understand the two components of the `for` statement that we ignored a moment ago.

First, let's look at the `range` function. If you ask Python to evaluate the expression `range(5)`, you will get back a **sequence object** representing the

numbers `[0, 1, 2, 3, 4]`.

The `range` function is very versatile: It can create all kinds of sequences depending on the parameters that we supply. We can call `range` in three ways:

- `range(stop)`: Creates a sequence of numbers beginning at 0 and going up to `stop-1`, incrementing by 1.
- `range(start, stop)`: Creates a sequence of numbers beginning at `start` and going up to `stop-1`, incrementing by 1.
- `range(start, stop, step)`: Creates a sequence of numbers beginning at `start` and going up to `stop-1`, incrementing by `step` each time.

Now that you have a better understanding of the `range` function, let's look at the **loop variable**—the variable that always follows the keyword `for`. In **Listing 1.3**, the loop variable is named `sideLength`. In Python, this loop variable starts as the name for the first item in the sequence produced by the `range` function the first time through the loop. The second time through the loop, the loop variable is the name for the second item in the sequence, and so on until there are no more items in the sequence. We can use any valid Python name for the loop variable.

```
1 def drawSpiral(myTurtle, maxSide):
2     for sideLength in range(1, maxSide + 1, 5):
3         myTurtle.forward(sideLength)
4         myTurtle.right(90)
```

Listing 1.3 A Python function to draw a spiral

Description

Figure 1.12 shows the sequence produced by the call to `range` and the element in the list named by `item` during the fourth repetition of the loop. Note that `55` is the last item in the sequence because the upper bound, 60, is not included in the result of the `range` function.

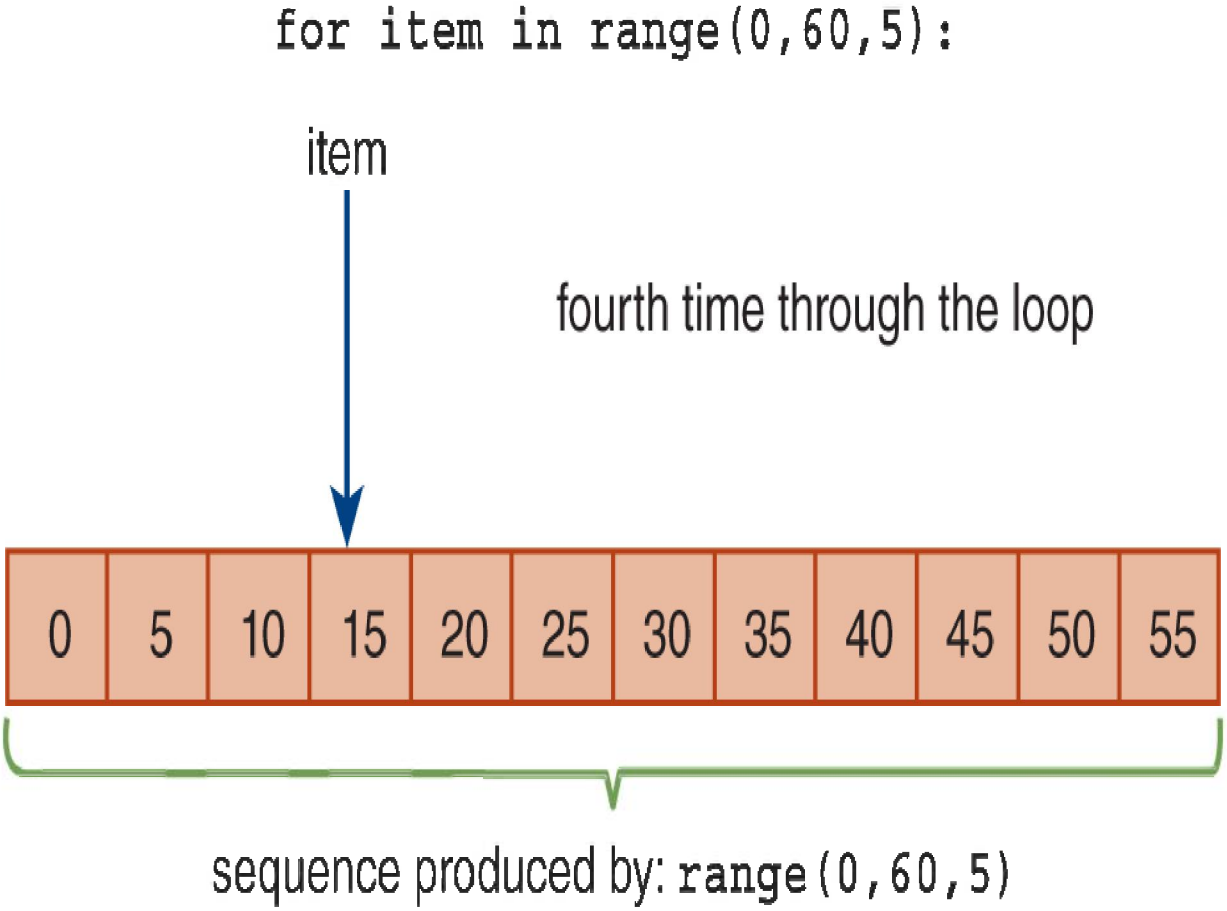


Figure 1.12 A loop variable naming each item in a sequence.

Description

Try It Out

1.31 Use the `range` function to create the sequence of numbers that are multiples of 5 up to 50.

1.32 Use the `range` function to create the sequence of numbers from −10 to 10.

1.33 Use the `range` function to create the sequence of numbers from 10 to −10.

Loop variables can be used in expressions and function calls like any other Python names. To solve the spiral problem, Listing 1.3 uses a loop variable in a function named `drawSpiral` that draws a spiral. The parameters to this function are the turtle and a bound for the longest side on the spiral. We start with the first side having length 1; each time through the loop, we increase the length of the next side of the spiral by 5.

Figure 1.13 illustrates the first four iterations of the `for` loop in Listing 1.3. In the first iteration, the turtle draws a line that is 1 unit long because `sideLength` refers to the first item in the sequence produced by `range(1, maxSide + 1, 5)`. In the second iteration, `sideLength` refers to the number 6, so the turtle draws a line that is 6 units long. In the third iteration, `sideLength` refers to 11; in the fourth iteration, `sideLength` refers to the fourth number in the sequence, which is 16.



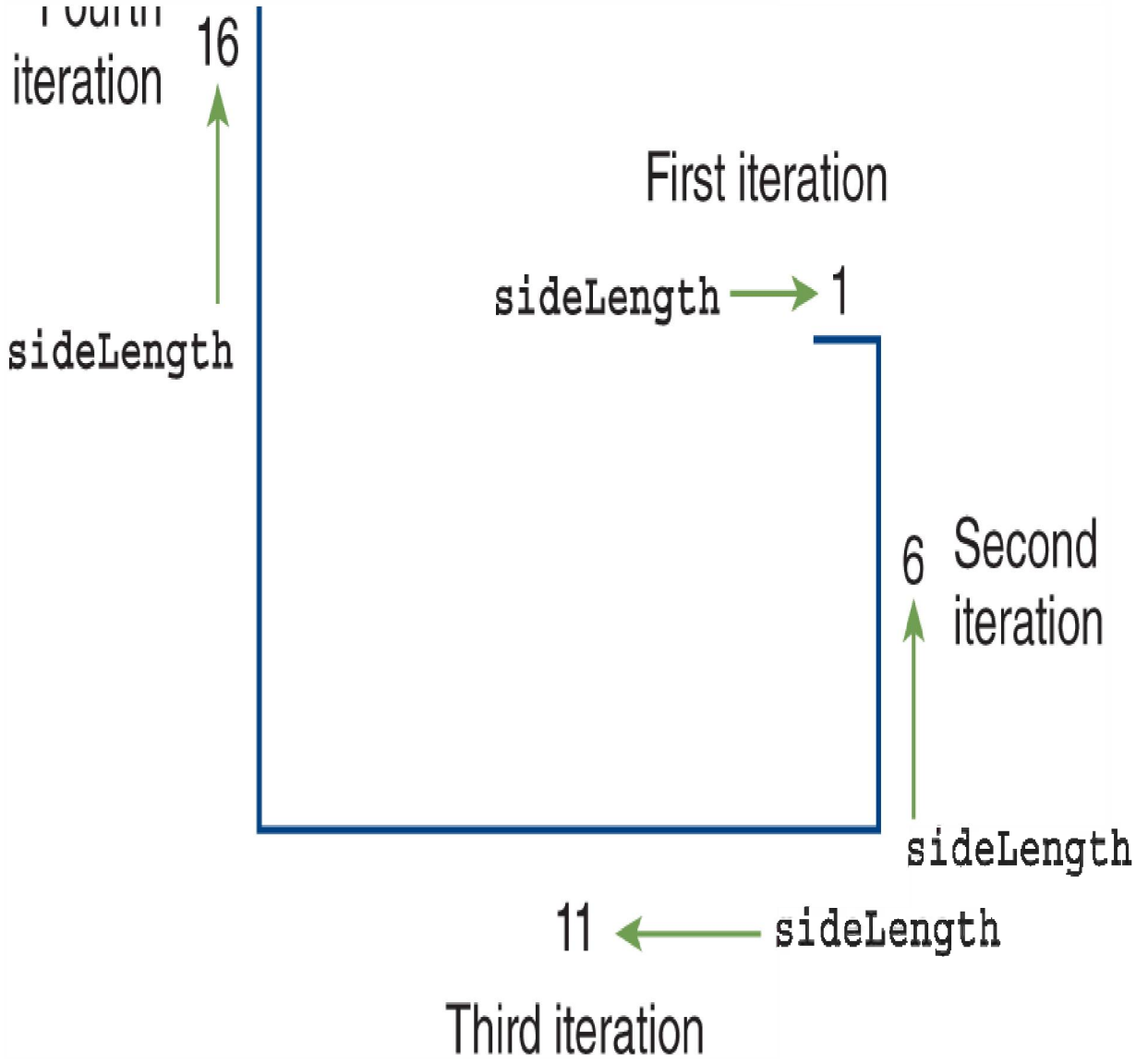


Figure 1.13 The first four iterations of the loop in `drawSpiral(t, 150)`.

Description

Try It Out

1.34 Modify the spiral function to turn more than 90 degrees for each iteration.

1.35 Modify the spiral function to turn less than 90 degrees for each iteration.

1.36 Modify the spiral function to use the loop variable as the number of degrees to turn.

1.37 Modify the spiral function to use a second turtle and create two spirals in opposite directions.

1.38 Write a function `drawTriangle` that takes two side lengths and the angle between them and draws the triangle. (*Hint:* You need to remember the starting point.)

1.39 Write a function that draws a series of 10 squares, with each square being 5 pixels smaller on each side. The squares should all start at the same location.

1.40 Redo the last question so that the squares are all centered.

1.41 Use the turtle to plot the function $y = x^2$.

1.42 Use the turtle to plot the function $y = \frac{x}{2} + 3$.

Drawing a Circle

Our final problem for this chapter is to write a function that uses a turtle to draw a circle of a given radius. Although this may seem like a daunting task, we will help ourselves by solving a simpler problem first and then using what we learn to solve the more general problem. The first challenge is that the turtle's functionality allows us to draw only straight lines. But by drawing many short straight lines, we can approximate a curved line.

Suppose that our problem was to draw a triangle rather than a circle. It is not too hard to imagine how we would modify `drawSquare` to write `drawTriangle`. We would change the call to `range(4)` to be `range(3)`, as we now need only three sides. We would also change the number of degrees we pass as the parameter for our right turn function.

How many degrees do we need to turn each time to draw an equilateral triangle? When the triangle is complete, we want our turtle to be pointed in the same direction it was when we started. So, as we draw the three sides of our triangle, the turtle will turn through 360 degrees. That matches our `drawSquare` function, where we made four 90-degree turns ($4 \times 90 = 360$).

So, to draw a triangle, we will use 360 ÷ 3 = 120 for our turning angle. **Listing 1.4** shows the small changes made to the `drawSquare` function to create the `drawTriangle` function.

```
1 def drawTriangle(myTurtle, sideLength):
2     for i in range(3):
3         myTurtle.forward(sideLength)
4         myTurtle.right(120)
```

Listing 1.4 A Python function to draw a triangle

Description

We now have two simple examples of polygons—the equilateral triangle and the square. It is relatively easy to imagine how we would write a function to draw a pentagon or an octagon by following the pattern we have established with the triangle and square. In fact, **Table 1.7** illustrates the values we would supply to the `range` and `right` functions for several different polygons.

TABLE 1.7 Number of Sides and Turning Angle for Several Polygons

Polygon	Sides	range()	Angle
triangle	3	3	360/3 = 120
square	4	4	360/4 = 90
pentagon	5	5	360/5 = 72
octagon	8	8	360/8 = 45

Description

Table 1.7 suggests that we can write a function that is more abstract than `drawSquare`, `drawTriangle`, or even `drawOctagon`. The abstraction we are using in this case is a regular polygon. Creating one function that can replace many simpler functions at a higher level of abstraction is a common and important problem-solving technique in computer science.

GOOD PROGRAMMING PRACTICE:
An important problem-solving technique is to create one function at a higher level of abstraction that replaces many simpler functions.

For example, we can write a function that draws any regular polygon if we pass a third parameter to the function. The third parameter tells the function how many sides we want. Once we know how many sides are required, we can easily calculate the turning angle by using the formula `turnAngle = 360 / numSides`. You can see the new `drawPolygon` function in **Listing 1.5**.

```
1 def drawPolygon(myTurtle, sideLength, numSides):
2     turnAngle = 360 / numSides
3     for i in range(numSides):
4         myTurtle.forward(sideLength)
5         myTurtle.right(turnAngle)
```

Listing 1.5 A Python function to draw a regular polygon of any size

Description

Session 1.11 and **Figure 1.14** illustrate several calls to the `drawPolygon` function, assuming we have saved the code into a file named `dp.py`. Before we call the `drawPolygon` function, however, we want to move the turtle to the left and make it face upward. To do so, we lift the turtle's tail so that the turtle will not leave a path when it moves: We move the turtle backward 200 pixels, turn the turtle 90 degrees to the left to face up, and then put the tail down.

```
In [1]: from dp import *   #contains drawPolygon

In [2]: import turtle

In [3]: t = turtle.Turtle()

In [4]: #move turtle to left and face up

In [5]: t.up()

In [6]: t.backward(200)
```

```
In [7]: t.left(90)

In [8]: t.down()

In [9]: #call drawPolygon with different sizes and sides

In [10]: drawPolygon(t, 100, 4)

In [12]: drawPolygon(t, 100, 8)

In [13]: drawPolygon(t, 50, 20)

In [14]: drawPolygon(t, 20, 20)
```

Session 1.11 Testing the `drawPolygon` function

Description



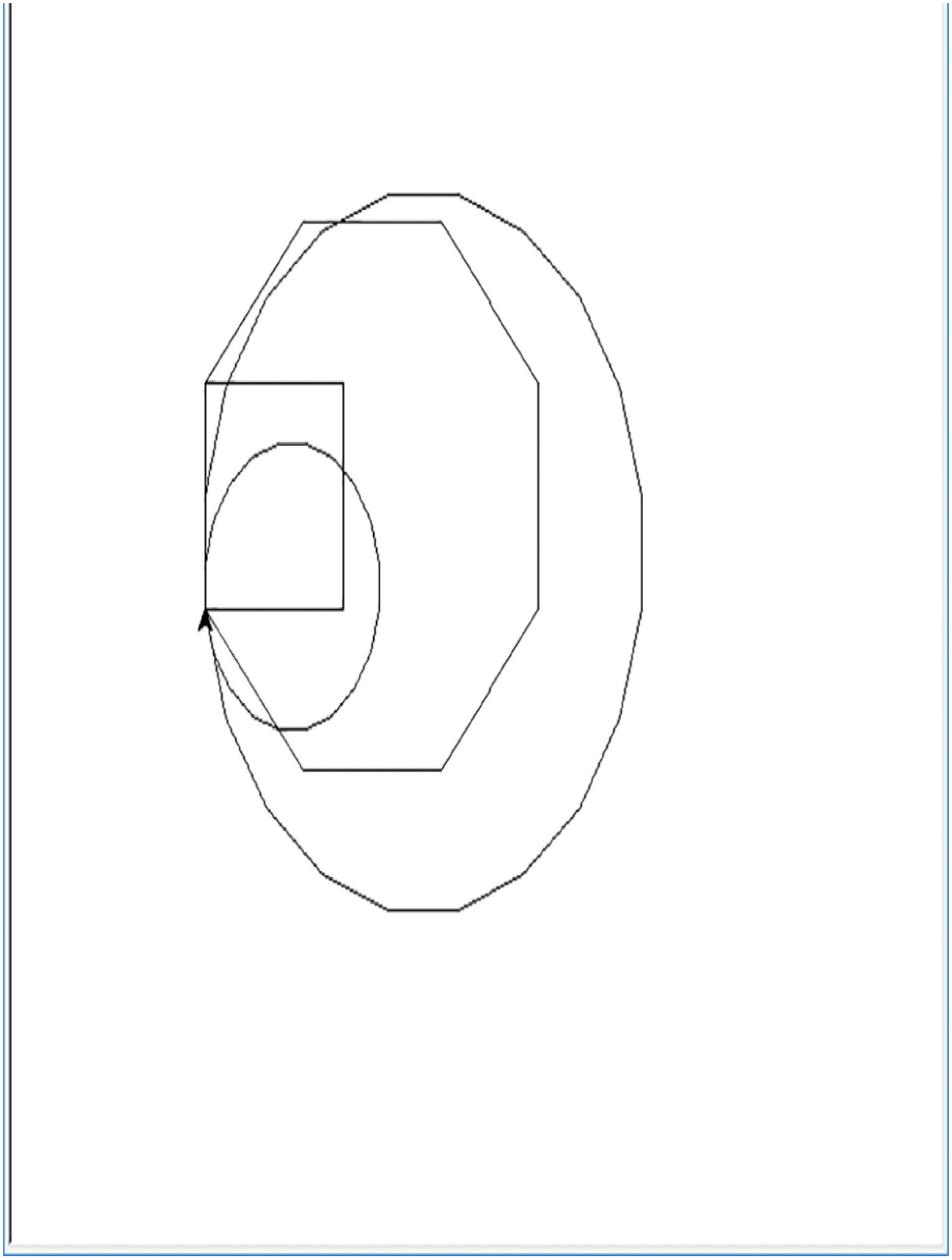


Figure 1.14 Several polygons drawn by the `drawPolygon` function.

© 2001–2025 Python Software Foundation; All Rights Reserved.

Description

Note that the call `drawPolygon(t,20,20)` makes a pretty good approximation of a circle.

So far, we have worked our way through the problem of drawing a circle by using a large-degree polygon as an approximation. Now let's return to the original problem statement, which asks us to draw a circle of a given radius. Our last trick is to figure out how to use the radius to compute the number of sides and the side length.

Suppose that to get the smoothest circle possible, even when we are drawing a very large circle, we choose to always have 360 sides to our polygon. With this specification, the turtle will always turn 1 degree and should give us a smooth circle even for a large radius.

Only one question remains: How do we decide what side length to use for a given radius? One good approximation is to use the relationship of the radius of a circle to the circumference. If we are doing a good job of approximating a circle, then the circumference of the circle should be very close to the sum of the individual sides of the polygon. Recall that the circumference of a circle can be calculated from the radius using the formula $circumference = 2 \times \pi \times radius$. Once we have the circumference, we can calculate an individual side length by dividing the circumference by the number of sides, which we have decided will be 360. **Listing 1.6** shows the `drawCircle` function, which makes use of our `drawPolygon` function to draw a circle with a given radius.

```
1 def drawCircle(myTurtle, radius):
2     circumference = 2 * 3.1415 * radius
3     sideLength = circumference / 360
4     drawPolygon(myTurtle, sideLength, 360)
```

Listing 1.6 A Python function to draw a circle

Description

The `drawCircle` function is extremely simple because we have reduced the problem of drawing a circle to that of drawing a polygon with a particular number of sides and side lengths. First, we calculate the circumference; then, we calculate the side length given the circumference and number of sides. Because we already know how to draw a polygon, we do not have to redo that work. We can build on what we have already accomplished and use the `drawPolygon` function to perform this tedious work. **Session 1.12** and **Figure 1.15** illustrate the use of the `drawCircle` function to draw both a large circle and a small circle. We assume that you have added the `drawCircle` function to the file `dp.py`.

```
In [1]: from dp import *
```

```
In [2]: import turtle
```

```
In [3]: t = turtle.Turtle()
```

```
In [4]: t.up()
```

```
In [5]: t.backward(200)
```

```
In [6]: t.left(90)

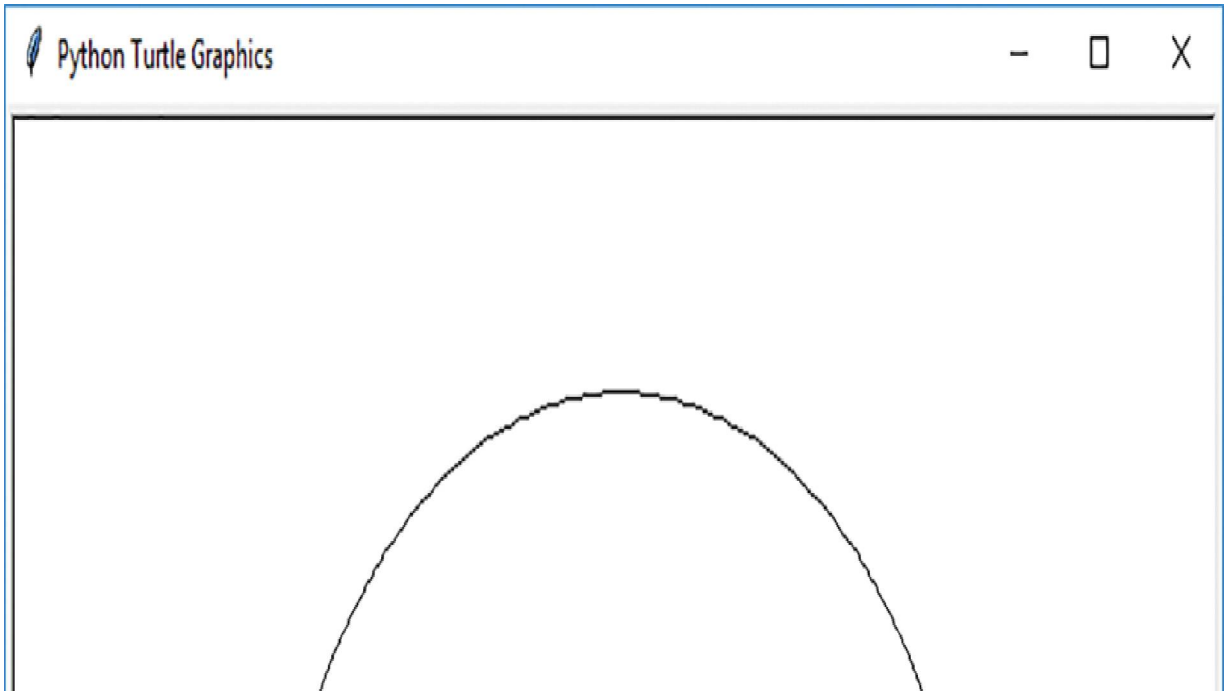
In [7]: t.down()

In [8]: drawCircle(t, 20)

In [9]: drawCircle(t, 200)
```

Session 1.12 Drawing two circles

Description



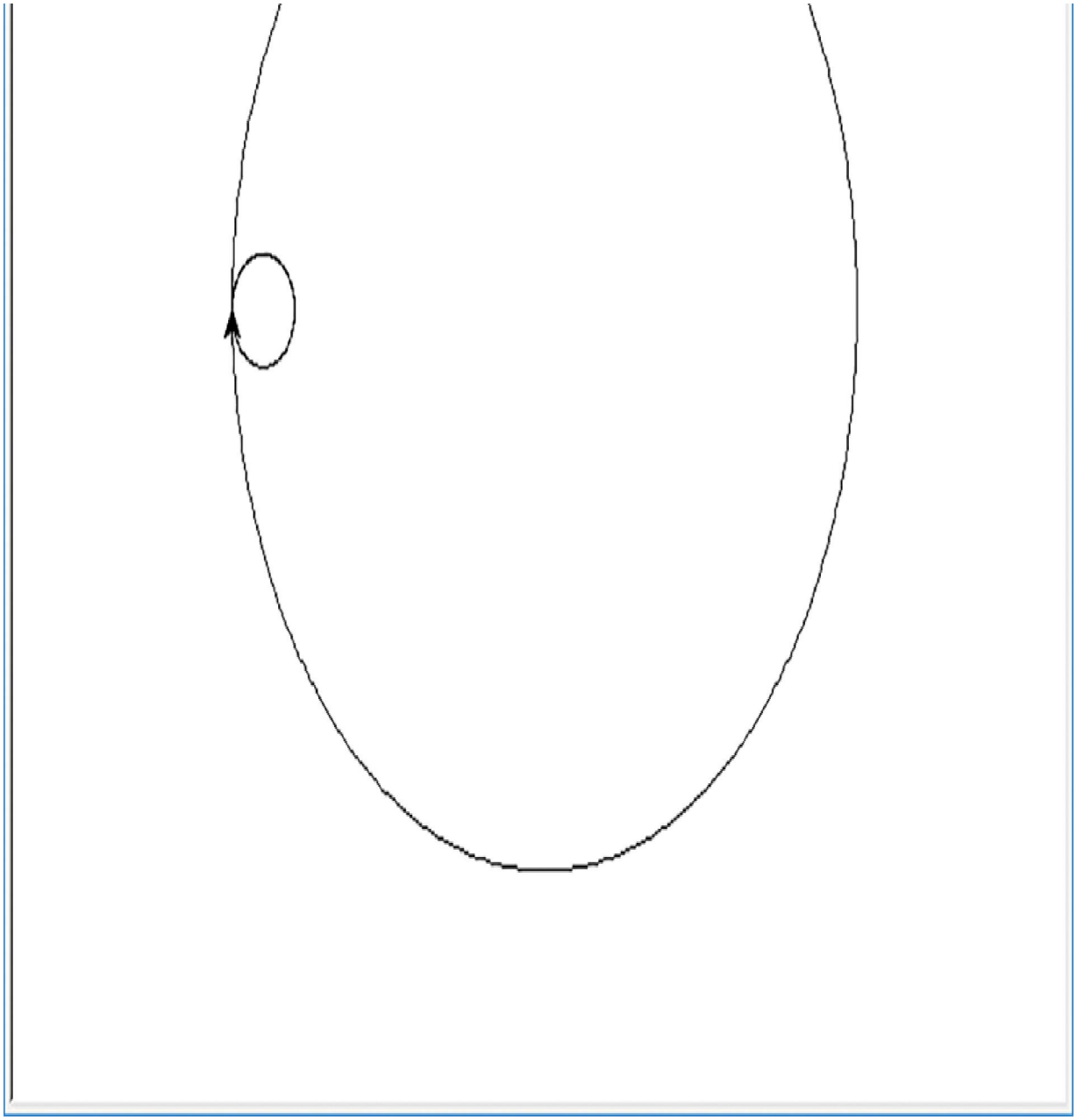


Figure 1.15 Drawing circles for a given radius.

© 2001–2025 Python Software Foundation; All Rights Reserved.

Description

Try It Out

1.43 Modify the `drawCircle` function so that the circle is drawn with the center at the turtle's present position.

1.44 The `drawCircle` function is somewhat inefficient: For small circles, drawing 360 sides is overkill; for very large circles, 360 sides might be too few. See if you can devise a way to make the number of sides and turning angle dependent on the radius so that smaller circles use fewer sides and larger circles use more sides.

1.6 Summary

This chapter introduced the following fundamental building blocks of programming and Python:

- Using primitive types
- Using expressions
- Naming objects
- Using the modules and functions provided by Python
- Writing your own functions to extend the functionality provided by Python
- Using the `for` statement to repeat a block of code multiple times

In addition, the approach we followed to use the turtle to draw a circle illustrates an important problem-solving pattern that you will use many times as you progress through this text. That pattern can be summarized as the following Good Programming Practice:

GOOD PROGRAMMING PRACTICE:
Use these problem-solving strategies when designing your programs:

- Simplify so that you can understand the problem better.
- Generalize to solve many problems with one function.
- Build on what you have learned to solve more complex problems.

We will continue to use these basic building blocks as we learn more about Python, and we will explore more tools that you can add to your toolbox. We have glossed over a few details on some of the ideas introduced in this chapter but will return to them later. As you delve into Python in more depth,

always keep in mind this critical idea: Focus on problem-solving while continually adding to your knowledge of programming and computer science.

Key Terms

abstraction
algorithms
assignment statement
attributes
block
calltips
camelCase
comment
complex number
computer science
constructor
data science
define
dereferencing
dot operator
expression
`for` loop
floating-point numbers
functions
function body
generalization
instances
integers
iteration

keywords
libraries
loop body
loop variable
machine learning
methods
modules
object
parameters
program
programming languages
Python
reference
repetition
representation
sequence object
simplification
statement
types
variables

Programming Exercises

1.1 Using the `drawSquare` function, you can have the turtle create an interesting flowerlike shape by drawing many squares. Each square is drawn after turning the turtle by some number of degrees between each square. Write a function `drawFlower` that takes the number of squares to draw as a parameter (`numSquares`) and draws a flower by repeating the square `numSquares` times. You will need to figure out how far to turn the turtle based on `numSquares`.

1.2 Write a function to make the turtle draw a five-pointed star.

1.3 Write a function to make the turtle draw an *n*-pointed star when *n* is restricted to be odd.

1.4 Write a function to have the turtle draw a simple line drawing of anything you want.

1.5 The turtle understands two other functions: `begin_fill()` and `end_fill()`. When you call `begin_fill`, the turtle keeps track of its starting point, and all the lines it draws, until you call `end_fill`. When `end_fill` is called, the turtle fills in the space enclosed by the lines that the turtle has drawn. Use these new functions to draw a more interesting picture.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 2
Playing by the Rules
Selection, Boolean Expressions, `while` Loops, an Introduction to Strings, Input, the `print` Function, and Random Numbers

2.1 OBJECTIVES

- To understand how rules can tell computers how to behave
- To introduce the string data type and perform simple searches
- To use Boolean expressions to create selection statements
- To learn to use the Python `print` function to output values
- To learn to use a `while` loop
- To understand the accumulator pattern
- To learn how functions can return a value
- To learn how to get input from the user
- To use random numbers to generate a game play

We know that computers can do only what we tell them to do. So, you may wonder, how is it that many computer programs/applications we interact with seem so "intelligent"? How can a computer play expert-level chess, or an NPC (nonplayable character) follow you on quests in your favorite role-

playing video game? How can Apple's Siri, Amazon's Alexa, the Google Assistant, or OpenAI's ChatGPT understand and respond to your questions? The answer is that we can make computers seem to be intelligent by giving them rules or having them learn patterns in data. This is the basis for the field of artificial intelligence (AI). Today, AI is used in many applications: diagnosing disease, suggesting products to online shoppers, detecting fraud in credit card transactions, identifying spam phone calls, and playing games. Covering the full gamut of AI is beyond the scope of this text, but we can apply some basic rules and strategies to approximate intelligent programs.

In this chapter, we will write some programs to make our computer seem intelligent, by creating rules for reacting to inputs and strategies for playing games.

2.2 The Chatbot

In our first AI-type application, we will create a simple rules-based chatbot. The chatbot can answer questions posed by the user. **Session 2.1** shows the finished program in action, assuming that the `chatbot` function is defined in the file `chatbot.py`. Each question is sent as a parameter to the `chatbot` function.

```
In [1]: from chatbot import *
```

```
In [2]: chatbot("Who brings gifts for lost teeth?")

        The tooth fairy
```

```
In [3]: chatbot("Where is the tooth fairy?")

        At the dentist's office
```

```
In [4]: chatbot("What is the meaning of life?")

        To be kind to one another
```

```
In [5]: chatbot("Why is the sky blue?")

        You'll understand when you're older.

In [6]: chatbot("Who is the best student in this class?")

        The tooth fairy

In [7]: chatbot("Where am I?")

        At the dentist's office

In [8]: chatbot("How did I get here?")

        You stumped me on that!
```

Session 2.1 The chatbot

Description

Can you tell what is going on in this program? At first, it appears that the chatbot is indeed intelligent. But if you look closely, you will see that the bot's answers depend only on the first word in the question. If the question starts with the word "Who," the answer is always "The tooth fairy." Similarly, the chatbot has predefined answers for any questions starting with "When," "Where," "What," or "Why." If the question does not start with any of these words, the chatbot answers, "You stumped me on that!" In other words, the computer is following rules, which dictate how the chatbot should respond to questions.

Now, can you imagine how we could write this program? It is obvious that the chatbot is working with words, rather than numbers. So, before we create this program, we need to explore the Python data type that handles words: the **string**.

It is also clear that the chatbot replies with different answers, depending on the first word in the question. To make this happen, we need to learn how a program can make decisions using a technique called **selection**.

Finally, the chatbot creates and outputs a response to each question, so we need to learn about the print statement.

We will explore each of these areas in turn.

2.3 The String Data Type

The Python data type for handling characters rather than numbers is the string. Strings are often used to represent words, names, or even a single character.

To define a string in Python, you surround the characters with either single, double, or triple quotes. Also, like `ints` and `floats`, strings are objects that can be named.

Strings are surrounded by quotes and can contain any characters: letters, spaces, numbers, and even punctuation marks. When you have a string that contains a single quote, use double quotes around the string. When you have a string that contains double quotes, use single quotes around the string. Strings can also be surrounded by triple single quotes (`'''`) or triple double quotes (`"""`), although this is rarely used. If the string does not contain either single or double quotes, it does not matter which quote style you use, as long as you start and end the string with the same quote style. **Session 2.2** illustrates some simple strings.

```
In [1]: "Hello"                    #use double quotes

Out[1]: 'Hello'

In [2]: 'world'                    #or single quotes

Out[2]: 'world'

In [3]: greeting = "Hello world"   #name a string
```

```
In [3]: greeting = 'Hello world'          #name a string

In [4]: greeting
Out[4]: 'Hello world'

In [5]: "Let's go"                        #embed ' inside "
Out[5]: "Let's go"

In [6]: 'She said, "How are you?", then left.'     #embed " inside '
Out[6]: 'She said, "How are you?", then left.'
```

Session 2.2 A variety of simple strings

Description

RECAP:
If a string contains a single quote, use double quotes around the string. If a string contains double quotes, use single quotes around the string. If a string does not contain either single or double quotes, you can use single or double quotes around the string, as long as the type of quotes is not mixed. In other words, if you start a string with a single quote, then you

must end the string with a single quote. Likewise, if you start a string with double quotes, then you must end that string with double quotes.

2.4 Making Decisions

Now we know the chatbot can handle each question as a string. We also know that we want the chatbot's answer to depend on the first word in the sentence. So, the chatbot needs some way to determine which word starts the question the user is asking. To do that, we can use Boolean expressions and a selection statement.

2.4.1 Boolean Expressions

A **Boolean expression** compares values of objects. The result is one of the two Boolean data values: `True` or `False`. These Boolean values are another primitive type in Python.

The easiest type of Boolean expression compares the results of two expressions. To make this comparison, we use operators you might recognize from mathematics: **equality operators** to compare if two values are equal, and **relational operators** to test how the two values compare. **Table 2.1** shows equality and relational operators and their meanings in Python.

TABLE 2.1 Equality and Relational Operators

Equality Operator	Meaning
==	Equal to
!=	Not equal to
Relational Operator	**Meaning**
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Statements that compare two data values using a relational or equality operator are called **relational expressions**. As with other expressions in Python, evaluating relational expressions produces a result—in this case, a

Boolean value. **Session 2.3** shows some simple examples of relational expressions.

```
In [1]: dogWeight = 25          #assign 25 to dogWeight
```

```
In [2]: dogWeight
```

```
Out[2]: 25
```

```
In [3]: dogWeight == 25         #is dogWeight equal to 25?
```

```
Out[3]: True
```

```
In [4]: dogWeight != 25         #is dogWeight not equal to 25?
```

```
Out[4]: False
```

```
In [5]: dogWeight < 25          #is dogWeight less than 25?
```

```
Out[5]: False
```

```
In [6]: dogWeight <= 25          #is dogWeight less than or equal to 25?
```

```
Out[6]: True
```

```
In [7]: dogWeight > 25           #is dogWeight greater than 25?
```

```
Out[7]: False
```

```
In [8]: dogWeight >= 25          #is dogWeight greater than or equal to 25?
```

```
Out[8]: True
```

```
In [9]: dogWeight == 5 * 5       #is dogWeight equal to result of calculation?
```

```
Out[9]: True
```

```
In [10]: catWeight = 18          #assign 18 to catWeight
```

```
In [11]: dogWeight == catWeight  #is dogWeight equal to another variable?
```

```
Out[11]: False

In [12]: today = "Monday"          #assign "Monday" to today

In [13]: today == "Monday"         #strings can also be compared

Out[13]: True

In [14]: today == "monday"         #comparing strings is case sensitive

Out[14]: False
```

Session 2.3 Evaluating simple relational expressions

Description

Note that the variable `dogWeight` is assigned the value `25` using the assignment operator (=) and is then compared to 25 using the equality operator (==). It is important to distinguish between the assignment operator, =, and the equality operator, ==. Note also that in statement 9 in the session, the value of the variable is compared against the result of an arithmetic expression—namely, `5 * 5`. With this statement, Python first computes the product, then makes the comparison.

Also note that strings, as well as numbers, can be compared using the relational and equality operators. Each character in a string has a value

assigned according to the **Unicode encoding system**. The equality operator compares each character in the first string to the corresponding character in the second string. If all characters match, the result is `True`, as shown in statement 13. The result for statement 14 is `False`, however, because capital letters are assigned different Unicode values from lowercase letters. This will be important in our simple chatbot program.

HEADS UP:

Do not confuse the assignment operator (=) with the equality operator (==). The assignment operator assigns a value to a variable, whereas the equality operator compares whether two expressions are equal and evaluates to `True` or `False`.

2.4.2 Compound Boolean Expressions and Logical Operators

In Python, **compound Boolean expressions** are composed of simple Boolean expressions that are connected by one of the **logical operators**: `and`, `or`, and `not`. **Table 2.2** defines the behavior of the Python logical operators. Note that *any* Python expression can be used as part of a Boolean expression, but most often the expressions will be relational expressions.

TABLE 2.2 Logical Operator Behavior

Expression	Evaluation
`x and y`	`True` if both `x` and `y` are `True`; otherwise `False`.
`x or y`	`True` if either `x` or `y` is `True`; otherwise `False`.
`not x`	`True` if `x` is `False`; otherwise `False`.

It is important to understand that Python uses **short-circuit evaluation** of a Boolean expression, which means it evaluates only as much of the expression, from left to right, as necessary to determine whether the expression is `True` or `False`.

For example, in this Boolean expression using `or`,

```
3 < 7 or 10 < 20
```

Python needs to evaluate only the expression `3 < 7` because this is `True`, and only one expression needs to be `True` for a compound expression with `or` to be `True`. It makes no difference to the outcome if `10 < 20` is `True` or `False`. Similarly, in this expression,

```
10 > 20 and 3 < 7
```

Python needs to evaluate only the `10 > 20` part of the expression, because `10 > 20` is `False`. Thus, the whole expression must be `False`, because with `and`, both expressions need to be `True` for the whole expression to be `True`. As in the previous example, it makes no difference to the outcome whether `3 < 7` is `True` or `False`. **Session 2.4** shows the logical operators in action.

```
In [1]: 6 < 10 and 3 < 7

Out[1]: True

In [2]: 6 < 10 and 10 < 6

Out[2]: False

In [3]: 4 != 4 or 5 < 8
```

```
Out[3]: True

In [4]: 4 != 4 or 8 < 5

Out[4]: False

In [5]: not 6 < 10

Out[5]: False

In [6]: not 6 > 10

Out[6]: True
```

Session 2.4 Evaluating compound Boolean expressions

Description

HEADS UP:
Note that the logical operators `and` and `or` take two operands. In contrast, `not` takes only one operand.

Try It Out

2.1 What is the result of evaluating the Boolean expression `False or True`?

2.2 What is the result of evaluating the Boolean expression `True and False`?

2.3 What is the result of evaluating the Boolean expression `not 7 > 3`?

2.4 What is the result of evaluating the Boolean expression `not (True or False)`?

2.5 What is the result of evaluating the Boolean expression `(not True) and (not False)`?

2.6 Write an equivalent Boolean expression for `(not (True and False))`.

2.7 Write a compound Boolean expression that evaluates to `True` if the value of the variable `count` is between `1` and `10` inclusive.

RECAP:
The equality, relational, and logical operators allow you to test values by creating expressions that evaluate to `True` or `False`.

2.4.3 Selection Statements

Once we can test values by writing a Boolean expression, we can use those results for making a decision. In computer science, decisions are often called selection because we wish to select between possible outcomes based on the result of a test we run. For example, when we go outside in the morning, we might ask the question, "Is it raining?" If it is raining, we will grab an umbrella. Otherwise, we will pack our sunglasses. In this case, we are selecting which item to take with us based on the condition of the weather.

Until now, the statements in our Python programs have been executed in sequence—one at a time, in the order they were written. We now introduce the **selection statement**, also known as an `if` statement. The selection statement contains a test, called a **condition**, and groups of statements that may or may not be executed, depending on whether the condition evaluates to `True` or `False`. Most programming languages provide two versions of this useful statement: the `if` and the `if-else`.

The `if-else` statement looks like this:

```
if ‹condition›:
    ‹statements›  #execute only if condition is true
else:
    ‹statements›  #execute if condition is false
```

Description

The keyword `if` is followed by a Boolean expression that is used as a condition for the selection and then a colon. When the Boolean expression is evaluated, the result will either be `True` or `False`. If the condition is `True`, the first block of statements is executed in sequence and the second block is skipped. If the condition is `False`, the first block of statements is skipped and the second block is executed in sequence (see **Figure 2.1**). In this way, it is possible to perform different actions based on the result of the condition that was tested.

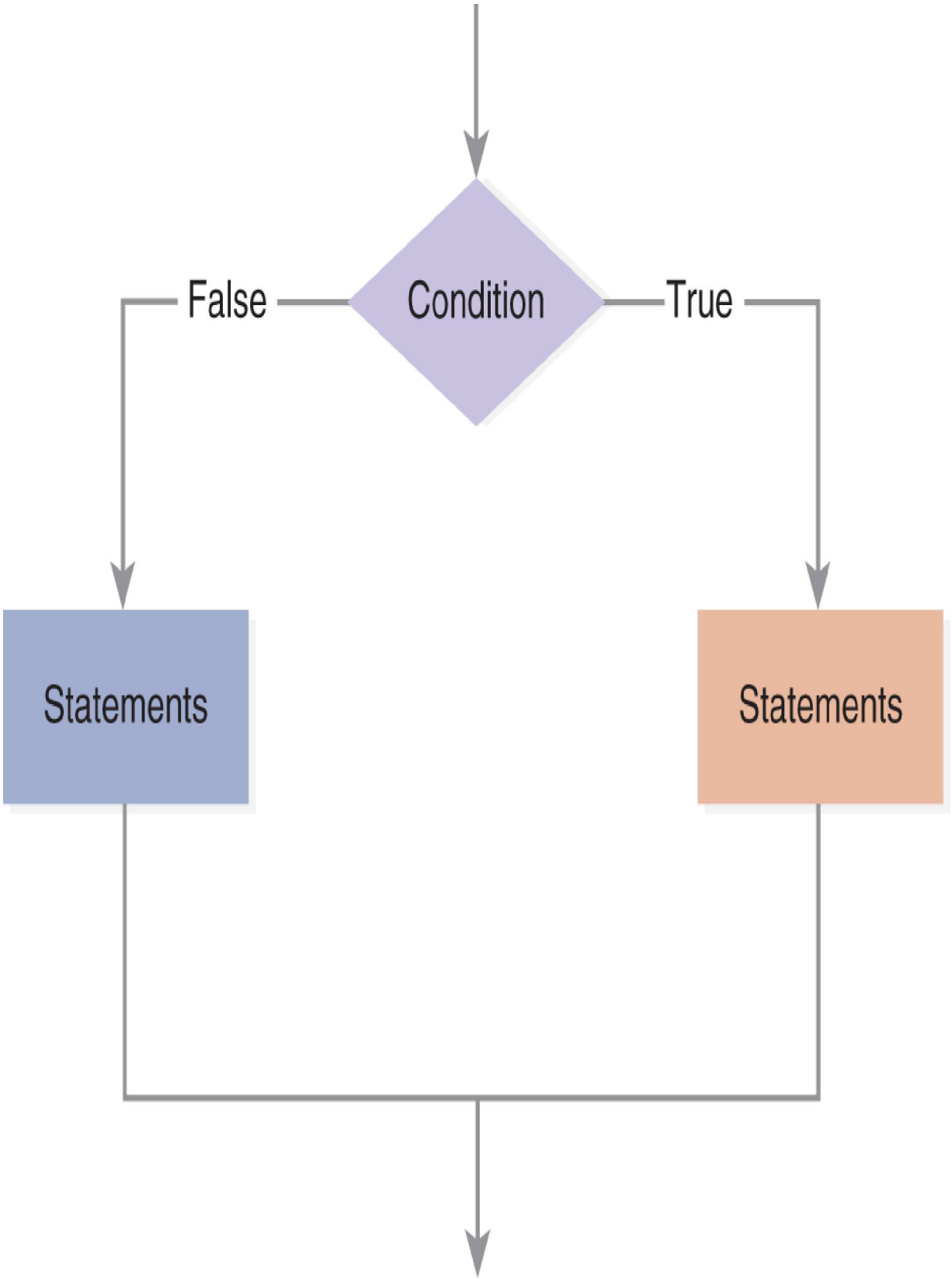


Figure 2.1 The logical flow of an `if-else` statement.

Notice that a colon follows the condition and that each block of statements is indented just like a `for` loop or function definition.

In **Session 2.5**, the variables `a` and `b` are assigned initial values. The `if-else` statement that follows asks whether `a` is greater than `b`. Because that is `True`, the first block of statements (in this case, only one statement) is executed, and the value of the variable `c` is set to `10`. We can verify that this did occur by printing the variable `c` after the statement is done. Then in statement 5, we change the value of `b` to be greater than `a`. When we re-execute the `if-else` statement, the condition is `False`, so the first statement is skipped and the statement after the `else` is executed. Again, we verify this by printing the value of `c`, which is now 20.

```
In [1]: a = 5
```

```
In [2]: b = 3
```

```
In [3]: if a > b:
            c = 10        #executed only if a > b
        else:
            c = 20        #executed only if a <= b
```

```
In [4]: c
```

```
Out[4]: 10

In [5]: b = 6

In [6]: if a > b:
            c = 10
        else:
            c = 20

In [7]: c

Out[7]: 20
```

Session 2.5 Executing a simple `if-else` selection statement

Description

The other variation of the `if` statement does not provide an `else` clause. It looks like the following:

```
if ‹condition›:
    ‹statements›  #execute only if condition is true
```

Description

As before, the condition is evaluated, and the result will be either `True` or `False`. If the result is `True`, the indented block of statements is executed in sequence. Conversely, if the condition is `False`, nothing is performed, the statement completes, and the program goes on to the next statement following the `if` statement (see **Figure 2.2**).

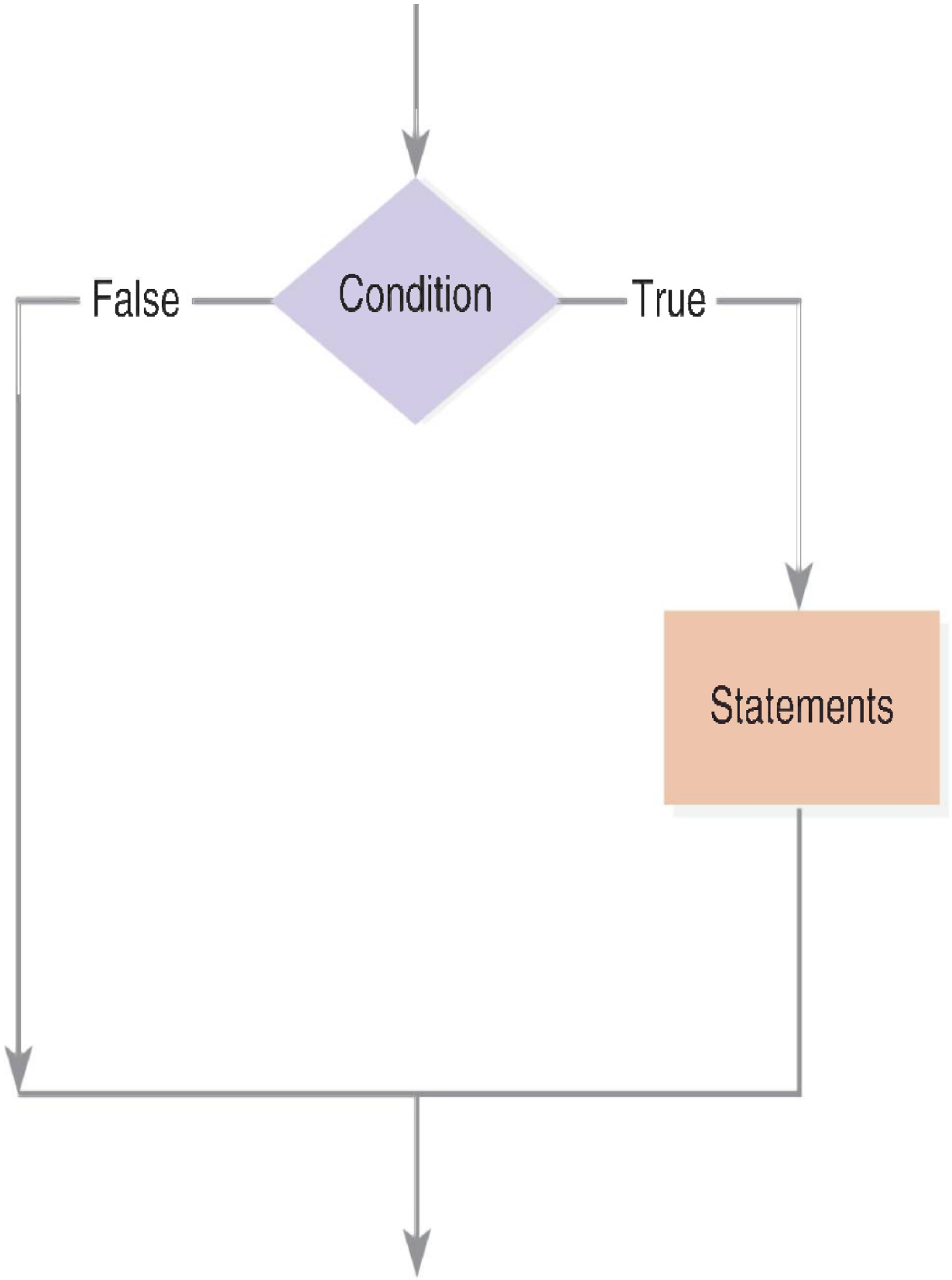


Figure 2.2 The logical flow of an `if` statement.

For example, in **Session 2.6**, the variables `a` and `b` are again assigned initial values. The `if` statement that follows asks whether `a` is greater than `b`. Because the answer is `True`, the block of statements (again, only one statement in this case) is performed, and the value of the variable `c` is set to `10`. In the second selection, however, the condition (`b > a`) evaluates to `False`, so the statement after the condition is not executed. In this case, the value of `c` remains the same as it was before the selection.

```
In [1]: a = 5
```

```
In [2]: b = 3
```

```
In [3]: if a > b:       #condition is True
            c = 10      # this statement is executed
```

```
In [4]: c
```

```
Out[4]: 10
```

```
In [5]: if b > a:       #condition is False
```

```
    c = 25    #this statement is not executed
```

```
In [6]: c
```

```
Out[6]: 10
```

Session 2.6 Executing a simple `if` selection statement

Description

Since the statements contained within a selection statement can be any legal Python statement, it is certainly possible to place one `if` statement inside another. This practice is sometimes referred to as **nested selection**. The following `if-else` statement has two nested `if-else` statements:

```
if <condition1>:
    if <condition2>:
        <statements>  #condition1 is true and condition2 is true
    else:
        <statements>  #condition1 is true and condition2 is false
else:
    if <condition3>:
        <statements>  #condition1 is false and condition3 is true
    else:
        <statements>  #condition1 is false and condition3 is false
```

Description

The result of this structure is to decide among four blocks of statements. If `condition1` is `True`, then the first nested `if-else` is performed. If the nested `condition2` is also `True`, then its first block of statements is performed. However, if the nested `condition2` is `False`, its second block of statements is executed. Likewise, if `condition1` is `False`, the second nested `if-else` will be performed. In this way, we can decide among four groups of statements by using the results of three conditions.

One common pattern for nested selection is called **tail nesting**. With tail nesting, all of the nested selection occurs in the `else` of the previous `if-else` statement. The structure looks like this:

```
if <condition1>:
    <statements>          #condition1 is true
else:
    if <condition2>:
        <statements>      #condition1 is false and condition2 is true
    else:
        if <condition3>:
            <statements>  #condition1 and condition2 are false
                          #and condition3 is true
        '''
```

Description

This tail nesting pattern is so common that Python provides a shorter version known as the `if-elif-else` statement. In this statement, the combination of the `else` and the next `if` allows us to simply list the conditions and associated statements one at a time without additional indentation. Note that the last case uses a simple `else` followed by a colon. The statements after the final `else` are executed if none of the previous conditions are `True`.

```
if ‹condition1›:
    ‹statements›      #condition1 is true
elif ‹condition2›:
    ‹statements›      #condition1 is false and condition2 is true
'''
elif ‹conditionN›:
    ‹statements›      #conditions 1 - (N - 1) are false
                      #and conditionN is true
else:
    ‹statement›       #conditions 1 - N are all false
```

Description

Session 2.7 shows a selection statement using `if-elif-else` to calculate a grade based on a score.

```
In [1]: score = 75
```

```
In [2]: if score >= 90:
            grade = 'A'
        elif score >= 80:
            grade = 'B'
        elif score >= 70:
            grade = 'C'
        elif score >= 60:
            grade = 'D'
        else:
            grade = 'F'
```

```
In [3]: grade

Out[3]: 'C'
```

Session 2.7 Executing an `if-elif-else` selection statement

Description

HEADS UP:

After a block of statements is selected to be executed, the remainder of the `if` statement is skipped.

RECAP:

Python provides three forms of selection statements:

`if`	For executing a set of statements only if a condition is `True`
`if-else`	For executing one set of statements if the condition is `True` and executing a separate set of statements if a condition is `False`
`if-elif-else`	For selecting a set of statements to execute based on multiple mutually exclusive conditions

Try It Out

2.8 How does Python know to which `if` an `else` belongs?

2.9 Write a selection statement that sets the value of a variable named `answer` to `1` if a variable named `result` is equal to `100`, and to `2` otherwise.

2.10 Write a nested selection using `if-else` that sets the value of a variable `gradePoint` to `4` if a variable named `score` is greater than or equal to `90`; to `3` if `score` is between `80` and `89`; to `2` if `score` is between `70` and `79`; to `1` if `score` is between `60` and `69`; and to `0` otherwise.

2.11 Rewrite the selection in Exercise 2.10 using an `elif` statement.

2.12 A year is a leap year if it is divisible by 4, unless it is divisible by 100, but not divisible by 400. For example, 2024 is a leap year (divisible by 4, but not by 100); 1900 is not a leap year (divisible by 4 and 100, but not by 400); 2000 is a leap year (divisible by 4, 100, and 400). Write a function that takes a year as a

parameter and sets a variable named `isLeapYear` to `True` if the year is a leap year and `False` otherwise.

2.13 A fruit company sells oranges for 32 cents per pound plus $7.50 per order for shipping. If an order weighs more than 100 pounds, the shipping cost is reduced by $1.50. Write a function that will take the number of pounds of oranges as a parameter and set a variable `totalCost` to the cost of the order.

2.14 Write a function that takes three integers as parameters and sets the variable `largest` to the largest integer.

2.15 Write a function that takes two parameters—a pay rate and the number of hours worked—and sets a variable named `totalPay` to the amount owed to the employee. Any hours over 40 are paid at time and a half.

2.5 Implementing the Chatbot

Now we are ready to complete our chatbot program. Again, we will use the abstraction idea to build a function that will accept a question as a parameter and respond with an answer.

Recall the basic rules for the chatbot. It selects a response based on the first word in the question. But how can we test for the first word in the sentence? If we build a selection condition using the equality operator to compare the question to our "W" words, we would be testing the entire question. We want to test only if the question *starts* with the word. Fortunately, like the `turtle` data type, the string data type has methods. For now, we will look at one method that is particularly helpful here: `startswith`, which tests whether a string starts with a specific set of characters. Because `startswith` returns `True` or `False`, we can use the method in a condition for our selection statement. The format of `startswith` is shown in **Table 2.3**.

TABLE 2.3 The String Data Type's `startswith` Method

Method	Description
`startswith(str)`	Returns `True` if the string starts with `str`; `False` otherwise.

Because `startswith` is a method of the string data type, we need to call `startswith` using the `dot` operator, as in the following code:

```
if question.startswith("Who"):
```

Because we want to test the first word of the question against each of our "W" words, it makes sense to use an `if-elif-else` selection statement. The statement to execute whenever the condition is `True` will be to output our predetermined answer for any question beginning with that word. If the first

word is not one of our "W" words, we execute our final `else` statement, which outputs a message saying the chatbot is stumped.

One final Python function to learn before writing our chatbot program is the `print` function. This is a Python function, not a string method, so `print` can be called without the dot operator. In its simplest form, the function merely outputs a message. The format of the `print` function is shown in **Table 2.4**.

TABLE 2.4 The `print` Function

Function	Description
`print(str)`	Outputs `str`.

Now we are ready to have the chatbot come to life. **Listing 2.1** shows the code. Try running Session 2.1 to see the predefined responses.

```
def chatbot(question):
    if question.startswith("Who"):
        print("The tooth fairy")
    elif question.startswith("What"):
        print("To be kind to one another")
    elif question.startswith("Where"):
        print("At the dentist's office")
    elif question.startswith("Why"):
        print("You'll understand when you're older.")
    elif question.startswith("When"):
        print("This afternoon")
    else:
        print("You stumped me on that!")
```

Listing 2.1 The `chatbot`

Description

We can add a bit more intelligence to our chatbot by adding some more rules. For one thing, we can check for more words. For example, we might want the chatbot to respond "The tooth fairy" not only when the question starts with "Who," but also if the question contains the word "name." We do this by combining two conditions using a logical `or`. Or, we could add a response to a question that starts with "How" and also contains the word "feeling" by combining two conditions using a logical `and`. We can add these

new rules and any other rules by simply adding conditions to the `chatbot` code to test for words both at the beginning and also within the sentence. To test for words within the question, we can use the string data type's `in` operator, which is shown in **Table 2.5**.

TABLE 2.5 The String Data Type's `in` Operator

Operator	Description
`str1 in str2`	Has the value `True` if `str1` is contained in `str2`; `False` otherwise.

One other important change to the chatbot is to make the question case-insensitive. Remember that capital and lowercase letters do not compare as equal because they have different Unicode values. As it stands now, our chatbot will not recognize a question that starts with "why," "WHY," or any variation of capital and lowercase letters other than specifically "Why." We do not want the user of this program to worry about capitalization. We can easily solve this problem and make the program more **user-friendly** by converting the question to lowercase and then comparing the first word to lowercase versions of `who`, `what`, `why`, `where`, and `when`. To do this, we can use another string method, `lower`, shown in **Table 2.6**, to convert the question to lowercase before doing any comparisons. Note that the `upper` method could also be used with the `W` words all capitalized. Either way will work fine.

TABLE 2.6 The String Data Type's `upper` and `lower` Methods

Method	Description
`lower()`	Returns the string with all letters lowercase.
`upper()`	Returns the string with all letters uppercase.

Listing 2.2 shows our revised `chatbot2` using compound conditions and the `in` operator and `lower` method.

```
1  def chatbot2(question):
2      question = question.lower()
3
4      if question.startswith("who") or "name" in question:
5          print("The tooth fairy")
6      elif question.startswith("what"):
7          print("To be kind to one another")
8      elif question.startswith("how") and "feeling" in question:
9          print("I don't feel anything. I'm a computer!")
10     elif question.startswith("where"):
11         print("At the dentist's office")
12     elif question.startswith("why"):
13         print("You'll understand when you're older.")
14     elif question.startswith("when"):
15         print("This afternoon")
16     else:
17         print("You stumped me on that!")
```

Listing 2.2 The revised `chatbot2`

Description

GOOD PROGRAMMING PRACTICE:

Making user input case-insensitive is one way to make your programs user-friendly.

Session 2.8 exercises the new features in our now "more intelligent" chatbot.

```
In [1]: from chatbot2 import *

In [2]: chatbot2("WHO is at the door?")

        The tooth fairy

In [3]: chatbot2("How are you feeling?")

        I don't feel anything. I'm a computer!

In [4]: chatbot2("What is your name?")

        The tooth fairy

In [5]: chatbot2("What is the meaning of life?")

        To be kind to one another
```

Session 2.8 Asking `chatbot2` some questions

Description

There are a few things to note about Session 2.8. In statement 2, the question uses a capitalized "WHO"—which `chatbot2` recognizes to be the same as "who"—because we converted the sentence to lowercase before beginning the comparisons. In statement 3, we get the new response for the words "how" and "feeling." Although statement 4 starts with "What," we get the response from line 4 of `chatbot2`, rather than from line 6, because the question contains the word "name." Remember that once an `if-elif-else` statement finds a match, the rest of the conditions are skipped. Statement 5 in Session 2.8, which starts with "What," but does not contain the word "name," simply matches line 6 in `chatbot2`.

Try It Out

2.16 The chatbot's responses are arbitrary. Rewrite the chatbot to output your own responses.

2.17 Rewrite the chatbot to handle a question that starts with the word "Where" and contains "world."

2.18 Rewrite the chatbot so that it capitalizes the question parameter.

2.19 Add a response to any question starting with "How."

2.6 Playing Games

All games have one or more goals and rules for players to follow. Think about your favorite game and recall when you first learned to play that game. It might have been hide-and-seek, a sport like baseball, checkers, a card game like Go Fish, or a video game. You might have watched others play and surmised some of those goals or rules, but more likely you asked someone, "How do you play?"

Now, imagine you are the one explaining that game to a new friend. You need to define the goals and to outline the rules clearly to your friend. Computers are no different. Recall that the field of artificial intelligence (AI) involves two major approaches: rules based and learned patterns ("machine learning"). Computers can "learn" a game from observing other games and discern patterns from those games, and/or you can give the computer explicit goals and rules for playing the game. In addition, you can also provide a "strategy" for winning (i.e., when this event happens, make sure you do this!). In the following sections, we will explore some of these rule-based approaches for playing and winning a game.

2.6.1 Guess a Number

Perhaps you are familiar with the Guess a Number game, where one person chooses a secret number and the other person tries to guess that number. With each guess, the first person responds with clues that the guess is too high, too low, or correct when the other person has guessed the number. We want to build a program for the computer to play this game, with the computer guessing the number.

Suppose the secret number is between 1 and 100. If the computer ignores the clues and randomly chooses numbers, the game could continue for some time without success. In this case, it may take ***up to*** 100 attempts for the computer to guess correctly. But there is an optimum approach to guessing the number that requires only a handful of guesses. Given that the secret number is between 1 and 100, the computer's first guess should be 50 (halfway between 1 and 100). The user will respond that the guess is either too high or

too low (or maybe correct if the secret number is, in fact, 50). If the guess is incorrect, the computer has now eliminated half of the possible numbers. If the guess is too low, the secret number must be between 51 and 100. So the computer's next guess should be 75 (halfway between 51 and 100). If the guess is too high, the secret number must be between 1 and 49. So the computer's next guess should be 25 (halfway between 1 and 49). As the computer continues to guess, it continues to eliminate half the remaining possible numbers. In this way, the computer will quickly find the secret number. In computer science, this approach is called a **binary search**, because the possible numbers are cut in half each time. The algorithm for a binary search is shown here:

```
find and guess the middle number (end + start) // 2
while the guess is not correct:
    If guess is too high,
        set end to guess - 1
    Else if guess is too low,
        set start to guess + 1
    Find and guess the middle number (end + start) // 2
```

Description

Notice that this algorithm uses looping. We want to repeat guessing a number until the guess is correct. So far, we have used a `for` loop when we have needed to repeat an operation a specific number of times. But in this case, we do not know how many guesses it will take to find the correct number. So we need to use a different kind of loop. The **`while` loop** is ideal for this type of repetition because it repeats its operations until an event occurs or a special value is found, often called a **sentinel value**. In our game, the event is guessing the correct number.

We will also need to use other techniques, such as getting input from the user and printing statements that combine strings and numbers. Let's look at each of these techniques in turn.

2.6.2 The `while` Loop

The structure of a `while` loop is as follows:

```
priming statement
while <condition>:
    statement 1
    statement 2
    . . .
    statement to update the condition
```

Description

As you might guess from the template shown here, the `while` loop continues to perform the statements in the body of the loop as long as the condition remains `True`. Because we want the `while` loop to execute until an event occurs, the loop condition should evaluate to `True` if the event has not occurred. Once the condition evaluates to `False`, the `while` loop stops. The condition can be any Python Boolean expression.

You might notice that the condition is evaluated at the beginning of the loop. Usually, with a `while` loop, we set up a **priming statement** before the loop that causes the condition to evaluate to `True` or `False` when the loop begins. If the condition evaluates to `False` when the `while` loop is first encountered, then the loop body is never executed.

The final thing to do in the loop body is to execute a statement that updates the loop condition, causing the condition to eventually evaluate to `False`. When the condition evaluates to `False`, the loop ends.

In our guessing game, the event to end the loop is guessing the correct number. Before the loop begins, we generate the first guess as a priming statement. The loop condition will be that our guess is not the correct number. As the last statement of the `while` loop body, we generate a new guess, which will update the loop condition for the next iteration. If we do not generate a new guess, we would continue to process the same guess again and again, causing an **infinite loop**. Infinite loops are almost never good, and it is important to make sure that something changes in the body of your loop so that the condition will eventually evaluate to `False`.

HEADS UP:
Be careful to update the `while` loop condition in the loop body to avoid an infinite loop.

2.6.3 Printing Numbers in Strings

Before writing the code for the guessing game, we need to explore one other technique. We want to print a message to the user asking if our guess is correct. It might look like this:

```
"Is 50 too high, too low, or correct?"
```

As we have learned, the `print` function takes a string as a parameter, but our guess is an `int`. So we need to convert the guess to a string. To do this and create our message to the user, we can use the string concatenation operator and the Python `str` function.

The **string concatenation operator** (+) combines two strings into one string. Although + is also the addition operator, when + is applied to two strings, Python recognizes + as the concatenation operator. The concatenation operator works only on strings. If you want to combine a string and a number, the number must first be converted to a string using the `str` function, as described in **Table 2.7**. **Session 2.9** uses the string concatenation operator and the `str` function to combine strings and numbers. In command 2, the parentheses around 1 + 1 indicates to Python that we want to add the two numbers, rather than concatenate them. Note that the concatenation operator does not insert a space between the two strings. To add a space, you must insert a space into the string itself.

TABLE 2.7 Python's Function for Converting a Number to a String

Function	Description
`str(num)`	Converts the integer or floating-point number `num` to a string.

```
In [1]: sum = 1 + 1
        '1 plus 1 equals ' + str(sum)

Out[1]: '1 plus 1 equals 2'

In [2]: '1 plus 1 equals ' + str(1 + 1)        # + in parentheses means addition

Out[2]: '1 plus 1 equals 2'

In [3]: '1 plus 1 DOES NOT equal ' + str(1) + str(1)

Out[3]: '1 plus 1 DOES NOT equal 11'

In [4]: votingAge = 18
        print("You can vote if you are " + str(votingAge) + " years old.")

        You can vote if you are 18 years old.
```

Session 2.9 Using concatenation and the `str` function

Description

HEADS UP:
The concatenation operator (+) does not automatically add a space between strings. Insert spaces in strings where you want them to appear.

2.6.4 Asking for Input

There is one more technique we need to cover before playing our Guess a Number game. The computer will ask the user whether each guess is "too high," "too low," or "correct."

So we need to learn how to get input from the user.

Python provides a function that allows us to ask a user to enter some data and returns a reference to the data in the form of a string. This function is called `input`.

As shown in **Table 2.8**, Python's built-in `input` function takes a single parameter that is a string. This string is often called the **prompt** because it contains some helpful text prompting the user to enter something. For example, you might call `input` as follows:

```
msg = input('Enter your question: ')
```

TABLE 2.8 The `input` Function

Function	Description
`input(prompt)`	Outputs the prompt, then returns a string containing any characters typed by the user when the user presses enter or return. The enter or return key is not part of the returned string.

This causes the prompt string to be displayed. Python then waits for the user to type something and press the enter or return key. Whatever the user types will be stored in the `msg` variable. Note that the `input` function always returns a string. If the string represents a value that we want to use as a number, we need to convert the string to a numeric type using the `float` or

`int` functions. **Session 2.10** shows the `input` function used with the `int` and `str` functions.

```
In [1]: name = input("What is your name? ")
        print("Hello, " + name)

        What is your name? Monica
        Hello, Monica
```

```
In [2]: test1 = input("Enter the first test score: ")
        test2 = input("Enter the second test score: ")
        test3 = input("Enter the third test score: ")
        average = (int(test1) + int(test2) + int(test3)) / 3
        print("The average test score is " + str(average))

        Enter the first test score: 92
        Enter the second test score: 85
        Enter the third test score: 88
        The average test score is 88.33333333333333
```

Session 2.10 Getting input from the user

Description

HEADS UP:
When inputting a string, the user does not enclose the input in quotes.

GOOD PROGRAMMING PRACTICE:
End the prompt with a space so that the user's input is separated from the prompt. This makes the input more readable.

Try It Out

2.20 Rewrite the chatbot to input the question rather than accept it as a parameter.

2.21 Write a function that validates an email address. The function should input an email address and return "Valid" if the email address contains an at sign (@) and a dot (.), or "Invalid" otherwise.

2.22 Write a function to determine if a string sent as a parameter is a hashtag. The hashtag should begin with a "#" and cannot contain spaces.

2.7 Completing Guess a Number

We are now ready to write our Guess a Number game. **Listing 2.3** shows our first pass at the game with the secret number being between 1 and 100.

```
1  def guessANumber():
2      start = 1
3      end = 100
4
5      print("Pick a number between " + str(start) + " and " + str(end) + ".")
6      print("I'll try to guess it in as few guesses as possible.")
7
8      guess = (end + start) // 2
9      response = input("Is " + str(guess) + " too high, too low, or correct? " )
10     response = response.lower()
11
12     while response != 'correct':
13         if response == 'too high':
14             end = guess - 1
15         else: # response == 'too low':
16             start = guess + 1
```

```
17
18          guess = (end + start) // 2
19          response = input("Is " + str(guess) + " too high, too low, or correct? " )
20          response = response.lower()
21
22      print("Your number is " + str(guess))
```

Listing 2.3 Guess a Number, pass 1

Description

In this program, we first set the `start` and `end` values to 1 and 100, respectively. Then we print a message to the user explaining the game. Next, in lines 8 to 10, we generate the first guess and ask the user to respond. This primes the `while` loop, whose condition is that the user does not respond with "correct." Inside the `while` loop, we check whether the user's response was "too high," in which case we adjust the `end` variable to be one less than the guess. Otherwise, we assume the user responded with "too low," and we adjust the `start` variable to be one more than the guess. Then we perform the loop update, which generates a new guess and prompts the user for a response. **Session 2.11** shows the game running when the user has chosen the secret number to be 73.

```
In [1]: from GuessANumberPass1 import *
```

```
In [2]: guessANumber()   #secret number is 73
```

```
Pick a number between 1 and 100.
I'll try to guess it in as few guesses as possible.
Is 50 too high, too low, or correct? too low
Is 75 too high, too low, or correct? too high
Is 62 too high, too low, or correct? too low
Is 68 too high, too low, or correct? too low
Is 71 too high, too low, or correct? too low
Is 73 too high, too low, or correct? correct
Your number is 73
```

Session 2.11 The computer guesses the secret number

Description

2.8 Validating User Input

In the program, we convert the user's response to lowercase to make the input case-insensitive. But we can further improve the user friendliness of the program. Note that the `while` loop condition checks for "correct," and in the loop body, we check for "too high." With this simple `if-else` structure, if the user enters anything other than "too high," we assume the user entered "too low." But what if the user enters something else, like "Hello"? The program would execute the `else` statement in the loop body and change `start` to be `guess + 1`. That is obviously not the correct behavior for the program. We do not want to generate a new guess because we have not received a valid response for the current guess. To solve this problem, we need to add a third check inside the `while` loop to verify that the user response is valid. Inside the loop, any response other than "too high" or "too low" should trigger a message asking the user to reenter the response, and we need a way to avoid executing the loop update. Fortunately, Python's `continue` statement, shown in **Table 2.9**, can be used to skip the remainder of the loop body, when needed.

TABLE 2.9 The `continue` Statement

Statement	Description
`continue`	Skips the remaining statements in the loop body and returns control to the loop condition.

The `if-else` statement inside the loop body now becomes an `if-elif-else` statement, as shown in **Listing 2.4**. **Session 2.12** exercises the new input checks in the Guess a Number game.

```
1 def guessANumber():
2     start = 1
```

```
3   end = 100
4
5   print("Pick a number between " + str(start) + " and " + str(end) + ".")
6   print("I'll try to guess it in as few guesses as possible.")
7
8   guess = (end + start) // 2
9   response = input("Is " + str(guess) + " too high, too low, or correct? ")
10  response = response.lower()
11
12  while response != 'correct':
13      if response == 'too high':
14          end = guess - 1
15      elif response == 'too low':
16          start = guess + 1
17      else:                       #invalid response
18          response = input("Please enter 'too high', 'too low', or 'correct' " )
19          response = response.lower()
20          continue                #skip the remainder of the loop
21
22      guess = (end + start) // 2
```

```
23      response = input("Is " + str(guess) + " too high, too low, or correct? ")
24      response = response.lower()
25
26  print("Your number is " + str(guess))
```

Listing 2.4 Guess a Number, pass 2

Description

```
In [1]: from GuessANumberPass2 import *
```

```
In [2]: guessANumber()    # number is 31
```

```
Pick a number between 1 and 100.
I'll try to guess it in as few guesses as possible.
Is 50 too high, too low, or correct? too high
Is 25 too high, too low, or correct? too low
Is 37 too high, too low, or correct? a little high
Please enter 'too high', 'too low', or 'correct' hello
Please enter 'too high', 'too low', or 'correct' too high
Is 31 too high, too low, or correct? correct
Your number is 31
```

Session 2.12 The user enters some invalid responses

Description

GOOD PROGRAMMING PRACTICE:
Wherever possible, verify that input from the user is valid. If not valid, allow the user to reenter the input.

2.9 Counting the Guesses

Using the binary search method, the computer is able to guess the secret number in just a few attempts. In fact, the maximum number of attempts is related to the range of numbers being searched. The maximum number of attempts to find the secret number in a group of `n` numbers should be no more than $\log_2(n)$. So for our range of 100 numbers, calculating $\log_2(100)$ is 6.6. Rounding off, the maximum number of guesses for 100 numbers is 7. We can verify this by counting the number of guesses the computer makes. To do that, we can use the accumulator pattern.

2.9.1 The Accumulator Pattern

An important problem-solving technique is the **accumulator pattern**. Your ability to recognize this commonly occurring pattern and then implement it will be especially useful as you encounter new problems that need to be solved.

As an example, consider the simple problem of computing the sum of the first five integer numbers. Of course, this is quite easy because we can just evaluate the expression 1 + 2 + 3 + 4 + 5. But what if we wanted to sum the first 10 integers? Or perhaps the first 100? The size of the expression would become quite long. To deal with this challenge, we can develop a more general solution that uses iteration.

Examine the Python code shown in **Session 2.13**, which calculates the sum of the numbers 1 to 5. As you can see, the variable `acc` starts with a value of 0, often called the **initialization**. Recall that the statement

```
In [1]: acc = 0

In [2]: for x in range(1, 6):
            acc = acc + x

In [3]: acc

Out[3]: 15
```

Session 2.13 Computing a running sum with a `for` loop and an accumulator variable

Description

```
for x in range(1, 6):
```

will cause the loop variable `x` to iterate over the values from 1 to 5. **Figure 2.3** shows how this statement can be used to create the running sum. Every time we pass through the body of the `for` loop, the assignment statement `acc = acc + x` is performed. Because the right-hand side of the statement is evaluated first, the current value of `acc` is used in the addition. To complete the assignment statement, the name `acc` is updated to refer to this new sum. The final value of `acc` is then 15.

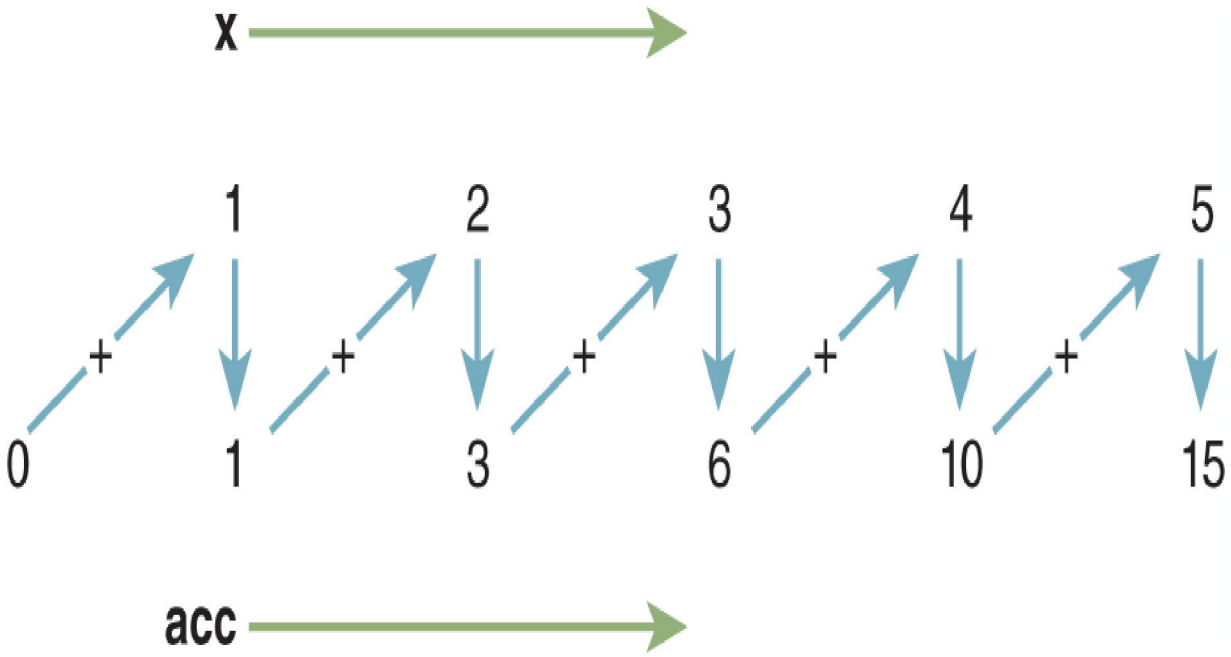


Figure 2.3 Using the accumulator pattern.

Description

It might seem strange to see the same name appearing on both the left-hand and right-hand sides of the assignment statement. However, if you remember the sequence of events for an assignment statement, you will not be confused:

1. Evaluate the right-hand side.
2. Let the variable on the left-hand side refer to the resulting object.

The variable `acc` is often referred to as the **accumulator variable** because it is continuously updated with the value of the running sum. Now, whether we want the sum of the first five integers or the first 5,000, the task is the same: Simply change the upper bound on the `range` and allow the accumulator pattern to do its work.

RECAP:
The accumulator pattern is useful for finding a sum of a group of numbers:

1. Set the accumulator variable to a starting number (typically 0).
2. Using a loop, add each number to the running sum in the accumulator variable.
3. When the loop finishes, the accumulator variable will hold the sum of all the numbers.

HEADS UP:
The entire accumulator process depends on correctly initializing the accumulator variable. If the accumulator does not start at an appropriate value (0 in many cases), the ending sum will be incorrect.

The accumulation pattern can also be used to count occurrences of something of interest, for example, how many numbers are in a set or how many words are in a sentence. In this pattern, our accumulator variable is actually a **counter**. We start the counter with an initial value, usually 0 or 1, and we increment (add 1) to the counter each time the event we want to count occurs. **Session 2.14** shows the accumulation pattern used for counting.

```
In[1]: countPassed = 0    #initialize both counting variables
       countTests = 0

       score = input("Enter a test score, -1 to stop: ")
       score = int(score)

       while score != -1:
           if score >= 60:
```

```
    if score >= 60:
        countPassed = countPassed + 1    #only count if passing grade

    countTests = countTests + 1          #count every score

    score = input("Enter another score, -1 to stop: ")   #loop update
    score = int(score)

print("You passed "+str(countPassed)+" out of "+str(countTests)+" tests.")
```

```
Enter a test score, -1 to stop: 75
Enter another score, -1 to stop: 60
Enter another score, -1 to stop: 59
Enter another score, -1 to stop: 82
Enter another score, -1 to stop: -1
You passed 3 out of 4 tests.
```

Session 2.14 Counting test scores with two counting variables

Description

In this session, we count the number of tests the user has passed, as well as the number of tests the user has taken. We use a `while` loop to allow the user

to enter any number of test scores. The user signals the end of the test scores by entering `-1`. We use two counting variables, one to count the passing test scores and one to count the total number of test scores. We add 1 to the `countPassed` variable only when the test score is at least 60, but we add 1 to `countTests` in every iteration of the loop.

RECAP:
The accumulator pattern is also useful for counting events:

4. Set the counter variable to a starting number (typically 0 or 1).
5. Using a loop, add 1 to the counter each time the event occurs.
6. When the loop finishes, the counter variable will hold the number of times the event occurred.

Try It Out

2.23 Compute the sum of the first 100 even numbers.

2.24 Compute the sum of the first 50 odd numbers.

2.25 Write a function to count how many numbers are divisible by 7 in a range sent as parameters.

2.26 Write a function to compute the average of the first 100 odd numbers.

We can use this accumulation pattern to count the attempts in the Guess a Number game. **Listing 2.5** shows the third pass of Guess a Number with the computer counting the number of attempts to guess the number. Our counter variable is `numAttempts`. We initialize `numAttempts` to 1 on line 9 when we make the first guess. Inside the loop on line 24, we add 1 to `numAttempts` each time the computer makes another guess. Notice that we do not change

`numAttempts` when the user input is invalid. When the computer guesses the number, `numAttempts` holds the count of attempts.

```
1  def guessANumber():
2      start = 1
3      end = 100
4
5      print("Pick a number between " + str(start) + " and " + str(end) + ".")
6      print("I'll try to guess it in as few guesses as possible.")
7
8      guess = (end + start) // 2
9      numAttempts = 1
10     response = input("Is " + str(guess) + " too high, too low, or correct? ")
11     response = response.lower()
12
13     while response != 'correct':
14         if response == 'too high':
15             end = guess - 1
16         elif response == 'too low':
17             start = guess + 1
18         else:                          #invalid response
```

```
19            response = input("Please enter 'too high', 'too low', or 'correct' ")
20            response = response.lower()
21            continue                #skip the remainder of the loop
22
23        guess = (end + start) // 2
24        numAttempts = numAttempts + 1
25        response = input("Is " + str(guess) + " too high, too low, or correct? ")
26        response = response.lower()
27
28    print("Your number is " + str(guess))
29    print("I guessed it in " + str(numAttempts) + " tries.")
```

Listing 2.5 Guess a Number, pass 3

Description

Session 2.15, where the secret number is 33, shows the worst case in number of guesses. As we had calculated earlier, the computer took the maximum number of attempts (7) to guess the secret number.

```
In [2]: from GuessANumberPass3 import *

In [3]: guessANumber() #secret number is 33

Pick a number between 1 and 100.
I'll try to guess it in as few guesses as possible.
Is 50 too high, too low, or correct? too high
Is 25 too high, too low, or correct? too low
Is 37 too high, too low, or correct? too high
Is 31 too high, too low, or correct? too low
Is 34 too high, too low, or correct? too high
Is 32 too high, too low, or correct? too low
Is 33 too high, too low, or correct? correct
Your number is 33
I guessed it in 7 tries.
```

Session 2.15 Counting the attempts to guess the secret number

Description

Our Guess a Number program does a lot of things. Notice that we did not try to write all the functionality in the first attempt. For pass 1, we wrote the main logic of the program. For pass 2, we added validation of the user's

input. Then in pass 3, we added a counter. By writing our game in steps, we could concentrate just on the functionality of the current pass. This approach, called **stepwise refinement**, is an important software engineering technique.

GOOD PROGRAMMING PRACTICE:

Use stepwise refinement to develop a program incrementally by adding small amounts of functionality to each pass.

Try It Out

2.27 What is the maximum number of guesses if the range of numbers is:

a. 1–50

b. 100–200

c. 1–300

2.28 Modify the `GuessANumber` function to take the start and end numbers as parameters to the function.

2.29 Modify the `GuessANumber` function to process other ways of saying too high or too low (for example, “higher” or “lower”).

2.10 The Game of Nim

Next, we will teach the computer to play Nim. The game of Nim begins with a pile of stones. Two players take turns nimming (stealing) 1, 2, or 3 stones from the pile. The player who steals the last stone wins the game. You can play this game with stones, coins, or any tokens you choose. We will play Nim with digital stones.

In this section, we will have the computer be both players so that the computer is essentially playing itself. But the computer will have two different strategies for making a play. Our first strategy will be to choose a random number of stones to remove. To do this, we need to explore how to generate a random number.

2.10.1 Random Numbers

To generate a random play, we will use the `random` module. This module generates what we call **pseudorandom numbers**—numbers that appear to be random, but in reality are not. You may find it useful to invoke the `help` command on this module to explore its functionality. For our purposes, we will use only the `randint` function—a random number generator that returns an integer between a start and end number. **Table 2.10** describes the `randint` method, and **Session 2.16** shows the `randint` method in action. Note that you need to import the `random` module to use the `randint` function. In statement 2, we generate a random number between 1 and 100. In statement 3, we generate 5 numbers between 1 and 10, and in statement 4, we generate 5 numbers between 20 and 50. Because these numbers are randomly generated, your results will likely differ from the numbers shown in Session 2.16 and will likely differ each time you execute this session's code.

TABLE 2.10 The `randint` Method

Function	Description
`randint(a, b)`	Returns a random integer ranging from `a` up to and including `b`.

In [1]:
```
import random
```

In [2]:
```
random.randint(1, 100)
```

Out[2]:
```
67
```

In [3]:
```
for i in range(5):
    print(random.randint(1, 10))
```

```
7
5
8
1
5
```

In [4]:
```
for i in range(5):
    print(random.randint(20, 50))
```

```
36
```

```
35
48
24
24
```

Session 2.16 Exercising the `randint` function

Description

Try It Out

2.30 Create a function to simulate flipping a coin, by generating either 0 or 1. Instead of printing the number, print "Heads" or "Tails" for each.

2.31 Create a function to simulate rolling two six-sided dice. Print the sum of the two dice.

2.10.2 Returning a Value from a Function

Our computer will act as both opponents. So, the code for taking a turn will be similar for each player. Rather than retyping all the statements for making a play, a better approach is to use abstraction.

Recall that abstraction allows us to think about a collection of steps as a logical group. In Python, we can define a function that not only serves as a name for a sequence of actions but also returns a value when it is called. We have already seen this type of behavior with the `randint` function. For example, when we call `randint(1, 10)`, it *returns* a number between 1 and 10.

The basic function definition template is shown below with one additional statement: `return`.

```
def functionName(param1, param2,...):
    statement1
    statement2
    ...
    return expression    #exit and return expression's value
```

Description

The `return` statement causes two related actions. First, the expression contained in the statement is evaluated, producing a result object. Second, the function terminates immediately, and the reference to the result object is returned to the calling statement. The bottom line is that the value of a function call becomes the value of the expression that is returned. In Session 2.16, the `randint` function returns a value, which we print.

It is important to realize that no matter where the `return` statement occurs, it will be the final statement performed because `return` causes the function to stop. For this reason, `return` is typically the last statement in the function.

RECAP:
A `return` statement causes a function to stop. The value of the expression in the `return` statement becomes the value of the function in the calling statement.

So, for our game of Nim, the program logic will be:

```
while there are more stones
    get Player 1's play, make the play,
            if pile is 0, winner is Player 1
    get Player 2's play, make the play,
            if pile is 0, winner is Player 2
```

Description

As you can see, after each turn, we check for an empty pile. If the pile is empty, the player who just took a turn must be the winner.

We separate the player moves into functions so that each player can move according to their predetermined strategy. The first player, Random Randy, randomly chooses to remove 1, 2, or 3 stones. Obviously, this does not seem like a winning strategy.

Let's talk a moment about what a winning strategy might be. The object of the game is to take the last stone from the pile. Because a player can remove 1, 2, or 3 stones at each turn, the winning strategy, then, is to ensure that after making a play, the pile still has more than 3 stones.

The second player, Nimble Nancy, has cleverly adopted the winning strategy. Because a player can remove 1, 2, or 3 stones, Nimble Nancy finds the modulus of 4 (the remainder after dividing by 4) and takes that number of stones. This always leaves either 0 stones or a pile that is a multiple of 4, which is more stones than the other player can take in one turn. **Listing 2.6** shows the Nim game.

```
1 def nim():
2     pile = 15
3     print("The pile has " + str(pile) + " stones.")
4
5     winner = ""
6     while pile != 0:
7         c1Move = randomMove(pile)
8         pile = pile - c1Move
9         print("Random Randy takes " + str(c1Move) + " stone(s). Pile is now " + str(pile))
```

```
        if pile == 0:
            winner = "Random Randy"
        else:
            c2Move = nimbleMove(pile)
            pile = pile - c2Move
            print("Nimble Nancy takes " + str(c2Move) + " stone(s). Pile is now " + str(pile))
            if pile == 0:
                winner = "Nimble Nancy"
    print("The winner is " + winner + "!")

def randomMove(pile):
    import random
    move = random.randint(1, 3)
    if move > pile:
        move = pile

    return move

def nimbleMove(pile):
    if pile % 4 != 0:
```

```
30          move = pile % 4
31      else:
32          move = 1
33
34      return move
```

Listing 2.6 The Nim game

Description

Session 2.17 shows the results of 3 games. As expected, no matter how many times we run `nim`, Nimble Nancy always wins.

```
In [2]: nim()
```

```
The pile has 15 stones.
Random Randy takes 3 stones(s). Pile is now 12
Nimble Nancy takes 1 stones(s). Pile is now 11
Random Randy takes 3 stones(s). Pile is now 8
Nimble Nancy takes 1 stones(s). Pile is now 7
Random Randy takes 1 stones(s). Pile is now 6
Nimble Nancy takes 2 stones(s). Pile is now 4
Random Randy takes 2 stones(s). Pile is now 2
```

```
Nimble Nancy takes 2 stones(s). Pile is now 0
The winner is Nimble Nancy!
```

In [3]:
```
nim()
```

```
The pile has 15 stones.
Random Randy takes 3 stones(s). Pile is now 12
Nimble Nancy takes 1 stones(s). Pile is now 11
Random Randy takes 1 stones(s). Pile is now 10
Nimble Nancy takes 2 stones(s). Pile is now 8
Random Randy takes 1 stones(s). Pile is now 7
Nimble Nancy takes 3 stones(s). Pile is now 4
Random Randy takes 2 stones(s). Pile is now 2
Nimble Nancy takes 2 stones(s). Pile is now 0
The winner is Nimble Nancy!
```

In [4]:
```
nim()
```

```
The pile has 15 stones.
Random Randy takes 1 stones(s). Pile is now 14
Nimble Nancy takes 2 stones(s). Pile is now 12
Random Randy takes 1 stones(s). Pile is now 11
Nimble Nancy takes 3 stones(s). Pile is now 8
Random Randy takes 1 stones(s). Pile is now 7
Nimble Nancy takes 3 stones(s). Pile is now 4
Random Randy takes 1 stones(s). Pile is now 3
Nimble Nancy takes 3 stones(s). Pile is now 0
The winner is Nimble Nancy!
```

Session 2.17 Playing a few Nim games

Description

Try It Out

2.32 Modify the `nim` function so that you can pass the beginning number of stones as a parameter. Check that the number of stones passed is greater than the number of stones you can pick.

2.33 Consider how changing the number of stones you can pick changes the strategy. Modify the Nim game to allow up to (and including) 5 stones to be taken with each play. What needs to change in the `randomMove` function to accommodate this rule change, and what needs to change in the `nimbleMove` function?

2.34 Devise another strategy and play the game using that strategy for one of the players.

2.11 Summary

This chapter introduced the following building blocks of programming and Python:

- An introduction to the string data type
- Boolean expressions using equality, relational, and logical operators
- Selection statements using `if`, `if-else`, and `if-elif-else`
- The Python `print` function
- The Python `input` function
- The `while` loop
- The accumulator pattern
- Generating random numbers
- Functions that return a value

Key Terms

accumulator pattern
accumulator variable
binary search
Boolean expression
compound Boolean expressions
condition
counter
equality operators
infinite loop
initialization
logical operators
nested selection
priming statement
prompt
pseudorandom numbers
relational expressions
relational operators
selection
selection statement
sentinel value
short-circuit evaluation
stepwise refinement
string
string concatenation operator

tail nesting

Unicode encoding system

user-friendly

`while` loop

Programming Exercises

2.1 Write a function that counts the number of days in a week that the Dow Jones Industrial Average (DJI) was above a threshold number. The function should take the threshold as a parameter and input five numbers representing the DJI for 5 days. Return the number of days that the DJI was equal to or above the threshold number.

2.2 Write a function `isEven` that takes a number as a parameter and returns `True` if the number is even, and `False` otherwise.

2.3 Write a function that asks the user which game they want to play: Nim or Guess a Number. The function should call the appropriate function to play the selected game. After playing a game, the function should repeat asking the user to play until the user enters "Stop."

2.4 Rewrite the `GuessANumber` function so that a human is guessing the secret number determined by the computer. The computer will need to reply with too high, too low, or correct based on the human's guess.

2.5 Write a function that counts the number of prime numbers that are less than a number sent as a parameter. Check that the parameter is non-negative. The function should return the count.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 3

Codes and Other Secrets

String Operators and Methods, the `len` Built-in Function, and Keyword Parameters

3.1 OBJECTIVES

- To further explore the string data type
- To introduce the `len` built-in function
- To introduce simple cryptographic algorithms

3.2 The String Data Type

For almost as long as people have been writing, people have also been trying to hide what they were writing from others. Herodotus describes the use of "secret writing" that saved Greece from being conquered by Xerxes and the Persians. The Kama Sutra, ancient Hindu writings, instructs women in the art of secret writing so that they can hide the details of their liaisons. In this chapter, we explore some simple forms of secret writing using Python.

Today this art of secret writing is called **cryptography**. You use cryptography nearly every day without even thinking about it. You may not be sending secret messages to your friends (although several cryptographic add-ons are available for email these days), but rather using encryption whenever you make an online purchase or check your grades or your bank balance online.

Before we dive into cryptography, we must learn more about the important Python string data type—a data structure that allows us to represent the written word in our programs.

Strings are simply sequences of **characters**, such as the letters of the alphabet and other symbols commonly used in writing. Most often, these sequences of characters are put together to form familiar words, but as we will see in this chapter, character sequences can be used for many other interesting purposes.

In Python, you will know that an object is a string because it is surrounded by either single quotes (`'`) or double quotes (`"`). Strings can also be surrounded by triple single quotes (`'''`) or triple double quotes (`"""`), although this is rarely used.

Strings can contain any characters: letters, spaces, numbers, punctuation marks, and even quotation marks. Like numbers, strings are objects that can be named as variables.

RECAP:

If a string contains a single quote, use double quotes around the string. If a string contains double quotes, use single quotes around the string. If a string does not contain either single or double quotes, it does not matter whether you use single or double quotes around the string, as long as the type of quotes is not mixed. In other words, if you start a string with a single quote, then you must end the string with a single quote. Likewise, if you start a string with double quotes, then you must end that string with double quotes. Alternatively, you could surround a string with either triple single quotes or triple double quotes.

As with numeric data types, strings in Python have operators that we can use to transform them. We have already explored the concatenation operator (+) for combining strings, the `in` operator, and several string methods (`startswith`, `upper`, and `lower`). In the next sections, we will examine some more commonly used string operators and methods.

3.2.1 Repetition

Another operator we can apply to a string is the repetition operator (`*`). As you might guess, this operator takes a string and repeats it as many times as you specify. For example, suppose you want to create the string `'gogogo'`. Rather than typing "go" multiple times, you can simply apply the repetition operator as follows: `'go'*3`.

As with their numeric counterparts, the repetition operator has a higher precedence than the concatenation operator. Thus, in an expression that uses both the repetition and concatenation operators, the repetition will be performed first. We can construct strings using both operators together, as shown in **Session 3.1**.

```
In [1]: "Hip "*2 + "Hooray!"
```

```
Out[1]: 'Hip Hip Hooray!'
```

```
In [2]: def rowYourBoat():
            print("Row, "*3 + 'your boat')
            print("Gently down the stream")
            print("Merrily, "*4)
            print("Life is but a dream")
```

```
In [3]: rowYourBoat()
```

```
Row, Row, Row, your boat
Gently down the stream
Merrily, Merrily, Merrily, Merrily,
Life is but a dream
```

Session 3.1 String repetition

Description

3.2.2 Indexing

We next examine the **index operator** `[]`, which does not have a mathematical counterpart. This operator is useful when we want to access characters in specific positions in a string. The first character in a string is at index *0*, the second character is at index *1*, and so on. For example, if we want to get the first character of a string, we can use the index operator as follows: `'John'[0]`. The result of this expression is `'J'`.

In addition, Python allows us to access the string in reverse. The last character in a string has index value -1, the second-to-last character has index value -2, and so on. **Figure 3.1** shows the relationship between index values and the characters in a string.

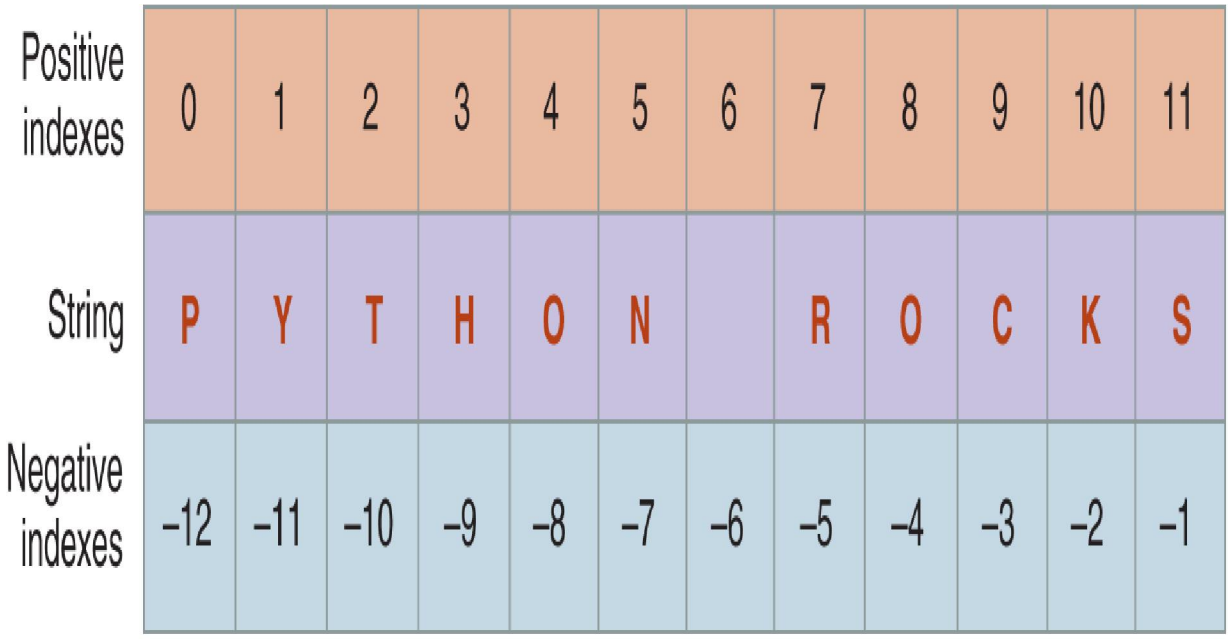


Figure 3.1 Index values for each character in a string.

Description

When indexing a string, it is often useful to know the overall length of that string. Python's `len` built-in function tells us how many characters are in a string. The `len` function is described in **Table 3.1**. It is important to understand the relationship between the length of a string and the index of the last character in the string. For example, although `len('abc')` returns 3, we would use `'abc'[2]` to access the last character, `'c'`.

TABLE 3.1 The `len` Function

Function	Description
`len(string)`	Returns the number of characters in `string`, including spaces and special characters.

We can go one step further and combine the `range` function with the `len` function to allow us to access each character in a string. **Session 3.2** illustrates some uses of the index operator and the `len` function.

```
In [1]: name = "Roy G Biv"

In [2]: firstChar = name[0]

In [3]: firstChar
Out[3]: 'R'

In [4]: middleIndex = len(name) // 2

In [5]: middleIndex
```

```
Out[5]: 4
```

```
In [6]: middleChar = name[middleIndex]
```

```
In [7]: middleChar
```

```
Out[7]: 'G'
```

```
In [8]: name[-1]
```

```
Out[8]: 'v'
```

```
In [9]: name[len(name) - 1]
```

```
Out[9]: 'v'
```

```
In [10]: for i in range(len(name)):
             print(name[i])
```

```
R
o
y

G


B
i
v
```

Session 3.2 String indexing

Description

RECAP:
The indexes of characters in a string range from 0 to `len(string) - 1` in the forward direction and from –1 to `-len(string)` in the reverse direction.

3.2.3 String Slicing

The **slice operator** `[:]` is similar to the index operator, except that it can extract multiple characters from the string. We often call these parts of a string **substrings**. We use the slice operator with two index values: a starting index before the `:` and an end index after the colon.

For example, we can get the first three characters of a string as follows:

```
In [1]: "mary"[0:3]

Out[1]: 'mar'
```

Description

The first number after the left square bracket gives the index of the first character in the string we want to access—in this case, index 0, the first character (`m`). The number after the `':'` is the index that is one more than the last character we want selected from the string. This is similar to the bounds of the `range` function where the upper bound is not included. So in this case, we want to start selecting characters at index 0 and end at index 2. As you can see, the result is the substring, `"mar"`.

If you omit the first index, Python assumes that you want to start at the beginning of the string, index 0. If you omit the end index, Python assumes that you want to continue selecting characters until the end of the string. A useful side effect of these shortcuts is that we can make a copy of a string using the following slice operator: `duplicateStr = oldStr[:]`.

The **proper prefixes of a string** are all the substrings that start with the first character of the string and are shorter than the original string. We can easily generate the prefixes of a string using the slice operator, as shown in **Session 3.3**. Notice that the first value in the range is 1 so that the first slice is `name[0:1]`. If we wanted to print the final `'v'`, we would need to use `len(name) + 1` for the `range` end so that the final slice would become `name[0:len(name)]`.

```
In [1]: name = "Roy G Biv"

In [2]: name[0:3]
```

Out[2]: 'Roy'

In [3]:
```
name[:3]    #start index omitted, start at 0
```

Out[3]: 'Roy'

In [4]:
```
name[6:9]
```

Out[4]: 'Biv'

In [5]:
```
name[6:]   #end index omitted, continue to string end
```

Out[5]: 'Biv'

In [6]:
```
for i in range(1, len(name)):
    print(name[0:i])
```

```
R
Ro
```

```
Ro
Roy
Roy
Roy G
Roy G
Roy G B
Roy G Bi
```

Session 3.3 String slicing

Description

3.2.4 String Searching

We have already learned about the `in` operator. Its companion operator, `not in`, can also be used to determine whether one string is contained in another. Both `in` and `not in` operate on two strings:

```
"sub" in "string"

"sub" not in "string"
```

Description

The exact substring must be found in the string and must match the initial string in case. These operators evaluate to `True` or `False`, so they can be used in `if-else` statements. **Session 3.4** demonstrates the `in` and `not in` operators.

In [1]:
```
name = "Roy G Biv"
```

In [2]:
```
"Biv" in name
```

Out[2]: True

In [3]:
```
"biv" in name    #b does not match B
```

Out[3]: False

In [4]:
```
"v" not in name
```

Out[4]: False

In [5]:
```
"3" in name
```

Out[5]: False

```
In [6]: "Bv" in name   #B and v are both in name, but not together

Out[6]: False
```

```
In [7]: if "y" in name:
            print("The letter y is in name.")
        else:
            print("The letter y is not in name.")

        The letter y is in name.
```

Session 3.4 String searching

Description

Table 3.2 provides a short summary of the string operators we have introduced so far.

TABLE 3.2 Summary of the String Operators

Operator/ Function	Description	Example	Result
`+`	**Concatenation:** joins two strings together	`"hello " + "world"`	`"hello world"`
`*`	**Repetition:** repeats the string	`"abc"*3`	`"abcabcabc"`
`[i]`	**Index:** returns the character at position `i`; indexes start at 0	`"abc"[1]`	`'b'`
`[i:j]`	**Slice:** returns the substring from `i` to `j - 1`	`"abcdef"[0:2]`	`"ab"`
`in`	`in` operator: determines whether one string contains another	`'bc' in 'abc'`	`True`
`not in`	`not in` operator: determines whether one string does not contain another	`'d' not in 'abc'`	`True`

Description

Try It Out

3.1 Create a string variable that is initialized to your entire name—first, middle, and last.

3.2 Using the slice operator, print your first name.

3.3 Using the slice operator, print your last name.

3.4 Using the slice and concatenation operators, print your name in the form `Lastname, Firstname`.

3.5 Print the length of your first name.

3.6 Assume you have two variables: `s='s'` and `p='p'`. Using concatenation and repetition, write an expression that produces the string `mississippi`.

3.7 Modify the prefix example in Session 3.3 to print all prefixes of `name` using only the end index.

3.2.5 String Methods

In addition to the string operators, strings have methods, because they are objects just like turtles. Some of these methods are summarized in **Table 3.3**. We have discussed `startswith`, `upper`, and `lower`. Several other methods help to format your strings to fit in a certain amount of space, such as `ljust`, `rjust`, and `center`. Several others replace characters.

TABLE 3.3 Summary of String Methods

Method	Use	Explanation
center	`aString.center(w)`	Returns the string `aString` evenly surrounded by spaces to make `aString` `w` characters long.
count	`aString.count(item)`	Returns the number of occurrences of item in `aString`.
ljust	`aString.ljust(w)`	Returns `aString` left-justified in a field of width `w`.
rjust	`aString.rjust(w)`	Returns `aString` right-justified in a field of width `w`.
upper	`aString.upper()`	Returns `aString` in all uppercase.
lower	`aString.lower()`	Returns `aString` in all lowercase.
index	`aString.index(item)`	Returns the index of the first occurrence of `item` in `aString`, or an error if not found.
rindex	`aString.rindex(item)`	Returns the index of the last occurrence of `item` in `aString`, or an error if not found.
find	`aString.find(item)`	Returns the index of the first occurrence of `item` in `aString`, or −1 if not found.
rfind	`aString.rfind(item)`	Returns the index of the last occurrence of `item` in `aString`, or −1 if not found.
contains	`aString.contains(item)`	Returns `True` if `item` is found in `aString`; `False` otherwise.
replace	`aString.replace(old, new)`	Replaces all occurrences of `old` substring with `new` substring in `aString`.
startswith	`aString.startswith(item)`	Returns `True` if the first characters in `aString` are `item`; `False` otherwise.

The most useful methods for us in this chapter are those that allow us to search for one string within another. We know that `startswith` determines whether a string starts with a substring. The `count` method finds the number of occurrences of one string within another. The `index` and `find` methods are similar in that they return the index of the first occurrence of one string within another. However, they differ in their behavior when the string we are looking for is not found. The `find` method returns −1 if the string is not found. In contrast, the `index` method generates an error and halts the program if the string is not found. **Session 3.5** illustrates the use of these string methods.

```
In [1]: 1 "hello".ljust(10)

Out[1]: 'hello     '
```

```
In [2]: 1 "hello".rjust(10)

Out[2]: '     hello'
```

```
In [3]: 1 teamSport = "Golden Gopher football"
```

```
In [4]: 1 teamSport.count('o')

Out[4]: 4
```

```
In [5]: 1 teamSport.count('oo')
```

```
Out[5]: 1
```

```
In [6]: 1 teamSport.lower()
```

```
Out[6]: 'golden gopher football'
```

```
In [7]: 1 teamSport                    #original string is not affected
```

```
Out[7]: 'Golden Gopher football'
```

```
In [8]: 1 teamSport = teamSport.upper() #assign result to original string
```

```
In [9]: 1 teamSport
```

```
Out[9]: 'GOLDEN GOPHER FOOTBALL'
```

```
In [10]: 1 teamSport.find('G')          #finds first G at index 0
```

```
Out[10]: 0
```

```
In [11]: 1 teamSport.rfind('G')         #finds last G at index 7
```

```
Out[11]: 7
```

```
In [12]: 1 teamSport.index('G')
```

```
Out[12]: 0
```

```
In [13]: 1 teamSport.rindex('G')
```

```
Out[13]: 7
```

```
In [15]: 1 teamSport.find('g')          #not found
```

```
Out[15]: -1
```

```
Out[15]: -1
```

```
In [16]: 1 teamSport.index('g')        #not found, index method throws an error
```

```
ValueError                                Traceback (most recent call last)
Input In [16], in <cell line: 1>()
----> 1 teamSport.index('g')

ValueError: substring not found
```

```
In [17]: 1 'ab cd ef'.replace('cd', 'xy') #replace cd with xy
```

```
Out[17]: 'ab xy ef'
```

Session 3.5 Demonstrating string methods

Description

Try It Out

3.8 Using the `count` method, find the number of occurrences of the character `'s'` in the string `'mississippi'`.

3.9 In the string `'mississippi'`, replace all occurrences of the substring `'iss'` with `'ox'`.

3.10 Find the index of the first occurrence of `'p'` in `'mississippi'`.

3.11 Make the word `'python'` centered and all capital letters in a string of length 20.

3.2.6 Character Functions

As you will see shortly, our string-based algorithms are able to convert a character into a number, and back again. The Python functions `ord` and `chr` do this job for us. In addition, as we have seen, the `str` function converts a number into its string form. **Table 3.4** summarizes these functions, and **Session 3.6** presents an example of their use.

TABLE 3.4 Character Functions

Function	Description
`ord(ch)`	Returns the Unicode value of a single-character string `ch`.
`chr(num)`	Returns the character represented by the Unicode value `num`.
`str(num)`	Converts a number `num` to a string.

```
In [1]: ord('a')

Out[1]: 97
```

```
In [2]: ord('c')
Out[2]: 99

In [3]: ord('A')
Out[3]: 65

In [4]: chr(104)
Out[4]: 'h'

In [5]: chr(97 + 13)
Out[5]: 'n'

In [6]: str(10960)
Out[6]: '10960'
```

Description

It might seem a bit odd to you that the letter `'a'` is represented by the number 97. Why not 1 or 0? Numbers and strings are all stored in the memory of the computer as a series of 1s and 0s. Because characters have no intrinsic numeric value, Python and other programming languages use an encoding system to map characters to numbers. The sequence of 1s and 0s that corresponds to the letter `'a'` when it is converted to a decimal number is 97. The mapping from letters to numbers was defined as the **American Standard Code for Information Interchange (ASCII)** in the 1960s. A limitation of ASCII is that it represents only 128 English characters. By comparison, the newest encoding system, Unicode, which was developed by the Unicode Consortium, can map characters in multiple languages. To maintain compatibility with previously written programs, the Unicode Consortium defined the first 128 characters in the Unicode system to be the same as the ASCII characters. You can see this mapping by visiting `unicode.org` and viewing the Basic Latin (ASCII) code chart.

For our purposes, we would much prefer the letter `'a'` to map to the number 0 and the letter `'z'` to map to the number 25. This would greatly simplify our code. Using the `find` method and string indexing, we can define our own helper functions for mapping from characters to numbers, and back again, in **Listing 3.1**. In both of these functions, we want to define a string containing the lowercase letters of the alphabet in order. Instead of typing a string containing all the letters of the alphabet, however, we can use a handy constant named `ascii_lowercase` in the `string` module, which is a string containing the lowercase letters of the alphabet in order from *a* to *z*. We also want our alphabet to have a space character, to which we will give the value 26. So, we concatenate a space character to the `ascii_lowercase` constant to make our `alphabet` string (see lines 3 and 11). To use this constant, we need to import the `string` module, as shown in lines 2 and 10.

```
1  def letterToIndex(letter):
2      import string
3      alphabet = string.ascii_lowercase + ' '
4      idx = alphabet.find(letter)
5      if idx == -1:   #not found
6          print("error: " + letter + " is not in the alphabet")
7      return idx
8
9  def indexToLetter(idx):
10     import string
11     alphabet = string.ascii_lowercase + ' '
12     letter = ''
13     if idx >= len(alphabet):
14         print("error: " + str(idx) + " is too large")
15     elif idx < 0:
16         print("error: " + str(idx) + " is less than 0")
17     else:
18         letter = alphabet[idx]
19     return letter
```

Listing 3.1 Helper functions (`mapper.py`) map characters to numbers

Description

In the `letterToIndex` function, we check whether the parameter character is in `alphabet` by passing it to the `find` method. Recall that if the `find` method does not find its parameter, the method returns –1. So if the return value is –1, we print an error message; otherwise, we return the index of the parameter in the `alphabet` string.

In `indexToLetter`, we use the `if-elif-else` form of the `if` statement. This form allows us to check for a sequence of conditions on a variable. For example, to verify that the index parameter corresponds to a letter in our `alphabet` string, we check whether the index parameter (`idx`) is greater than or equal to the length of the `alphabet` string, or less than 0. In either case, we print an error message. Otherwise, we return the letter from the `alphabet` at the index provided.

In **Session 3.7**, we test our two helper functions. We have saved these two functions in a file named `mapper.py`, which we import at the beginning of the session. To have confidence that the code works correctly, we test **boundary values**—that is, the edge values that should cause different results. For example, in the `letterToIndex` function, we have two possible outcomes: The letter is found, or the letter is not found. The boundary values that we can use to test this function are the letter `a` (the lowest possible index) and a space (the highest possible index). Then we test the value 5, which we know is not in the alphabet, and we correctly receive an error that 5 is not found. In the `indexToLetter` function, we have three possible outcomes: The index is less than 0, the index is greater than the last letter in our `alphabet`, or the index is within the bounds of the `alphabet`. We can test this function with the values 0 (first letter), 26 (last character in the `alphabet`), –1 (less than 0), and 27 (greater than the highest index).

```
In [1]:  1 from mapper import *
```

```
In [2]: 1 letterToIndex('a')
```

```
Out[2]: 0
```

```
In [3]: 1 letterToIndex(' ')   #space character
```

```
Out[3]: 26
```

```
In [4]: 1 letterToIndex('5')
```

```
error: 5 is not in the alphabet
```

```
Out[4]: -1
```

```
In [5]: 1 indexToLetter(0)
```

```
Out[5]: 'a'
```

```
In [6]: 1 indexToLetter(26)
```

```
Out[6]: ' '

In [7]:  1 indexToLetter(-1)

        error: -1 is less than 0

Out[7]: ''

In [8]:  1 indexToLetter(27)

        error: 27 is too large

Out[8]: ''
```

Session 3.7 Testing our helper functions

Description

GOOD PROGRAMMING PRACTICE:
To verify that your code is correct, test boundary values.

Try It Out

3.12 What is the difference between `ord('A')` and `ord('a')`?

3.13 Write a function that takes a single character digit and returns its integer value.

3.14 Write the `letterToIndex` function using `ord`.

3.15 Write the `indexToLetter` function using `ord` and `chr`.

3.16 Write a function that takes an exam score from 0 to 100 and returns the corresponding letter grade. Use the same grading scale your professor does for this class.

3.3 Encrypting and Decrypting Messages

Cryptography is the science of making messages secure. It works by transforming a readable message into an unreadable message, which is then transmitted securely to a receiver. At that point, the receiver transforms the unreadable message back into the original readable message. We refer to messages that are readable as **plaintext**, whereas messages that are unreadable are called **ciphertext**. The process of turning plaintext into ciphertext is called **encryption**; the reverse process of turning ciphertext into plaintext is called **decryption**. **Figure 3.2** provides an overview of the encryption and decryption processes. Algorithms for both encryption and decryption are the subject of this section.

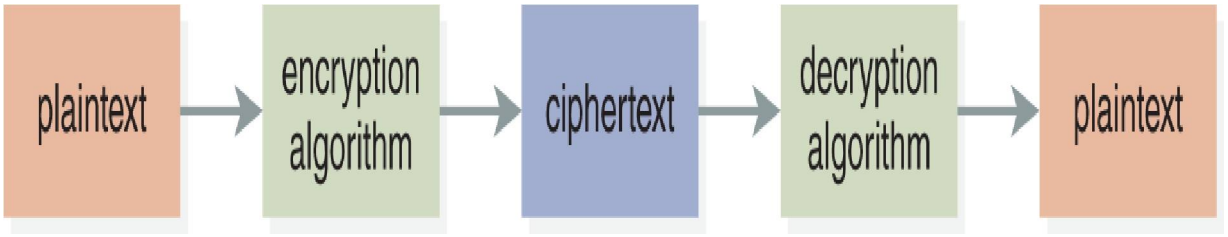


Figure 3.2 Encrypting and decrypting a message.

3.4 Transposition Cipher

3.4.1 Encrypting Using Transposition

One of the easiest ways to encrypt a message is simply to scramble the letters. For example, the word "house" could be randomly transformed to "suheo." In fact, there are 120 different possible arrangements of the word "house." However, if the encryption algorithm randomly scrambles the letters, the task of the decryption algorithm becomes difficult. Encryption and decryption algorithms must work together in some agreed-upon way, with the encryption algorithm using a scheme to scramble the letters and the decryption algorithm knowing the scheme.

One way to scramble the letters of a message is to separate the message into two groups of characters, where the first group is composed of the even-numbered characters and the second group is composed of the odd-numbered characters. If we create one string out of the even-numbered characters and another out of the odd-numbered characters, we can concatenate the two new strings to form the ciphertext string. Because this results in a string with the characters shuffled to new positions, we call this a **transposition cipher**, also sometimes called the **rail fence cipher**. **Figure 3.3** illustrates the idea behind this encryption algorithm.

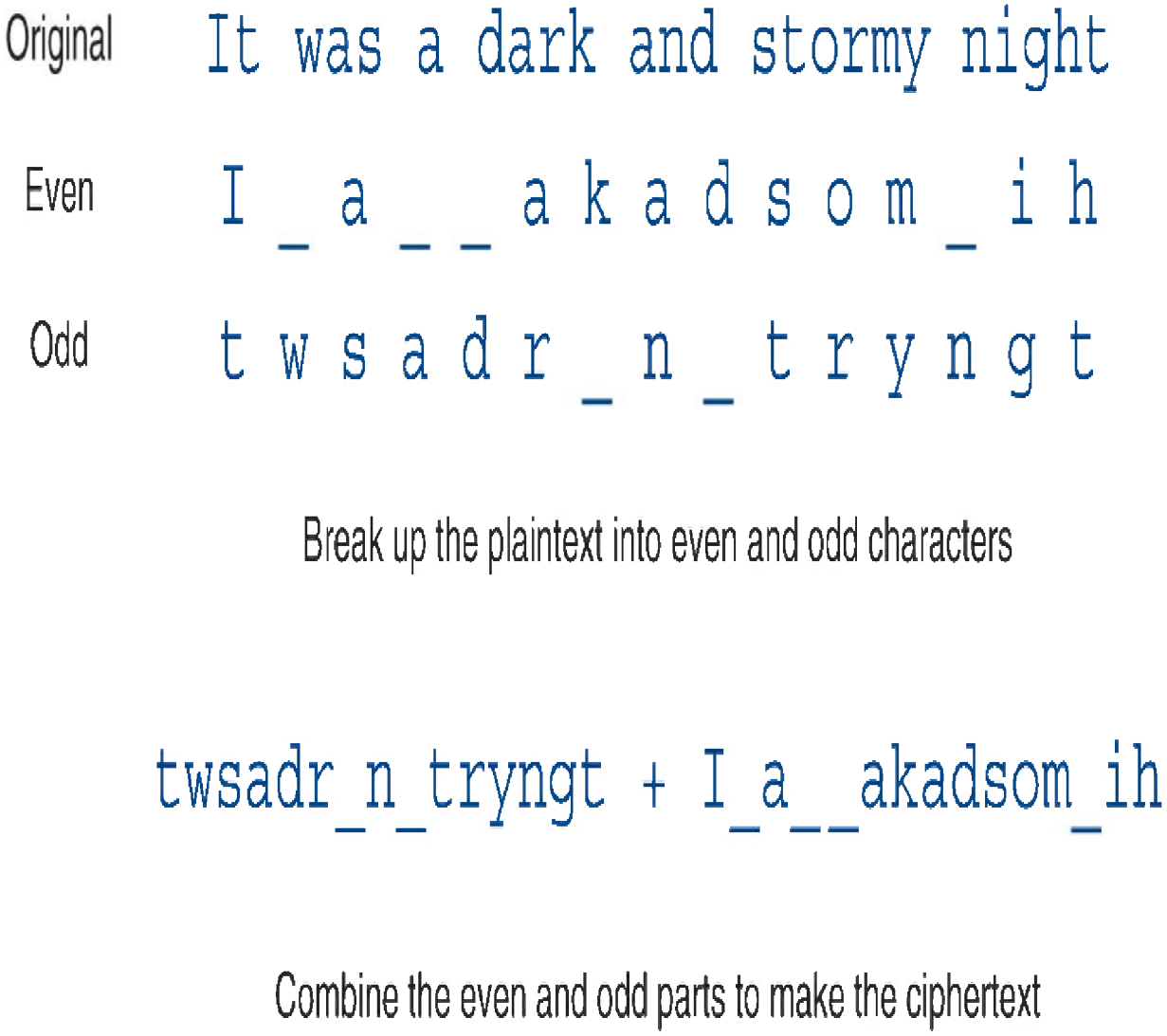


Figure 3.3 Encrypting a string using an even–odd shuffle.

Description

Now that we have a scheme for scrambling the plaintext message, we will write a Python function that takes the plaintext message as a parameter and returns the ciphertext message. The key to this algorithm lies in our ability to put the even characters from the plaintext into one string and the odd characters into another. One way to do this is to use a counter and a **string iterator**. A string iterator is a `for` loop in which the loop variable takes on the value of each character in the string.

The string iterator uses this form of the `for` loop:

```
for ch in stringVariable:
```

With each iteration of the `for` loop, the loop variable `ch` takes on the value of one character in the string `stringVariable`. For the first iteration, `ch` becomes the character at index 0 of `stringVariable`; for the second iteration, `ch` becomes the character at index 1 of `stringVariable`; and so on.

We start with two empty strings, one that will hold the even-numbered characters and one that will hold the old-numbered characters. We use a counter to keep track of whether the current character is even or odd numbered. As we iterate over each character of the string, we increment our counter. If the counter is an even number, we concatenate the current character to our even-character string. If the counter is odd, we concatenate the current character to our odd-character string. In this so-called rail fence cipher, each rail contains part of the original string. In this case, because we are dividing the string into two parts, we call it a two-rail cipher.

How can we tell if a number is even or odd? Recall that for any even number *N*, when you divide *N* by 2, there is no remainder. For any odd number *N*, *N* divided by 2 has a remainder of 1. In Python, we can obtain the remainder by using the modulo operator. For any even number *N*, `N % 2` is 0. For any odd number *N*, `N % 2` is 1. We can use the expression `charCount % 2 == 0` as the condition in an `if` statement to test whether the character counter is even or odd. **Listing 3.2** shows the encryption algorithm using the strategy described here.

```
1  def scramble2Encrypt(plaintext):
2      evenChars = ""
3      oddChars = ""
4      charCount = 0
5      for ch in plaintext:
6          if charCount % 2 == 0:
7              evenChars = evenChars + ch
8          else:
9              oddChars = oddChars + ch
10         charCount = charCount + 1
11
12     cipherText = oddChars + evenChars
13     return cipherText
```

Listing 3.2 Scrambling a plaintext message

Description

The `scramble2Encrypt` function makes use of the accumulator pattern in several places. First, `evenChars` and `oddChars` start as empty strings and accumulate characters as the `for` loop executes. The accumulator pattern works just as well for concatenating strings as it does for adding numbers. When we apply the accumulator pattern to strings, we build up a string that starts from nothing (the empty string, `""`), growing the string one character at a time. The next use of the accumulator pattern is a familiar counting accumulator using the `charCount` variable.

HEADS UP:
The empty string has no characters in it and can be defined as either `""` or `''` with no spaces between the quotes. The length of an empty string is 0.

On line 5 of Listing 3.2 we begin a loop that will iterate over each character in the `plainText` string. When we iterate over a string, the loop variable references each character in the string one after another. On line 6, we make use of the test we devised to decide whether `charCount` is even or odd. If `charCount` is even, we concatenate the current character referenced by the loop variable `ch` to the existing value of `evenChars`. Because concatenation creates a new string, `evenChars` becomes this newly constructed string when we perform the assignment. If `charCount` is odd, we concatenate `ch` to `oddChars` to create a new string referenced by `oddChars`.

Finally, after all characters in the plaintext string have been processed, on line 12 we create the `cipherText` string by concatenating `oddChars` and `evenChars`. The order of variables in the concatenation is not accidental. We put `oddChars` first for a reason: If our string has an odd number of characters, `oddChars` will be one character shorter than the number of even characters. Can you explain why? We will see in a moment that this point is important when we want to split the ciphertext back into two pieces.

Session 3.8 shows the output of the `scramble2Encrypt` function for five test inputs. Notice that some of the test cases are just nonsense strings, but they are chosen carefully to make it easy to see if the function is working the way we think it should. Also notice that we test some boundary cases such as a string of length 1 and even an empty string.

```
In [1]:  1 from transposition import *
```

```
In [2]:  1 scramble2Encrypt('abababab')
```

```
Out[2]: 'bbbbaaaa'
```

```
In [3]: 1 scramble2Encrypt('abababababc')
```

```
Out[3]: 'bbbbaaaaac'
```

```
In [4]: 1 scramble2Encrypt('I do not like green eggs and ham')
```

```
Out[4]: ' ontlk re gsadhmId o iegeneg n a'
```

```
In [5]: 1 scramble2Encrypt('a')
```

```
Out[5]: 'a'
```

```
In [6]: 1 scramble2Encrypt('')
```

```
Out[6]: ''
```

Session 3.8 Testing `scramble2Encrypt`

Description

3.4.2 Decrypting a Transposed Message

Our next task is to write a function to decrypt a message that was encrypted by our `scramble2Encrypt` function. The input to our decryption function will be the ciphertext produced by our encryption function. The decrypt function will return the restored plaintext copy of the string.

To restore the plaintext string, we start by splitting the ciphertext in half. The first half of the string contains the odd characters from our original message, and the second half of the string contains the even characters. To restore the plaintext version of the string, we start with an empty plaintext string and concatenate the characters from the even and odd strings onto the end of the plaintext string. We alternate taking a character from the even string first, then the odd string, and so on. **Figure 3.4** gives an example of putting the plaintext back together again.

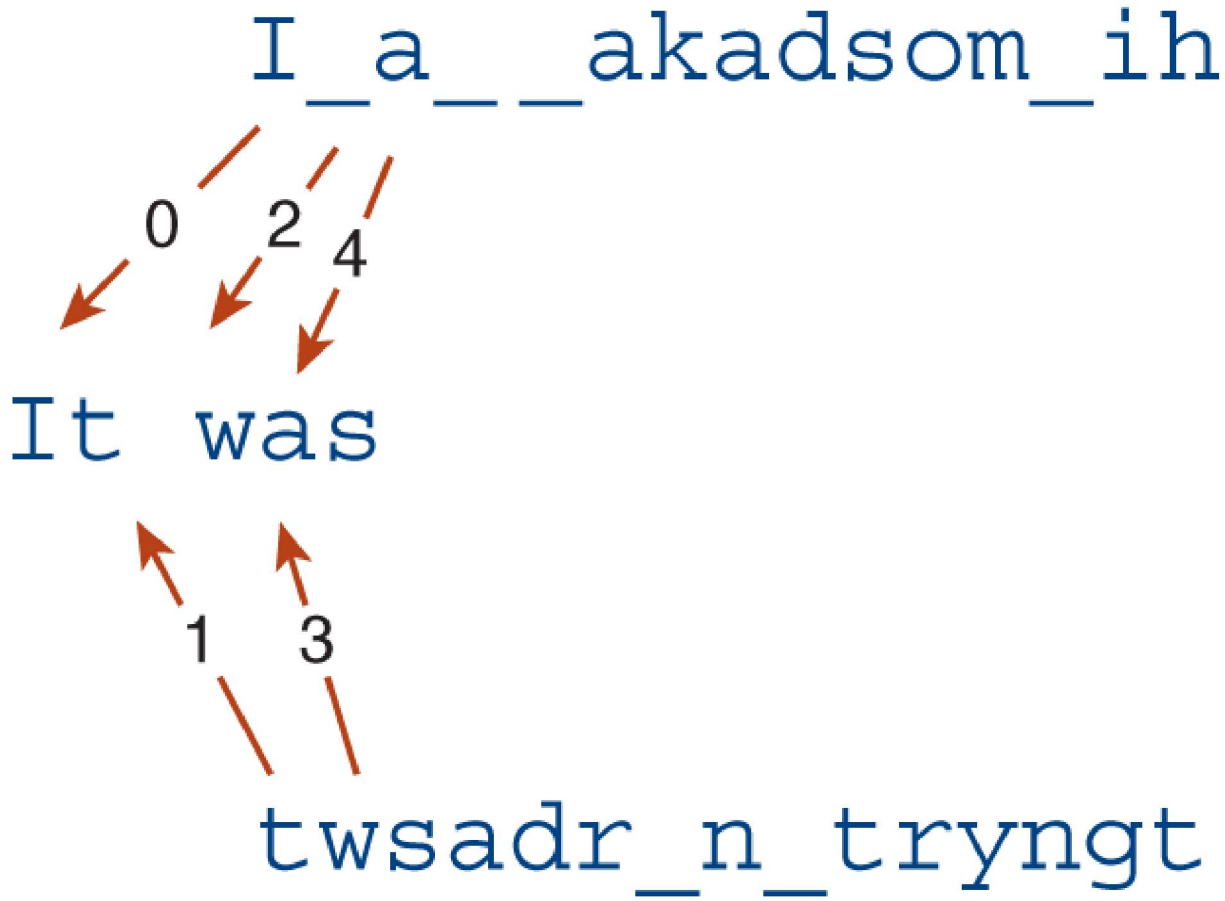


Figure 3.4 Decrypting a message by alternating characters from the ciphertext.

Description

One detail to consider when reconstructing the plaintext message is that we may have one more character in the even-character string than we do in the odd-character string. We can easily check for this by comparing the lengths of the two strings. If the odd-numbered character string is shorter than the even-numbered character string, we simply concatenate the last character from the even string onto the plaintext. **Listing 3.3** shows the Python code for implementing the decryption function.

```
1  def scramble2Decrypt(cipherText):
2      halfLength = len(cipherText) // 2
3      evenChars = cipherText[halfLength:]   #halfLength to end
4      oddChars = cipherText[:halfLength]    #0 to halfLength - 1
5      plainText = ""
6
7      for i in range(halfLength):
8          plainText = plainText + evenChars[i]
9          plainText = plainText + oddChars[i]
10
11     if len(oddChars) < len(evenChars):
12         plainText = plainText + evenChars[-1]   #Last even character
13
14     return plainText
```

Listing 3.3 Decrypting a transposed message

Description

To find the middle of the string, we use integer division to divide the length of the string by 2. Then to split the ciphertext string into two halves, we use the slicing operator, as shown on lines 3 and 4 of Listing 3.3. We use a simple slicing operator shortcut on these lines. Recall that when the number is omitted before the :, the slice operator starts at the beginning of the string;

in contrast, when the number after the `:` is omitted, the slice operation continues until the end of the string.

In the loop that starts on line 7, we use the string indexing operator to get the next character from each of the two halves. Once again, we are using an accumulator pattern to build the plaintext string. Finally, on lines 11 and 12, we check for the case in which the number of characters in the ciphertext is odd, which would make the `evenChars` string longer than the `oddChars` string. If so, we add the final character from `evenChars` to the end of the plaintext string.

Session 3.9 (on the next page) shows how you can test the decrypt function. Notice that since `scramble2Encrypt` returns a string, we can use the function call directly as a parameter to the `scramble2Decrypt` function. In essence, we are passing the return value from `scramble2Encrypt` to the `scramble2Decrypt` function. It is easy to test whether the decryption function works because it should return the same string that we provided as a parameter to the encryption function. Once again, we have tried an easy case, plus several boundary cases.

In [1]:
```
from transposition import *
```

In [2]:
```
scramble2Decrypt(scramble2Encrypt('abababab'))
```

Out[2]: 'abababab'

In [3]:
```
scramble2Decrypt(scramble2Encrypt('a'))
```

Out[3]: 'a'

In [4]:
```
scramble2Decrypt(scramble2Encrypt(''))
```

Out[4]: ''

In [6]:
```
scramble2Decrypt(scramble2Encrypt('I do not like green eggs \
and ham'))
```

Out[6]: 'I do not like green eggs and ham'

Description

Note that in the last test, we cannot fit the entire string on one line. To indicate that we are not finished with the statement we want to execute and that the statement is continued on the next line, we use the **backslash continuation character** (\).

HEADS UP:
In Python, strings must start and end on the same line, unless you insert a continuation character (`\`), which indicates that there is more to the string on the next line. The continuation character can also be used in other Python statements that extend over multiple lines.

Try It Out

3.17 With paper and pencil, use the transposition algorithm to encrypt the sentence "the quick brown fox jumps over the lazy dog." Check your answer by calling the `scramble2Decrypt` function.

3.18 Write a Python function `stripSpaces(myString)` that takes a string representing a phrase as a parameter and returns the phrase with the order of the letters intact but the spaces between each word removed.

3.19 The transposition cipher can be generalized to any number of rails. Write a function to implement a three-rail fence cipher that takes every third character and puts it on one of the three rails.

3.5 Substitution Cipher

The transposition cipher provides us with a simple example, but it is not a very secure form of encryption. An attacker who is trying to read your secret messages and knows you are using a transposition cipher will find it easy to use the same algorithm to decrypt your message.

Another kind of cipher, called the **substitution cipher**, substitutes one letter for another throughout a message. For example, you might decide that `'a' => 't'`. In this case, the letter `'t'` would be inserted into the ciphertext in each place where an `'a'` appears in the plaintext. Substitution ciphers have been used since the time of Caesar, and they remain popular today—just look at the "Cryptoquip" in the daily newspaper.

The substitution cipher has one big advantage over the transposition cipher: It uses a **key**, which is a scrambled version of the alphabet. If attackers are trying to read your secret messages, knowing that you are using a substitution cipher helps them only a little bit, because they would also need to know how the alphabet letters were rearranged. **Figure 3.5** illustrates the encryption process using a key. If we just consider the 26 letters of the alphabet, there are 26 factorial (26!), or 403,291,461,126,605,635,584,000,000 different possible rearrangements of the alphabet. So, we have a lot of keys to choose from!

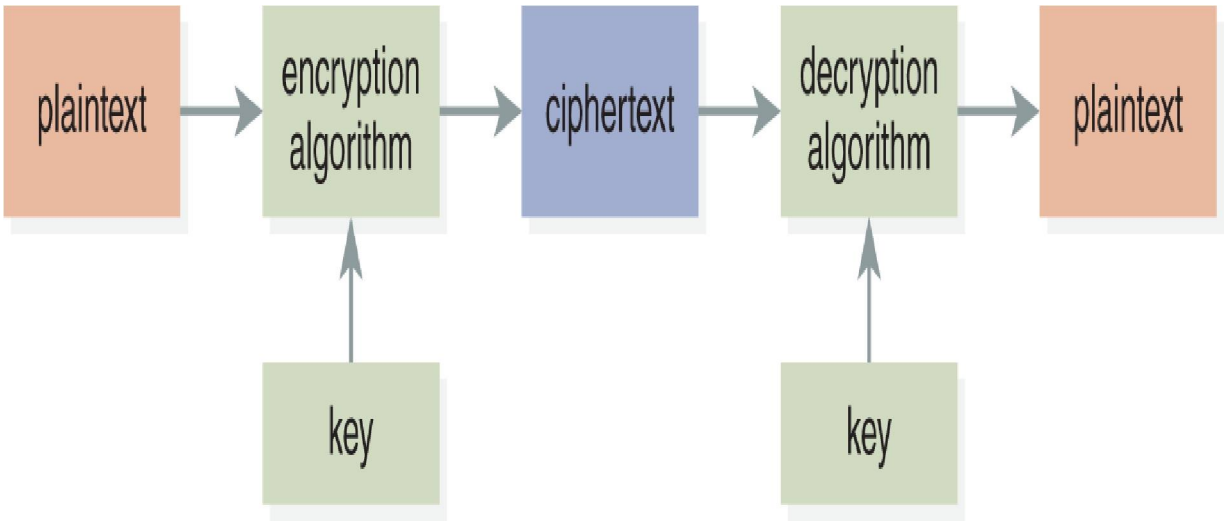


Figure 3.5 Using a key to make encryption more secure.

Description

Let's look at a simple example of a substitution cipher in action. We will encrypt the word `'flow'` using the key `'bpzhgocvjdqswkimlutneryaxf'`. Letters are matched up according to their position in the alphabet. For example, the sixth letter in the plaintext alphabet (index 5) is `'f'`, which corresponds to the sixth letter (index 5) in the key, which is `'o'`. **Figure 3.6** illustrates the process. Each letter in the word `'flow'` is mapped from the regular plaintext alphabet into the corresponding letter in the key. In this case, `'f'` maps to `'o'`, `'l'` maps to `'s'`, `'o'` maps to `'i'`, and `'w'` maps to `'y'`.

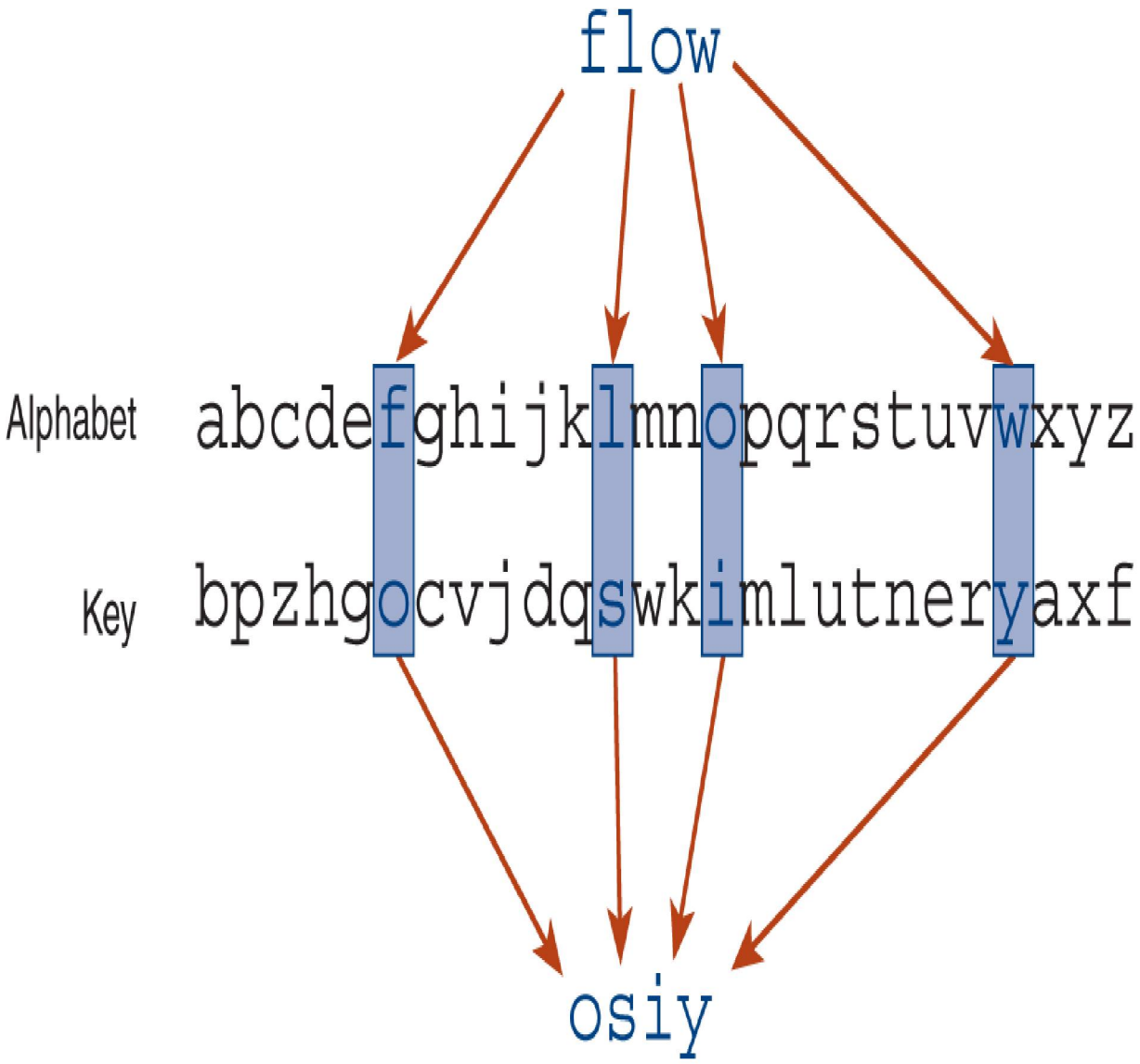


Figure 3.6 A substitution cipher.

Description

Our next step is to turn this mapping process into a Python function. The important step in the encryption process is to take a letter from the plaintext alphabet and map it into a letter in the ciphertext alphabet. We can do this by using the string `find` method. Recall that `aString.find(ch)` returns the index of the first occurrence of `ch` in the string `aString`, or −1 if the string is not found. To use this technique, we can create a string containing all the letters in the alphabet in order and then use the `find` method to return the

position of any letter in the alphabet. Once we know the position of a letter in the plaintext alphabet, we can use the index operator [ ] to look up the substitution letter in the key.

For example, in **Session 3.10**, we find a mapping from the plaintext alphabet to the key alphabet for the letter `'h'`. Note that we define `alphabetString` to contain the letters of the alphabet in order. We include a space at the end so that we can encrypt messages containing spaces. For simplicity, our key here is the reverse of the `alphabetString`. In Session 3.10, we use the `find` method to find the location of `'h'` in `alphabetString`. We then print the return value to see that the letter `'h'` was found at index 7. Using 7 as the index into the `key` string, we find that the letter `'t'` should be substituted for all occurrences of the letter `'h'` in the plaintext message.

```
In [1]: alphabetString = "abcdefghijklmnopqrstuvwxyz "

In [2]: key = " zyxwvutsrqponmlkjihgfedcba"

In [3]: i = alphabetString.find('h')

In [4]: print(i)

        7

In [5]: print(key[i])

        t
```

Session 3.10 Mapping a letter from the alphabet to the key

Description

Now that we know how to map characters from the plaintext to the ciphertext alphabet, what remains is to apply this mapping process to each character of the plaintext message and to construct the final message. For these last two steps, we use iteration and the accumulator pattern. **Listing 3.4** shows a function to encrypt a string using a given key.

```
1  def substitutionEncrypt(plainText, key):
2      import string
3      alphabet = string.ascii_lowercase + " "
4      plainText = plainText.lower()
5
6      cipherText = ""
7      for ch in plainText:
8          idx = alphabet.find(ch)
9          cipherText = cipherText + key[idx]
10
11     return cipherText
```

Listing 3.4 Encrypting a message using a substitution cipher

Description

In line 3, we add the space at the end of the alphabet so that spaces will also be encrypted. Although the ciphertext will contain spaces, those spaces will represent some other character and consequently will give no hints to an attacker. Another approach would be to delete the space from the alphabet and key and to strip out all the spaces from plaintext before applying any substitutions. This would effectively reduce the string `"the quick brown fox"` to the string `"thequickbrownfox"`. As you can see, removing the spaces would make the decrypted message more difficult to read.

On line 4, we convert any uppercase letters in the plaintext to lowercase letters so that uppercase and lowercase letters will have the same substitution value; that is, if the key substitutes `'t'` for `'b'`, then `'t'` will be

substituted for both `'b'` and `'B'`. Also notice that we do not check whether the character is found on line 8. For simplicity, we assume that the plaintext contains only letters and spaces. To handle digits, punctuation, or other characters, our key would need to provide a unique substitution character for each character we might receive in the plaintext message.

Session 3.11 shows the `substitutionEncrypt` algorithm in action. Notice that even though we encrypt the same plaintext message, we get different ciphertext messages when we use different keys. This is what makes the substitution cipher superior to the transposition cipher. Even though attackers may know that we are using a substitution cipher, that information does not do them any good if they do not have the key. The `substitutionDecrypt` function is nearly the same as the `substitutionEncrypt` function.

```
In [1]: from substitution import *
```

```
In [2]: testKey1 = "zyxwvutsrqponml kjihgfedcba"
```

```
In [3]: testKey2 = "ouwckbjmpzyexa vrltsfgdqihn"
```

```
In [4]: plainText = input("Enter the message to encrypt: ")
```

```
Enter the message to encrypt: The quick brown fox
```

```
In [5]: cipherText = substitutionEncrypt(plainText, testKey1)

In [6]: print(cipherText)

        hsvakgrxpayjlemauld

In [7]: cipherText = substitutionEncrypt(plainText, testKey2)

In [8]: print(cipherText)

        smknrfpwynul danb q
```

Session 3.11 Using the `substitutionEncrypt` function

Description

Try It Out

3.20 Make up your own key and encrypt a message. Exchange your key and the ciphertext with a partner to see if you can decrypt each other's message.

3.21 Write the `substitutionDecrypt` function.

3.22 Rewrite the `substitutionEncrypt` function, but remove all spaces from the plaintext message.

3.6 Creating a Key

The strength of the substitution cipher lies in the ability of two people to share a key. That is, both the sender and the receiver of the encrypted message must have access to the same key. That key must represent a scrambled version of the 26 letters of the alphabet plus a space. As a first approach to creating a key, let's write a Python function that returns a randomly scrambled version of the alphabet.

The main idea in this algorithm is that we want to randomly select one character from the alphabet to include in our key, and then to randomly select another character, and another, until we have selected all 27 characters in a random order. This seems pretty simple at first, but we must carefully keep track of the characters we have already selected so that our key does not contain duplicates.

We can think of the problem in a slightly different way. Suppose we start with a string that contains all 27 characters in the plaintext alphabet and an empty string for the key. First, we pick a random number from 0 to 26. This random number gives us the index of the first letter we will use in the key. We concatenate the letter retrieved from the alphabet to our key string. To make sure that we do not pick the same letter again, we remove the selected letter from our alphabet string. Because the alphabet string is now one character shorter, we must pick a random number from 0 to 25. We can repeat this process of picking random numbers and removing characters until we have no more letters in our original alphabet string.

To implement this solution, we can break it up into two subproblems. First, we will write a function that takes a string and an index value as parameters and returns a new string with the character at the given index position removed. Next, we will write a function that starts with a full alphabet and returns the scrambled alphabet as our key. **Listing 3.5** gives a short Python function for removing a character from a string. This version of the function uses the slice operator first to get the substring from the beginning up to, but not including, the character we want to delete, and then to get the substring from the character after the one we want to delete up to

the end. We can use string concatenation to combine these two pieces into our new, smaller string.

```
1 def removeChar(string, idx):
2     return string[:idx] + string[idx + 1:]
```

Listing 3.5 Removing one character from a string

Description

Let's test the `removeChar` function using a simple string and several different index values as shown in **Session 3.12**. Notice that we get an unexpected result when we use the index value −1. Before reading on, see if you can explain why we get this result.

```
In [1]: from keyGeneration import *
```

```
In [2]: removeChar('abcdefg', 0)    #remove first character
```

```
Out[2]: 'bcdefg'
```

```
In [3]: removeChar('abcdefg', 6)   #remove last character
```

```
Out[3]: 'abcdef'
```

```
In [4]: removeChar('abcdefg', 3)   #remove from the middle

Out[4]: 'abcefg'

In [5]: removeChar('abcdefg', -2)  #remove second to last character

Out[5]: 'abcdeg'

In [6]: removeChar('abcdefg', -1)  #try to remove last character???

Out[6]: 'abcdefabcdefg'
```

Session 3.12 Testing the `removeChar` function

Description

This example illustrates the importance of testing even the simplest of functions. The reason we get the result we do when using character index −1 is as follows. The first slice operation (`string[:-1]`) works as intended: It returns the entire string, except for the last character. However, the second slice operation (`string[-1+1:]`) becomes (`string[0:]`), which returns the entire string. We can move forward with this broken version of the `removeChar` function because our key-generating function will never try to remove a negative index.

Now that we have the `removeChar` function, we turn our attention to the main key-generating function, `keyGen()`. **Listing 3.6** shows the Python code

for the strategy we outlined previously. We will successively remove one character from `alphabet` until it is empty, so for the `for` loop, we use `range(len(alphabet)- 1, -1, -1)`, which assigns the loop variable `i` the numbers from 26 to 0 in descending order, keeping step with the decreasing size of `alphabet`. We use the function `randint` from the `random` module to generate a random integer between 0 and the current size of `alphabet`, indicated by `i`. Because we generate random numbers using the call `random.randint(0, i)`, we get a smaller range for the random number as `i` gets smaller. Finally, we use an accumulator pattern `key = key + alphabet[idx]` to build up the final key.

```
1  def keyGen():
2      import string, random
3      alphabet = string.ascii_lowercase + " "
4      key = ""
5
6      for i in range(len(alphabet) - 1, -1, -1):
7          idx = random.randint(0, i)
8          key = key + alphabet[idx]
9          alphabet = removeChar(alphabet, idx)
10
11     return key
```

Listing 3.6 Function to generate a random key from the alphabet

Description

HEADS UP:
Because the `keyGen` function uses the `randint` method and the `string` constant `ascii_lowercase`, we need to `import` the `random` and `string` modules. This can be done in the function itself, as in line 2 in Listing 3.6. Alternately, for any function stored in a `.py` module, the `import` statement could be written in the `.py` module before the function definition.

Session 3.13 shows a few random keys generated by calling our `keyGen` function. In statement 5, we verify that each key is 27 characters long by calling the `len` function on the return value from the `keygen` function. We can also verify manually that each key uses each letter of the alphabet once.

```
In [1]: from keyGeneration import *

In [2]: keyGen()
Out[2]: 'ntod vxpefmaqukbgcwslryzjih'

In [3]: keyGen()
Out[3]: 'mr zapdkequnlofwxchjbgsviyt'

In [4]: keyGen()
Out[4]: 'pv wlkydimcazhxbfeutorsjgnq'

In [5]: len(keyGen())
Out[5]: 27
```

Session 3.13 Generating random keys

Description

Remembering 27 randomly scrambled characters is probably beyond the capabilities of most people, so we will look at another approach to our key-generating problem. Rather than generating a random sequence of 27 characters, suppose that we choose a word or short phrase as our password. The question now becomes, "How can we use this password as the basis for scrambling the alphabet?" The answer is that we will use two different strings to construct the key: the password and the alphabet.

The first step is to remove all duplicate letters in the password. Next, we remove all the letters from the alphabet that are in the password. We then use the alphabet, in order, starting with the letter after the last letter in the password. When we get to the end of the alphabet, we go to the beginning of the alphabet and use the rest of the characters.

Here is an example of how we do this. Suppose our password is `'topsecret'`. Removing the duplicates leaves us with `'topsecr'`. Now if we remove the letters in our password from the alphabet, we have `'abdfghijklmnquvwxyz '`. The next letter in our alphabet after the last letter in our password is `'u'`, so the three strings we will put together to make our final key are `'topsecr'`, `'uvwxyz '`, and `'abdfghijklmnq'`. Putting these all together gives us `'topsecruvwxyz abdfghijklmnq'`.

This may seem like a daunting task, but it is manageable if we break it down into smaller problems:

1. Remove duplicates from the password.
2. Split the alphabet into parts `beforeLast` and `afterLast`.
3. Remove the letters found in the password from `beforeLast` and `afterLast`.
4. Put together the three parts of the key.

Breaking down a problem into smaller parts, in the way that we have done here, is often referred to as **decomposition**. We are dividing the large problem into smaller tasks that are easier to solve.

GOOD PROGRAMMING PRACTICE:

Use decomposition to break a large problem into smaller, manageable tasks.

We already know how to do Steps 2 and 4, so let's solve the problem of removing duplicate letters from a string. We will start by reconstructing the string, one character at a time, using the accumulator pattern. However, we will modify the accumulator pattern by adding one simple rule: If a character is not already part of the reconstructed string, we will add it to the new string; otherwise, the character has already been added to the reconstructed string, so we will ignore it.

For example, consider the string `"book"`. We start by creating an empty string, `newStr`, to build on. Then we iterate through the characters in `"book"`. Because `"b"` is not already part of `newStr`, we add it, giving us `"b"`. The next character is `"o"`. Once again, `"o"` is not in `newStr` so we add it, giving us `"bo"`. Next, we have our second `"o"`, but `newStr` already has an `"o"`, so we ignore this one. The final character, `"k"`, is new, so we add it to `newStr`, giving us `"bok"`.

The key to this approach is asking the question, "Is the current character not already a part of the string we are creating?" To answer this question, we can use the string `not in` operator as the condition of an `if` statement. **Listing 3.7** shows the Python code we use to create a string with duplicates removed.

```
1  def removeDupes(myString):
2      newStr = ""
3
4      for ch in myString:
5          if ch not in newStr:
6              newStr = newStr + ch
7
8      return newStr
```

Listing 3.7 Removing duplicate letters from a string

Description

We use a similar idea to remove the characters from one string that are in another string (Step 3 above). Our `removeMatches` function takes two parameters, the original string and a string containing all the characters to remove from the original string. In this case, the second parameter is our password after duplicates have been removed. Again, we begin with an empty string, `newStr`, and apply the accumulator pattern to build up a new string. If the next character in the original string is not one of the characters in `removeString`, we add it to `newStr`. If the next character in `myString` is in `removeString`, we will simply ignore it. **Listing 3.8** shows the Python code for this process.

```
1  def removeMatches(myString, removeString):
2      newStr = ""
3
4      for ch in myString:
5          if ch not in removeString:
6              newStr = newStr + ch
7
8      return newStr
```

Listing 3.8 Remove the characters in one string from another

Description

Let's test the two new functions we have just written. **Session 3.14** shows both `removeDupes` and `removeMatches` in action.

```
In [1]: from keyGeneration import *
```

```
In [2]: import string
```

```
In [3]: alphabet = string.ascii_lowercase + " "
```

```
In [4]: removeDupes("topsecret")

Out[4]: 'topsecr'

In [5]: removeDupes("wonder woman")

Out[5]: 'wonder ma'

In [6]: removeMatches(alphabet, 'topsecr')

Out[6]: 'abdfghijklmnquvwxyz '

In [8]: removeMatches(alphabet, removeDupes('wonder woman'))

Out[8]: 'bcfghijklpqstuvxyz'
```

Session 3.14 Testing `removeDupes` and `removeMatches`

Description

Now that we have implemented `removeDupes` and `removeMatches`, the rest of our alphabet scrambling algorithm is fairly easy. **Listing 3.9** shows

the Python code to provide a scrambled version of the alphabet using a password as the starting point.

```
1  def genKeyFromPass(password):
2      import string
3      alphabet = string.ascii_lowercase + " "
4      password = password.lower()
5
6      password = removeDupes(password)
7
8      lastChar = password[-1]
9      lastIdx = alphabet.find(lastChar)
10     afterString = removeMatches(alphabet[lastIdx + 1:], password)
11     beforeString = removeMatches(alphabet[:lastIdx], password)
12
13     key = password + afterString + beforeString
14
15     return key
```

Listing 3.9 Generating a key starting from a password

Description

The `genKeyFromPass` function is a good example of problem solving by **top-down design**. Look at the code in Listing 3.9. Now imagine how much longer the code would be if we tried to include the code for `removeDupes` and `removeMatches`. The function would be so long it would be much more difficult to understand what it is doing. Not only do `removeDupes` and `removeMatches` make `genKeyFromPass` more readable, but they can also be used in other functions you write.

GOOD PROGRAMING PRACTICE:
Look for opportunities to write small, reusable functions.

Session 3.15 shows the `genKeyFromPass` function in action. Note that our function does, in fact, generate the key we predicted from the password, `'topsecret'`.

```
In [1]: from keyGeneration import *

In [2]: genKeyFromPass('topsecret')
Out[2]: 'topsecruvwxyz abdfghijklmnq'

In [3]: genKeyFromPass('wonder woman')
Out[3]: 'wonder mabcfghijklpqstuvxyz'
```

Session 3.15 Running the `genKeyFromPass` function

Description

To complete this section, we will consider the security of the substitution cipher. Suppose we do not know the key, but we want to try to read the secret message anyway. One possibility would be simply to use **brute force**, a method in which we try all possible keys until we get plaintext that makes sense. Because the number of keys is equivalent to the number of different ways we can rearrange the 26 letters of the alphabet (ignoring the space at this point), there are 26! or 403×10^{24} possible keys. If we were able to try 1,000,000 keys every second it would take us 12,788,288,341,153 years to try all possible keys. That is 12 trillion years, roughly a thousand times older than estimates of the age of the universe!

However, as we will see in the *Cryptanalysis chapter,* the substitution cipher is quite easy to break by exploiting some simple characteristics of the English language. It is so easy, in fact, that the substitution cipher regularly appears in daily newspapers next to the crossword puzzle under the name "Cryptoquip." The main reason the substitution cipher is easy to break is that there is a one-to-one mapping between letters in the key and letters in the plaintext alphabet. This means that if plaintext "e" maps to ciphertext "k," then it will do so consistently throughout the message. But because "e" is the most common letter in the language, "k" will probably be the most common letter in the ciphertext. This observation gives us a head start in figuring out the rest of the letters.

Try It Out

3.23 Write the `removeChar` function using `for` loops rather than slice operators.

3.24 Modify the `removeChar` function so that it works for negative character indexes.

3.25 Modify the `substitutionCipher` function to use the `genKeyFromPass` function.

3.26 Encryption often involves the Caesar cipher—named after Julius Caesar, who used the system to encrypt military messages. Many early Internet users also adopted this cipher. Called `rot13`, the cipher encrypts a message by rotating the plaintext character by 13 positions in the alphabet. For example, "a" becomes "n"; likewise, "n" becomes "a." The nice thing about `rot13` is that the same function can be used to encrypt and decrypt a message. Write a function called `rot13` that takes a message as a parameter and rotates all the characters by 13 places. Note that `rot13` does not handle spaces.

3.27 Rewrite the Caesar cipher so that it takes the number of places to rotate as a parameter. You will have to write separate encrypt and decrypt functions.

3.7 The Vigenère Cipher

The key to cracking a substitution cipher was first discovered by Arab scholars in the ninth century, but it was not widely known in the West until the 15th century. Once frequency analysis of ciphers became widespread, the plain substitution cipher became useless. To fix the problems inherent in the simple substitution cipher, Blaise de Vigenère invented the strategy of using multiple letter mappings. Rather than using one key for the whole message, Vigenère's idea was to use a different key for each letter of the message. Using a different key for each letter makes the frequency analysis much more difficult.

The key to the Vigenère cipher is the Vigenère square, shown in **Table 3.5**. Each row in the table corresponds to a different key, such that we might encode the first letter of our secret message using row 12, the second letter of the message using row 7, and so on. The rows that we have chosen here are simple rotations of the alphabet. Each row in the table shifts the letters of the alphabet one place to the left. When a letter shifts out of the first column, it moves around to the end of the right side.

TABLE 3.5 The Vigenère Square

	Plaintext																								
Key	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**
a	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
b	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
c	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
d	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
e	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
f	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e

g	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
h	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
i	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
j	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
k	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
l	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
m	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
n	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
o	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
p	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
r	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
s	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
t	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
u	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
v	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
w	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
x	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Description

We will now see how to use the Vigenère square to encode the message, “the eagle has landed.” The first step is to decide on our key. Choosing the word “davinci” as our key, we next align the key across the top of the

message, repeating the letters of the key as needed to cover the message. **Table 3.6** shows how the letters in "davinci" cover the letters in our message. Notice that in this scheme, we will not encode spaces, so we deleted the spaces from the message.

TABLE 3.6 Matching Key Letters with Message Letters

d	a	v	i	n	c	i	d	a	v	i	n	c	i	d	a	v
t	h	e	e	a	g	l	e	h	a	s	l	a	n	d	e	d

Description

To encode the message, we use the row corresponding to the letter in the key and the column corresponding to the plaintext letter to find the ciphertext letter. For example, the first letter in the message is "t," so we look in column "t" and row "d" and find the letter "w." So, "w" is the first character in the ciphertext. We next look at column "h" and row "a" to find that "h" is the second character. The third character is in column "e" and row "v," which means that "z" is the third character. Notice that the next character in the message is also an "e" but this time we use column "e" and row "i," so this "e" is encoded as an "m." If we continue following this pattern, the entire phrase is encoded as "`whz mnith hva ycvgey`".

Although the Vigenère cipher was invented in the 1500s, it went unused for 200 years because the cipher secretaries found it difficult to implement. Obviously, they did not have Python. Let's write a function to encrypt a message using the Vigenère cipher. We will begin by doing a top-down analysis of the procedure needed to encode a message.

1. Initialize an empty result string.
2. For each letter in the plaintext message:
   a. Determine which letter of the key we should use.

b. Using the key letter row, look up the ciphertext letter in the Vigenère square in the plaintext character column.

c. Add the ciphertext letter to the ciphertext message.

3. Return the result string as the ciphertext message.

At first glance, it might seem as if looking up a character in Table 3.5 would be the most difficult part of this procedure, but it is not especially difficult. Let's think about what we know:

1. Each row of the table rotates the letters one more position to the left.
2. The row of the table corresponds to the ordinal value of the key letter.
3. The column of the table corresponds to the ordinal value of the plaintext letter.

(Note that the term *ordinal position* refers to the position of the letter in the range from 0 to 25. This is mapped using our helper functions developed earlier.)

Before we write the Python code for this procedure, we can make a simplification. Using modulo arithmetic, we can avoid copying and shifting letters in a string. We can simply use the formula:

```
cipherTextLetter = (plainTextLetterIndex +
keyLetterIndex) % 26
```

This is how this would work. Suppose we want to encrypt the letter "e" using the key letter "j." The ordinal value for "e" is 4, and the ordinal value for "j" is 9. Applying the formula `(9 + 4) % 26 = 13`, we find the letter with the ordinal value of 13, which is "n." If you check row "j" and column "e" of Table 3.5, you will see this is exactly right. Let's look at another example. Suppose we want to encrypt the letter "t" using the key letter "s." The ordinal value of "t" is 19, and the ordinal value of "s" is 18. This means that `(18 + 19) % 26 = 37 % 26 = 11`, and 11 is the ordinal value for the letter "l," which you will find in row "s" and column "t."

The code for the `vigenereIndex` function is shown in **Listing 3.10**. The `vigenereIndex` simply uses our helper functions `letterToIndex` and `indexToLetter`, which are stored in `mapper.py`, to do the lookups we need.

```
1 from mapper import *
2
3 def vigenereIndex(keyLetter, plainTextLetter):
4     keyIndex = letterToIndex(keyLetter)
5     ptIndex = letterToIndex(plainTextLetter)
6
7     newIndex = (ptIndex + keyIndex) % 26
8
9     return indexToLetter(newIndex)
```

Listing 3.10 Looking up a letter in the Vigenère square

Description

The final step in implementing the Vigenère cipher is to write a function that takes a key and a plaintext message as parameters and returns the ciphertext message. As before, we will take a character-by-character approach to encrypting the message, applying our `vigenereIndex` function to each character. The only remaining question is how to decide which letter of the key to use in conjunction with each letter of the message.

One way to solve this problem is to duplicate the key as many times as needed so we can cover all the letters in the message, as we did in Table 3.6. If our message is long, however, creating another long string by duplicating the key is wasteful. We can avoid duplicating the key by using modulo arithmetic. If our key is K letters long, we want an index counter for our key to repeatedly cycle through the numbers 0 through $K - 1$. The modulo operator allows us to do exactly that. Try **Session 3.16** for yourself.

```
In [1]: for i in range(100):
            print(i % 7, end = ' ')
```

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6
0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6
0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6
0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1
```

Session 3.16 Demonstrating circular counting with modulo arithmetic

Description

In Session 3.16, we send a new type of parameter to the `print` function: `end = ' '`, which is called a **keyword parameter**. Python functions can accept both keyword parameters and **positional parameters**. Positional parameters, which are the type of parameters we have used so far, are simply values given in the order listed in the function's header. For the `print` function, the positional parameters are the items to be printed. After the positional parameters are given, a function can optionally accept keyword parameters, which consist of (1) the keyword, (2) the assignment operator, and (3) a value. The `print` function accepts four keyword parameters, as shown in **Table 3.7**; all of these parameters are optional. If we call the `print` function without any of the optional keyword parameters, the `print` function outputs all the positional parameters to the standard Python output, separating each value by a space, and then appends a newline character to move to the next line. In Session 3.16, we use the `end` keyword and assign a space as its value to specify that we want a space to follow each value, rather than a newline character.

TABLE 3.7 The `print` Function and Its Positional and Keyword Parameters

Function	Description
`print( value1, …,` (optional keyword parameters separated by commas)	Outputs each value Optional keyword parameters: ▪ `sep`: the string to separate each value; default is a space. ▪ `end`: the string to output after all values are outputted; default is a newline. ▪ `file`: where to send the output; default is the Python's standard output. ▪ `flush`: `True` to output immediately, or `False` to place output temporarily in a holding buffer; default is `False`. This parameter is meaningful when writing to a file rather than to the standard output.

Using the modulo arithmetic demonstrated in Session 3.16, we can use a counter `k` to keep track of the current character in the message and then use `counter % k` as the index into the key. **Listing 3.11** shows the entire function needed to encode a message using the Vigenère cipher. Instead of passing the key and plaintext as parameters, we prompt the user to enter the values; this makes it easier to encrypt a long message. **Session 3.17** shows the `encryptVigenere` function in action.

```
1  def encryptVigenere():
2      key = input('Enter the key: ')
3      plainText = input('Enter the message to be encrypted: ')
4      cipherText = ""
5      keyLen = len(key)
6      k = 0 # this represents which key row should be used in the table
7      for ch in plainText:
8          if ch == ' ':
9              # remember spaces are not used in the encryption
10             cipherText = cipherText + ch
11         else:
12             cipherText = cipherText \
13                 + vigenereIndex(key[k % keyLen], ch )
14             k = k + 1
15     return cipherText
```

Listing 3.11 Encrypting a message using the Vigenère cipher

Description

```
In [1]: from vigenere import *

In [2]: encryptVigenere()

        Enter the key: davinci
        Enter the message to be encrypted: the eagle has landed

Out[2]: 'whz mnith hva ycvgey'

In [3]: encryptVigenere()

        Enter the key: calendar
        Enter the message to be encrypted: one small step for man

Out[3]: 'qnp wzdlc utpt srr dcn'
```

Session 3.17 Encrypting a message using the `encryptVigenere` function

Description

Try It Out

3.28 Write a function `undoVig(keyLetter, ctLetter)` that takes a letter from the key, a ciphertext letter, and returns the plaintext letter.

3.29 Write a function `decryptVigenere` that takes a keyword, the ciphertext for the message, and returns the plaintext message.

3.8 Summary

In this chapter, we explored an important Python data type called strings—sequences of characters that provide a wide range of built-in functions, string operators, and methods.

- String operators:
  - Concatenation (`+`)
  - Repetition (`*`)
  - Indexing (`[]`)
  - Slicing (`[:]`)
  - Membership (`in` and `not in`)
- Length (`len`) built-in function
- String search methods:
  - `find/rfind`
  - `index/rindex`
- String formatting methods:
  - `upper`
  - `lower`
  - `center`
  - `ljust`
  - `rjust`
- Other string methods
  - `replace`
  - `count`
  - `startswith`
  - `countains`

We also introduced Python built-in character functions:

- `ord`

- `chr`

We distinguished between positional and keyword parameters with an example using the `print` function.

We used decomposition to build up a collection of small functions that could be used to solve larger problems.

We used string manipulation to implement some basic encryption/decryption schemes for converting plaintext to ciphertext, and back again. These algorithms allowed us to illustrate iteration over characters in a string, as well as the accumulator pattern as it applies to strings.

- Transposition cipher
- Substitution cipher
- The Vigenère cipher

Key Terms

American Standard Code for Information Interchange (ASCII)
backslash continuation character
boundary values
brute force
characters
ciphertext
cryptography
decomposition
decryption
encryption
index operator
key
keyword parameter
plaintext
positional parameters
proper prefixes of a string
rail fence cipher
slice operator
string iterator
substitution cipher
substrings
top-down design
transposition cipher

Programming Exercises

3.1 Research the “Playfair cipher.” Write a program to implement this cipher.

3.2 Research the UPC codes that are used on products of all kinds. Use the `turtle` module to draw a UPC code for a given product name and price.

3.3 Research the U.S. postal bar codes used to encode a zip code. Use the `turtle` module to draw a postal code for a given zip code.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 4

Introducing the Python Collections

Lists, Dictionaries, Tuples, and Statistics

4.1 OBJECTIVES

- To use Python lists as a means of storing data
- To use Python dictionaries to store associative data
- To use Python tuples as a means of storing data
- To implement algorithms to compute elementary statistics
- To use functions from the Python `statistics` module
- To learn how to produce a histogram using the `matplotlib` library
- To learn how to call functions from the `math` module

4.2 What Is Data?

Every year, thousands of earthquakes occur around the world. Most of them are so mild that they are barely noticed except by scientists with monitoring equipment. If we consider only those earthquakes that people feel, then perhaps 35 occur on a typical day. Some days, there may be only 10; on other days, there may be 40 or 50.

What we have just described is a way that earthquake occurrence might be reported through **data**—an assortment of items, many times numeric, that have been observed, measured, or collected by some means. These data items pertain to some experiment, event, or activity that we are interested in exploring. Data, sometimes referred to as "raw data," represents the starting point for analysis that can be performed in an attempt to discover underlying characteristics. Data can contain information in various forms, and the information can be used in a variety of ways. Sometimes this information allows us to make generalizations about the data items. We may also be able to make predictions about future events based on the data. This kind of analysis is based on the mathematical science of statistics.

This chapter centers on data, information, and statistics. We will focus on ways that a programming language like Python can help us perform some of the basic data processing tasks that are commonly done with large amounts of data.

4.3 Storing Data for Processing

Whenever we work with large amounts of data, we need some means of organizing the data's storage so that we can process the data in an orderly and efficient manner. Python's built-in **collections** provide ways to organize and store data values. We have already introduced one type of collection: strings. In this chapter, we will further explore strings and consider three more Python collections: lists, dictionaries, and tuples.

4.3.1 Strings Revisited

We have introduced the idea of a string as a sequential collection of characters. Python calls this a **sequence**. Each string is considered to be ordered from left to right, and each character can be accessed using the indexing operation. Provided that our data items can be thought of as characters, a string might make the perfect collection mechanism.

As an example, assume that we have just taken a multiple-choice exam where each question has five possible answers labeled A, B, C, D, and E. Our answer sheet might look like this:

1. A
2. B
3. E
4. A
5. D
6. B
7. B
8. A
9. C
10. E

To store our answers for later processing, we could use a simple string of 10 characters. The string will contain the 10 data items gathered from the

student:

```
In [1]: myAnswers = "ABEADBBACE"
        for answer in myAnswers:
            print(answer)
```

```
A
B
E
A
D
B
B
A
C
E
```

Description

This technique allows us easy access to each answer. Furthermore, using iteration over strings, we can process the entire exam. However, some potential drawbacks arise with this approach. What if the exam was in math class, where the answers are numbers instead of characters? Can a string still be used?

For example, suppose this is our answer sheet for a math exam:

1. 34
2. 56
3. 2
4. 652
5. 26
6. 1
7. 99
8. 865
9. 22
10. 16

If we attempt to use the same string-based storage technique, we might get something like this:

```
myAnswers = "34562652261998652216"
```

Description

It is easy to see that we have a problem. How can we distinguish which numbers correspond to each answer?

In this case, we need a way to store collections of integers instead of being restricted to just using characters. As with the string, where there is one character per position, we would like an organization technique that can provide one integer per position. Fortunately, there is such a collection—the list.

4.3.2 Lists

A **list** is a collection that is similar to a string in general structure, but has some differences that must be understood for the list to be used properly.

Figure 4.1 shows that strings and lists are examples of sequential collections.

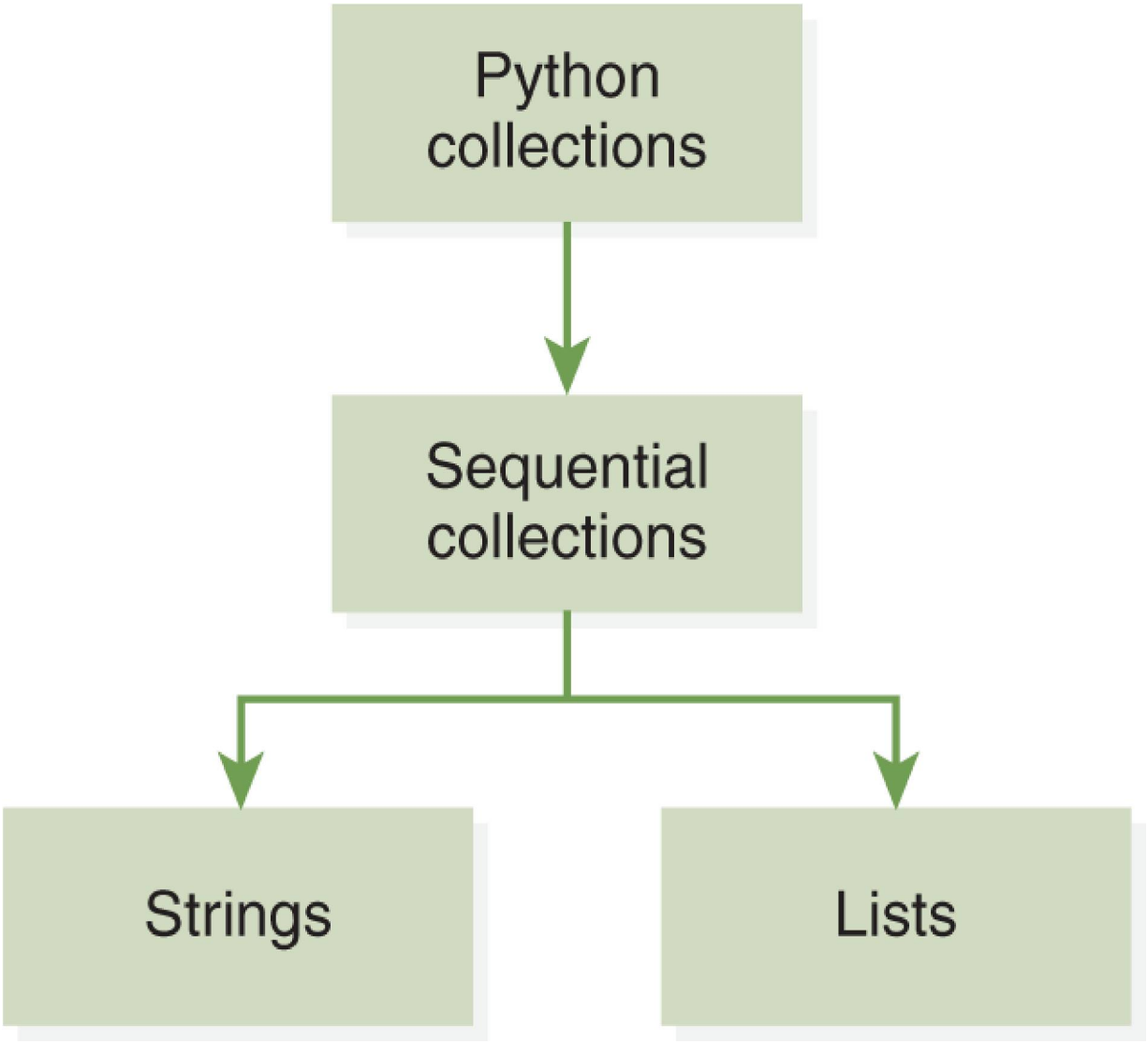


Figure 4.1 Lists and strings as sequential collections.

A list is an ordered, sequential collection of zero or more Python data objects. Lists are written as comma-delimited values enclosed in square brackets. We call a list with zero data objects the **empty list**, which is represented simply by `[]`. Recall that strings are **homogeneous collections** because each item in the collection is the same type of object—a character. Lists, by contrast, are **heterogeneous collections** because lists can be

composed of a mixture of any kind of object. In **Session 4.1**, the list `myList` consists of two integers, a floating-point value, and a string.

```
In [1]: [3, "cat", 6.5, 2]      #define an unnamed list

Out[1]: [3, 'cat', 6.5, 2]

In [2]: myList = [3, "cat", 6.5, 2]

In [3]: myList                  #assign the list to a name

Out[3]: [3, 'cat', 6.5, 2]
```

Session 4.1 A Python list

Description

RECAP:
A list is a sequence consisting of heterogeneous objects. Lists are written as comma-delimited values enclosed in square brackets.

As with other values in Python, asking the interpreter to evaluate a list will simply return the list itself. To remember the list for later processing, we need to assign the list to a variable. Evaluating the variable then returns the list. **Figure 4.2** shows the sequential organization of the items in the list example in Session 4.1.

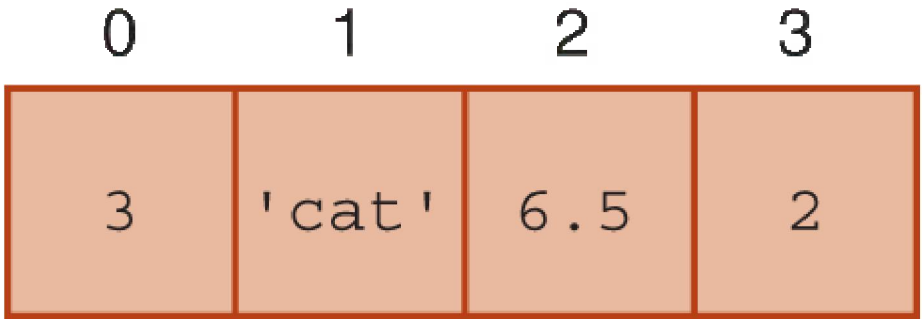


Figure 4.2 Sequential storage of the elements in a list with indexes.

Description

Since lists are sequential, many operations that can be applied to any Python sequence, such as strings, can be applied to lists as well. **Table 4.1** reviews these operations, and **Session 4.2** gives examples of their use. Note that the last example in Session 4.2 introduces the `del` statement (see **Table 4.2**), which allows you to delete an item from a list. This operation is not permitted on strings. Strings are **immutable collections** of data in which individual items within the string cannot be changed. That is not true for lists: We can change an individual member of a list by using the assignment statement and placing the indexed location on the left-hand side. Thus lists, unlike strings, are **mutable collections** of data; that is, lists can be modified.

TABLE 4.1 Operations on Any Python Sequence

Operation Name	Operator/Function	Explanation
Indexing	`[ ]`	Accesses an element of a sequence by its position.
Concatenation	`+`	Combines sequences of the same type.
Repetition	`*`	Concatenates a repeated number of times.
Membership	`in`	Asks whether an item is in a sequence.
Membership	`not in`	Asks whether an item is not in a sequence.
Length	`len`	Asks the number of items in the sequence.
Slicing	`[ : ]`	Extracts a part of a sequence.

```
In [1]: myList = [3, 'cat', 6.5, 2]

In [2]: myList[2]            #item 2 using indexing
Out[2]: 6.5

In [3]: myList + ["read", 7] #concatenate list with another list
Out[3]: [3, 'cat', 6.5, 2, 'read', 7]

In [4]: myList * 3           #repeat list 3 times
Out[4]: [3, 'cat', 6.5, 2, 3, 'cat', 6.5, 2, 3, 'cat', 6.5, 2]

In [6]: len(myList)          #number of items in the list
Out[6]: 4

In [7]: myList[1:3]          #slice items 1 to 2
```

```
In [7]: myList[1:3]          #slice items 1 to 2
```

```
Out[7]: ['cat', 6.5]
```

```
In [8]: 3 in myList          #test whether item is in list
```

```
Out[8]: True
```

```
In [9]: "dog" in myList
```

```
Out[9]: False
```

```
In [10]: "dog" not in myList
```

```
Out[10]: True
```

```
In [11]: del myList[2]       #delete item 2
```

```
In [12]: myList
```

```
Out[12]: [3, 'cat', 2]
```

Session 4.2 Using sequence operators and functions with lists

Description

TABLE 4.2 An Additional Operation on Lists

Operation Name	Statement	Explanation
Delete	`del`	Deletes an item.

Note that the indexes for lists, as with strings, start with 0. The slice operation `myList[1:3]` returns a list of items starting with the item indexed by 1 up to, but not including, the item indexed by 3. Also note that after lines 3 and 4 the list has not been altered, because we did not assign the results of the concatenation and repetition to `myList`.

Session 4.3 shows an assignment statement modifying the item at index 2 in the list `changeList`. The reference diagram in **Figure 4.3** shows that a list is a collection of references to Python objects. Changing an item in the list simply changes the reference stored at that position. Note that after item 2 has been changed, the previous reference is deleted, as indicated by a dotted line in the figure. As shown in Session 4.3, attempting to change an item in a string using the same operation will not work; the Python interpreter reports an error because strings do not support the ability to change a single character.

```
In [1]: changeList = [1, 2, "buckle my shoe", 3, 4, "shut the door"]
```

```
In [2]: changeList
```

```
Out[2]: [1, 2, 'buckle my shoe', 3, 4, 'shut the door']
```

```
In [3]: changeList[2] = "the sky is blue"    #change item 2
```

```
In [4]: changeList
```

```
Out[4]: [1, 2, 'the sky is blue', 3, 4, 'shut the door']
```

```
In [5]: name = "Monte"
```

```
In [6]: name[2] = "x"     #attempt to change character in a string
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [6], in <cell line: 1>()
----> 1 name[2] = "x"

TypeError: 'str' object does not support item assignment
```

Session 4.3 Mutating a list

Description

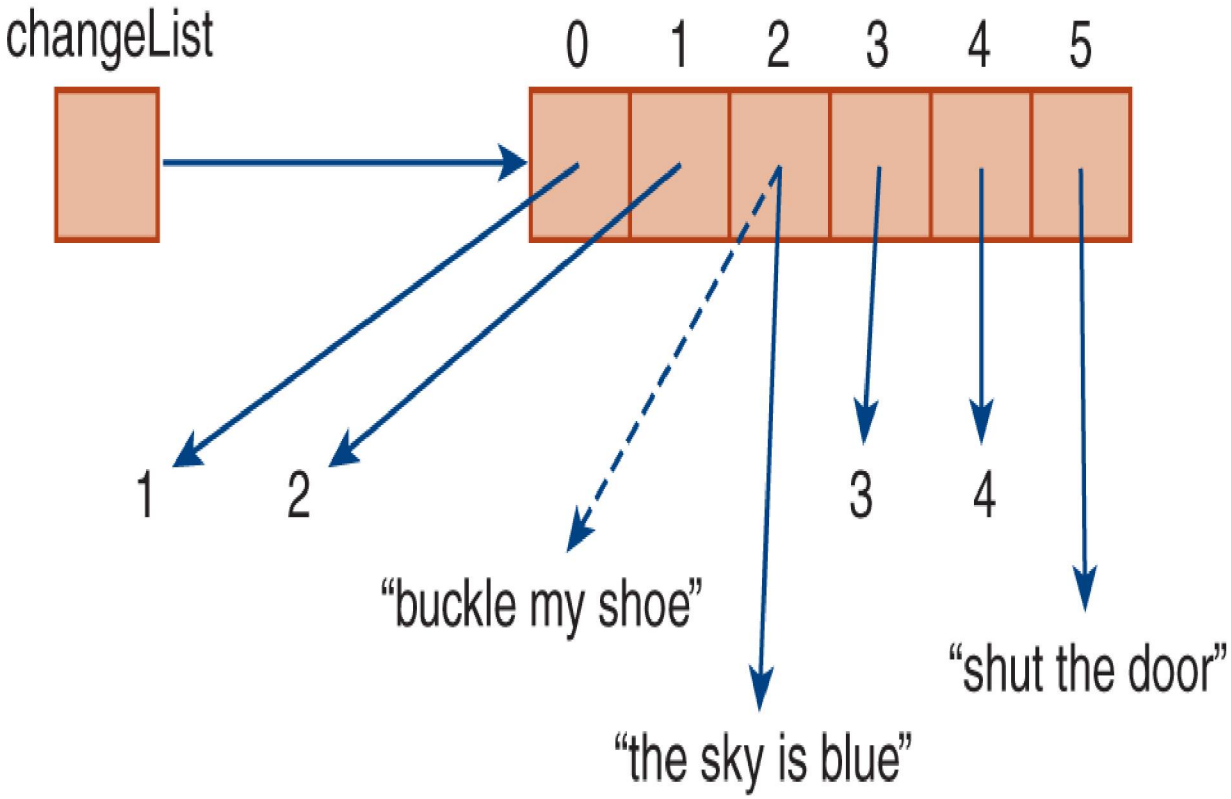


Figure 4.3 The collection of references forming a list.

Description

The use of the repetition operator (*) is also affected by this idea of a collection of references. The result of using the * operator is a repetition of references to the data objects in the sequence. This can best be seen by considering the statements in **Session 4.4**.

```
In [3]: myList = [1, 2, 3, 4]

In [4]: myList

Out[4]: [1, 2, 3, 4]

In [5]: listOfMyList = [myList] * 3

In [6]: listOfMyList

Out[6]: [[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]

In [7]: myList[2] = 45      #change one item in myList

In [8]: listOfMyList

Out[8]: [[1, 2, 45, 4], [1, 2, 45, 4], [1, 2, 45, 4]]
```

Session 4.4 Mutating a list created with repetition

Description

The notation `[myList]` means we have a list containing one item, whose value is a reference to the list `myList`. Hence, it is a list containing a list. The variable `listOfMyList` holds a collection of three references to the original list, `myList`. Note that a change to one element of `myList` shows up in all three occurrences in `listOfMyList`. This outcome occurs because the repetition operation is implemented as a list of three references to the same list, as shown in **Figure 4.4**.

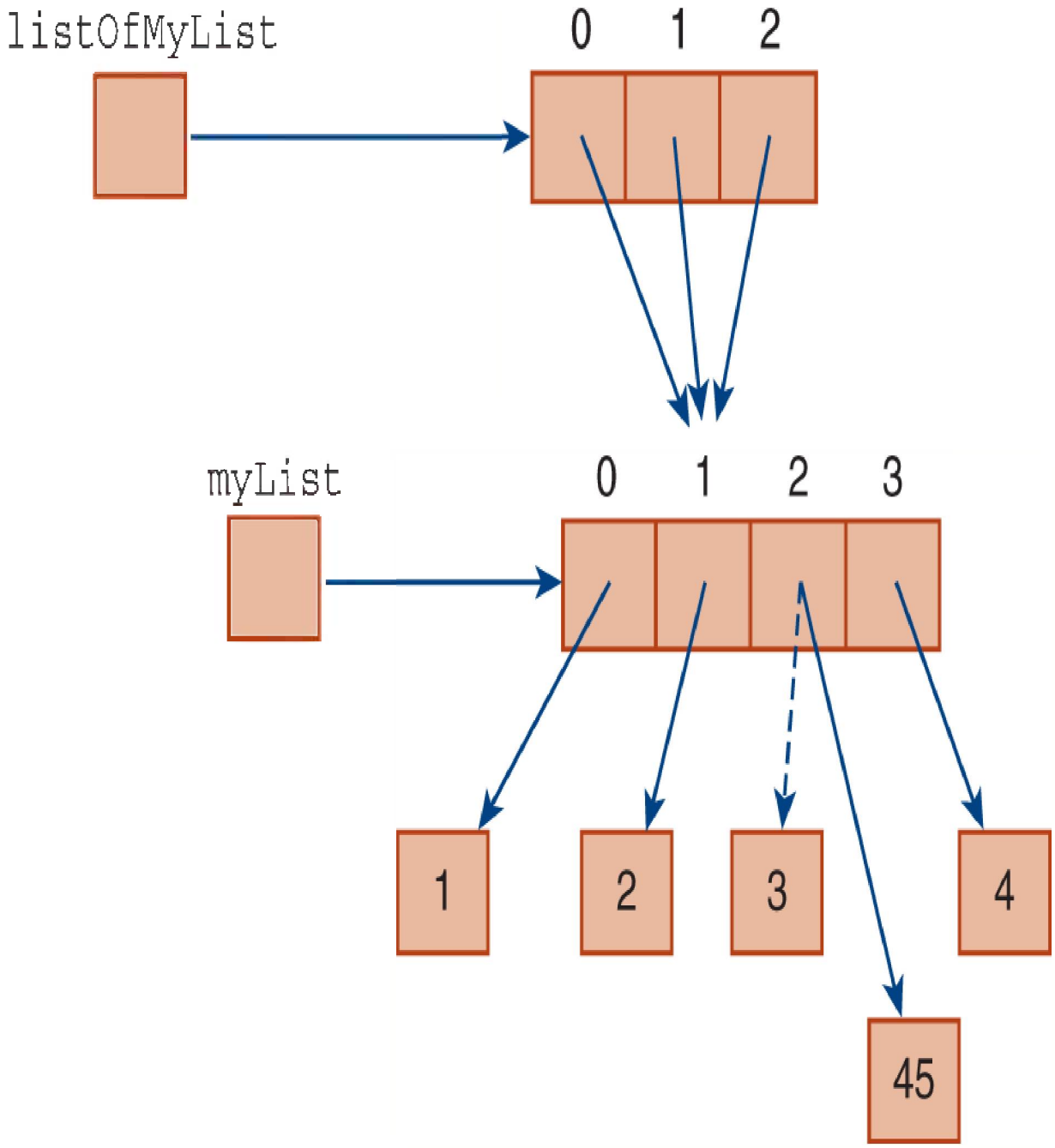


Figure 4.4 The repetition operator copies references.

Description

A useful function for creating lists is the `list` built-in function, which converts other sequences to lists. We have already seen two such sequences: strings and **ranges**. Recall that the `range` function returns an object

representing a sequence of integers. **Session 4.5** demonstrates the use of `list` to create some simple lists from strings and ranges.

```
In [1]: range(10)              #range is a sequence

Out[1]: range(0, 10)

In [2]: list(range(10))        #create a list from a range

Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [3]: list(range(10, 2, -2))

Out[3]: [10, 8, 6, 4]

In [4]: list("the quick fox")  #create a list from a string

Out[4]: ['t', 'h', 'e', ' ', 'q', 'u', 'i', 'c', 'k', ' ', 'f', 'o', 'x']
```

Session 4.5 Using the `list` function

Description

Lists support a number of other useful methods, as summarized in **Table 4.3**. Examples of their use can be seen in **Session 4.6**.

TABLE 4.3 Methods and Function Provided by Lists in Python

Method or Function Name	Use	Explanation
`list`	`list(sequence)`	Function that creates a list from the elements in `sequence`.
`append`	`aList.append(item)`	Adds an `item` to the end of a list.
`insert`	`aList.insert(i, item)`	Inserts `item` at the `i`th position in a list.
`pop`	`aList.pop()`	Removes and returns the last item in a list. Raises an `IndexError` if the list is empty.
`pop`	`aList.pop(i)`	Removes and returns the `i`th item in a list. Raises an `IndexError` if the list is empty or there is no `i`th item in the list.
`sort`	`aList.sort()`	Arranges a list of numbers into numeric order, a list of strings into alphabetic order, and a list of Booleans into `True`, then `False` order.
`reverse`	`aList.reverse()`	Modifies a list to be in reverse order.
`index`	`aList.index(item)`	Returns the index of the first occurrence of `item`. Raises a `ValueError` if `item` is not found.
`count`	`aList.count(item)`	Returns the number of occurrences of `item`.
`remcve`	`aList.remove(item)`	Removes the first occurrence of `item`. Raises a `ValueError` if `item` is not in the list.
`clear`	`aList.clear()`	Removes all items in the list.
`copy`	`aList.copy()`	Returns a copy of the list.

```
In [1]: 1 myList = list( (1024, 3, True, 6.5) )
```

```
In [2]: 1 myList
```

Out[2]: [1024, 3, True, 6.5]

In [3]:
```
myList.append(False)        #add False to end of list
```

In [4]:
```
myList
```

Out[4]: [1024, 3, True, 6.5, False]

In [5]:
```
myList.insert(2, 4.5)       #insert 4.5 at index 2
```

In [6]:
```
myList
```

Out[6]: [1024, 3, 4.5, True, 6.5, False]

In [7]:
```
myList.pop()                #remove and return last item
```

Out[7]: False

In [8]:
```
1 myList
```

Out[8]: [1024, 3, 4.5, True, 6.5]

In [9]:
```
1 myList.pop(3)          #remove and return item at index 3
```

Out[9]: True

In [10]:
```
1 myList
```

Out[10]: [1024, 3, 4.5, 6.5]

In [11]:
```
1 myList.sort()          #sort the list
```

In [12]:
```
1 myList
```

Out[12]: [3, 4.5, 6.5, 1024]

```
In [13]:
1 myList.reverse()          #reverse the order of the list
```

```
In [14]:
1 myList
```

```
Out[14]: [1024, 6.5, 4.5, 3]
```

```
In [15]:
1 myList.count(6.5)         #count the occurrences of 6.5
```

```
Out[15]: 1
```

```
In [16]:
1 myList.index(4.5)         #return index of 4.5
```

```
Out[16]: 2
```

```
In [17]:
1 myList.remove(6.5)        #delete 6.5
```

```
In [18]:
1 myList
```

```
Out[18]: [1024, 4.5, 3]
```

```
In [19]: 1 myList.clear()          #delete all items

In [20]: 1 myList

Out[20]: []
```

Session 4.6 Examples of `list` methods

Description

You can see that some of the methods, such as `pop`, return a value and also modify the list. Others, such as `reverse` and `sort`, simply modify the list with no return value. Note that by default `sort` will not work on lists that have both numeric and string items because strings and numbers cannot be compared. The `pop` method without parameters will remove the last item in the list. You can also remove and return an item at a specific index location by sending the index as a parameter. Also notice the familiar "dot" notation, which is used when asking an object to invoke a method. You can read `myList.append(False)` as "ask the object `myList` to perform its `append` method using the value `False` as a parameter." All data objects invoke methods in this way.

Before leaving this section, we will describe one additional string method —`split`, shown in **Table 4.4**. This method takes a string as a parameter that indicates the places to break the string into substrings; the substrings are returned in a `list`. By default, if no parameter is passed to `split`, it will break the string using one or more spaces or tabs as the **delimiter**. In Python, spaces and tabs are considered to be **whitespace**. The `split` method

discards the delimiters; that is, the delimiter is not included in the resulting list. **Session 4.7** demonstrates how the `split` method works.

TABLE 4.4 The String `split` Method

Method	Explanation
`split()`	Returns a list of substrings created by splitting the string using whitespace as the delimiter. The whitespace is not part of the list.
`split(delim)`	Returns a list of substrings created by splitting the string using `delim` as the delimiter, and omitting occurrences of `delim` from the substrings.

```
In [1]:  1 team = "Minnesota Vikings"
```

```
In [2]:  1 team.split()          #split at spaces or tabs
```

```
Out[2]: ['Minnesota', 'Vikings']
```

```
In [3]:  1 team.split('i')       #split using i as delimiter
```

```
Out[3]: ['M', 'nnesota V', 'k', 'ngs']
```

```
In [4]:  1 team.split('nn')      #split using nn as delimiter
```

```
Out[4]: ['Mi', 'esota Vikings']
```

Session 4.7 Using the string `split` method

Description

Try It Out

4.1 Create a list with the following five items: 7, 9, `'a'`, `'cat'`, `False`. Assign this list to the variable `myList`.

4.2 Write Python statements to do the following:

a. Append 3.14 and 7 to the list.

b. Insert the value `'dog'` at position 3.

c. Find the index of `'cat'`.

d. Count the number of 7s in the list.

e. Remove the first 7 from the list.

f. Remove `'dog'` from the list using `pop` and `index`.

4.3 Split the string `"the quick brown fox"` into a list of words.

4.4 Split the string `"mississippi"` into a list using `'i'` as the split point.

4.5 Write a function that takes a sentence as a parameter and returns the number of words in the sentence.

4.6 Although Python provides us with many list methods, it is good practice and very instructive to think about how they are implemented. Implement a Python function that works like the following:

a. `count`

b. `in`: return `True` if the item is in the list

c. `reverse`

d. `index`: return `-1` if the item is not in the list

e. `insert`

4.7 Write a function `shuffle` that takes a list as a parameter and returns a new list with the elements shuffled into a random order.

4.8 Write a function `shuffle` that takes a list as a parameter and shuffles the list in place.

4.9 Draw a reference diagram to illustrate the differences between the previous two exercises.

4.10 Suppose you initialize the following list: `myList = [[]]*3`. Evaluate this expression.

4.11 Evaluate the expression `myList[1].append(2)`. Explain the result.

4.12 Draw a reference diagram to illustrate what is happening in the previous two exercises.

4.13 Write a code fragment to initialize `myList` so that each sublist is independent—that is, so that changes to one list do not affect the others.

4.4 Calculating Statistics on Data

In the next few sections, we perform simple statistics on data sets stored in list structures. For example, let's assume that we want to analyze statistics on earthquakes worldwide. During one seven-day period, we find that there were 37 earthquakes on Monday, 32 on Tuesday, 46 on Wednesday, 28 on Thursday, 37 on Friday, 41 on Saturday, and 31 on Sunday. This data can be stored in the simple list:

```
[37, 32, 46, 28, 37, 41, 31]
```

Python's `statistics` module can help with our analysis of the data. **Table 4.5** highlights some useful functions from this module. To use these functions, we need to `import` the `statistics` module. In addition, we can get help from some built-in Python functions, as shown in **Table 4.6**. All of these functions operate on lists.

TABLE 4.5 Some Useful Functions of the `statistics` Module

Function	Explanation
`mean(data)`	Returns the average of the data values.
`median(data)`	Returns the middle value of ordered numeric data. If the number of data items is even, returns the average of the two middle values.
`mode(data)`	Returns the value that appears most often. If more than one value appears with the same highest frequency, returns the first mode in the list.
`multimode(data)`	Returns a list of the values that appear most often.
`stdev(data)`	Returns the standard deviation.
`stdev(data, mean)`	Returns the standard deviation, given the mean.

TABLE 4.6 The `min`, `max`, and `sum` Built-in Functions

Function	Explanation

Function	Explanation
`min(sequence)`	Returns the smallest item in the sequence.
`max(sequence)`	Returns the largest item in the sequence.
`sum(sequence)`	Returns the sum of the items in a numeric sequence.

HEADS UP:
To use the functions in the `statistics` module, be sure to `import statistics`.

4.4.1 Simple Dispersion

We now turn our attention to the simple idea of **dispersion**—a measure of how spread out the data values are.

The easiest way to look at dispersion is to consider the range of values and simply compute the difference between the largest and smallest values in the data set. In Python, this is easy: We just use the built-in functions `max` and `min`, shown in Table 4.6, which work on any sequence and return the maximum and minimum values, respectively, in the collection.

As shown in **Session 4.8**, we can compute the range of the integer list by subtracting the minimum from the maximum. Note that these functions work with strings as well as with lists, because both are sequences. Characters are "ordered" by their ordinal values.

```
In [1]: 1 earthquakes = [37, 32, 46, 28, 37, 41, 31]
```

```
In [2]: 1 max(earthquakes)
```

```
Out[2]: 46
```

```
In [3]: 1 min(earthquakes)

Out[3]: 28

In [4]: 1 max(earthquakes) - min(earthquakes)

Out[4]: 18

In [5]: 1 max("house")          #max/min also work on characters in strings

Out[5]: 'u'

In [6]: 1 min("house")

Out[6]: 'e'
```

Session 4.8 Using the `min` and `max` functions

Description

To construct a function to return the range of a data set, we will assume that the data values have been placed in a list. The function can then take the

list as a parameter and return the range. **Listing 4.1** shows such a function, and **Session 4.9** illustrates its use.

```
1 def getRange(aList):
2     return max(aList) - min(aList)
```

Listing 4.1 A function to return the range of a list

Description

```
In [1]:  1 from getRange import *
```

```
In [2]:  1 getRange( [2, 5] )
```

```
Out[2]: 3
```

```
In [3]:  1 earthquakes = [37, 32, 46, 28, 37, 41, 31]
```

```
In [4]:  1 getRange(earthquakes)
```

```
Out[4]: 18
```

Session 4.9 Using the `getRange` function

Description

An interesting question arises if we consider what would happen if we did not have a built-in function to return the maximum value in the list. In such a case, we could construct our own `getMax` function by iterating through the items in the list and keeping track of the largest value seen. This function, shown in **Listing 4.2**, assumes that the list contains at least one item (otherwise the idea of a "maximum" makes no sense). The variable `maxSoFar` does most of the work. We begin by letting `maxSoFar` be the first item in the list (`aList[0]`). Then, as we "visit" other items, we check whether they are greater than `maxSoFar`. If they are, we reassign the `maxSoFar` variable to the new larger item. When we reach the end of the list, `maxSoFar` will hold the maximum of all the items.

```
1  def getMaxByIndex(aList):
2      maxSoFar = aList[0]
3      for pos in range(1, len(aList)):
4          if aList[pos] > maxSoFar:
5              maxSoFar = aList[pos]
6
7      return maxSoFar
```

Listing 4.2 Constructing a function to return the maximum value in a list

Description

Listing 4.3 shows an alternative implementation for the `getMax` function. This time, the iteration is performed "by item" instead of "by index." In either case, each element of the list is eventually processed and compared

against the maximum item seen up to that point. A similar argument can be made to implement the `getMin` function; that is, we would simply keep track of the minimum value seen so far.

```
1  def getMaxByItem(aList):
2      maxSoFar = aList[0]
3      for item in aList[1:]:
4          if item > maxSoFar:
5              maxSoFar = item
6
7      return maxSoFar
```

Listing 4.3 Alternative function to return the maximum value in a list

Description

Try It Out

4.14 Implement the `getMin` function using iteration by index.

4.15 Implement the `getMin` function using iteration by item.

4.16 Rewrite the `getRange` function using `getMin` and `getMax`.

4.5 Central Tendency

One of the most often used measures of a collection of data is known as **central tendency**—a measure that estimates where the "typical value" of a collection will be found. Three common ways to compute central tendency are (1) the mean, (2) the median, and (3) the mode.

4.5.1 Mean

The most common measure of central tendency is the **mean** (more commonly called the average). The mean is simply the sum of all the values in the data collection divided by the number of items. As shown in **Listing 4.4**, to compute the mean, we add up the values in the collection and divide by the length. Specifically, given a list of values, we use the built-in `sum` function, shown in Table 4.6, to compute a total and the `len` function shown in Table 4.1 to compute the number of values. **Session 4.10** shows our `mean` function in action, as well as the `mean` function from the `statistics` module.

```
1 def mean(aList):
2     mean = sum(aList) / len(aList)
3     return mean
```

Listing 4.4 Computing the mean of a list

Description

```
In [1]: 1 from statsfunctions import * #our mean function is in statsfunctions.py

In [2]: 1 earthquakes = [37, 32, 46, 28, 37, 41, 31]

In [3]: 1 mean(earthquakes)          #call our mean function

Out[3]: 36.0

In [4]: 1 import statistics

In [5]: 1 statistics.mean(earthquakes) #call the statistics mean function

Out[5]: 36
```

Session 4.10 Calculating the mean

Description

HEADS UP:
Our `mean` function and the `mean` function in the `statistics` module have the same name. To call the `mean` function in the `statistics` module, we

use the dot operator with the name of the module.

Try It Out

4.17 Create a list of the number of students in each of your classes. Use the `mean` functions on that list to find the average number of students in your classes.

4.18 Replace the call to the `sum` function with an accumulation iteration that computes the total of the values in `aList`.

4.5.2 Median

The second measure of central tendency is called the **median**; it is found by locating the item that occurs in the exact middle of a collection that is in numeric order. In other words, the median is the value located at a point where half of the values lie above it and half of the values lie below it.

One way to compute the median is to put the items in order, from lowest to highest, and then find the value that falls in the middle. If there is an odd number of items, the median will be a distinct value. However, if there is an even number of items, the median will be the average of the two middle values. **Figure 4.5** shows the two cases.

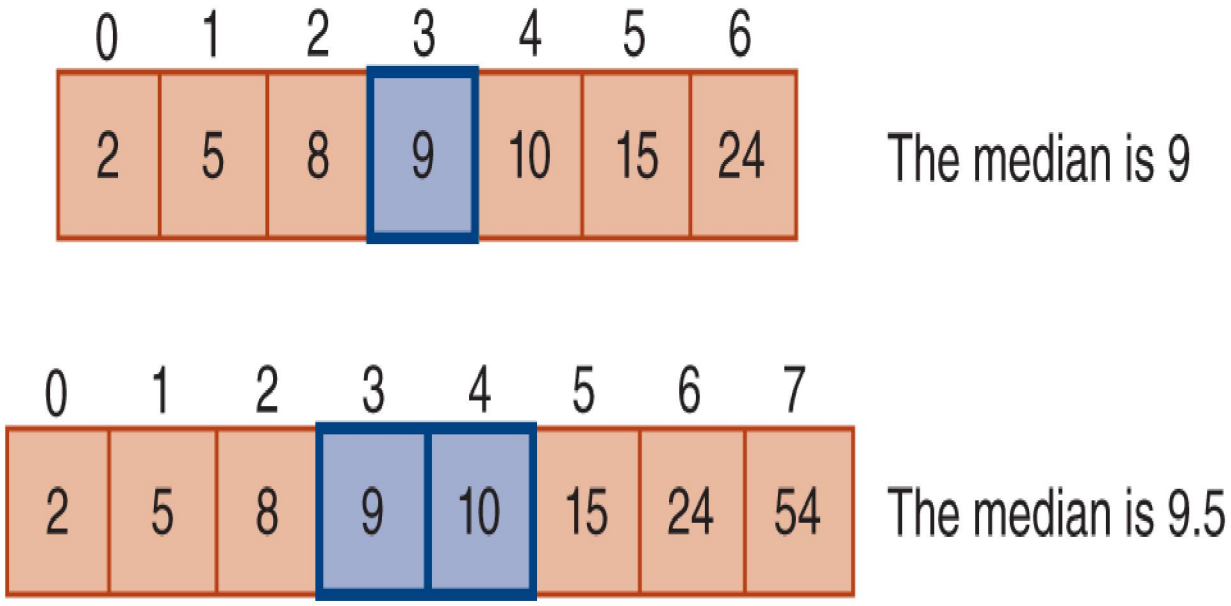


Figure 4.5 Finding the median of a list of values.

Description

The process of ordering values is often referred to as **sorting**. Sorting is a very important topic in computer science, and a number of algorithms can be used to place the values in a list of data in order. As you know, the list object has a `sort` method that allows you to put the list in order (see Table 4.3). Although we will not cover sorting techniques here, you will undoubtedly study them in your algorithms and data structures courses.

The Python statements in **Session 4.11** show some of the computations that will be required to find the median. With an odd number of values, integer division by 2 will provide the correct index into the list. In contrast, with an even number of items, integer division by 2 gives the index of the rightmost item of the middle pair.

```
In [1]: 1 aList = [24, 5, 8, 2, 9, 15, 10]  #odd number of items
```

```
In [2]: 1 aList.sort()                      #arrange list in order
```

In [3]:
```
aList
```

Out[3]: [2, 5, 8, 9, 10, 15, 24]

In [4]:
```
len(aList)                    #number of items in aList
```

Out[4]: 7

In [5]:
```
len(aList) // 2               #find middle item's index
```

Out[5]: 3

In [6]:
```
aList[len(aList) // 2]        #middle item is median
```

Out[6]: 9

In [7]:
```
bList = [2, 5, 8, 9, 10, 15, 24, 54]    #even number of sorted items
```

In [8]:
```
1 len(bList)
```

Out[8]: 8

In [9]:
```
1 len(bList) // 2                #index of right middle item
```

Out[9]: 4

In [10]:
```
1 bList[4]                       #higher of two middle items
```

Out[10]: 10

In [11]:
```
1 bList[3]                       #lower of two middle items
```

Out[11]: 9

In [12]:
```
1 median = (bList[4]+bList[3])/2  #median is average of 2 middle items
```

```
In [13]: 1 median

Out[13]: 9.5
```

Session 4.11 Computation required to find the median item

Description

Listing 4.5 shows our Python function to compute and return the median value of a list. As noted previously, the list items must be in sorted order. To do this, we simply use the `sort` method from the `list` class. Because the `sort` method modifies the list, we must be careful to make a copy of the original list and sort the copy in case the original order is needed for later processing.

```
1  def median(aList):
2      copyList = aList.copy()
3      copyList.sort()              #sort the copy
4      if len(copyList) % 2 == 0:   #even length?
5          rightMid = len(copyList) // 2
6          leftMid = rightMid - 1
7          median = (copyList[leftMid] + copyList[rightMid]) / 2
8      else:                        #odd length
9          mid = len(copyList) // 2
10         median = copyList[mid]
11     return median
```

Listing 4.5 Computing the median of a list

Description

The other important part of the solution is the decision about whether the list has an even or odd number of items. To make this choice, we can take advantage of the modulo operator (%) and check the result of dividing by 2. Even numbers will have no remainder. Line 4 in Listing 4.5 shows a simple selection statement to check for a remainder of zero.

HEADS UP:

To determine whether a value is even or odd, use the modulo operator (%) to divide the value by 2 and check whether there is a remainder. If

there is no remainder, the value is even. If there is a remainder (1), the value is odd.

Session 4.12 shows our `median` function in action. With data for seven days, our median number of earthquakes is 37. With 8 days of data, the median is 34.5, which is the average of 32 and 37. We check our results by also calling the `median` function in the `statistics` module.

```
In [1]: 1 from statsfunctions import *        #import our median function
```

```
In [2]: 1 import statistics
```

```
In [3]: 1 earthquakes7 = [37, 32, 46, 28, 37, 41, 31]      #7 items
```

```
In [4]: 1 earthquakes8 = [37, 32, 46, 28, 37, 41, 31, 29]  #8 items
```

```
In [5]: 1 median(earthquakes7)
```

```
Out[5]: 37
```

```
In [6]: 1 statistics.median(earthquakes7)
```

```
In [6]: 1 statistics.median(earthquakes7)

Out[6]: 37

In [7]: 1 median(earthquakes8)

Out[7]: 34.5

In [8]: 1 statistics.median(earthquakes8)

Out[8]: 34.5
```

Session 4.12 Using the `median` function

Description

Try It Out

4.19 Draw a reference diagram to illustrate why it is important to make a copy of a list before sorting.

4.20 Find the mean age of 10 people near you.

4.21 Find the mean age of 10 people near you, plus your professor, who is 39 years old.

4.22 Find the median age of 10 people near you.

4.23 Find the median age of 10 people near you, plus your young professor.

4.5.3 Mode

The **mode** of a data set is the most frequently occurring value. Some data sets may have more than one mode. For example, in the list

```
[1, 5, 2, 1, 1, 6, 3, 1, 5]
```

the mode is 1 because the number 1 occurs more than any other value. However, with the list

```
[1, 5, 2, 1, 1, 5, 5, 6, 3, 1, 5]
```

1 and 5 would both be considered the mode values because each occurs four times.

To compute the mode, we need to process each data item in our list and keep a count of how many times it has occurred thus far. Unfortunately, this is not quite as easy as it might sound. Since we do not know how many different values the collection holds, it is not possible to create individual variables to hold each count. Instead, we need to create a collection of counting variables and try to keep them associated with the correct data value. Python's dictionary collection will serve our purpose well.

Python Dictionaries and Tuples

To solve the mode problem, we need to associate an occurrence count with each data value. This can be done with an associative collection called a **dictionary**. Python dictionaries are collections of pairs of items where each pair consists of a key and a related value. **Figure 4.6** shows how Python dictionaries, lists, and strings are related to one another.

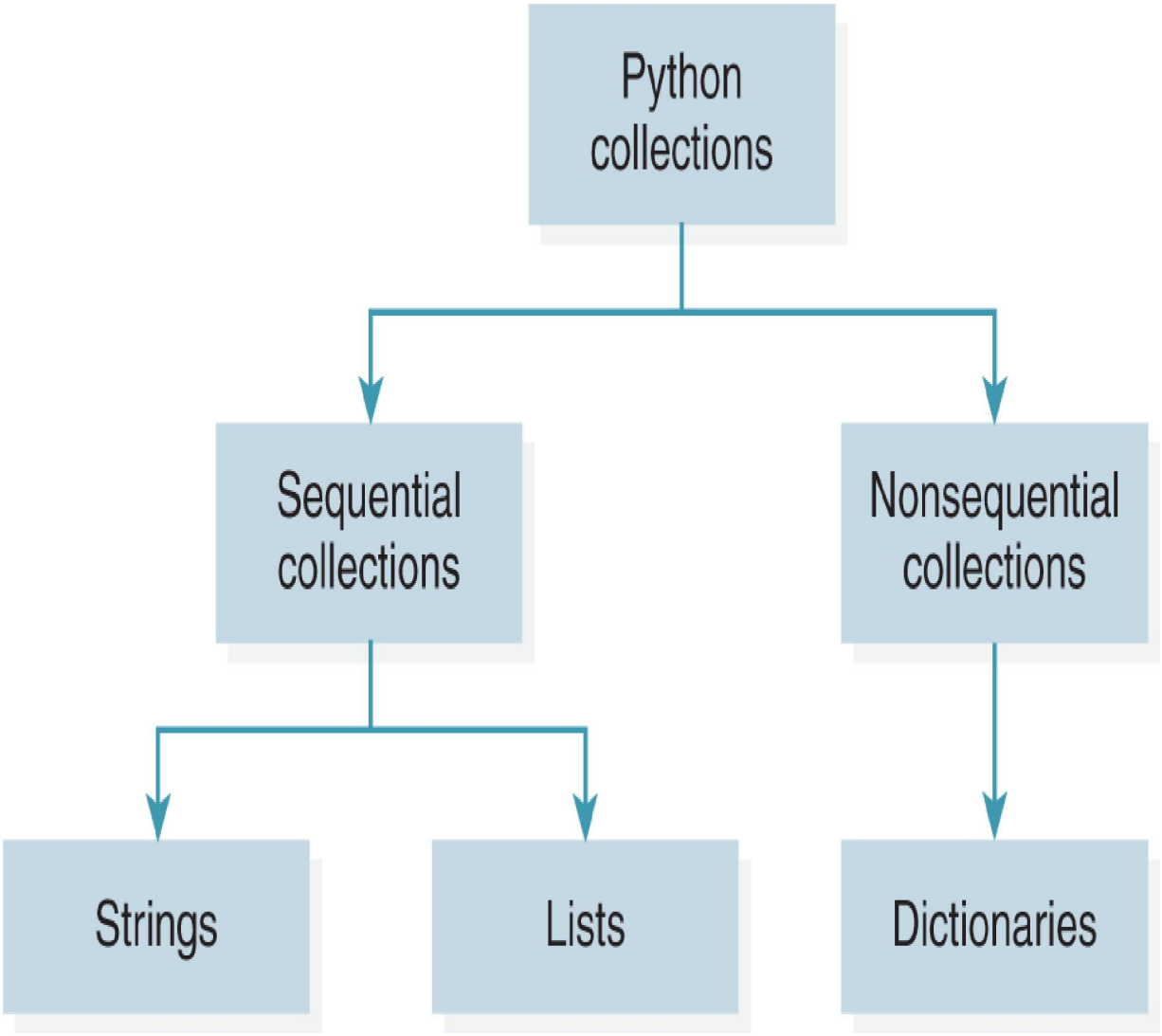


Figure 4.6 Sequential and nonsequential Python collections.

Description

In dictionary notation, a key–value pair is written as `key:value`. Dictionaries are written as comma-delimited `key:value` pairs enclosed in curly braces. The empty dictionary is written as `{}`. The `key:value` pairs appear in the order they were added to the dictionary. Dictionaries with the same `key:value` pairs in a different order are considered equal.

RECAP:

Dictionaries consist of `key:value` pairs. Dictionaries are defined using comma-delimited `key:value` pairs enclosed in curly braces.

Table 4.7 shows some operators and built-in functions that work on dictionaries. The `list` function creates a list from a **dictionary view object**, which is an object that can be used to **iterate** through a dictionary using a `for` loop. To iterate means to visit each object one at a time. The possible dictionary view objects are the keys, values, or items. The merge operator (|) creates a single new dictionary from two dictionaries. Unique entries in the second dictionary are appended to the first dictionary. Entries with the same key in both dictionaries are assigned the value in the second dictionary.

TABLE 4.7 Some Dictionary Operators and Functions in Python

Operator	Use	Explanation
`in`	`key in aDict`	Returns `True` if key is in the dictionary; `False` otherwise.
`not in`	`key not in aDict`	Returns `True` if key is not in the dictionary; `False` otherwise.
`[](index)`	`aDict[key]`	On the left-hand side of an assignment statement, adds or updates an item. On the right-hand side of an assignment statement, returns the value associated with `key`. Raises a `KeyError` if `key` is not in the dictionary.
| (merge)	`aDict | bDict`	Merges the two dictionaries. Unique entries in `bDict` are appended to `aDict`. Entries with the same key in both dictionaries are assigned the value in `bDict`.

Built-in Function	Use	Explanation
`del`	`del aDict[key]`	Statement that removes the entry from the dictionary. Raises a `KeyError` if `key` is not in the dictionary.
`list`	`list(view object)`	Creates a list from the keys, values, or items in the dictionary.
`length`	`len(aDict)`	Returns the number of pairs (items).

In **Session 4.13**, we first create a dictionary with two entries, each of which is an association between the key, which in this case is a name (a string), and a value, which in this case is an age (an integer). Notice that the pairs appear in the order they were added. One benefit of dictionaries is that they offer quick access to data values by key. Like lists, dictionaries are mutable—that is, they can be modified.

```
In [1]: 1 parents = {'David':45, 'Brenda':46}        #create dict with 2 items
```

```
In [2]: 1 parents
```

```
Out[2]: {'David': 45, 'Brenda': 46}
```

```
In [3]: 1 parents['David'] = 46                      #change value using key
```

```
In [4]: 1 parents['David']                           #access value using key
```

```
Out[4]: 46
```

```
In [5]: 1 children = {'Kelsey':19, 'Hannah':16}      #create a second dict
```

```
In [6]: 1 children['Rylea'] = 7                    #add item to dict
```

```
In [7]: 1 children
```

```
Out[7]: {'Kelsey': 19, 'Hannah': 16, 'Rylea': 7}
```

```
In [8]: 1 family = parents | children              #merge 2 dicts
```

```
In [9]: 1 family
```

```
Out[9]: {'David': 46, 'Brenda': 46, 'Kelsey': 19, 'Hannah': 16, 'Rylea': 7}
```

```
In [10]: 1 len(family)                             #len is number of pairs
```

```
Out[10]: 5
```

```
In [11]: 1 del family['David']
```

```
In [12]: 1 family

Out[12]: {'Brenda': 46, 'Kelsey': 19, 'Hannah': 16, 'Rylea': 7}

In [13]: 1 'David' in family

Out[13]: False
```

Session 4.13 Examples of adding and modifying values in a dictionary

Description

We can manipulate a dictionary in a number of ways. Accessing a dictionary value by its key looks much like a sequence index operation, except that dictionaries are not restricted to numeric indices. To access a value in the dictionary, we use the key inside the square bracket index operator. If the key is not in the dictionary, Python raises a `KeyError`.

To add a new key–value pair to an existing dictionary, we use the index operator with the new key on the left-hand side of the assignment statement and the new value on the right-hand side.

We can also change a value by simply assigning a new value to the existing key using the same syntax we used to add the value to the dictionary.

To join two dictionaries, we can use the merge operator (|). Unique items in the second dictionary are appended to the items in the first dictionary. Items with the same key in both dictionaries are assigned the value in the second dictionary.

In Session 4.13, we also show the `len` function performing the same role it had in previous collections: In this case, it counts the number of key–value pairs.

As with lists and strings, dictionaries have a set of methods that are commonly used to work with them. **Table 4.8** describes these methods, and **Session 4.14** shows the methods in action. Notice that there are two variations of the `get` method, as shown in Table 4.8. If the key is not present in the dictionary, `get` will return `None`. `None`, also referred to as the **null object**, is a Python keyword that represents an object with no value. A second, optional parameter for the `get` method can specify a return value in the case where the key is not present. The `reversed` built-in function lets you iterate through the dictionary keys, items, or values in reverse order.

TABLE 4.8 Dictionary Methods and a Built-in Python Function

Method Name	Use	Explanation
`keys`	`aDict.keys()`	Returns a `dict keys` object.
`values`	`aDict.values()`	Returns a `dict values` object.
`items`	`aDict.items()`	Returns a `dict items` object.

`reversed`	`reversed(aDict.keys())`
	`reversed(aDict.values())`
	`reversed(aDict.items())`
		Built-in function used to return an iterator for a `for` loop with keys, values, or items in reverse order.

`get`	`aDict.get(k)`	Returns the value associated with `k` or `None` if `k` is not in the dictionary.
`get`	`aDict.get(k, alt)`	Returns the value associated with `k`, or `alt` if `k` is not in the dictionary.
`clear`	`aDict.clear()`	Removes all items from the dictionary.

```
In [1]: 1 family = {'David':45, 'Brenda':46, 'Kelsey':19, 'Hannah':16, 'Rylea':7}
```

```
In [2]: 1 family
```

Out[2]: {'David': 45, 'Brenda': 46, 'Kelsey': 19, 'Hannah': 16, 'Rylea': 7}

In [3]:
```
family.keys()                    #request keys
```

Out[3]: dict_keys(['David', 'Brenda', 'Kelsey', 'Hannah', 'Rylea'])

In [4]:
```
list(family.keys())              #create list of keys
```

Out[4]: ['David', 'Brenda', 'Kelsey', 'Hannah', 'Rylea']

In [5]:
```
list(family.values())            #create list of values
```

Out[5]: [45, 46, 19, 16, 7]

In [6]:
```
list(family.items())             #create list of items
```

Out[6]: [('David', 45), ('Brenda', 46), ('Kelsey', 19), ('Hannah', 16), ('Rylea', 7)]

In [7]:
```
family.get('Lena', 'No item')          #get key with default value
```

Out[7]: 'No item'

In [8]:
```
for name in family.keys():             #iterate through keys
    print(name)

```

```
David
Brenda
Kelsey
Hannah
Rylea
```

In [9]:
```
for name in family:                    #also iterate through keys
    print(name)

```

```
David
Brenda
```

```
Kelsey
Hannah
Rylea
```

In [10]:
```
for item in reversed(family.items()):    #iterate through items
    print(item)

```

```
('Rylea', 7)
('Hannah', 16)
('Kelsey', 19)
('Brenda', 46)
('David', 45)
```

In [11]:
```
family.clear()                           #delete all items
```

In [12]:
```
family
```

Out[12]:
```
{}
```

Description

HEADS UP:
`None` is a special value in Python that represents the null object—that is, an object with no value.

The `keys` method returns a `dict_keys` object. To see the contents of a `dict_keys` object, we can convert it to a list using the built-in `list` function, described in Table 4.7. Alternatively, we can iterate over a `dict_keys` object using a `for` loop without first converting it to a list. The `values` and `items` methods work in a similar fashion.

The `items` method returns a `dict_items` object. Applying the `list` function to a `dict_items` object returns a list of key–value pairs that are stored as **tuples**. Tuples are similar to lists in that they are heterogeneous sequences of data, but a tuple is immutable. In other words, like a string, a tuple element cannot be changed. Tuples are written as comma-delimited values enclosed in parentheses. As sequences, they can use the operations that were described earlier for lists and strings.

RECAP:
Tuples are immutable sequences of heterogeneous values. Tuples are written as comma-delimited values enclosed in parentheses.

Try It Out

4.24 You have been given the following lists of students and their test scores:

```
names = ['joe', 'tom', 'barb', 'sue', 'sally']
scores = [10, 23, 13, 18, 12]
```

Description

Write a function, `makeDictionary`, that takes the two lists and returns a dictionary with the names as the keys and the scores as the values. Assign the result of `makeDictionary` to `scoreDict`, which will be used in the exercises that follow.

4.25 Using `scoreDict`, find the score for `'barb'`.

4.26 Add a score of 19 for `'john'`.

4.27 Create a sorted list of all the scores in `scoreDict`.

4.28 Calculate the average of all the scores in `scoreDict`.

4.29 Update the score for `'sally'` to be 13.

4.30 Tom has just dropped this class. Delete `'tom'` and his score from `scoreDict`.

4.31 Print a table of students and their scores with the students listed in alphabetical order.

4.32 Write a function called `getScore` that takes a name and a dictionary as parameters and returns the score for that name if it is in the dictionary. If the name is not in the dictionary, print an error message and return `-1`.

Computing the Mode

We can now return to our problem of computing the mode of a data set. Using a dictionary of counts, one for each data item, we will be able to track the information necessary to find the mode. We define our `mode` function to receive a list of data values. It will return a list of modes since it is possible that there can be more than one mode. **Listing 4.6** shows the initial implementation of the `mode` function.

```
1 def mode(aList):
2     countDict = {}
3     for item in aList:
4         if item in countDict:
5             countDict[item] = countDict[item] + 1
6         else:
7             countDict[item] = 1
8     # more code to come
```

Listing 4.6 Starting the `mode` function

Description

To count each occurrence of the data items, we use a dictionary in which the key is the data item itself and the associated value is the occurrence count, that is, the number of times the data item occurs in the parameter list. To process each item, we first check whether it already exists in the dictionary as a key. If it does, we simply increment the associated count. If the item is not in the dictionary, this is the first occurrence of this item, so we make a new entry in the dictionary with an initial count of 1.

An alternative way to implement this `if` statement would be to use the second form of the `get` method, as described earlier. In this form, if the key is present, its associated value is returned. If the key is not present, the method will allow us to include a default value to be returned. In our case, if the key is not present, we can just return a zero since it has not occurred. Either way, when we add 1, the count will be correct. Notice that this is another variation on the accumulator pattern, except that the `get` method allows us to include the initialization of the accumulator variable right in the initialization of the item:

```
countDict[item] = countDict.get(item, 0) + 1
```

To complete our mode calculation, we need to go through our dictionary, key by key, and find the largest count. We will append the key or keys associated with that count to a list of modes that will be returned at the end of the function. Fortunately, we can rely once again on Python's collections to provide methods that will do most of the work. Recall that the `values()` method returns a `dict_values` object that behaves much like a list of the values in a dictionary. By using the built-in `max` function, we can then learn the maximum count. To find all the keys with the `max` value, it is just a matter of iterating through the dictionary keys while looking for a key with a value that matches the maximum count. When we find such a key, we place that key in the mode list to be returned.

Listing 4.7 shows the complete function. Note that the `modeList` starts out empty. We can iterate through the dictionary keys by using a `for` loop, appending keys to the `modeList` as needed.

```
1  def mode(aList):
2      countDict = {}
3      for item in aList:
4          if item in countDict:
5              countDict[item] = countDict[item] + 1
6          else:
7              countDict[item] = 1
8
9      countList = countDict.values()
10     maxCount = max(countList)        #find the maximum count
11
12     modeList = []
13     for item in countDict:
14         if countDict[item] == maxCount:
15             modeList.append(item)
16     return modeList
```

Listing 4.7 Computing the mode(s) of a list

Description

Session 4.15 shows the `mode` function in use. The third example in this session uses our earthquake data `[37, 32, 46, 28, 37, 41, 31]` from

the beginning of this chapter. Each value occurs once except for 37, which occurs twice; thus, 37 is the only mode.

```
In [1]: 1 from statsfunctions import *    #import our mode function
```

```
In [2]: 1 mode([1, 1, 4, 5, 6, 2, 4, 7, 1, 4, 6, 1])
```

```
Out[2]: [1]
```

```
In [3]: 1 mode([1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6])
```

```
Out[3]: [4, 5]
```

```
In [4]: 1 earthquakes = [37, 32, 46, 28, 37, 41, 31]
```

```
In [5]: 1 mode(earthquakes)
```

```
Out[5]: [37]
```

```
In [6]: 1 import statistics     #use multimode function in statistics module
```

```
In [7]: 1 statistics.multimode([1, 1, 4, 5, 6, 2, 4, 7, 1, 4, 6, 1])

Out[7]: [1]
```

```
In [8]: 1 statistics.multimode([1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6])

Out[8]: [4, 5]
```

Session 4.15 Using the `mode` and `multimode` functions

Description

At the end of Session 4.15, we import the `statistics` module and call the `multimode` function in that package, receiving the same results as from our `mode` function.

Try It Out

4.33 If you do not have a dictionary to work with, you can still compute the mode by creating a list of integers for which the index value into the list is the key. You need a list of size `max(value)` plus one. This approach will work only when the keys are integers and positive. Note that this is a very inefficient use of memory when the data values are sparse. Implement the `mode` function using the approach described.

4.6 Frequency Distribution

As we worked through the steps required to compute the mode of a data set, we actually solved the basic statistics problem of finding the **frequency distribution**—a representation of the number of times each value occurs in the data.

4.6.1 Using a Dictionary to Compute a Frequency Table

One way to display a frequency distribution is to show a two-column table. The first column gives the data item, and the second gives the associated count. This appears to be similar to our count dictionary. In fact, the only difference is that the table will need to show the items in order. Recall that the dictionary maintains items in the order they were inserted. Our challenge, then, is to sort the data in key order.

To write this function, we can start with the `mode` function from Listing 4.7. Instead of extracting the counts and looking for the maximum, however, we will extract the keys, convert the `dict_keys` object to a list, and use the list `sort` method to provide the ordering. We can then iterate over the sorted list of keys and print a table entry consisting of the key and the associated count from the count dictionary. **Listing 4.8** shows the complete function.

```
1  def frequencyTable(aList):
2      countDict = {}
3
4      for item in aList:
5          if item in countDict:
6              countDict[item] = countDict[item] + 1
7          else:
8              countDict[item] = 1
9
10     itemList = list(countDict.keys())
11     itemList.sort()
12
13     print("ITEM", "FREQUENCY")
14
15     for item in itemList:
16         print(item, "       ", countDict[item])
```

Listing 4.8 Using a dictionary to compute the frequency distribution

Description

Note that we are not ordering the dictionary. Instead, we are creating a list of keys using the `keys` method and the `list` function and then ordering that

list. Once the list is in order, we can go back to the dictionary to find the associated counts using the keys. **Session 4.16** shows the function in action.

```
In [1]: 1 from freqTables import *
```

```
In [2]: 1 frequencyTable([3,1,1,5,3,1,2,2,3,5,3,5,4,4,6,7,6,7,5,7,8,3,8,2,3,4,1,5,6,7])
```

ITEM	FREQUENCY
1	4
2	3
3	6
4	3
5	5
6	3
7	4
8	2

Session 4.16 Demonstrating the `frequencyTable` function

Description

4.6.2 Computing a Frequency Table Without a Dictionary

It is possible to implement the frequency table function without using a dictionary, but doing so requires a bit of ingenuity in list processing. In this section, we will work through this alternative approach and introduce a useful pattern for sequential list processing.

Our plan is to start with a sorted copy of the list. Because the list is sorted, all occurrences of distinct data values will be contiguous (next to one another) in the list. As we iterate through the list, we will look for transitions between values. The difficult part is knowing when these transitions occur.

Figure 4.7 shows how we track the locations of the transitions between groups of equal data values in the list. To do so, we use two variables: `current` and `previous`. Both variables will always refer to contiguous pairs of data values. The current data value will be compared against the previous data value. If they are equal, we know that we are still processing a group of equal data values and we increment our count. However, when the current data value no longer matches the previous data value (or when we run out of items in the list), we know that we have found the end of a group. At that time, we print the previous data value and the count for that value.

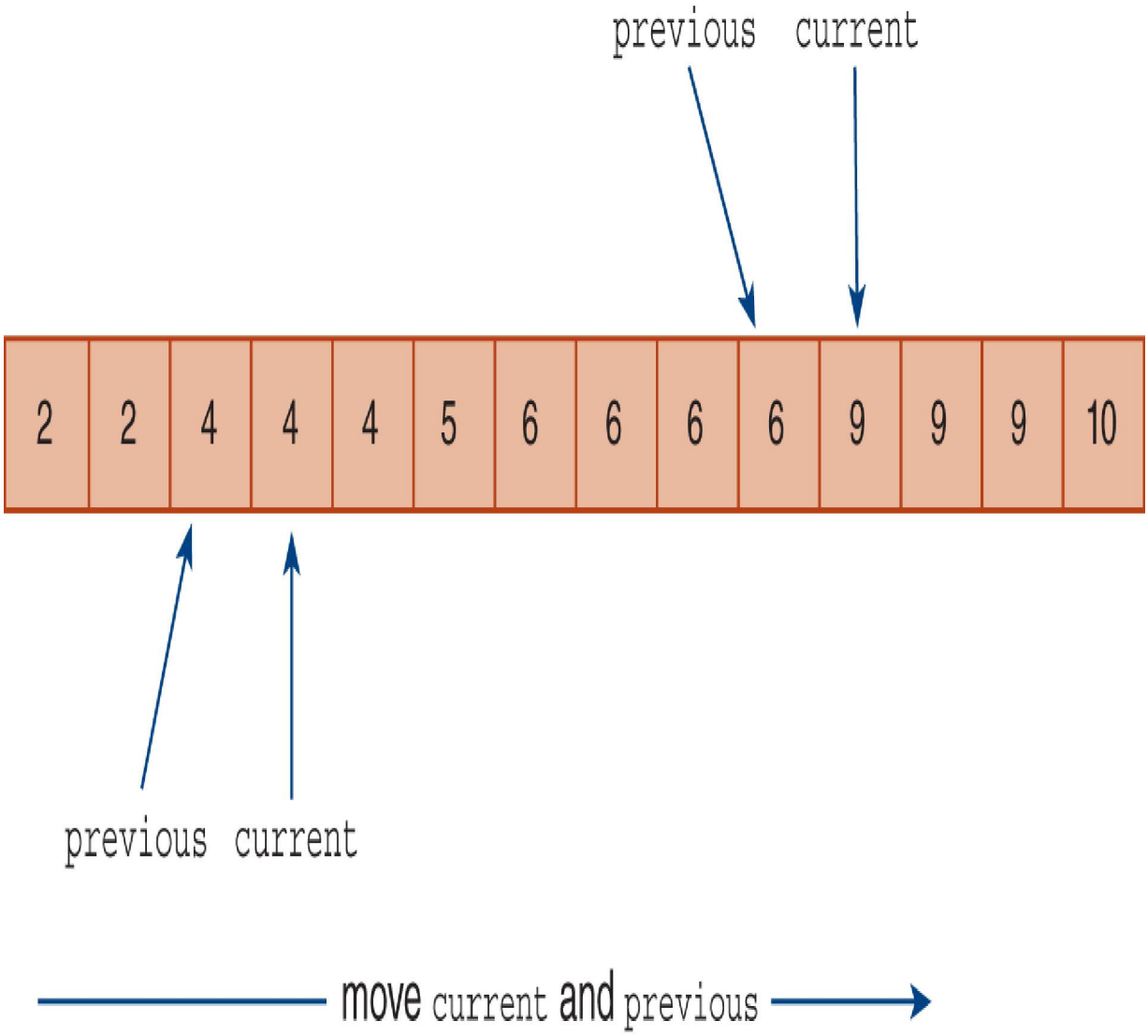


Figure 4.7 Using `previous` and `current` to locate transitions.

Description

Listing 4.9 shows the complete function. To begin, `previous` needs to be initialized so that the first data value in the list is considered to be the start of a new group of equal data values. To do this, we can simply set `previous` to the first item in the list. In addition, an accumulator variable, `groupCount`, needs to be initialized to 0 because no data values have been processed.

```
1  def frequencyTableAlt(aList):
2      print("ITEM", "FREQUENCY")       #print table headers
3
4      sortedList = aList.copy()
5      sortedList.sort()
6
7      previous = sortedList[0]
8      groupCount = 0
9      for current in sortedList:
10         if current == previous:     #same value as group
11             groupCount = groupCount + 1   #count the item
12         else:                       #new group detected
13             print(previous, "      ", groupCount)
14             previous = current
15             groupCount = 1
16     print(current, "      ", groupCount)  #print last group
```

Listing 4.9 Alternative method for computing a frequency table

Description

We can now process each data value in the list. Recall the two possibilities. On the one hand, if `current` matches `previous`, then we are still in a group of identical data values. In that case, we should increment `groupCount`. On the other hand, if `current` does not match `previous`, we know that we have located a transition. At this point, we can print a row in our table consisting of `previous` and `groupCount`. We also need to assign `current` to `previous` and initialize `groupCount` to 1 so that the next group of data values will be counted correctly.

When all the data values have been processed, the last row corresponding to the last group will not have been printed, so we can simply print out the value of `current` and `groupCount`. Try running this new function with the same list as in Session 4.16 to verify that the same results are produced.

Try It Out

4.34 Modify `frequencyTableAlt` so that it returns a list of key–count tuples.

4.35 Suppose you have a list of key–score tuples like the following:

```
[('maria',10), ('bob',8), ('maria',17),
('bob',5),...]
```

Write a function that takes such a list as a parameter and prints out a table of average scores for each person.

4.6.3 Visualizing a Frequency Distribution

Frequency distributions are best visualized through a graphical representation known as a **histogram**. To create the visualization, we can use the popular `matplotlib` package to draw a **bar chart** using vertical bars for each key. If you are using Jupyter Notebook, the `matplotlib` libraries are already installed. Appendix A has instructions for installing `matplotlib`, `if necessary`. The height of the bar will represent the frequency with

which that key appears in the data set. By visually comparing the heights of the lines, we can see the relationships among the frequencies.

We will use our dictionary approach to calculate frequencies, as we did in the `frequencyTable` function. However, instead of printing the table, we will draw a histogram. As an example, consider the following data list:

```
[3,1,1,5,3,1,2,2,3,5,3,5,4,4,6,7,6,7,5,7,8,3,8,2,3,4,1,5,6,7]
```

Figure 4.8 shows the bar chart that we would like to produce. Note that each bar represents the count (previously the second column in the frequency table).

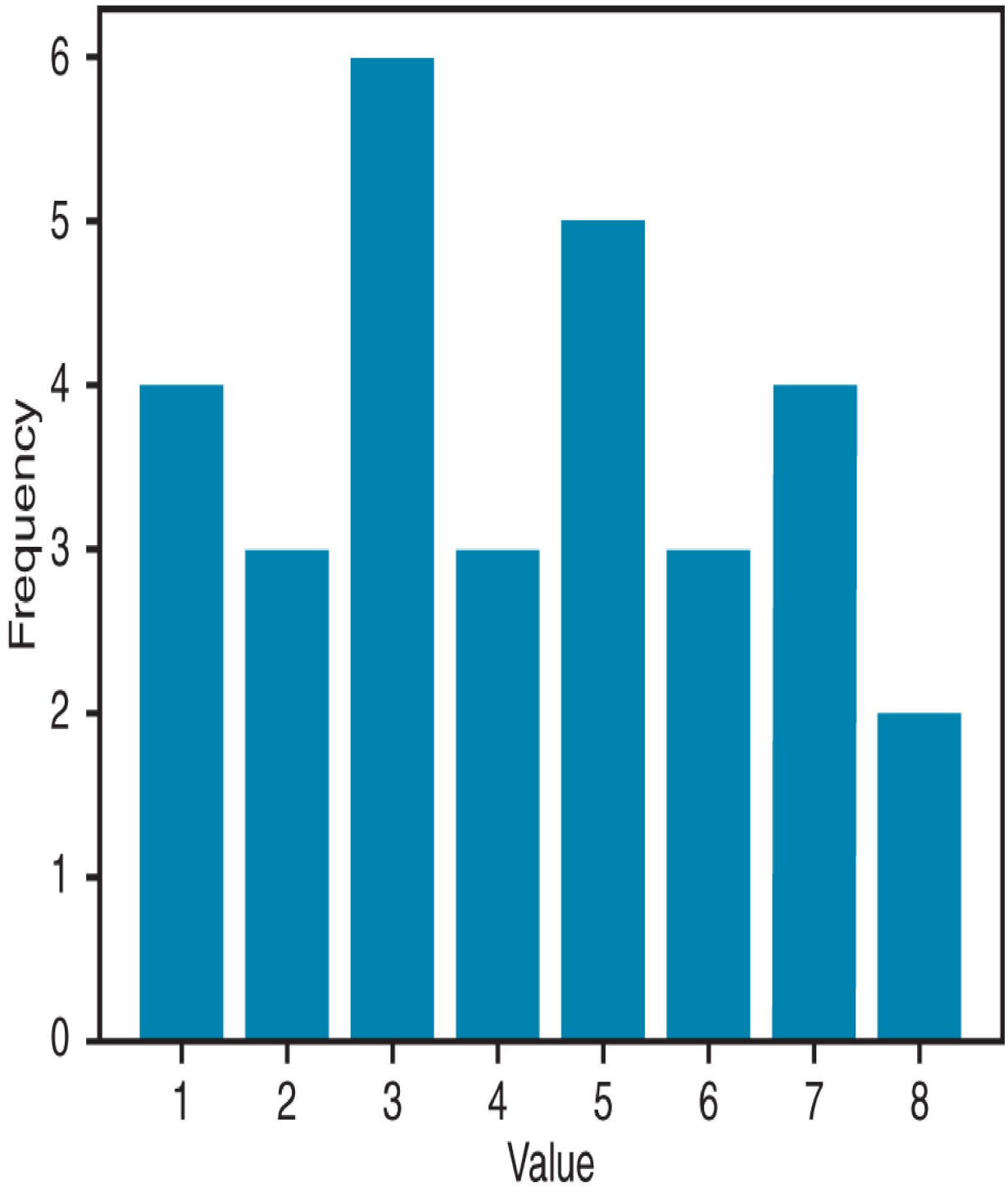


Figure 4.8 A frequency histogram.

© 2012–2023 The Matplotlib.

Description

Listing 4.10 shows the complete Python function used to create Figure 4.8. Note that when we import `matplotlib`, we use the phrase "`as plt`." This sets up a name, `plt`, that we can use as a prefix for calling `matplotlib` functions. Lines 5–11 should look familiar: They simply create our count dictionary as before. The remainder of the function does the work of defining the axes and drawing the bar chart.

```python
import matplotlib.pyplot as plt

def frequencyChart(aList):

    countDict = {}

    for item in aList:
        if item in countDict:
            countDict[item] = countDict[item] + 1
        else:
            countDict[item] = 1

    keys = list(countDict.keys())       # the x-axis
    values = list(countDict.values())  # the y-axis

    plt.bar(x = keys,height = values)
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.show()
```

Listing 4.10 Creating a histogram

Description

To create the histogram, we first need to determine what values we are using to represent the *x*-axis and what we are using to represent the bar height (the *y*-axis). We will use the dictionary keys—the values we are counting—as the *x*-axis and the frequency count of each key as the *y*-axis. Lines 13–14 extract the list of keys from the dictionary and a list of its values (the frequency counts).

Table 4.9 shows some common visualization functions of `matplotlib` for drawing various kinds of graphs. Note that the parameters listed for each function are only a subset of the possible parameters that can be set to customize a visualization. See https://matplotlib.org for information on other options available.

TABLE 4.9 Common Visualization Functions Provided by `matplotlib.pyplot`

Function Name	Use	Explanation
`bar`	`bar(x = xValue(s), height = hValues(s), color = colorName)`	Creates a bar chart given *x* values and their corresponding heights. The `color` of the bars can be set using a string representation of that color (e.g., `"red"`, `"green"`, `"blue"`). If not set, the default color is blue.
`scatter`	`scatter(x = xValue(s), y = yValue(s), color = colorName(s))`	Creates a scatterplot of points for each (`x`, `y`) coordinate. The `color` of the points can be set using a string representation of that color, as described above.

Function Name	Use	Explanation
axvline	`axvline(x, color = colorName, ls = lineStyle)`	Plots a vertical line at the provided *x* value. The `color` of the line can be set using a string representation of that color, as described above. The line style (`ls`) can be set using a string representation of the line (e.g., `"solid"`, `"dashed"`). If the line style is not set, a solid line will be generated.
axhline	`axhline(y, color = colorName, ls = lineStyle)`	Plots a horizontal line at the provided *y* value. The `color` of the line and the line style (`ls`) can be set using string representations, as described above.
xlabel	`xlabel(str)`	Sets the title of the *x*-axis to the provided string (`str`).
ylabel	`ylabel(str)`	Sets the title of the *y*-axis to the provided string (`str`).
show	`show()`	Displays the visualization.

Lines 16–19 of Listing 4.10 create the visualization using `matplotlib`. They specify the type of chart, label each axis, and then display the visualization. Line 16 specifies we are using the `bar` chart function (see Table 4.9). We will pass the list of keys as the `x` parameter, and the `height` parameter will be represented as the counts. We can independently label the *x*- and *y*-axes through the `xlabel` and `ylabel` functions, which both take a string as a parameter. Finally, once we finish setting up the chart, we invoke the `show` function to display the visualization.

Try calling the `frequencyChart` function using the following statement and verify that the histogram matches Figure 4.8:

```
frequencyChart([3,1,1,5,3,1,2,2,3,5,3,5,4,4,6,7,6,7,5,7,8,3,8,2,3,4,1,5,6,7])
```

Try It Out

4.36 Another way to compute the frequency table is to obtain a list of key–value pairs using the `items` method. This list of tuples can be sorted and printed without returning to the original dictionary. Rewrite the frequency table function using this idea.

4.37 A good programming practice is to take a step back and look at your code and to factor out the parts that you have repeated into separate functions. This is the case with our `frequencyTable` function. Modify `frequencyTable` so that it returns the dictionary of frequency counts it creates.

4.38 Now write a separate function to print the frequency table using the dictionary created by the `frequencyTable` function in the previous problem.

4.39 Modify `frequencyChart` from Listing 4.10 to use the new `frequencyTable` function.

4.7 Dispersion: Standard Deviation

We now turn to **standard deviation**—an alternative statistical measure of dispersion that shows how much the individual data items of a collection differ from the mean. The larger the spread of the data around the mean, the larger the standard deviation. Data that has a small standard deviation is clustered tightly around the mean. Data that has a large standard deviation is more dispersed.

In mathematics, the equation for computing the standard deviation for a sample of data, usually denoted by s, is as follows:

$$s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$$

where $\bar{x}$ is the mean of the data values.

Expressed in words, this equation says that we should perform the following steps:

1. Find the mean of the data values.
2. For each data value in the collection:
   a. Subtract the mean from the data value.
   b. Square this difference.
   c. Add the result to a running sum.
3. Divide the sum by $n - 1$, where n is the number of data values.
4. Compute the square root of the quotient from Step 3 to get the standard deviation.

To compute the square root, we can use the `sqrt` function in Python's `math` module. **Table 4.10** shows the `sqrt` function along with some other useful functions in the `math` module, and **Table 4.11** shows some useful constants provided in the `math` module. To see all the functions and constants in the `math` module, use the `help` command:

```
import math
help("math")
```

TABLE 4.10 Some Math Functions

Function Name	Use	Explanation
`log10`	`math.log10(x)`	Returns the base 10 logarithm of `x`.
`log2`	`math.log2(x)`	Returns the base 2 logarithm of `x`.
`dist`	`math.dist(p1, p2)`	Returns the Euclidean distance between 2 points, `p1` and `p2`, specified as a tuple.
`sqrt`	`math.sqrt(x)`	Returns the square root of `x`.

TABLE 4.11 Some Math Constants

Constant	Use	Explanation
pi	`math.pi`	Evaluates to `3.141592653589793`.
e	`math.e`	Evaluates to `2.718281828459045`.

To implement a function to compute the standard deviation, we will again expect a list of values. Because we are using the square root function, we will need to import the `math` library. Finally, Step 1 can be easily accomplished by calling on the `mean` function in the statistics module.

Now we can iterate through each value in the data list and perform the subtraction and square operations. We use the accumulator pattern to keep track of the running sum of these squares (sometimes called the "sum of squares"). **Listing 4.11** shows the complete function that returns the computed standard deviation.

```
1  def standardDev(aList):
2      import statistics as stats, math
3      theMean = stats.mean(aList)
4
5      total = 0
6      for item in aList:
7          difference = item - theMean     #subtract from mean
8          diffSq = difference ** 2        #square the difference
9          total = total + diffSq          #add to running total
10
11     sDev = math.sqrt(total / (len(aList) - 1) )
12     return sDev
```

Listing 4.11 Computing the standard deviation

Description

Session 4.17 provides two examples of finding the standard deviation. Note that the mean is the same for both data lists—namely, 12.0. However, the standard deviations are different because the values in the first list are much closer to the mean than are the values in the second list. The final example shows the standard deviation for the earthquake count data. We also send the earthquake data to the standard deviation functions of the `statistics` module, calling the first version of `stdev` that takes a data list and the second version of `stdev` that takes both the data list and the mean.

We see that our `standardDev` function and the `stdev` functions from the `statistics` module return the same values.

```
In [1]:
1 from statsfunctions import standardDev,mean
```

```
In [2]:
1 dataList = [7, 11, 9, 18, 15, 12]
2 mean(dataList)
```

```
Out[2]: 12.0
```

```
In [3]:
1 standardDev(dataList)
```

```
Out[3]: 4.0
```

```
In [4]:
1 dataList2 = [2, 10, 6, 24, 18, 12]
2 mean(dataList2)
```

```
Out[4]: 12.0
```

```
In [5]:
1 standardDev(dataList2)
```

```
Out[5]: 8.0
```

```
In [6]: 1 earthquakes = [37, 32, 46, 28, 37, 41, 31]
        2 standardDev(earthquakes)
```

```
Out[6]: 6.2182527020592095
```

```
In [7]: 1 import statistics
        2 statistics.stdev(earthquakes)
```

```
Out[7]: 6.2182527020592095
```

```
In [8]: 1 statistics.stdev(earthquakes, statistics.mean(earthquakes))
```

```
Out[8]: 6.2182527020592095
```

Session 4.17 Using the `standardDev` and `stdev` functions

Description

Try It Out

4.40 Modify the frequency chart function to include a red vertical line representing the mean. *Hint:* You can layer `matplotlib` plots together by simply calling them sequentially before invoking the `show` method. For example, adding a horizontal line to a scatterplot would be implemented as follows:

```
plt.scatter(x = xData, y = yData)
plt.axhline(y = theMean)
plt.show()
```

Description

4.41 Modify the frequency chart function to draw a line representing plus or minus 1 standard deviation from the mean. Make these lines dashed.

4.42 Using `random.uniform`, generate a list of 1,000 numbers in the range 0–50. Graph the frequency distribution, along with the mean and standard deviation.

4.43 Using `random.gauss`, generate a list of 1,000 numbers in the range 0–50. Graph the frequency distribution, along with the mean and standard deviation.

4.8 Summary

This chapter introduced three types of Python collections: lists, tuples, and dictionaries. Together with strings, these four data types provide us with a powerful set of tools for solving problems. To investigate the use of these collections, we developed a set of algorithms for performing basic statistical analysis. In particular, we implemented functions to compute the following:

- Dispersion (simple range and standard deviation)
- Central tendency (mean, median, and mode)
- Frequency distribution

We introduced the functions of the `statistics` module, which also perform these operations. We used `matplotlib` to create a graphical representation of a frequency histogram. Finally, we used functions of the `math` module to calculate the standard deviation.

Key Terms

bar chart
central tendency
collections
data
delimiter
dictionary
dictionary view object
dispersion
empty list
frequency distribution
heterogeneous collections
histogram
homogeneous collections
immutable collections
iterate
list
mean
median
mode
mutable collections
null object
ranges
sequence
sorting

standard deviation
tuples
whitespace

Programming Exercises

4.1 We used `matplotlib`'s `bar chart` method to visualize histograms. However, `matplotlib` has a histogram method (`hist`) that we could have used instead. Read the documentation at https://matplotlib.org to learn how to use the `hist` method instead of the `bar` method used in this chapter.

4.2 Given a list of points of the form $[(x_1, y_1),(x_2, y_2),\ldots,(x_n, y_n)]$, write a function called `plotRegression` that accepts a list as a parameter and uses `matplotlib`'s `scatter` method to plot those points, and `matplotlib`'s `plot` method to plot the best-fit line according to the following formulas:

$$y = \bar{y} + m\left(x - \bar{x}\right)$$

$$m = \frac{\sum x_i y_i - n\overline{xy}}{\sum x_i^2 - n\bar{x}^2}$$

where $\bar{x}$ is the mean of the x values, $\bar{y}$ is the mean of the y values, and n is the number of points. The Greek letter Σ represents the sum operation. For example, Σx_i indicates the addition of all the x values.

Your program should analyze the point data, plot the points using `matplotlib`'s `scatter` method, and use the `plot` method to draw the best-fit line. You should use a different color than the points for the best-fit line.

4.3 Have you ever tried to talk like a pirate? Write a function, `toPirate`, that takes an English sentence in the form of a string as a parameter. The `toPirate` function should return a string containing the pirate translation of your sentence. Use the following table to construct a translation dictionary. You can also consult an online website for more pirate phrases.

	English	Pirate Translation
Greetings		
	hello	avast
	excuse	arrr
People		
	sir, boy, man	matey
	madam	proud beauty
	officer	foul blaggart
Articles		
	the	th’
	my	me
	your	yer
	is	be
	are	be
Places		
	kitchen	galley
	hotel	fleabag inn

4.4 Now try an even more challenging exercise: Translate two-word phrases into their pirate equivalents. For example, the two-word phrase “excuse me” would be translated into simply “arrrr.”

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock

CHAPTER 5

Bigger Data: File I/O

String Formatting; Reading Data in Text, CSV, and JSON Formats; Using the `pandas` Library

5.1 OBJECTIVES

- To read and store large data sets as text files
- To introduce string formatting
- To access data sources in CSV and JSON formats
- To use the `pandas` library to read and manipulate data

5.2 Using Files for Large Data Sets

All of the examples in the *Introducing the Python Collections* chapter used small data sets so we could concentrate on developing and testing the statistical methods needed to analyze the data. We assumed our data items were stored in lists, and we constructed our functions to process those lists accordingly. Now that we have those functions in place, we can turn our attention to using statistical tools to describe larger data sets.

We have already seen that Python provides some powerful collections in which to store and manipulate data. However, as the amount of data gets larger, having the user enter the data interactively requires a substantial amount of effort. Instead, large data sets are usually stored in files. We can read such data from the file and fill our collections for processing.

5.2.1 Text Files

One popular format is a **text file**—that is, a file containing characters. For example, the Python programs we write are stored as text files. We can create text files in a number of ways—storing data entered via a keyboard, for example, or downloading text from a website. Regardless of how the file is created, to read the file, we must know the file's format—that is, how the data is stored in the file. For example, the data items could be organized as one item per line, or as a series of data items separated by spaces, or in some other format. Once we know the format of the file, we can use Python methods to read and manipulate the data in the file.

In Python, we must **open a file** before we can use it and **close a file** when we are finished with it. As you might expect, once a file has been opened, it becomes a Python object just like all other data. **Table 5.1** shows the functions that can be used to open and close files.

TABLE 5.1 Opening and Closing Files in Python

Function or Method Name	Use	Explanation
`open`	`fileRef = open(filename, mode, encoding = 'e')`	Built-in function. Opens a file called `filename` and returns a reference to a file object. If the file does not exist, raises an `OSError`. Mode and encoding are optional parameters. The mode can be: `'r'`: open the file for reading. If no mode is specified, r is the default. `'w'`: open the file for writing. If the file exists, it will be overwritten. `'a'`: open the file for writing. If the file exists, the new data will be appended to the end. `encoding`, if used, is typically `'utf-8'`, which is the encoding used for Unicode characters.
`close`	`fileRef.close()`	Method that signals that file use is complete. Memory or other resources associated with the file are released.

As an example, suppose we have a text file named `rainfall.txt` that contains the data shown in **Figure 5.1**, which represents the total annual rainfall (in inches) for 25 towns in Iowa. The first item on each line is the rain gauge location, usually a town name, and the second is the rainfall amount. The two values are separated by a space.

```
Akron 25.81
Albia 37.65
Algona 30.69
Allison 33.64
```

```
Alton 27.43
AmesW 34.07
AmesSE 33.95
Anamosa 35.33
Ankeny 33.38
Atlantic 34.77
Audubon 33.41
Beaconsfield 35.27
Bedford 36.35
BellePlaine 35.81
Bellevue 34.35
Blockton 36.28
Bloomfield 38.02
Boone 36.30
Brighton 33.59
Britt 31.54
Buckeye 33.66
```

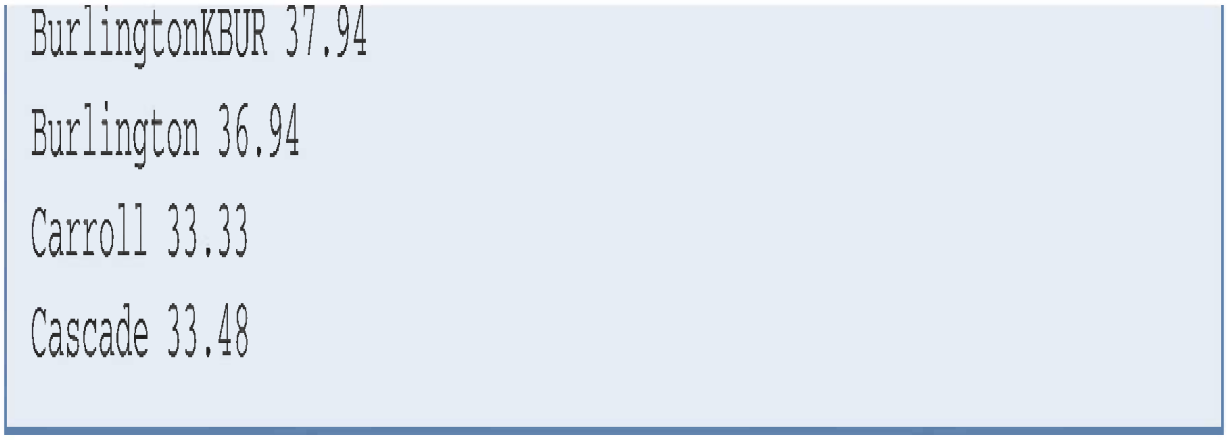


Figure 5.1 The contents of the file `rainfall.txt`.

Description

To open this file, we call the `open` function. The variable `fileRef` then holds a reference to the file object returned by `open`. When we are finished with the file, we can close it by using the `close` method. After the file is closed, any further attempts to use `fileRef` will result in an error. These two operations are shown in **Session 5.1** as option 1.

```
1 #option 1: open, then close file
2 fileRef = open("rainfall.txt", "r")
3
4 #read and process the data in the file
5
6 fileRef.close()
```

```
1 #option 2: use with/as block
2 with open("rainfall.txt", "r") as fileRef:
3
4     #read and process data in the file
5     #file is automatically closed when block finishes
```

Session 5.1 Opening and closing a file

Description

If we do not close the file after we have finished processing its data, Python will eventually close it. The benefit of formally closing the file is that we release any resources, such as memory, that have been assigned to the file.

Another option for opening a file is to call the `open` function using the following syntax, which is also shown as option 2 in Session 5.1:

```
with open(filename, mode) as fileRef:
```

The `open` function returns the file object, `fileRef`, which then becomes our reference to the file. This syntax has two significant differences: The data processing is performed in a block, and the file is automatically closed when the block finishes. This approach is the preferred method for opening a file, so we will use it when reading and writing files.

RECAP:

Using the syntax `with/as` is the preferred approach for opening and automatically closing files:

```
with open(filename, mode) as fileRef:
```

5.2.2 Iterating over Lines in a File

We now use this file as input in a program that will do some data processing. In the program, we will read each line of the file and print it with some additional text. Because text files consist of sequences of lines of text, we can use the `for` loop to iterate through each line in the file.

A line in a file is defined as a sequence of characters up to and including a special character called the **newline** character. If you evaluate a string that contains a newline character, you will see the character represented as `\n`. If you print a string that contains a newline character, you will not see the `\n`; instead, you will see that printing continues on the next line. When you are typing a Python program and you press the enter or return key on your keyboard, the editor inserts a newline character into your text at that point.

HEADS UP:

The newline character terminates a line of text in a file. The newline character is represented in a string by the character `\n`.

As the `for` loop iterates through each line of the file, the loop variable will contain the current line of the file as a string of characters. The general pattern for processing each line of a text file is as follows:

```
for line in inFile:
    statement1
    statement2
    ...
```

Description

To process our rainfall data, we will use a `for` loop to iterate over the lines of the file. Because the city and rainfall amount on each line are separated by a space, we can use the `split` method to break each line into a list containing the city and the rainfall amount. We can then use these values to construct a simple sentence, as shown in **Session 5.2**.

```
1 with open("rainfall.txt", "r") as rainFile:
2     for aLine in rainFile:
3         values = aLine.split()
4         print(values[0], "had", values[1], "inches of rain.")
5
```

```
Akron had 25.81 inches of rain.
```

Albia had 37.65 inches of rain.
Algona had 30.69 inches of rain.
Allison had 33.64 inches of rain.
Alton had 27.43 inches of rain.
AmesW had 34.07 inches of rain.
AmesSE had 33.95 inches of rain.
Anamosa had 35.33 inches of rain.
Ankeny had 33.38 inches of rain.
Atlantic had 34.77 inches of rain.
Audubon had 33.41 inches of rain.
Beaconsfield had 35.27 inches of rain.
Bedford had 36.35 inches of rain.
BellePlaine had 35.81 inches of rain.
Bellevue had 34.35 inches of rain.
Blockton had 36.28 inches of rain.
Bloomfield had 38.02 inches of rain.
Boone had 36.30 inches of rain.
Brighton had 33.59 inches of rain.
Britt had 31.54 inches of rain.
Buckeye had 33.66 inches of rain.

```
BurlingtonKBUR had 37.94 inches of rain.
Burlington had 36.94 inches of rain.
Carroll had 33.33 inches of rain.
Cascade had 33.48 inches of rain.
```

Session 5.2 Simple program to read rainfall data from a file

Description

5.2.3 Writing a File

Let's think about another example of file processing: converting our rainfall data from inches to centimeters. To do this, we will read the file contents as before, but instead of printing a message, we will do some computation and then write the results to another file. The new file will need to be opened for writing. To write a line to a file, we use the `write` method, shown in **Table 5.2**.

TABLE 5.2 The `write` Method

Method Name	Use	Explanation
`write`	`nChars = fileVar.write(string)`	Writes a string to a file that is open for writing using the file object returned from the `open` function. Returns the number of characters written.

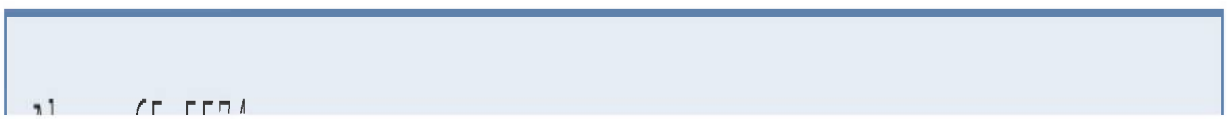
Listing 5.1 shows the Python program to perform the inches-to-centimeters conversion. The new file will be called `rainfallInCM.txt`. A `for` loop is again used to iterate through the input file. Each line of the file is split, and the rainfall value in inches is converted to the equivalent value in centimeters (lines 7–8).

```
1  with open("rainfall.txt", "r") as rainFile:
2      with open("rainFallInCM.txt", "w") as outFile:
3          for aLine in rainFile:
4              values = aLine.split()
5
6              inches = float(values[1])
7              cm = 2.54 * inches              #convert to CMs
8
9              nChars = outFile.write(values[0] + " "  \
10                                    + str(cm) + "\n")
```

Listing 5.1 Writing data into a new file

Description

The `write` statement on lines 9–10 does all the work to create a new line in the output file. We use string concatenation to build up the output line piece by piece. The first piece, `values[0]`, is the city name. We then add a space to separate the city from the rainfall amount. Next, the floating-point value called `cm` is converted to a string using the `str` function. Finally, the entire line is completed by adding a newline character (`\n`). The `write` method returns the number of characters that were written in the line, but because we do not process that number, we simply assign the return value to a variable `nChars`, which we ignore. **Figure 5.2** shows the contents of the newly created file.



```
Akron 65.5574
Albia 95.631
Algona 77.9526
Allison 85.4456
Alton 69.6722
AmesW 86.5378
AmesSE 86.233
Anamosa 89.73819999999999
Ankeny 84.7852
Atlantic 88.31580000000001
Audubon 84.86139999999999
Beaconsfield 89.5858
Bedford 92.32900000000001
BellePlaine 90.9574
Bellevue 87.24900000000001
Blockton 92.1512
Bloomfield 96.5708
Boone 92.202
```

```
Brighton 85.3186
Britt 80.1116
Buckeye 85.4964
BurlingtonKBUR 96.3676
Burlington 93.82759999999999
Carroll 84.6582
Cascade 85.0392
```

Figure 5.2 The contents of the new text file, `rainfallInCM.txt`.

Description

5.2.4 String Formatting

As you can see on lines 9–10 of Listing 5.1, converting values to strings and concatenating these strings together can be tedious. Fortunately, Python provides a better alternative: **formatted strings**. Instead of concatenating strings, we build a template. Words or spaces that will remain constant are combined with placeholders for variables to be inserted into the string. For example, the statement

```
print(values[0], "had", values[1], "inches of rain.")
```

outputs the words `"had"` and `"inches of rain."` every time, but the city name and the amount are different for each line printed.

Using a formatted string, we can write the previous statement as

```
print( "{0} had {1:2.2f} inches of rain.".format(city, rain) )
```

A formatted string uses the string `format` method shown in **Table 5.3**. The curly braces `{}` serve as placeholders for values to be inserted into the string. The characters within curly braces specify the values and associated formats to substitute for the placeholders. The parameters of the `format` method specify the actual values to be substituted.

TABLE 5.3 The String `format` Method

Method Use	Explanation
`str.format(replacementField, ...)`	Substitutes replacement fields for placeholders in the string `str` using the formats specified within the placeholder.

Note that the number of placeholders in the string corresponds to the number of replacement fields sent as parameters. By default, the replacement fields are substituted for the placeholders in the same order as the replacement field parameters are given to the `format` method. To avoid confusion, it is recommended that you insert the number of the replacement field in each placeholder, as shown by `{0}` and `{1}` in the previous example. The replacement fields can be specified as a tuple or a dictionary. If a dictionary is provided, then the dictionary name should be preceded by ** and the keys will be used to identify the value to be replaced.

Following the replacement field number (if any), the placeholder can contain a colon followed by a conversion specification. The conversion specification consists of an alignment, width, and type. **Table 5.4** summarizes some common type specifications. In the first placeholder in the preceding example, no conversion specification is given, because string is the default conversion type. The second placeholder, `{1:2.2f}`, specifies that the second parameter should be formatted as a floating-point number (`f`) with 2 places to the right of the decimal point.

TABLE 5.4 Common Type Specifications

Type of Replacement Field	Type Conversion Character	Output Format
String	`s`	String. This is the default format.
Integer	`c`	Character. Converts an integer to its Unicode equivalent.
	`d`	Decimal integer. This is the default for a number.
	`n`	Number. Same as decimal integer.
Floating-point	`e` or `E`	Scientific notation with a default precision of 6, as `m.ddddde+/-xx` or `m.dddddE+/-xx`.
	`f` or `F`	Fixed-point with default precision of 6 digits after the decimal point, as `m.dddddd`.
	`g`	General format. Uses scientific or fixed-point depending on the magnitude of the number.
	`n`	Same as `g`.
	`%`	Percentage. Multiplies the number by 100, then displays as `f` format and appends a percent sign.

To these format characters, we can add other modifier characters to specify numeric precision and output width and alignment. **Table 5.5** lists commonly used modifier characters.

TABLE 5.5 Common Formatting Modifiers

Modifier Type	Modifier Character	Output Format
Alignment	`<`	Left-align value in its space. This is the default for strings.
	`>`	Right-align value in its space. This is the default for numbers.
	`^`	Center the value within its space.
Width	`w`	Value will use the space of `w` characters. The default is to use the minimum space required to display the value.
Precision	`.n`	Display `n` numbers after the decimal point.

Session 5.3 shows some examples of formatted strings in use.

In [6]:
```
#format item price to 2 decimal points
item = 'a dozen eggs'
itemPrice = 2.4
print("The price of {0} is ${1:.2f}".format(item, itemPrice))
```

```
The price of a dozen eggs is $2.40
```

In [8]:
```
#format dict values using keys, note **
myDict = {'name':'candy bar', 'price':95}
print("The {name} costs {price} cents.".format(**myDict))
```

```
The candy bar costs 95 cents.
```

In [10]:
```
#align center, right, and left
print("{0:^40s}".format("This text is centered."))
print("{0:>40s}".format("This text is right justified."))
print("{0:<40s}".format("This text is left justified."))
```

```
                This text is centered.
                This text is right justified.
```

```
This text is left justified.
```

```
In [11]: 1 #format percentage
         2 quizGrade = 7.5
         3 totalPoints = 12
         4 print("{0:.1F} is {1:.2%} of {2:d} total points.".format(
         5     quizGrade, quizGrade / totalPoints, totalPoints))
         6
```

```
7.5 is 62.50% of 12 total points.
```

Session 5.3 Demonstrating string formatting

Description

RECAP:
The format for a placeholder is

```
{position:alignmentWidthPrecisionType}
```

Try It Out

5.1 Write a program to read the `rainfall.txt` file and then write out a new file called `rainfallFmt.txt`. The new file should format

each line so that the city is right justified in a field that is 25 characters wide, and the rainfall data should be printed in a field that is 5 characters wide with 1 digit to the right of the decimal point.

5.2 Write a function that writes a temperature conversion table called `tempConv.txt`. The table should include temperatures from −300 to 212 degrees Fahrenheit and their Celsius equivalents, presented in two columns with appropriate headings. Each column should be 10 characters wide, and each temperature should have 3 digits to the right of the decimal point.

5.2.5 Alternative File-Reading Methods

In addition to the `for` loop, Python provides three methods to read data from the input file. The `readline` method reads a specified number of characters or up to one line from the file and returns it as a string. The string returned by `readline` will contain the newline character at the end if an entire line was read. This method returns the empty string when it reaches the end of the file.

The `readlines` method returns the contents of the entire file as a list of strings, where each item in the list represents one line of the file; the newline character appears at the end of each item in the list.

It is also possible to read the entire file into a single string with `read`. **Table 5.6** summarizes these methods, and **Session 5.4** shows them in action.

TABLE 5.6 Methods for Reading Files in Python

Method Name	Use	Explanation
`read(n)`	`fileRef.read()`	Reads and returns a string of `n` characters, or the entire file as a single string if `n` is not provided.
`readline(n)`	`fileRef.readline()`	Returns the next line of the file with all text up to and including the newline character. If `n` is provided as a parameter, then only `n` characters will be returned. Returns the empty string when the end of the file is reached.

Method Name	Use	Explanation
`readlines(n)`	`fileRef.readlines()`	Returns a list of `n` strings, each representing a single line of the file. If `n` is not provided, then all lines of the file are returned.

In [1]:
```
1 with open('rainfall.txt', 'r') as inFile:
2     aLine = inFile.readline()       #read one line
```

In [2]:
```
1 aLine
```

Out[2]:
```
'Akron 25.81\n'
```

In [3]:
```
1 with open('rainfall.txt', 'r') as inFile:
2     lineList = inFile.readlines()   #read all lines as a list
```

In [5]:
```
1 print(lineList)
```

```
['Akron 25.81\n', 'Albia 37.65\n', 'Algona 30.69\n', 'Allison
33.64\n', 'Alton 27.43\n', 'AmesW 34.07\n', 'AmesSE 33.95\n',
'Anamosa 35.33\n', 'Ankeny 33.38\n', 'Atlantic 34.77\n', 'Audubon
```

```
33.41\n', 'Beaconsfield 35.27\n', 'Bedford 36.35\n', 'BellePlaine
35.81\n', 'Bellevue 34.35\n', 'Blockt on 36.28\n', 'Bloomfield
38.02\n', 'Boone 36.30\n', 'Brighton 33.59\n', 'Britt 31.54\n',
'Buckeye 33.66\n', 'BurlingtonKBUR 37.94\n', 'Burlington 36.94\n',
'Carroll 33.33\n', 'Cascade 33.48']
```

In [6]:
```
1 with open('rainfall.txt', 'r') as inFile:
2     fileString = inFile.read()      #read entire file as a string
```

In [7]:
```
1 fileString
```

Out[7]:
```
'Akron 25.81\nAlbia 37.65\nAlgona 30.69\nAllison 33.64\nAlton 27.43
\nAmesW 34.07\nAmesSE 33.95\nAnamosa 35.33\nAnkeny 33.38\nAtlantic
34.77\nAudubon 33.41\nBeaconsfield 35.27\nBedford 36.35\nBelle
Plaine 35.81\nBellevue 34.35\nBlockton 36.28\nBloomfield 38.02
\nBoone 36.30\nBrighton 33.59\nBritt 31.54\nBuckeye 33.66
\nBurlingtonKBUR 37.94\nBurlington 36.94\nCarroll 33.33\nCascade
33.48'
```

Session 5.4 Using more `read` methods

Description

Note that we need to reopen the file before each read so that we start from the beginning of the file. Each file has a marker, called a **file pointer**, that denotes the current read position in the file. Whenever one of the read methods is called, the file pointer moves to the character immediately following the last character returned. In the case of `readline`, the file pointer moves to the first character of the next line in the file. In the case of `read` or `readlines`, the file pointer moves to the end of the file or to the end of the data read. Reopening the file sets the file pointer to the beginning of the file.

HEADS UP:

The string returned by the `readline` method includes the terminating newline character (`\n`). Each list item returned by the `readlines` method also includes the terminating newline character. The string returned by the `read` method has the newline characters embedded in the string.

Try It Out

5.3 Open a file during a Python session. Call the `readline` method twice on that file, then call the `readlines` method. Which lines does the list returned by `readlines` include?

5.4 Open the file in Exercise 5.3 again, but call `readlines` immediately. Compare the lines returned in this exercise with the previous one.

5.5 Write a program that reads in the contents of a file and writes a new file in which all the characters are in uppercase.

5.6 Write a program that reads in a file and then prints out the number of lines, words, and characters in the file.

5.7 Write a program that creates a file with a concordance—an index that tells you which line of the file each word appears on. If a word appears on more than one line, the concordance will show you

all the lines containing that word. *Hint:* Use a dictionary keyed by each word to solve this problem.

5.3 Reading Data from the Internet

Earlier, we used statistical functions on a week's worth of data about earthquakes from the U.S. Geological Survey (USGS) website (usgs.gov). One advantage of using such a small set of values was that we could easily demonstrate the concepts of mean, median, mode, and standard deviation. In particular, we used summary data as our data set: the number of earthquakes per day for a seven-day period. The real power of statistics and other methods of analysis, however, can be realized on larger raw (not summarized) collections of data.

Internet sites often provide large sets of data that can be downloaded for analysis. These data sets are usually made available in several formats: text files in **CSV (comma-separated values)** format or in **JSON (JavaScript Object Notation)**, which is a standardized data transfer format with self-identifying tags.

The USGS makes raw data available for earthquakes that occur around the world. It is also possible to filter the data to include only earthquakes meeting or exceeding minimum magnitudes or only earthquakes that occurred within a specific time period. The data about each earthquake includes the date and time, location, magnitude, depth of the quake, and other identifying data.

5.3.1 Using CSV Files

One format that the USGS provides for downloading earthquake data is CSV. This format is ideal for importing data into a spreadsheet, but it is also useful for gaining programmatic access to the data.

In CSV format, the first line of the file includes **metadata**, which is descriptive information about the data. For CSV files, the metadata specifies the labels and relative positions for each piece of data. The remaining lines in a CSV file contain the actual data. The USGS file provides one line of data for each earthquake. As the name CSV implies, data fields are separated, or delineated, by a comma. The positions of the labels in the first line correspond to the positions of the data in each succeeding line. Thus, the

first label corresponds to the first data field in the subsequent lines of the CSV file; the second label corresponds to the second data field, and so on. Optionally, all data fields can be surrounded by quotes, but at minimum, data containing spaces or commas are surrounded by quotes.

For example, **Figure 5.3** shows the first few lines of a downloaded CSV file with raw data about earthquakes with magnitudes of 4.5 and greater for November 2022. Information such as this is downloadable on the Real-Time Feeds page on the USGS website. This file contains information for 500 earthquakes. The first line, colored in blue in the figure, provides the labels and relative positions of the data in the succeeding lines. Each succeeding line has information about one earthquake. In Figure 5.3, we have shown the metadata line plus data for the first three earthquakes in the file. As the metadata indicates, the first value for each earthquake is the date and time followed by a comma, then the latitude followed by a comma, then the longitude followed by a comma, and so on. Although it appears in Figure 5.3 that the lines break at various places in the data, the file actually has newlines only at the end of the label line and after each earthquake's data. Note that the "`place`" field is enclosed by quotes because it could contain embedded commas.

```
time,latitude,longitude,depth,mag,magType,nst,gap,dmin,rms,net,id,updated
,place,type,horizontalError,depthError,magError,magNst,status,
locationSource,magSource
2022-11-
30T22:30:36.483Z,5.5812,127.1411,150.253,4.6,mb,103,113,2.144,0.54,us,
us6000j5w0,2022-12-11T03:42:29.040Z,"Philippine Islands
region",earthquake, 6.82,5.018,0.06,81,reviewed,us,us
2022-11-30T21:07:22.712Z,-35.5874,-
110.1823,10,4.7,mb,18,61,31.304,0.4,us,us6000j73b,2022-12-
10T06:03:14.040Z,"southern East Pacific
Rise",earthquake,15.48,1.306,0.166,11,reviewed,us,us
2022-11-
30T18:10:30.406Z,40.2398,142.1853,58.173,4.5,mb,24,154,1.016,0.85,us,
us6000j5ui,2022-12-05T17:57:13.040Z,"64 km ESE of Hachinohe,
Japan",earthquake,6.65,8.811,0.172,10,reviewed,us,us
```

Figure 5.3 The first four lines of the `earthquakes.csv` file.

Description

Fortunately, Python's `csv` module makes it easy to read and process CSV files. The module provides a `reader` function, shown in **Table 5.7**, that can be used to interpret each line of the CSV file. It is also possible to define the field separator, line terminator, and other format elements of the file. For our file, we do not need to define any special formatting, however, because by

default, the `reader` function assumes correctly that our file is in a Microsoft® Excel format. The `reader` function returns an **iterator**, which is an object that can return one item at a time from a group. In our case, the iterator will return one line at a time from the file as a list of values for one earthquake.

TABLE 5.7 The `reader` Function of the `csv` Module

Method Name	Explanation
`iterator = reader(file)`	Returns an iterator object for moving through the rows in the file. Each row is returned as a list containing its fields.

Session 5.5 shows how we can use the `csv` module and the `reader` function to read the file. We open the file as usual, then pass the file to the `reader` function. We then can use the `csvReader` iterator in a `for` loop. The `reader` function converts each line from CSV format to a list of strings. Because the output is so long, Jupyter Notebook creates a box with a scroll bar for viewing the file.

```
In [1]: 1 import csv
```

```
In [2]: 1 with open("earthquakes.csv", "r") as inFile:
        2     csvReader = csv.reader(inFile)   #pass file to csv reader
        3     for line in csvReader:           #read each line
        4         print(line)
        5
```

['time', 'latitude', 'longitude', 'depth', 'mag', 'magType', 'nst', 'gap', 'dmin', 'rms', 'net', 'id', 'updated', 'place', 'type', 'horizontalError', 'depthError', 'magError', 'magNst', 'status', 'locationSource', 'magSource']
['2022-11-30T22:30:36.483Z', '5.5812', '127.1411', '150.253', '4.6', 'mb', '103', '113', '2.144', '0.54', 'us', 'us6000j5w0', '2022-12-11T03:42:29.040Z', 'Philippine Islands region', 'earthquake', '6.82', '5.018','0.06', '81', 'reviewed', 'us', 'us']
['2022-11-30T21:07:22.712Z', '-35.5874', '-110.1823', '10', '4.7', 'mb', '18', '61', '31.304', '0.4', 'us', 'us6000j73b', '2022-12-10T06:03:14.040Z', 'southern East Pacific Rise', 'earthquake', '15.48', '1.306', '0.166', '11', 'reviewed', 'us', 'us']
['2022-11-30T18:10:30.406Z', '40.2398', '142.1853', '58.173', '4.5', 'mb', '24', '154', '1.016', '0.85', 'us', 'us6000j5ui', '2022-12-05T17:57:13.040Z', '64 km ESE of Hachinohe, Japan', 'earthquake', '6.65', '8.811', '0.172', '10', 'reviewed', 'us', 'us']
['2022-11-30T17:18:04.181Z', '26.8456', '55.1789', '16.891', '4.6', 'mb', '94', '37', '2.106', '0.76', 'us', 'us6000j5ua', '2022-12-05T17:37:06.040Z', '43 km NE of Bandar-e Lengeh, Iran', 'earthquake', '7.7', '3.768', '0.058', '88', 'reviewed', 'us', 'us']
['2022-11-30T16:22:16.241Z', '8.8228', '126.8520', '10', '4.8',

```
['2022-11-30T10:23:10.341Z', '0.0220', '120.0339', '10', '4.0',
'mb', '84', '66', '2.151', '0.68', 'us', 'us 6000j5sz', '2022-12-
09T00:50:11.040Z', 'Mindanao, Philippines', 'earthquake', '7.93',
'1.804', '0.063', '78', 'reviewed', 'us', 'us']
['2022-11-30T15:44:48.650Z', '-27.9013', '63.6965', '10', '4.5', 'mb',
'10', '177', '15.257', '0.23', 'us','6000j72w' '2022-12-09T00:39:32.040Z'
'Southwest Indian Ridge', 'earthquake', '14.22' '1.934' '0.243', '5',
'reviewed', 'us', 'us']
```

Session 5.5 Using the `csv reader` function

Description

HEADS UP:
You might be tempted to forgo the `csv` module, and read each line using the `readline` method, removing the terminating newline character, and splitting each line on a comma. That approach is risky, however, because some data items might have an embedded comma. And in fact, the "`place`" data in the earthquakes file does have an embedded comma. In this case, splitting on a comma would give us an incorrect result. Thus, it is better to use the `csv` module.

5.3.2 Using a `while` Loop to Read a File

For this example, we will focus on the severity of each earthquake. Seismologists measure severity as magnitude using the Richter scale. The larger the magnitude, the greater the severity of the earthquake.

Our first task in analyzing this real data source is to process the data file by extracting the pertinent data from the file and storing it in an appropriate

collection. That means we need to read the file and create a list of magnitudes so that we can analyze the data using statistics.

Let's think about how we can create a list of the magnitudes for the earthquakes in the file. First, we should find the column that contains the magnitude data. We can determine that by searching the metadata line, counting each label until we find the string `"mag"`. After we find the string we are looking for, we do not need to look at any of the remaining data names. The `while` loop will be ideal for this type of processing.

If we use a `while` loop to read a file, then our loop condition should detect when we have reached the end of the file. That loop condition would be that `readline` returned an empty string, meaning that there are no more lines to read. Before the loop begins, we read a line as the priming action for the loop condition. If there was a line to read, then inside the loop body, we process that line. As the last statement of the `while` loop body, we read the next line, which updates the loop condition for the next iteration. Following is the logic for reading a file using a `while` loop:

```
read a line
while not end of file:
    process the line
    read another line
```

Description

For the analysis of the earthquakes, we will use a `while` loop to find the column containing the earthquakes' magnitudes. **Session 5.6** shows how we can use a `while` loop to locate the column containing the magnitude. We know that its title is `"mag"`. In this session, we need only to read the first line that contains the titles. We can do that using the built-in function: `next`. The `next` function, as described in **Table 5.8**, returns the next item from an iterator. The iterator in this case is the `csvReader`.

```
In [1]: 1 import csv
```

```
In [2]: 1 with open("earthquakes.csv", "r") as inFile:
        2     csvReader = csv.reader(inFile)  #feed file to csv reader
        3
        4     titles = next(csvReader)        #read 1st line with titles
        5     colNum = 0
        6     while titles[colNum] != "mag":
        7         colNum = colNum + 1
        8
        9     print("The magnitude is found in col", colNum)
```

```
The magnitude is found in col 4
```

Session 5.6 Searching for a column name

Description

TABLE 5.8 The `next` Built-in Function

Method Name	Explanation
`next(iterator)`	Returns the next item from the iterator.

For this example, we first import the `csv` module. Then we open the file and pass the file to the `reader` function. We use the `next` function to read just the first line in the file, which contains the column names. The line is returned as a list of column names, so we will search the list using an index. Starting with a column number (`colNum`) of 0, we use the `while` loop condition to check whether the column name is `"mag"`. The entire loop body consists of incrementing the column number, which is also the loop update.

When we find the column name of `"mag"`, the condition becomes `False` and the `while` loop ends. The current value of `colNum` is the column containing the magnitude for each earthquake. The advantage of using a `while` loop is that as soon as we find the value we are looking for, we stop the loop. This avoids unnecessary processing of the remainder of the column names.

Now that we are correctly finding the column name, we can improve our loop in two ways. First, we can let the user pass the data name as a parameter. This will make the function reusable for any earthquake numeric data. Second, we can let the user know if we do not find the requested data name. We will know that we have not found the data name if we get to the end of the list without exiting the loop. So, in our loop, we need to check for two conditions: (1) we have not found the data name, and (2) we have not reached the end of the list. If either condition is `False`, we exit the loop. Assuming that the data name we are looking for is `dataName`, the condition now becomes:

```
while colNum < len(titles) and titles[colNum] != dataName:
```

Description

The order in which we check these two conditions is critical. Recall that for two conditions joined by an `and`, Python begins by checking the first condition. If that condition is `False`, the second condition is not evaluated. This works in our favor, because if the `colNum` is equal to the length of the `titles` list, we do not want to evaluate the second condition because the `colNum` would be invalid as an index and the second condition would

generate the error `"IndexError: list index out of range"`. If we reversed the order of the conditions, that error would be generated every time we reached the end of the `titles` list.

HEADS UP:
When a `while` loop can end either by reaching the end of a list or by matching a search condition, check for the end of the list first to avoid an `IndexError`.

The improved code is shown in **Listing 5.2**. On line 1, we added a parameter to specify the data name to find. On lines 6–7, we pass the file to the `csv` reader and read the titles line. Line 9 primes the `while` loop by setting `colNum` to 0. Line 10 includes our new `while` loop condition, and line 11 is the `while` loop body, which is also the loop update.

```
1  def makeDataList(dataName):
2      import csv
3      with open("earthquakes.csv", "r") as inFile:
4          dataList = []
5
6          csvReader = csv.reader(inFile)  #get iterator
7          titles = next(csvReader)        #read titles line
8
9          colNum = 0
10         while colNum < len(titles) and titles[colNum] != dataName:
11             colNum = colNum + 1
12
13         if colNum == len(titles):       #was end of titles reached?
14             print("Error:", dataName, "not found")
15         else:
16             for line in csvReader:
17                 dataList.append(float(line[colNum])) #create list of values
18     return dataList
```

Listing 5.2 Making a list of earthquake data

Description

The loop will terminate when one of the conditions becomes `False`. After the loop, we need to determine whether the loop terminated because we found the column or because we reached the end of the titles list without finding the column. If we reached the end of the list, `colNum` will be equal to the length of the list. We test for that condition on line 13, and print an error message if that is true.

If we found the column, we process the data lines in the file, appending the value in the `colNum` index for each earthquake to the `dataList` list (lines 16–17). Because all data is returned as a string, we convert the string to a number using the `float` function.

5.3.3 List Comprehension

We can make one more improvement to the `makeDataList` function. Notice that for every line in the file, we read the line and append the appropriate column to the `dataList` (lines 16–17). We can combine these two operations using a Python feature called **list comprehension**.

Before changing this code, let's look first at the general form of a list comprehension and some simpler examples of its use. The general form of a list comprehension is

```
[<expression> for <item1> in <sequence1>
              for <item2> in <sequence2>
              ...
   if <condition> ]
```

Description

Any items that correspond to a loop variable in one of the `for` loops can be used in the expression. You need to have only one `for` loop in the list comprehension, and the `if` is optional.

List comprehensions allow you to easily create one list based on doing some processing or applying selection criteria to another list. For example,

suppose that we want to create a list of the cubes of the first 10 numbers. One way to do this is shown in **Session 5.7**.

```
In [1]:  1 cubes = []
         2 for x in range(1, 11):
         3     cubes.append(x * x * x)
         4

In [2]:  1 cubes

Out[2]: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Session 5.7 Creating a list of cubes

Description

Using list comprehensions, we can do this in a single step, as shown in **Session 5.8**.

```
In [1]:  1 cubes = [x * x * x for x in range(1,11)]

In [2]:  1 cubes

Out[2]: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Session 5.8 A list of cubes using list comprehension

Description

The variable `x` takes on the values 1 through 10, as you might expect from the `for x in range(1, 11)` clause in the list comprehension. Each time through the loop, the expression `x * x * x` is computed and added to the list that is being constructed.

List comprehensions can also be used with an `if` statement to create a shorter list by keeping only those values from a longer list that meet certain criteria. This operation is often referred to as **filtering**. For example, suppose you wanted to create a list of all the even cubes, starting from the list of cubes we created previously. **Session 5.9** shows this code.

```
In [1]:  1 cubes = [x * x * x for x in range(1,11)]
         2 evenCubes = [x for x in cubes if x % 2 == 0]

In [2]:  1 evenCubes

Out[2]: [8, 64, 216, 512, 1000]
```

Session 5.9 Using `if` in list comprehension

Description

With a better understanding of list comprehensions, we can now return to the `makeDataList` function. We can combine reading the file and appending to the `dataList` list into one statement using list comprehension. The new `makeDataList` function is shown in **Listing 5.3**.

```
1  def makeDataList(dataName):
2      import csv
3      with open("earthquakes.csv", "r") as inFile:
4          dataList = []
5
6          csvReader = csv.reader(inFile)  #get iterator
7          titles = next(csvReader)        #read titles line
8
9          colNum = 0                      #search for dataName column
10         while colNum < len(titles) and titles[colNum] != dataName:
11             colNum = colNum + 1
12
13         if colNum == len(titles):       #was end of titles reached?
14             print("Error:", dataName, "not found")
15         else:                           #column was found
16             dataList = [(float(line[colNum])) for line in csvReader]
17     return dataList
```

Listing 5.3 `makeDataList` with list comprehension

Description

Line 16, which replaces lines 16–17 in the original function, creates a list consisting of the identified column from each line in the file, then assigns that list to `dataList`.

Session 5.10 shows several calls for our final `makeDataList` function. First, we create a list of magnitudes that contains 500 items. We print just the first 10 items. Then we do the same for the depth data. Finally, we call `makeDataList` for a column that is not found.

```
In [1]:  1 from makeDataList2 import *
```

```
In [2]:  1 magList = makeDataList("mag")
```

```
In [3]:  1 len(magList)
```

```
Out[3]: 500
```

```
In [4]:  1 for i in range(10):
         2     print(magList[i], end = " ")
         3
```

```
4.6 4.7 4.5 4.6 4.8 4.5 4.9 5.7 4.6 4.6
```

```
In [5]:  1 depthList = makeDataList("depth")
```

```
In [6]:  1 len(depthList)
```

```
Out[6]: 500
```

```
In [7]:  1 for i in range(10):
         2     print(depthList[i], end = " ")
         3
```

```
150.253 10.0 58.173 16.891 10.0 10.0 10.0 12.077 14.685 35.0
```

```
In [8]:  1 makeDataList("notATitle")
```

```
Error: notATitle not found
```

```
Out[8]: []
```

Session 5.10 Calling the `makeDataList` function

Description

We can now use basic statistics functions to learn about the earthquakes that have occurred during this period of time. **Session 5.11** uses the `makeDataList` function to create a list of magnitudes and then compute values for the maximum, minimum, mean, median, mode, and standard deviation. We can see that the range of earthquake magnitudes is 2.8 units, from a maximum of 7.3 to a minimum of 4.5. The three measures of central tendency seem to show that the majority of values are near the minimum. In fact, the mode of 4.5 is also the minimum, and the standard deviation is only .4.

```
In [1]:  1 from makeDataList2 import *
```

```
In [2]:  1 import statistics
```

```
In [3]:  1 magList = makeDataList("mag")
```

```
In [4]:  1 max(magList)
```

```
Out[4]: 7.3
```

```
In [5]:  1 min(magList)
```

```
Out[5]: 4.5
```

```
In [6]: 1 statistics.mean(magList)

Out[6]: 4.8302

In [7]: 1 statistics.median(magList)

Out[7]: 4.7

In [8]: 1 statistics.multimode(magList)

Out[8]: [4.5]

In [9]: 1 statistics.stdev(magList)

Out[9]: 0.40605137043342404
```

Session 5.11 Statistics from the earthquake data

Description

The reason for this difference between the mean and the mode becomes apparent when we look at our frequency table (**Session 5.12**). In this session, we have slightly modified the previous `frequencyTable` function to use string formatting. Although most of the earthquakes reported during the period were the minimum recorded by this report, some very strong earthquakes occurred that have magnitudes of 5 and greater. Thus, for this period, the mean is shifted up by these less frequent but strong events.

In [1]:

```
def frequencyTable(aList):
    countDict = {}
    for item in aList:
        if item in countDict:
            countDict[item] = countDict[item] + 1
        else:
            countDict[item] = 1

    itemList = list(countDict.keys())
    itemList.sort()

    print("ITEM", "FREQUENCY")
    for item in itemList:
        print("{0:4.1f} {1:6d}".format(item, countDict[item]))

```

In [2]:
```
1 from makeDataList import *
```

In [3]:
```
1 magList = makeDataList("mag")
```

In [4]:
```
1 frequencyTable(magList)
```

```
ITEM FREQUENCY
 4.5    115
 4.6     96
 4.7     60
 4.8     45
 4.9     59
 5.0     34
 5.1     26
 5.2     16
 5.3      8
 5.4      6
 5.5     12
```

```
5.6     4
5.7     3
5.8     2
5.9     1
6.0     1
6.1     3
6.2     2
6.6     1
6.8     1
6.9     1
7.0     3
7.3     1
```

Session 5.12 Frequency table for earthquake magnitude data

Description

Try It Out

5.8 Frequency tables are often created by placing data items in a range. Implement a function that will group the earthquake data by the following criteria: light (4.5–4.9), moderate (5–5.9), major (6–6.9), and strong (7–7.9). Use `matplotlib` to plot the frequency.

5.9 Write a function to process the earthquake data file and create lists of earthquake magnitudes, one for each date. Your function should return a list of lists that looks something like this:

```
[[date1, magnitude1, magnitude2, magnitude3,…],
[date2, magnitude1, magnitude2 ],

…]
```

Description

5.10 Modify the function in the previous exercise to return a dictionary of key–value pairs in which the key is the date and the value is a list of magnitudes for the earthquakes that occurred on that date.

5.11 Assuming that the following data is in a CSV file, write a function that creates a list of dictionaries, one per line of data.

```
product,color,price
suit,black,250
suit,gray,275
shoes,brown,75
shoes,blue,68
shoes,tan,65
```

For example, the dictionary for the first data line would be:

```
{'product': 'suit', 'color': 'black', 'price': '250'}
```

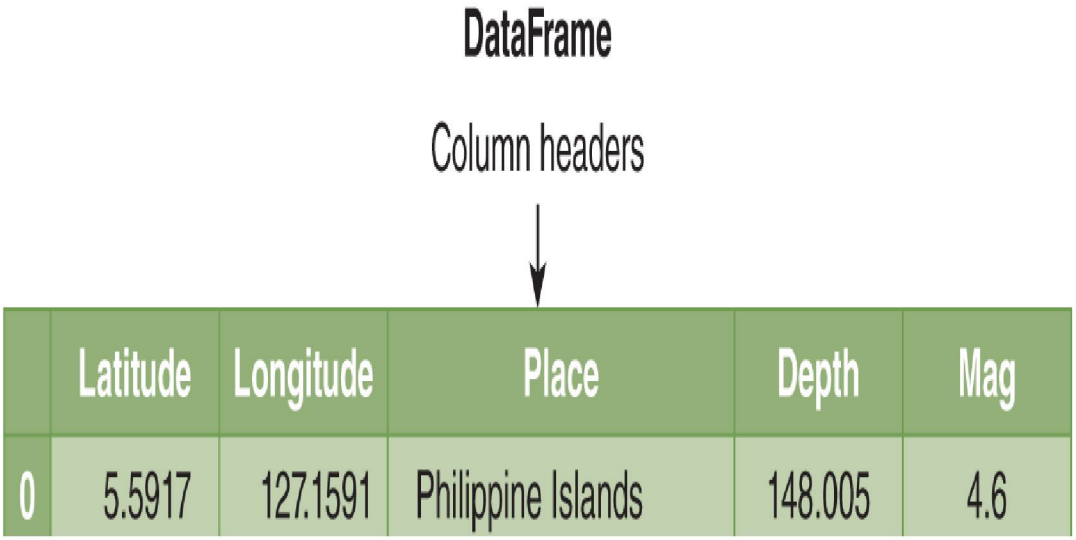
Description

5.4 Reading and Manipulating Data with `pandas`

In the previous sections, we showed how to explicitly read files line by line, and how to extract a column's data value as a list. This was important to illustrate for educational purposes. In practice, though, professionals (like data scientists) using large data sets typically use a Python package called `pandas` to read and manipulate data. In this section, we will discuss how to use the popular `pandas` library to do what we just did in the previous sections. If you are working with Jupyter Notebook, you already have `pandas` installed. If not, you may want to install `pandas` (see Appendix A for instructions).

5.4.1 `pandas` Data Structures

`Pandas` stores data in a two-dimensional tabular form called a **DataFrame**. Think of a DataFrame as similar to a spreadsheet, with data organized as labeled columns and rows. DataFrames can be sliced into subsets in the same way a collection or list can be sliced, and individual columns or rows can be extracted as a **Series**. A `pandas` series is a one-dimensional representation of a column or row with axis labels. **Figure 5.4** illustrates the differences between a `pandas` DataFrame and `pandas` series, and their relationships.



Row indices →

1	40.2538	142.1337	Honshu, Japan	57.997	4.6
2	26.8592	55.1776	Bandar-e Lengeh, Iran	17.467	4.6
3	8.8248	126.8707	Aras-asan, Philippines	10.000	4.8
4	-15.9117	176.9719	Fiji region	10.000	4.8

Series

By column

	Depth
0	148.005
1	57.997
2	17.467
3	10.000
4	10.000

By row

	Latitude	Longitude	Place	Depth	Mag



Figure 5.4 The `pandas` DataFrame and Series objects.

Description

The `pandas` library is powerful and extensive with numerous methods to read, manipulate, and process tabular data. We will not be able to cover all the available methods, but we will highlight some of the more commonly used functions, methods, and properties in **Tables 5.9A**, **5.9B**, and **5.9C**. We encourage you to look at https://pandas.pydata.org/ for more information on `pandas`'s full capabilities.

TABLE 5.9A A Few `pandas` Library Functions for Reading Files

Method Name	Explanation
`read_csv(filename, delimiter)`	Reads and parses the CSV file name, using the delimiter specified (defaults to comma-separated if nothing is specified). Returns a DataFrame object.
`read_excel(filename, sheetName)`	Reads a Microsoft® Excel file, and parses the worksheet specified by sheetName. Returns a DataFrame object.

TABLE 5.9B A Few Useful `pandas` DataFrame Properties and Methods

Method or Property Name	Explanation
`columns`	Property that returns a list of column names for this DataFrame.
`index`	Property that returns a list of row labels.
`iloc[]`	Property that allows for selection by integer position.
`shape()`	Method that returns the number of rows and columns in the DataFrame as a tuple.
`head(n)`	Method that gets the first `n` rows of the DataFrame.
`tail(n)`	Method that gets the last `n` rows of the DataFrame.

TABLE 5.9C A Few Useful `pandas` Series Methods

Method Name	Explanation
`sum()`	Returns the sum of all numeric data in the series.
`mean()`	Returns the mean of all numeric data in the series.
`mode()`	Returns the mode of all numeric data in the series.
`median()`	Returns the median of all numeric data in the series.
`std()`	Returns the standard deviation of all numeric data in the series.

5.4.2 Reading CSV Data with `pandas`

In Section 5.3.1, we used the `reader` function in the `csv` module to read and parse the USGS earthquake data. We can use the `read_csv` function in `pandas` to do the same, as shown in **Session 5.13**. The result of this function is a `pandas` DataFrame containing the earthquake data. Note that the `read_csv` function reads through every line automatically, so there is no need to create a `for` loop or `while` loop to sequentially read each line.

```
In [1]: 1 import pandas as pd
```

```
In [2]: 1 df = pd.read_csv("earthquakes.csv") #read entire csv file
        2 df                                 #display the result
```

Out[2]:

	time	latitude	longitude	depth	mag	magType	nst	gap	dmin	rms	...	updated	place	type	horizontalError
0	2022-11-30T22:30:36.483Z	5.5812	127.1411	150.253	4.6	mb	103.0	113.0	2.144	0.54	...	2022-12-11T03:42:29.040Z	Philippine Islands region	earthquake	6.82

1	2022-11-30T21:07:22.712Z	-35.5874	-110.1823	10.000	4.7	mb	18.0	61.0	31.304	0.40	...	2022-12-10T06:03:14.040Z	southern East Pacific Rise	earthquake	15.48
2	2022-11-30T18:10:30.405Z	40.2398	142.1853	58.173	4.5	mb	24.0	154.0	1.016	0.85	...	2022-12-05T17:57:13.040Z	64 km ESE of Hachinohe, Japan	earthquake	6.65
3	2022-11-30T17:18:04.181Z	26.8456	55.1789	16.891	4.6	mb	94.0	37.0	2.106	0.76	...	2022-12-05T17:37:06.040Z	43 km NE of Bandar-e Lengeh, Iran	earthquake	7.70
4	2022-11-30T16:23:16.341Z	8.8228	126.8539	10.000	4.8	mb	84.0	66.0	2.151	0.68	...	2022-12-09T00:50:11.040Z	Mindanao, Philippines	earthquake	7.93
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
495	2022-11-09T16:31:54.719Z	-20.3302	-177.6724	492.057	4.5	mb	45.0	111.0	3.163	0.48	...	2022-11-16T05:35:11.040Z	271 km WNW of Haveluloto, Tonga	earthquake	8.88
496	2022-11-09T10:14:33.810Z	-25.5781	178.2612	624.464	6.6	mww	131.0	27.0	4.998	0.65	...	2022-11-23T12:01:26.092Z	south of the Fiji Islands	earthquake	9.57
497	2022-11-											2022-11-	south of		

497	09T09:51:04.068Z	-26.0901	178.3427	660.000	7.0	mwb	142.0	26.0	4.578	0.66	...	23T11:52:40.121Z	the Fiji Islands	earthquake	10.35
498	2022-11-09T09:50:50.958Z	-25.6944	178.3159	634.472	4.7	mb	70.0	89.0	4.879	1.02	...	2022-11-23T11:26:46.040Z	south of the Fiji Islands	earthquake	11.81
499	2022-11-09T09:38:42.828Z	-26.0077	178.2776	630.379	6.8	mww	136.0	22.0	4.678	0.74	...	2022-11-23T11:38:17.587Z	south of the Fiji Islands	earthquake	8.79

500 rows × 22 columns

Session 5.13 Reading a CSV file with the `read_csv` function

Description

In Session 5.13, we import the `pandas` library. The next code block invokes the `read_csv` function where we pass the name of the CSV file (`earthquakes.csv`). The `read_csv` function returns a DataFrame object containing the data from the entire file, and we store that result in the variable `df`. When we want to take a look at the `df` variable, Jupyter Notebook automatically formats the output of a `pandas` DataFrame as a table with bolded column headers and bolded row indices.

Upon inspection, you may notice a few things about this DataFrame output. First, row indices start at zero, just like list and collection indices. Second, if the number of columns cannot fit on the Jupyter Notebook page, a scroll bar at the bottom of the table will allow you to see the rest of the columns. Third, if the data file is large, Jupyter truncates the output. For the earthquakes data, you see the first 5 rows, then an ellipsis separator indicating that some rows were hidden, and then the last 5 rows. Note that this does not mean only 10 rows were loaded. In fact, just below the loaded DataFrame, the number of rows (500) and columns (22) contained in the DataFrame are shown. Imagine

if the Jupyter Notebook did not truncate the output and you had to scroll through 500 rows to get to the next code block. So, *thank you* to Jupyter for automatically truncating the output!

HEADS UP:
There are other ways to control the number of printed rows of a `pandas` DataFrame. For example, you can use the `head(n)` method of the DataFrame, where `n` is the number of rows to show from the top of the DataFrame.

5.4.3 Data Manipulation with `pandas`

Once data is loaded into a DataFrame, `pandas` provides useful methods for manipulating and filtering that data. Let's do a little bit of data manipulation on the earthquake data set.

In **Session 5.14, Part 1**, we read the earthquake data set as before. We can get information about the names of the columns in the DataFrame by calling the `columns` property, which returns a list of the column names (as strings). If we only want to extract one of the columns as a series, we can specify the column name in brackets, just as we would for a dictionary object. A Series object also provides some statistical methods to calculate the mean, median, and mode, as shown in Table 5.9C. As you can see in the third code block, we can get the mean of all the earthquake magnitudes by invoking the `mean` method.

```
In [1]:  1 import pandas as pd
         2 df = pd.read_csv("earthquakes.csv")
```

```
In [2]:  1 # get a list of the column names in the DataFrame
```

```
2 df.columns
```

Out[2]:
```
Index(['time', 'latitude', 'longitude', 'depth', 'mag', 'magType', 'nst',
       'gap', 'dmin', 'rms', 'net', 'id', 'updated', 'place', 'type',
       'horizontalError', 'depthError', 'magError', 'magNst', 'status',
       'locationSource', 'magSource'],
      dtype='object')
```

In [3]:
```
1 # get only the magnitude column data as a series
2 magnitudes = df["mag"]
3 # calculate the mean of the magnitude series.
4 magnitudes.mean()
```

Out[3]:
```
4.8302
```

In [4]:
```
1 # get the 5th row (index 4) as a series
2 fifthRow = df.iloc[4]
3 # retrieve the "place" column value of the fifth row
4 fifthRow["place"]
```

```
Out[4]: 'Mindanao, Philippines'
```

Session 5.14, Part 1 Manipulating data using `pandas`

Description

Just as we can extract single columns, we can extract single rows using the `iloc` property. The fourth code block in Session 5.14, Part 1, illustrates how we can extract the 5th row of the DataFrame by referencing its index (remember indices start at 0). This returns a series of that row, and we can retrieve a specific column's value in the same way that we extracted a column from the DataFrame. The value of the `place` column for the 5th row is `Mindanao Philippines`.

Session 5.14, Part 2, shows how we can further use `pandas` to manipulate data. In code block 5, we create a new column. The original depth column was measured in kilometers, but we might want to represent that data column in different units. So, we can simply call the new column something appropriate ("`depth_m`") and define it as the original "`depth`" column multiplied by 1,000. `Pandas` automatically applies that calculation to every row of the DataFrame.

```
In [5]: 1 # create a new column based on a calculation using an existing column
        2 df["depth_m"] = df["depth"] * 1000.0
```

```
In [6]: 1 # subset the DataFrame to show only a few columns
        2 myColumns = ["latitude", "longitude", "place", "depth_m", "mag"]
        3 subsetDf = df[myColumns]
        4 subsetDf.head(5)
```

Out[6]:

	latitude	longitude	place	depth_m	mag
0	5.5812	127.1411	Philippine Islands region	150253.0	4.6
1	-35.5874	-110.1823	southern East Pacific Rise	10000.0	4.7
2	40.2398	142.1853	64 km ESE of Hachinohe, Japan	58173.0	4.5
3	26.8456	55.1789	43 km NE of Bandar-e Lengeh, Iran	16891.0	4.6
4	8.8228	126.8539	Mindanao, Philippines	10000.0	4.8

In [7]:

```
# retrieve the first 7 rows
subsetDf.iloc[0:7]
```

Out[7]:

	latitude	longitude	place	depth_m	mag
0	5.5812	127.1411	Philippine Islands region	150253.0	4.6
1	-35.5874	-110.1823	southern East Pacific Rise	10000.0	4.7
2	40.2398	142.1853	64 km ESE of Hachinohe, Japan	58173.0	4.5

3	26.8456	55.1789	43 km NE of Bandar-e Lengeh, Iran	16891.0	4.6
4	8.8228	126.8539	Mindanao, Philippines	10000.0	4.8
5	-27.9013	63.6965	Southwest Indian Ridge	10000.0	4.5
6	-16.1908	176.8334	Fiji region	10000.0	4.9

In [8]:
```
# retrieve only the even rows using list comprehension
subsetDf.iloc[ [(i % 2 == 0) for i in subsetDf.index] ]
```

Out[8]:

	latitude	longitude	place	depth_m	mag
0	5.5812	127.1411	Philippine Islands region	150253.0	4.6
2	40.2398	142.1853	64 km ESE of Hachinohe, Japan	58173.0	4.5
4	8.8228	126.8539	Mindanao, Philippines	10000.0	4.8
6	-16.1908	176.8334	Fiji region	10000.0	4.9
8	21.3009	121.1190	87 km SSE of Hengchun, Taiwan	14685.0	4.6

...	...	...	...	...	...
490	-15.6096	-173.0045	93 km ENE of Hihifo, Tonga	10000.0	4.7
492	-26.0355	178.4719	south of the Fiji Islands	656429.0	4.9
494	-10.9543	113.3845	south of Java, Indonesia	10000.0	4.8
496	-25.5781	178.2612	south of the Fiji Islands	624464.0	6.6
498	-25.6944	178.3159	south of the Fiji Islands	634472.0	4.7

Session 5.14, Part 2 Manipulating data using `pandas`

Description

If we do not need every column in the original data set, we can subset the DataFrame by passing a list of just the column names we want. Suppose we only care to see the "`latitude`", "`longitude`", "`place`", our new depth field ("`depth_m`"), and the magnitude ("`mag`"). The code blocks 6 and 7 in Session 5.14, Part 2, show how we can use the `iloc` property to extract the first 7 rows using list splitting.

We can also use list comprehension to extract only the even rows of the DataFrame, as shown in code block 8. The `index` property in this statement returns a range of row labels from the DataFrame, which by default are simply the row numbers.

In **Session 5.14, Part 3**, we continue looking at how to use conditional expressions to filter rows. This is useful if we want to select specific rows based on one or more columns' values. Code block 9 filters for all rows where the earthquake magnitude is greater than 6.5. Conditional statements for a DataFrame are placed within brackets, and usually reference the same

DataFrame variable (`subsetDf`). This is critical, because the conditional statements used to filter need to return a Boolean result for *each* row. In fact, the statement

```
subsetDf[subsetDf["mag"] > 6.5]
```

returns a list of `True` and `False` values that represent whether each row met that condition. `Pandas` will show only the rows that evaluated as `True`. Finally, we can string together more complex conditional statements using *and* (`&`) and *or* (`|`). The last code block (10) filters for all places that contain the name "`Fiji`" and have a magnitude greater than 6.5. We use the `str` function to cast the values in the `place` column to be strings (the default data type for text is `object`), which then allows us to use standard string methods like `contains`.

```
In [9]: 1 # retrieve all rows where the magnitude is greater than 6.5
        2 subsetDf[subsetDf["mag"] > 6.5]
```

Out[9]:

	latitude	longitude	place	depth_m	mag
261	-9.8078	159.5927	Solomon Islands	14000.0	7.0
332	-4.9043	100.7862	204 km SW of Bengkulu, Indonesia	25000.0	6.9
446	-20.1112	-178.3453	Fiji region	579000.0	7.0
474	-19.2881	-172.1471	205 km ESE of Neiafu, Tonga	37000.0	7.3
496	-25.5781	178.2612	south of the Fiji Islands	624464.0	6.6

497	-26.0901	178.3427	south of the Fiji Islands	660000.0	7.0
499	-26.0077	178.2776	south of the Fiji Islands	630379.0	6.8

```
In[10]: 1 subsetDf[(subsetDf["place"].str.contains("Fiji")) & (subsetDf["mag"] > 6.5)]
```

Out[10]:

	latitude	longitude	place	depth_m	mag
446	-20.1112	-178.3453	Fiji region	579000.0	7.0
496	-25.5781	178.2612	south of the Fiji Islands	624464.0	6.6
497	-26.0901	178.3427	south of the Fiji Islands	660000.0	7.0
499	-26.0077	178.2776	south of the Fiji Islands	630379.0	6.8

Description

HEADS UP:
It is a good coding practice to surround each conditional statement in a `pandas` filter with parentheses. This improves readability and reduces ambiguity on how the conditional statement is interpreted by `pandas`.

Try It Out

5.12 Filter the earthquake data to return only earthquakes that occurred in the Southern Hemisphere and at a depth shallower than 10 km (10,000 m).

5.13 Filter the earthquake data to return only earthquakes above 60 degrees latitude or below −60 degrees latitude.

5.14 Use `pandas` to find the mean, median, and standard deviation of the earthquake's depth column.

5.5 Reading JSON Data from the Internet

As mentioned earlier, another popular format for making data available on the Internet is JSON (JavaScript Object Notation). Like CSV files, JSON embeds identifying fields with the data. Whereas CSV files provide data names at the top of the file, JSON provides the data names with each piece of data using a dictionary-like syntax.

To demonstrate the use of data in JSON format, let's use the same earthquake data from the USGS. The USGS provides the earthquake data in GeoJSON format, which is designed for communicating geographic information. GeoJSON uses standard JSON formatting. The JSON data is provided under the Real-Time Feeds on the USGS website (https://earthquake.usgs.gov/earthquakes/).

We will examine the same earthquake data as we downloaded in the CSV file: all earthquakes with at least a 4.5 magnitude for the same time period. **Figure 5.5** shows the main description information (shown in blue)—the metadata—and the specific data for the first three earthquakes in JSON formatting. The ellipsis in the final gray section indicates information for the remaining earthquakes, which is not shown here. The data for each earthquake is provided as name–value pairs separated by commas, similar to a dictionary. The last field in the metadata is `"count"` with the value 500, which is the number of earthquakes for which data is provided.

```
{"type":"FeatureCollection","metadata":{"generated":1673714111000,"ur
l":"https://earthquake.usgs.gov/fdsnws/event/1/query.geojson?starttime
=2022-11-01%2000:00:00&endtime=2022-11-30%2023:59:59&minmagnitude=4.5
&orderby=time&limit=500","title":"USGS Earthquakes","status":200,"api
":"1.13.6","limit":500,"offset":1,"count":500},
"features":[
{"type":"Feature","properties":{"mag":4.6,"place":"Philippine Islands
```

{"type":"Feature","properties":{"mag":4.6,"place":"Philippine Islands region","time":1669847436483,"updated":1670730149040,"tz":null,"url":"https://earthquake.usgs.gov/earthquakes/eventpage/us6000j5w0","detail":"https://earthquake.usgs.gov/fdsnws/event/1/query?eventid=us6000j5w0&format=geojson","felt":null,"cdi":null,"mmi":null,"alert":null,"status":"reviewed","tsunami":0,"sig":326,"net":"us","code":"6000j5w0","ids":",us6000j5w0,","sources":",us,","types":",origin,phase-data,","nst":103,"dmin":2.144,"rms":0.54,"gap":113,"magType":"mb","type":"earthquake","title":"M 4.6 - Philippine Islands region"},"geometry":{"type":"Point","coordinates":[127.1411,5.5812,150.253]},"id":"us6000j5w0"},

{"type":"Feature","properties":{"mag":4.7,"place":"southern East Pacific Rise","time":1669842442712,"updated":1670652194040,"tz":null,"url":"https://earthquake.usgs.gov/earthquakes/eventpage/us6000j73b","detail":"https://earthquake.usgs.gov/fdsnws/event/1/query?eventid=us6000j73b&format=geojson","felt":null,"cdi":null,"mmi":null,"alert":null,"status":"reviewed","tsunami":0,"sig":340,"net":"us","code":"6000j73b","ids":",us6000j73b,","sources":",us,","types":",origin,phase-data,","nst":18,"dmin":31.304,"rms":0.4,"gap":61,"magType":"mb","type":"earthquake","title":"M 4.7 - southern East Pacific Rise"},"geometry":{"type":"Point","coordinates":[-110.1823,-35.5874,10]},"id":"us6000j73b"},

```
{"type":"Feature","properties":{"mag":4.5,"place":"64 km ESE of Hachi
nohe, Japan","time":1669831830406,"updated":1670263033040,"tz":null,"
url":"https://earthquake.usgs.gov/earthquakes/eventpage/us6000j5ui","
detail":"https://earthquake.usgs.gov/fdsnws/event/1/query?eventid=us
6000j5ui&format=geojson","felt":null,"cdi":null,"mmi":null,"alert":nul
l,"status":"reviewed","tsunami":0,"sig":312,"net":"us","code":"6000j5
ui","ids":",us6000j5ui,","sources":",us,","types":",origin,phase-data
,","nst":24,"dmin":1.016,"rms":0.85,"gap":154,"magType":"mb","type":"
earthquake","title":"M 4.5 - 64 km ESE of Hachinohe, Japan"},"geometr
y":{"type":"Point","coordinates":[142.1853,40.2398,58.173]},"id":"us
6000j5ui"},
…],"bbox":[-179.9729,-64.0075,4,179.9617,72.5191,660]}
```


Figure 5.5 Earthquake data in JSON format.

Description

For each earthquake, we see that for the key `"type"`, the value is `"Feature"`, and for the key `"properties"`, the value is a dictionary itself. One of the dictionary keys within the `properties` value is `"mag"`, which represents the magnitude of the earthquake. These items are highlighted in red in Figure 5.5. As you can see, the magnitudes of the first three of the 500 earthquakes are 4.6, 4.7, and 4.5, respectively.

To download this JSON data, we use the **Uniform Resource Locator (URL)** of the data. This URL is displayed in the address line of the browser as you select the desired time period and severity of the earthquake data from

the USGS site. For our data, the URL is shown on the second line of Figure 5.5.

Note that the USGS website allows you to select data for specific or relative periods of time, such as the previous 30 days or the past week.

Fortunately, Python makes it easy to get online data using the `urllib.request` module. With this module, we can read data from web pages just as easily as we can read data from a file. Rather than using the `open` function as we did for files, however, we call the `urllib.request.urlopen` function. The `urlopen` function, shown in **Table 5.10**, hides all the networking details needed to communicate with the website and returns an object that acts just like a file reference. To call `urlopen`, we need to import `urllib.request`.

TABLE 5.10 The `urlopen` Function in the `urllib.request` Module

Function Name	Explanation
`urlopen(url)`	Opens a web address and returns a file reference. Raises a `URLError` if unsuccessful.

Once we read the JSON data, we translate the JSON objects into Python objects. An important function in the `json` module is `loads`, which performs this translation. This function is shown in **Table 5.11**. Of course, we will need to `import json`.

RECAP:

JSON data is easily translated into Python lists, dictionaries, and built-in data types.

TABLE 5.11 The `loads` Function in the `json` Module

Function Name	Explanation
`loads(jsonData)`	Converts JSON objects to Python objects. JSON objects are translated to Python dictionaries; JSON arrays are translated to Python lists; and JSON data types are translated to Python data types.

Session 5.15 shows the initial steps to download and translate the JSON data into Python objects. It also shows that the translated data, `eData`, is a Python dictionary and that for the `features` key, the value is a list with 500 elements, one element for each earthquake.

In [1]:
```
1 import json
2 import urllib.request
```

In [2]:
```
1 handle = urllib.request.urlopen("https://earthquake.usgs.gov/fdsnws/
2                                  event/1/query.geojson? \starttime=
3                                  2022-11-01%2000:00:00&endtime=2022
4                                  -11-30%2023:59:59 \ &minmagnitude=4.5&
5                                  orderby=time&limit=500")
6 data = handle.read()
7 eData = json.loads(data)
```

In [3]:
```
1 eData.keys()
```

```
Out[3]: dict_keys(['type', 'metadata', 'features', 'bbox'])

In [4]: 1 earthquakeList = eData.get('features')

In [5]: 1 len(earthquakeList)

Out[5]: 500
```

Session 5.15 Loading the JSON data

Description

Now we can create our list of magnitudes. To do this, we need to walk through the dictionary structures to reach the `mag` key. **Listing 5.4** shows the code to do this. In line 3, we get the list that is the value of the key `'features'`. Each item in the list has data for one earthquake, so we use a `for` loop to visit each item in the list. In the `for` loop on line 5, we get the next earthquake in the list. Then we find the value for the `'properties'` key for this earthquake. Recall that the value for the `'properties'` key is itself a dictionary; from that dictionary, we retrieve the value for the `'mag'` key, which is the magnitude for the earthquake.

```
1  def makeMagList(earthquakeData):
2      magList = []
3      earthquakes = earthquakeData.get('features')
4      for i in range(len(earthquakes)):
5          earthquake = earthquakes[i]
6          properties = earthquake.get('properties')
7          mag = properties.get('mag')
8          magList.append(mag)
9
10     return magList
```

Listing 5.4 Making a list of JSON earthquake data

Description

Now we are ready to create the magnitude list and run the same statistics as we did for the data in the CSV file. **Session 5.16** shows the results of these calculations. As we would expect, these results are the same as we received for the data in CSV format. In this session, instead of reading the JSON data directly from the USGS site, we have previously saved the downloaded data in a file named `earthquakes.json`. Now we can open and read the file as a text file. The only difference is that we use the `encoding` parameter to the `open` function to specify that the data is in UTF-8 format.

```
In [1]:  1 from makeJsonMagList import *
         2 import json, statistics
```

```
import json, statistics
```

In [2]:
```
with open("earthquakes.json", encoding='utf-8') as handle:
    data = handle.read()
    eData = json.loads(data)
```

In [3]:
```
magList = makeMagList(eData)
```

In [4]:
```
statistics.mean(magList)
```

Out[4]: 4.8302

In [5]:
```
statistics.median(magList)
```

Out[5]: 4.7

In [6]:
```
statistics.multimode(magList)
```

Out[6]: [4.5]

```
In [7]: 1 statistics.stdev(magList)

Out[7]: 0.40605137043342404
```

Session 5.16 Running statistics on earthquake magnitudes

Description

Try It Out

5.15 Using the JSON data, create a list of earthquake depths.

5.16 Write a function that will create a list of any of the `"properties"` values for earthquakes. Your function should take the property name as a parameter.

5.17 In the JSON format of the earthquake data, the longitude, latitude, and depth are reported in that order in a list, which is the value of the `"coordinates"` key under the `"geometry"` key (see Figure 5.5). Write a function to create a list of longitudes from this data.

5.18 The "felt" property measures how many people reported feeling the earthquake. But for many of the earthquakes, the `"felt"` value is `None`, meaning there is no measurement for this property. Create a list that contains only those `"felt"` values that are not `None`.

5.6 Correlating Data

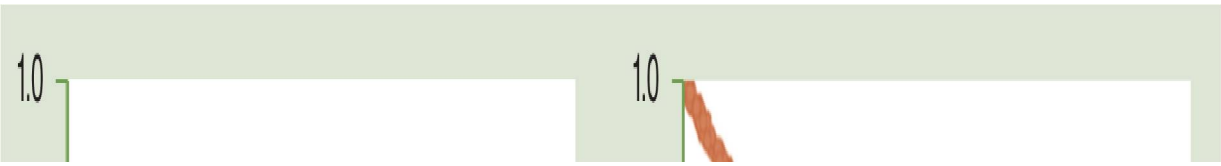
Correlation measures the strength and direction of the relationship between two variables. Expressed another way, correlation measures the tendency of two variables to increase or decrease at the same time. This measure is often called the **correlation coefficient**. Returning to the earthquake data, we could ask the question "Is the magnitude of an earthquake correlated with its depth?"

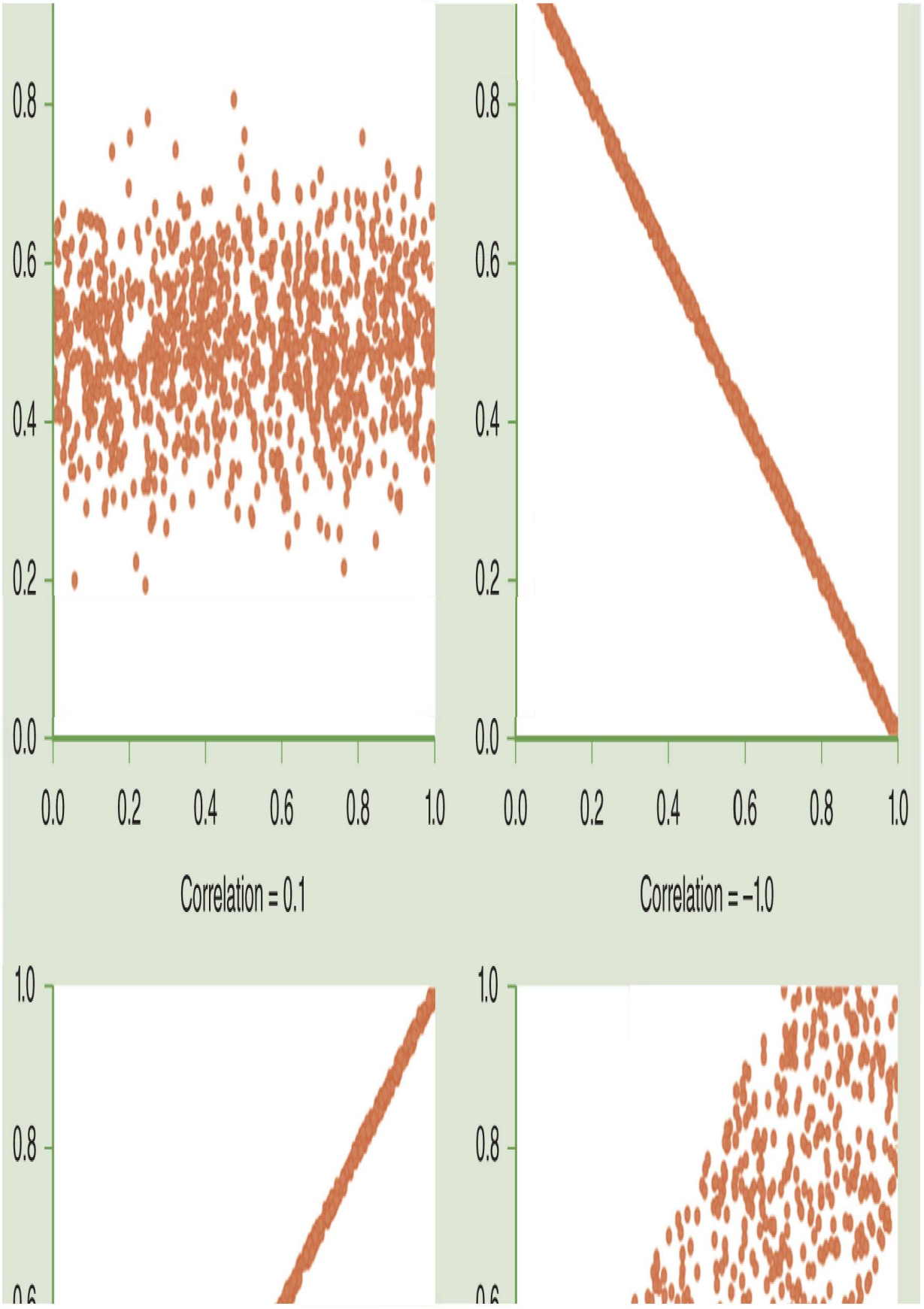
Although several algorithms are available for calculating a correlation coefficient for a sample, we will use the **Pearson correlation coefficient**, which yields a number between −1 and 1 to measure the relationship between two variables. The formula for calculating this coefficient is

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)S_x S_y}$$

where $\bar{x}$ and $\bar{y}$ are the means of the two variables x and y, and S_x and S_y are the standard deviations.

Figure 5.6 illustrates some of the correlational values that are possible for two variables. The values from one variable are used as the x coordinate, and the values from the other variable are used as the y coordinate. Each of the four examples shows 1,000 pairs of values of the x, y variables. When the variables are highly correlated, they nearly form a line, as shown in the bottom left and top right boxes. When they are not correlated, the points form a cloud or a wide band, as shown in the top left and bottom right boxes. A value of 1.0 indicates that the two variables are positively correlated—that is, the values of the variables move in the same direction. A value of −1.0 indicates that the two variables are negatively correlated—that is, the values of the variables move in opposite directions. A value of 0.0 indicates that there is no correlation between the two values.




0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Correlation = 0.1
Correlation = −1.0
1.0
0.8
0.6

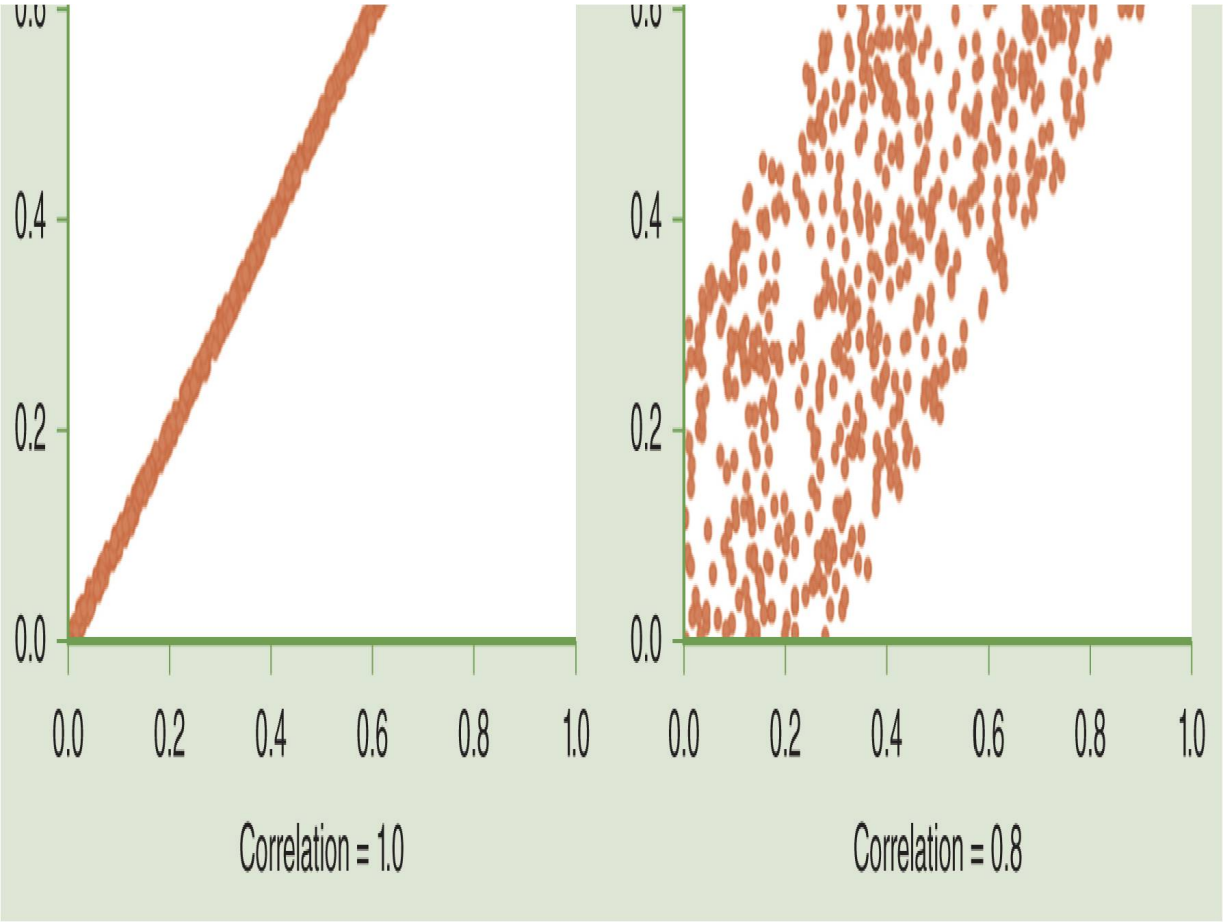


Figure 5.6 Examples of points and their correlation.

Description

Listing 5.5 implements the Pearson correlation formula. The two parameters `xList` and `yList` represent the values that we want to correlate. The two lists have the same length, and they are parallel, which means that the *x* value at a particular position in the list and a *y* value at the same position represent a single *x*, *y* data point. For example, using our earthquake data, the values at a particular index in `xList` and `yList` represent two measures for a particular earthquake.

```
1  def correlation(xList, yList):
2      import statistics as st
3      xBar = st.mean(xList)
4      yBar = st.mean(yList)
5      xStd = st.stdev(xList)
6      yStd = st.stdev(yList)
7
8      num = 0.0
9      for i in range(len(xList)):
10         num = num +(xList[i] - xBar) * (yList[i] - yBar)
11
12     corr = num / ((len(xList) - 1) * xStd * yStd)
13     return corr
```

Listing 5.5 The Pearson correlation function

Description

In **Session 5.17**, we run the correlation function between the magnitudes and depth of earthquakes, extracting the data from the CSV file. We find a result of 0.14, which means that the two values are not correlated.

```
In [1]:  1 from correlation import *
         2 from makeDataList import *
```

```
In [2]:  1 magList = makeDataList("mag")
         2 depthList = makeDataList("depth")
```

```
In [3]:  1 correlation(magList, depthList)
```

```
Out[3]: 0.1392354238478085
```

Session 5.17 Correlating magnitude and depth

Description

Try It Out

5.19 For each week of the semester so far, create a list of your average grade earned that week and another list tracking the number of hours you worked on your courses that week. Run the correlation function to see if those two data items are correlated.

5.7 Summary

This chapter used external data sources as input into Python programs. Some of these data sources, called text files, can reside on a local machine or may be available via the Internet. We showed how to create formatted output by using format strings, which allows for structured output, as well as formatted numeric values, columns of data, and justification. We used list comprehension to filter the external data. We also accessed real earthquake data on the Internet in CSV, `pandas`, and JSON formats and used statistical functions to analyze the data. Finally, we developed another statistical function called correlation.

Key Terms

close a file
correlation
correlation coefficient
CSV (comma-separated values)
DataFrame
file pointer
filtering
formatted strings
iterator
JSON (JavaScript Object Notation)
list comprehension
metadata
newline
open a file
Pearson correlation coefficient
Series
text file
Uniform Resource Locator (URL)

Programming Exercise

5.1 Write a program to correlate gross domestic product (GDP) and personal income—specifically, wages and salaries. GDP is the value of all goods and services in the U.S. economy. The Bureau of Economic Analysis makes this data available in the public domain.

To work with the latest data, go to the website of the Bureau of Economic Analysis and find the page for "Interactive Data Application" (https://apps.bea.gov/itable/). Click on "GDP and Personal Income"; then click "Interactive Data Tables." In Section 1, choose the table for "Gross Domestic Product." Download this table in CSV format. In Section 2, select the table for "Personal Income and Its Disposition." Download this table in CSV format. Write functions to extract the GDP and Wages and Salaries data into two lists. Then run the chapter's `correlation` function to see whether GDP and Wages and Salaries are related.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock

CHAPTER 6
Image Processing
Nested `for` Loops, Functions as Parameters, Namespaces, and Lists of Lists

6.1 OBJECTIVES

- To understand pixel-based image processing
- To use nested iteration
- To implement a number of image processing algorithms
- To create and use lists of lists
- To understand passing functions as parameters
- To understand the mechanics of parameter passing

6.2 What Is Digital Image Processing?

Digital photography is a popular way to produce photographs. Indeed, it sometimes seems as if almost everyone has either a camera on their cell phone or a digital camera, or both, as well as software that can organize and manipulate photographs. In this chapter, we consider digital images and many of the techniques that can be used to modify and enhance them.

Digital image processing uses algorithms to edit and manipulate digital images. A **digital image** is a collection of tiny, discrete picture elements called **pixels**. These pixels are organized in a two-dimensional grid and represent the smallest amount of picture information that can be referenced. If you look closely at an image, you may notice that pixels sometimes appear as small "dots." More pixels in your image mean more detail or higher **resolution**.

Digital cameras are often rated according to how much resolution they provide. Typically, resolution is expressed as a number of **megapixels**. One megapixel means that the picture you take is composed of 1 million pixels. A 20-megapixel camera is capable of taking a picture composed of as many as 20 million pixels.

6.2.1 The RGB Color Model

Each pixel in the digital image has a single color. Its specific color depends on a formula that mixes various amounts of the colors red, green, and blue. Defining colors as a combination of red, green, and blue is often referred to as the **RGB color model**.

The amount of each color component in a pixel is referred to as its **color intensity**. Intensities range from a minimum of 0 to a maximum of 255. For example, a color with 255 red intensity, 0 green intensity, and 255 blue intensity will be purple (or magenta). Black has zero intensity for all color components, whereas white has full color intensity, 255, for all three elements. **Table 6.1** shows some common RGB color combinations.

TABLE 6.1 Red, Green, and Blue Intensities for Some Common Colors

Color	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
Magenta	255	0	255
Yellow	255	255	0
Cyan	0	255	255
White	255	255	255
Black	0	0	0

Description

An interesting fact arises when you consider how many colors are possible using the RGB color model. With each of the three colors having 256 intensity levels, there are $256^3 = 16{,}777{,}216$ different combinations of red, green, and blue intensities.

6.2.2 The `cImage` Module

To manipulate images, we will use a group of objects found in our `cImage` module. See "Installing the Python Image Library and `cImage`" in Appendix A for instructions on installing `cImage.py`. We also provide `cImage` with the source code for this chapter. This module contains objects that allow us to construct and manipulate pixels. We can also construct an image from a file or create a blank image that we can fill in later. In addition, we can create windows where images can be displayed.

HEADS UP:
For all the code in this chapter, you will need to import `cImage`:

```
from cImage import *
```

The `Pixel` Object

Images are collections of pixels. To represent a pixel, we need a way to collect the red, green, and blue components of its color. The `Pixel` object in `cImage` provides a constructor that allows us to create `Pixel` objects and methods that allow us to manipulate the color components of pixels. **Table 6.2** shows the constructor and methods provided by the Pixel object, and **Session 6.1** shows them in action. The constructor requires the three color components; it returns a reference to a `Pixel` object that can be accessed or modified. We can extract the color intensities using the `getRed`, `getGreen`, and `getBlue` methods. Similarly, we can modify the individual components within a pixel using the `setRed`, `setGreen`, and `setBlue` methods.

TABLE 6.2 The `Pixel` Object

Method Name	Example Use	Explanation
`Pixel(r, g, b)`	`p = Pixel(25, 200, 143)`	Creates a pixel (`p`) with color component intensities of 25 red, 200 green, and 143 blue.
`getRed()`	`r = p.getRed()`	Returns the red component intensity.
`getGreen()`	`g = p.getGreen()`	Returns the green component intensity.
`getBlue()`	`b = p.getBlue()`	Returns the blue component intensity.
`setRed(r)`	`p.setRed(100)`	Sets the red component intensity to 100.
`setGreen(g)`	`p.setGreen(45)`	Sets the green component intensity to 45.
`setBlue(b)`	`p.setBlue(87)`	Sets the blue component intensity to 87.

```
In [1]: from cImage import *

In [2]: p = Pixel(200, 100, 150)   #create a Pixel object

In [3]: p

Out[3]: (200, 100, 150)

In [4]: p.getRed()                 #get the red component

Out[4]: 200

In [6]: p.setBlue(20)              #set the blue component to 20

In [7]: p

Out[7]: (200, 100, 20)
```

Session 6.1 Creating and using a pixel

Description

RECAP:

A `Pixel` object is represented by various values of red, green, and blue intensities.

The `ImageWin` Object

Before creating images, we will create a window that can be used to display our images. The `ImageWin` object in `cImage` provides a constructor that produces a window with a title, width, and height. When a window is constructed, it is shown immediately. The following code produces an empty window that is 600 pixels wide and 400 pixels high and has the title "Image Processing":

```
from cImage import *
myWin = ImageWin("Image Processing", 600, 400)
```

Description

Table 6.3 shows additional methods for the `ImageWin` object. Note that the `getMouse` method returns a coordinate position within the window itself and is not related to any particular image that might be displayed within the window.

TABLE 6.3 The `ImageWin` Object

Method Name	Example Use	Explanation
`ImageWin(title, width, height)`	`myWin = ImageWin("Pictures", 800, 600)`	Creates a window to display images that are 800 pixels wide and 600 pixels high with the title "Pictures."

Method Name	Example Use	Explanation
`exitOnClick()`	`myWin.exitOnClick()`	Closes the image window and exits when the mouse is clicked.
`getMouse()`	`pos = myWin.getMouse()`	Waits for a mouse click and then returns an (*x*, *y*) tuple representing the (col, row) of the mouse click in the window.

Now that we have a window, we need to create images to display. The `cImage` module provides two kinds of image objects: `FileImage` and `EmptyImage`. These objects allow us to create and manipulate images, and they give us simple access to the pixels in each image. **Table 6.4** shows the two constructors for creating images, as well as other methods that are provided by both objects.

TABLE 6.4 `FileImage` and `EmptyImage` Objects

Method Name	Example Use	Explanation
`FileImage(filename)`	`im = FileImage ("pic.gif")`	Creates an image object from a file named `pic.gif`.
`EmptyImage(width, height)`	`im = EmptyImage(300, 200)`	Creates an empty image that is 300 pixels wide and 200 pixels high.
`getWidth()`	`w = im.getWidth()`	Returns the width of the image in pixels.
`getHeight()`	`h = im.getHeight()`	Returns the height of the image in pixels.
`getPixel(col, row)`	`p = im.getPixel(150, 100)`	Returns the pixel from column 150, row 100.
`setPixel(col, row, Pixel)`	`im.setPixel(150, 100, Pixel(255, 255, 255))`	Sets the pixel at column 150, row 100 to be white.
`setPosition (col, row)`	`im.setPosition(20, 30)`	Positions the top-left corner of the image at column 20, row 30 in the window.
`draw(ImageWin)`	`im.draw(myWin)`	Displays the image `im` in the `myWin` image window. It will default to the upper-left corner.
`save(fileName)`	`im.save(fileName)`	Saves the image to a file.

The `FileImage` Object

The `FileImage` object is an image that is constructed from files such as those that are created by digital cameras or that reside on web pages. For this chapter, we are using an image named `butterfly.png`. You can find this file in the Chapter 6 source code folder. But you could also work with an image of your choice. **Session 6.2** shows that the `FileImage` constructor needs only the name of the image file. It converts the image stored in that file to an image object. In this case, `butterfly` is the reference to that object.

```
In [1]: from cImage import *

In [2]: myWin = ImageWin("Butterfly", 300, 224)

In [3]: butterfly = FileImage("butterfly.png")

In [4]: butterfly.draw(myWin)
```

Session 6.2 Creating and showing a file image

Description

We can use the `draw` method to ask the image object to show itself in an image window. The default positioning is to place the image in the upper-left corner of the window. **Figure 6.1** shows the result.

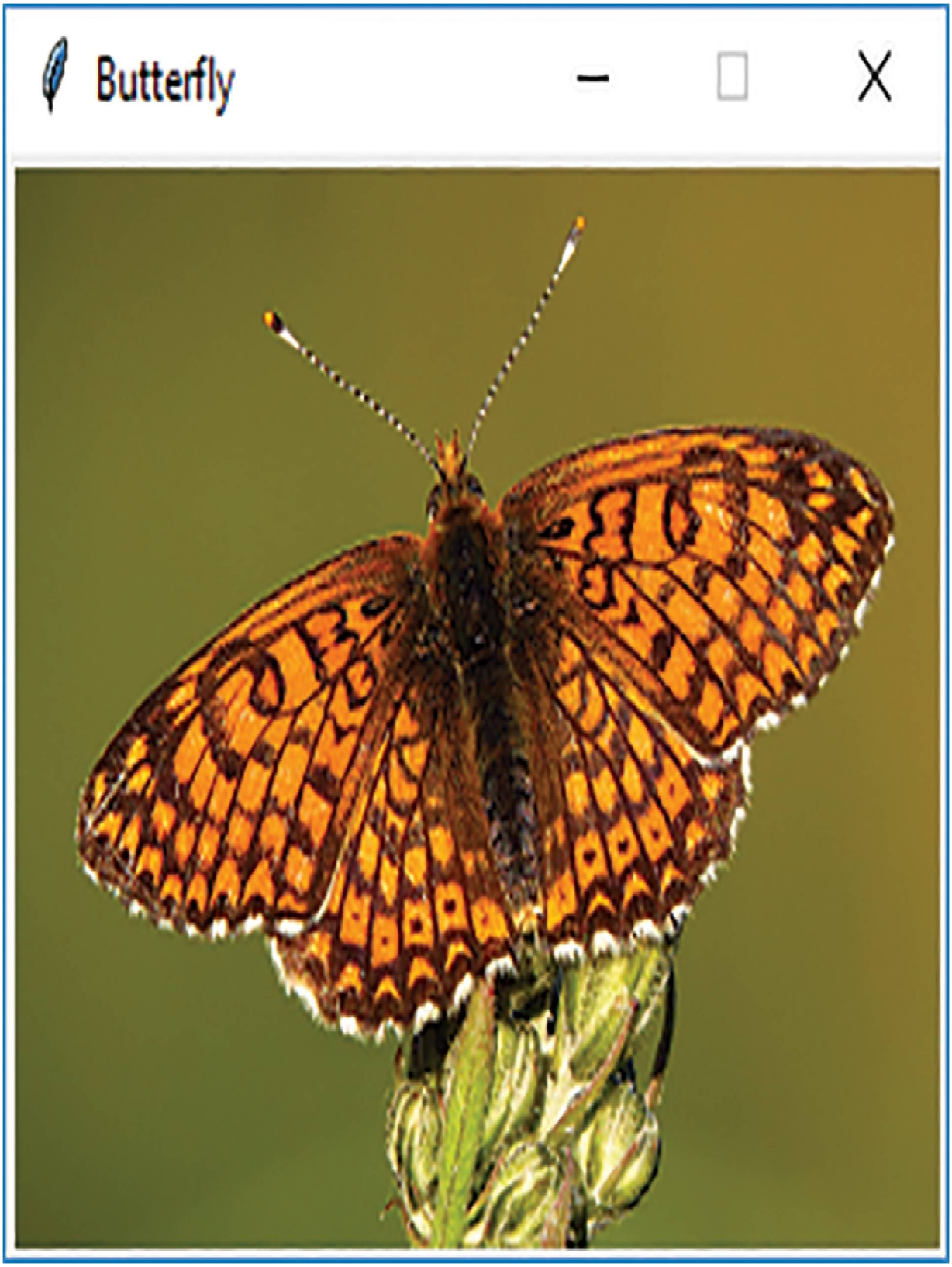
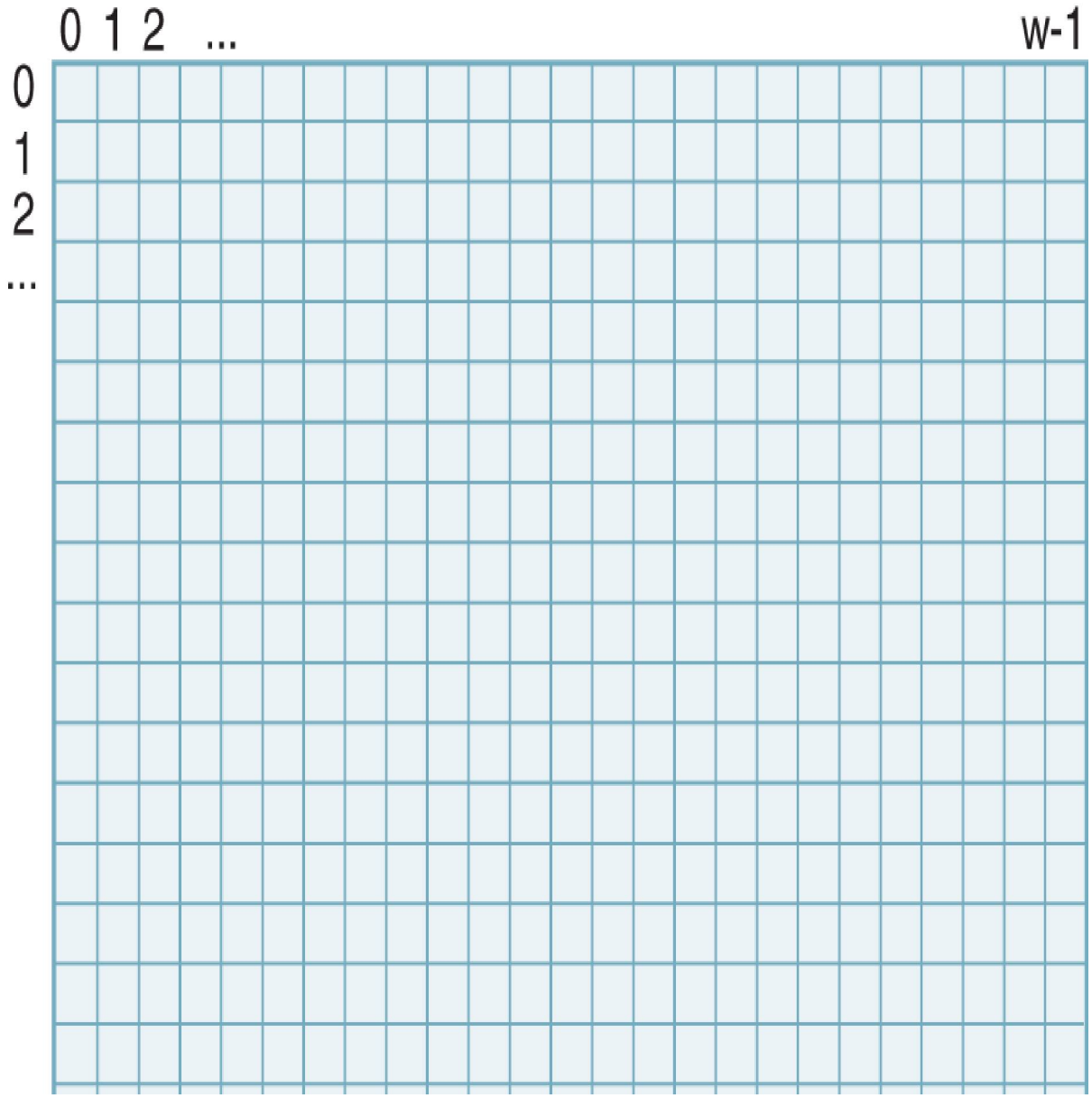


Figure 6.1 Drawing an image in a window.

© 2001–2025 Python Software Foundation; All Rights Reserved. (Original image: © Luka Hercigonja/Shutterstock.)

As noted earlier, an image is a two-dimensional grid of pixel values. Each small square in **Figure 6.2** represents a pixel that can take on any one of the millions of colors in the RGB color model. We can access information about an image by using the `getWidth` and `getHeight` methods (see **Session 6.3**). The image in Figure 6.1 is 300 pixels across and 224 pixels from top to bottom. Rows are numbered from 0 to 1 less than the height of the image. Columns are numbered from 0 to 1 less than the width.



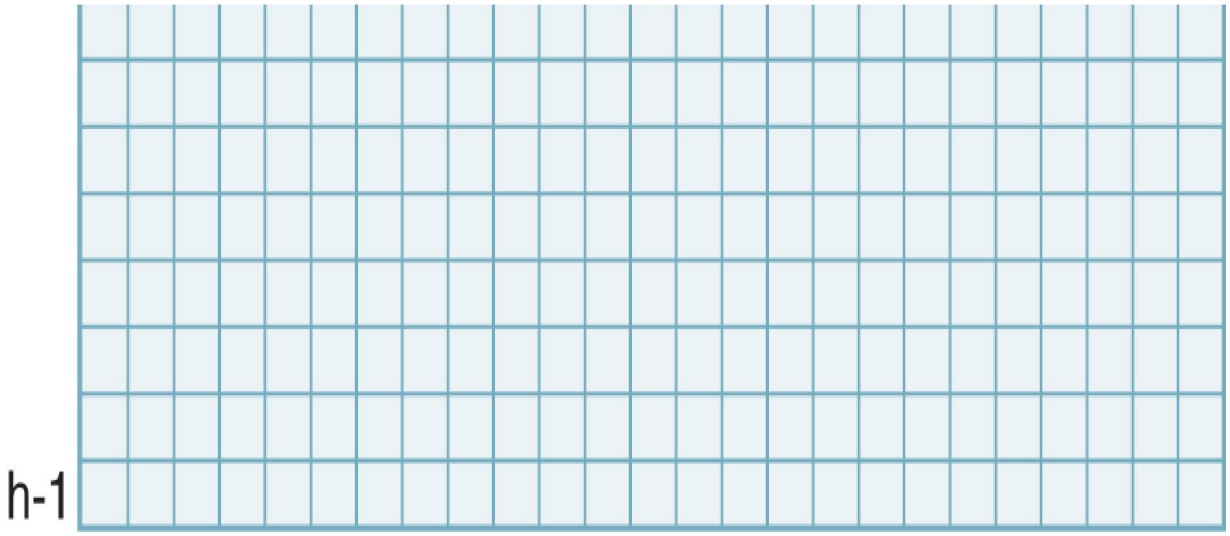


Figure 6.2 Details of an image.

```
In [1]: from cImage import *

In [2]: butterfly = FileImage("butterfly.png")

In [3]: butterfly.getWidth()

Out[3]: 300

In [4]: butterfly.getHeight()

Out[4]: 224

In [5]: butterfly.getPixel(124, 165)

Out[5]: (86, 38, 25)
```

Session 6.3 Accessing information about an image

Description

We can access a particular pixel by using the `getPixel` method. To use `getPixel`, we must provide the location of the pixel of interest. The location will be a pair of values, one specifying the column and one specifying the row. Each unique column–row pair provides access to a single pixel.

In Session 6.3, we are accessing the pixel located at column 124, row 165. In other words, this pixel is 125 pixels from the left and 166 pixels from the top—remember that we start counting at 0. The value that is returned in our example shows the red, green, and blue components of the `Pixel` at that location in our image.

RECAP:
Pixels in an image are referenced by column and row, with the column and the row both starting at 0.

The `EmptyImage` Object

We often want to build a new image pixel by pixel, starting with a "blank" image. Using the `EmptyImage` constructor, we can create an image that has a specific width and height, but where each pixel is black. In other words, each `Pixel` object has the value `(0, 0, 0)`. The statements in **Session 6.4** create an empty image with all black pixels.

```
In [2]: from cImage import *

In [4]: myImWin = ImageWin("Empty Image", 300, 300)

In [5]: emptyIm = EmptyImage(300, 300)

In [6]: emptyIm.draw(myImWin)
```

Session 6.4 Creating and displaying an empty image

Description

As an example to show the basic use of the image methods, we will construct an image that starts out empty (all black) and then is filled with white pixels at specific locations. **Session 6.5** begins by creating a window and an empty image that is sized to fit the window. To create a line of white pixels, we use a loop variable `i` and iterate over the range of the image height (from 0 to 299). We can call the `setPixel` method, using the value of `i` for both column and row with a pixel called `whitePixel` that has been created with its red, green, and blue components each set to 255. We draw the image in the window as shown in **Figure 6.3**. Finally, we save the image to a file using the `save` method.

```
In [1]: from cImage import *
```

```
In [2]: myImWin = ImageWin("Line Image", 300, 300)
```

```
In [3]: lineImage = EmptyImage(300, 300)
```

```
In [4]: whitePixel = Pixel(255, 255, 255)
```

```
In [5]: for i in range(lineImage.getHeight()):
            lineImage.setPixel(i, i, whitePixel)
```

```
In [6]: lineImage.draw(myImWin)
```

```
In [8]: lineImage.save("lineImage.gif")
```

Session 6.5 Using `EmptyImage`

Description

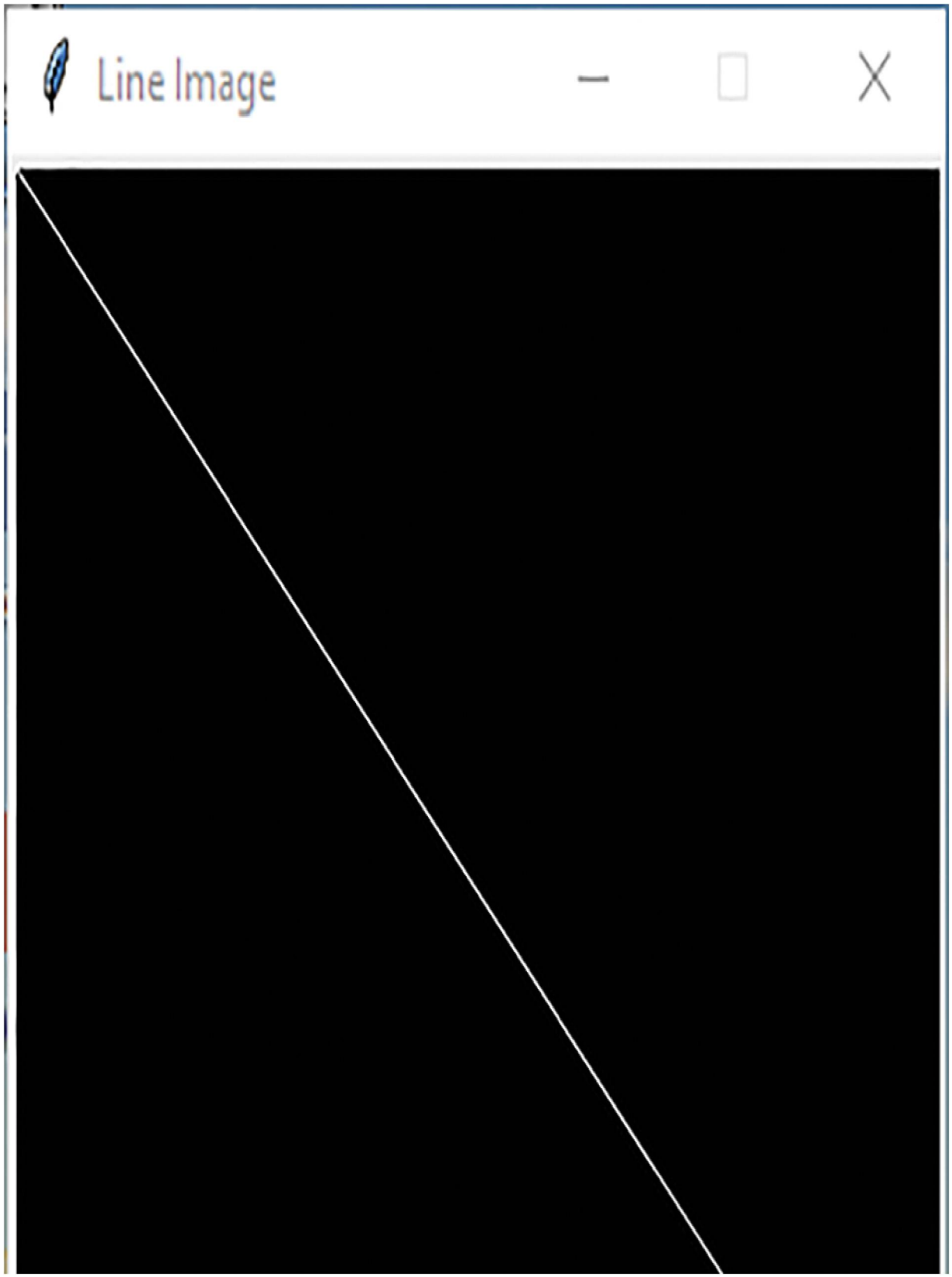




Figure 6.3 Creating a diagonal white line.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Try It Out

6.1 Modify Session 6.5 to create a line with random pixel colors.

6.2 Write a function to create an image of a rectangle. Start with an `EmptyImage`. Challenge: Create a filled rectangle.

6.3 Write a function to create an image of a circle.

6.4 Download an image from the web or from your personal image collection, and display it in a window.

6.5 Modify some pixels in the image from the previous question and save it.

6.6 Why is the number of color intensities limited to the range 0–255? Do some research to find the answer to this question.

6.3 Basic Image Processing

We now have all the tools necessary to do some simple image processing. Our first examples will perform color manipulations on an image. In other words, we want to modify the pixels in some way to change the appearance of the original image. The basic idea will be to systematically process each pixel one at a time and perform the following operations:

1. Extract the color components.
2. Build a new pixel with modified color components.
3. Place that pixel in a new image at the same location as in the original.

Once we see the general pattern, our options for image processing are endless. Note that all of the newly constructed images in this section will have the same dimensions as the original image.

6.3.1 Negative Images

When images are placed on film and then developed, a set of negatives is produced. A **negative image** is also known as a color-reversed image. In such an image, red becomes cyan, where cyan is a mixture of green and blue. Likewise, blue becomes yellow, where yellow is a mix of red and orange, and green becomes magenta, where magenta is a mix of red and blue. Regions that are white turn black, black turns white, light turns dark, and dark turns light. This color reversal continues for all possible color combinations.

At the pixel level, the negative operation is just a matter of "reversing" the red, green, and blue components in that pixel. Thus, to create a negative pixel, we just subtract each of the red, green, and blue intensity values from 255. Since color intensity ranges from 0 to 255, a pixel with a large amount of a specific color—say, red—will have a small amount of that color in the negative. At the maximum, a pixel with a red intensity of 255 will have a red intensity of 0 in the negative. The color-reversed results can then be placed in a new pixel.

Listing 6.1 shows a function that will take a `Pixel` as a parameter and return the negative pixel using the process just described. Note that the function expects to receive an entire `Pixel`; it then decomposes the color components, performs the subtractions, and builds and returns a new `Pixel`. We can easily test this function, as shown in **Session 6.6**.

```
1 def negativePixel(oldPixel):
2     newRed = 255 - oldPixel.getRed()
3     newGreen = 255 - oldPixel.getGreen()
4     newBlue = 255 - oldPixel.getBlue()
5     newPixel = Pixel(newRed, newGreen, newBlue)
6     return newPixel
```

Listing 6.1 Constructing a negative pixel

Description

```
In [1]: from negative import *

In [2]: aPixel = Pixel(155, 23, 255)

In [3]: negativePixel(aPixel)

Out[3]: (100, 232, 0)
```

Session 6.6 Testing the `negativePixel` function

Description

To create the negative image, we call the `negativePixel` function on each pixel. We then need to come up with a pattern that will allow us to process each pixel in an image. To do so, we can think of the image as having a specific number of rows equal to the height of the image. Each row, in turn, has a number of columns equal to the width of the image. With this idea in mind, we can build an iteration that will systematically move through all of the rows and, within each row, will move through all of the columns. This gives rise to the notion of **nested iteration**—placing an iteration inside of another iteration. In other words, for each pass of the "outer" iteration, the "inner" iteration will run to completion. That means that the inner iteration will run from start to finish for each pass of the outer iteration.

RECAP:
With a nested iteration, the inner iteration runs from start to finish for each pass of the outer iteration.

As an example, consider the code fragment in **Session 6.7** using `for` statements. The outer iteration moves over the list `[0,1,2]`, which was produced by `range(3)`. For each item in that list, the inner loop iterates over the characters `'c'`, `'a'`, `'t'`. As a consequence, the `print` function is called for each character in the string `"cat"` for each number in the list `[0,1,2]`.

```
In [1]: for num in range(3):
            for ch in "cat":
                print(num, ch)
```

```
0 c
0 a
0 t
1 c
1 a
1 t
2 c
2 a
2 t
```

Session 6.7 Showing nested iteration with lists and strings

Description

The resulting output shows nine lines. Each group of three represents one pass of the outer loop. Within each group, the value of `num` stays the same. For each value of `num`, the entire inner loop completes, and therefore each of the three characters of the string is output.

We can now apply this idea to construct a function to compute the negative of each pixel in an image (see **Listing 6.2**). This function will take a single parameter that gives the name of a file containing an image. It will not return anything, but will simply display both the original image and the negative image side by side.

```
1  def makeNegative(imageFile):
2      oldImage = FileImage(imageFile)
3      width = oldImage.getWidth()
4      height = oldImage.getHeight()
5
6      myImageWindow = ImageWin("Negative Image", width * 2, height)
7      oldImage.draw(myImageWindow)
8      newIm = EmptyImage(width, height)
9
10     for row in range(height):
11         for col in range(width):
12             oldPixel = oldImage.getPixel(col, row)
13             newPixel = negativePixel(oldPixel)
14             newIm.setPixel(col, row, newPixel)
15
16     newIm.setPosition(width + 1, 0)
17     newIm.draw(myImageWindow)
18     myImageWindow.exitOnClick()
```

Listing 6.2 Constructing a negative image

Description

The first step (lines 2–8) is to get the width and height of the original image and create a window that has the same height, but has twice the width as the original image, so that the negative image can be displayed next to the original. We then draw the original image in the window and create an empty image that has the same width and height as the original.

Using the idea of nested iteration (lines 10–14), we will first iterate over the rows, starting with row 0 and extending down to `height - 1`. For each row, we will process all of the columns from 0 to `width - 1`.

```
for row in range(height):
    for col in range(width):
```

Description

Each pixel is accessed by `col` and `row`. We can get the original color tuple at that location using the `getPixel` method (line 12). Once we have the original pixel, we can use the `negativePixel` function to transform it into the negative pixel. Finally, using the same `col` and `row`, we can place the new negative pixel in the new image (line 14) using the `setPixel` method. Once the iteration is complete for all pixels, the new image is drawn in the window. Note that we use the `setPosition` method (line 16) with a column value of `width + 1` to place the new image next to the original. **Figure 6.4** shows the resulting images.

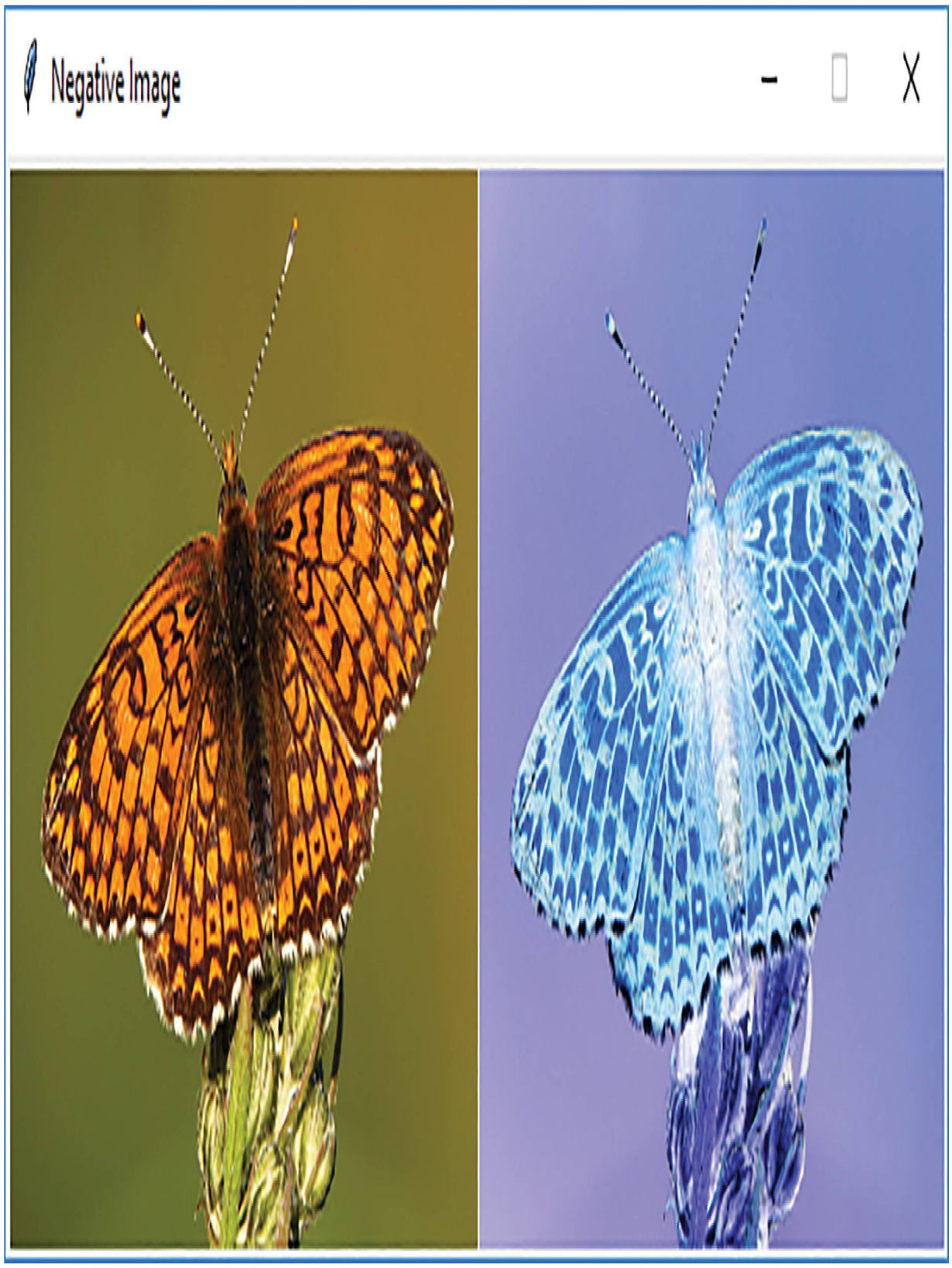


Figure 6.4 A negative image.

© 2001–2025 Python Software Foundation. All Rights Reserved. Images A and B © Luka Hercigonja/Shutterstock.

6.3.2 Grayscale

Another common color manipulation is to convert an image to **grayscale**, where each pixel will be a shade of gray, ranging from very dark (black) to very light (white). With grayscale, the red, green, and blue color components will take on the same value. In other words, there are 256 different gray color values ranging from darkest (0, 0, 0) to lightest (255, 255, 255). The standard color known as "gray" is typically coded as (128, 128, 128).

Our task, then, is to convert each color pixel into a gray pixel. The easiest way to do this is to consider that the intensity of each red, green, and blue component needs to play a part in the intensity of the gray image. If all the color intensities are close to zero, the resulting color will be very dark, which should in turn show as a dark shade of gray. In contrast, if all of the color intensities are closer to 255, the resulting color will be very light and the resulting gray should be light as well.

This analysis gives rise to a simple but accurate formula for converting a color image to grayscale: We will simply take the average intensity of the red, green, and blue components. This average can then be assigned to all three color components in a new pixel, resulting in a shade of gray. **Listing 6.3** shows a function, similar to the `negativePixel` function described previously, that takes a `Pixel` object and returns the grayscale equivalent. **Session 6.8** shows the function in use.

```
1 def grayPixel(oldPixel):
2     intensitySum = oldPixel.getRed() + oldPixel.getGreen() \
3                    + oldPixel.getBlue()
4     aveRGB = intensitySum // 3
5     newPixel = Pixel(aveRGB, aveRGB, aveRGB)
6     return newPixel
```

Listing 6.3 Constructing a grayscale pixel

Description

```
In [1]: from grayscale import *

In [2]: grayPixel(Pixel(34, 128, 74))

Out[2]: (78, 78, 78)

In [3]: grayPixel(Pixel(200, 234, 165))

Out[3]: (199, 199, 199)

In [4]: grayPixel(Pixel(23, 56, 77))

Out[4]: (52, 52, 52)
```

Session 6.8 Testing the `grayPixel` function

Description

Now the process of creating a grayscale image can proceed in the same way as described for creating the negative image (see **Listing 6.4**). After opening and drawing the original image, we create a new, empty image. Using nested iteration, we process each pixel, this time converting the pixel to the corresponding grayscale value (line 13). The final image is shown in **Figure 6.5**.

```
1  def makeGrayScale(imageFile):
2      oldImage = FileImage(imageFile)
3      width = oldImage.getWidth()
4      height = oldImage.getHeight()
5
6      myImageWindow = ImageWin("Grayscale", width * 2, height)
7      oldImage.draw(myImageWindow)
8      newIm = EmptyImage(width, height)
9
10     for row in range(height):
11         for col in range(width):
12             oldPixel = oldImage.getPixel(col, row)
13             newPixel = grayPixel(oldPixel)
14             newIm.setPixel(col, row, newPixel)
15
16     newIm.setPosition(width + 1,0)
17     newIm.draw(myImageWindow)
18     myImageWindow.exitOnClick()
```

Listing 6.4 Constructing a grayscale image

Description

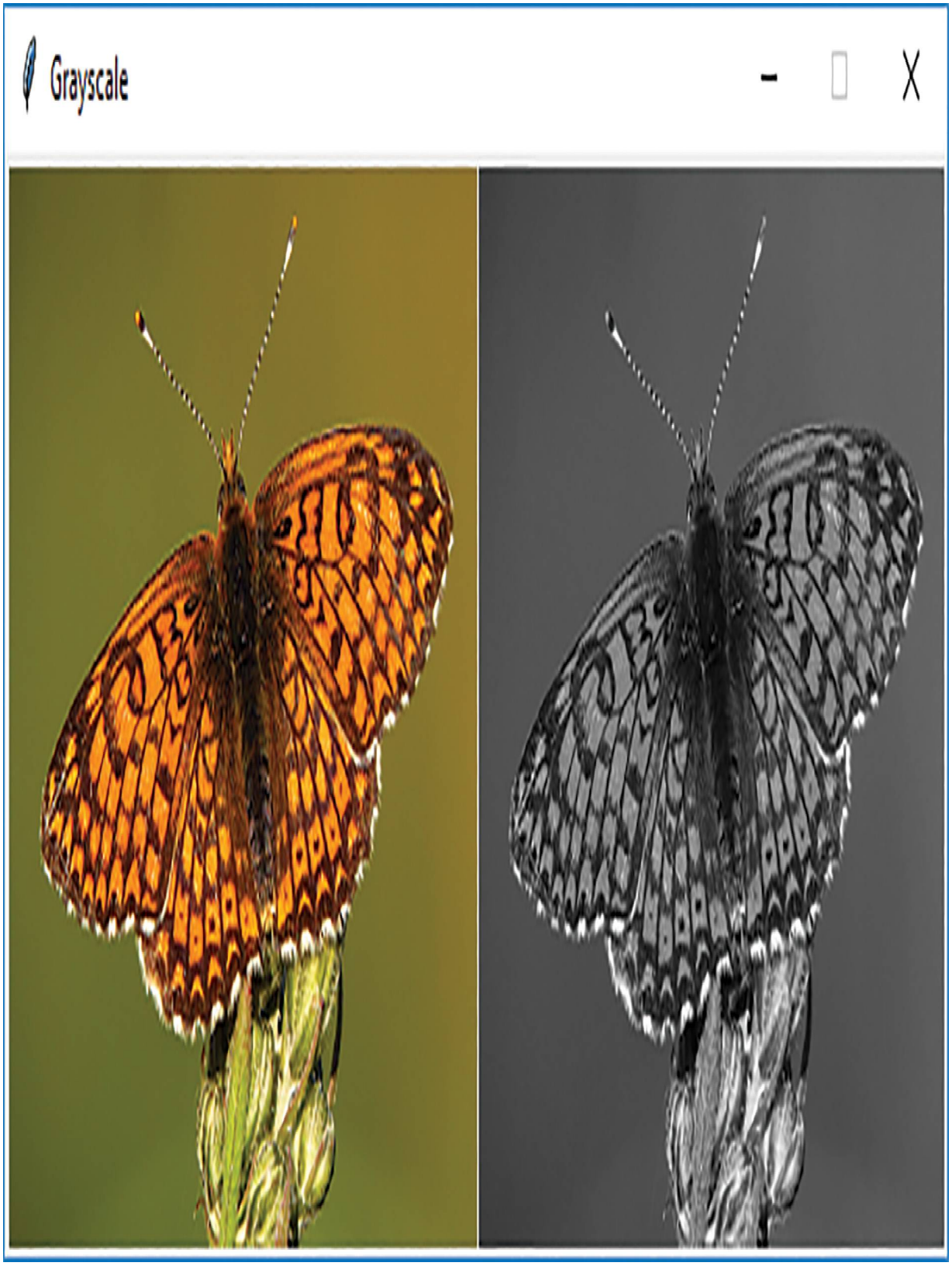


Figure 6.5 A grayscale image.

© 2001–2025 Python Software Foundation. All Rights Reserved. Images A and B © Luka Hercigonja/Shutterstock.

We developed the previous examples by continually building upon and extending a framework of simple ideas. We started with the pixel, then created a function to transform the color components of a pixel, and finally applied that function to all of the pixels in the image. This type of stepwise construction is a methodology that is widely used in writing computer programs. Building upon those functions that already work to create more complex functionality that can again be used as a foundation for additional functions allows programmers to be very efficient. We take this one step further in the next section.

GOOD PROGRAMMING PRACTICE:

To develop complex functionality, use stepwise refinement by building on functions that already work, then continuing to add your new code.

Try It Out

6.7 Write a function that removes all red from a pixel.

6.8 Write a function that enhances the red intensity of a pixel.

6.9 Write a function that diminishes the blue intensity of a pixel.

6.10 Write a function that manipulates all three color intensities in a pixel using a strategy of your choice.

6.11 Write a function that takes a colored pixel and returns a black or white pixel. *Hint:* You may want to start by converting the pixel to grayscale. Any pixel with a gray value less than some threshold will become black. All other pixels will become white.

6.3.3 A General Solution: The Pixel Mapper

If you compare the Python listings for the `makeGrayScale` and `makeNegative` functions, you will notice quite a bit of redundancy. In fact, the same steps were followed with only one exception—namely, the function that was called to map each original pixel into a new pixel. This similarity causes us to think that we could factor out the code that is the same and create a more general Python function. This is another example of using abstraction to solve problems.

GOOD PROGRAMMING PRACTICE:
Look for common code that can be factored out to create a more general solution.

Figure 6.6 shows how such a function might be constructed. We will create a function called `pixelMapper` that will take two parameters: an original image and an RGB function. The `pixelMapper` function will transform the original image into a new image using the RGB function. After applying the RGB function to every pixel, we will return the transformed image. In this way, we can create a single function that is capable of transforming an image given any function that manipulates the color intensities of a single pixel.



Figure 6.6 A general pixel mapping function.

To implement this general pixel mapper, we need to be able to pass a function as a parameter. Up to this point, all of our parameters have been data

objects such as integers, floating-point numbers, lists, tuples, and images. The question we must consider here is whether there is any difference between a function and a typical data object.

The answer to this question is simple: There is no difference. To understand why, we will first look at a simple example. The function `squareIt`, shown below, takes a number and returns the square.

```
def squareIt(n):
    return n * n
```

Description

In **Session 6.9**, we invoke the `squareIt` function with the usual syntax, placing the actual value to be squared as a parameter. However, as shown in statement 4, if we evaluate the name of the function without invoking it (without the parameters in parentheses), we notice that the result is a function definition. The name of a Python function is a reference to a data object—in particular, the function definition (see **Figure 6.7**).

```
In [1]: def squareIt(n):
            return n * n
```

```
In [2]: squareIt(3)
```

```
Out[2]: 9
```

```
In [3]: squareIt(squareIt(3))

Out[3]: 81

In [4]: squareIt        #evaluate name without parameters

Out[4]: <function __main__.squareIt(n)>

In [5]: z = squareIt    #give squareIt another name

In [6]: z(3)

Out[6]: 9

In [7]: z

Out[7]: <function __main__.squareIt(n)>
```

Session 6.9 Evaluating the `squareIt` function

Description

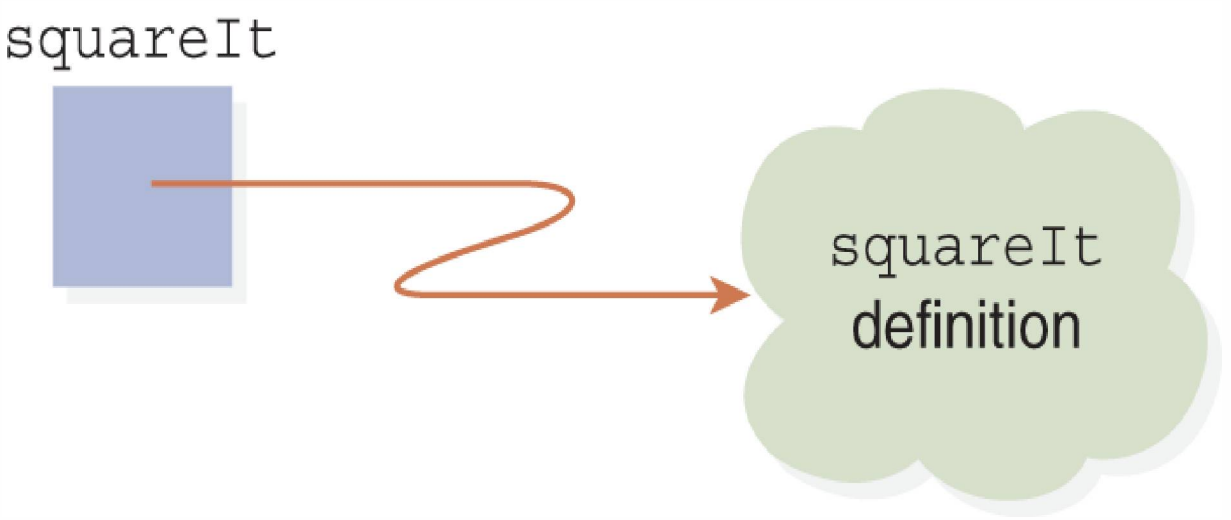


Figure 6.7 A function is a data object.

Because a function is simply another data type, you might wonder which kinds of operators you can use with it. In fact, you can use only two operators with a function. The parentheses are actually an operator that tells Python to apply the function to the supplied parameters. In addition, because a function is an object, you can use the assignment operator to give a function another name, as shown in Session 6.9. Note that now the variable `z` is a reference to the same data object as `squareIt` and can be used with the parentheses operator.

RECAP:
A function is another data type. Two operators can be used on a function: parentheses and the assignment operator.

As any Python object can be passed as a parameter, it is certainly possible to pass the function definition object. We just need to be careful not to invoke the function prior to passing it. To illustrate this point (**Session 6.10**), we create a simple function called `test` that expects two parameters: a function

object and a number. The body of `test` will invoke the function object using the number as a parameter and return the result.

```
In [1]: def squareIt(n):
            return n * n
```

```
In [2]: def test(functionParam, n):
            return functionParam(n)
```

```
In [3]: test(squareIt, 3)    #pass function name without parentheses
```

```
Out[3]: 9
```

```
In [4]: test(squareIt, 5)
```

```
Out[4]: 25
```

```
In [5]: test(squareIt(3), 5)   #actually passing (9, 5)
```

```
In [5]: test(squareIt(3), 5)  # Actually passing (2, 3)
```

```
TypeError                                 Traceback (most recent call last)
Input In [5], in <cell line: 1>()
----> 1 test(squareIt(3), 5)

Input In [2], in test(functionParam, n)
      1 def test(functionParam, n):
----> 2     return functionParam(n)

TypeError: 'int' object is not callable
```

Session 6.10 Using a function passed as a parameter

Description

We can then use our `test` function by passing the `squareIt` function definition. In addition, we will pass the integer 3. Remember, when we pass the function definition object, we do not include the parentheses pair. **Figure 6.8** shows the references immediately after `test` has been called and the parameters have been received. A copy of the reference to the actual parameter `squareIt` is received by `functionParam`, and `n` contains a reference to the object `3`.

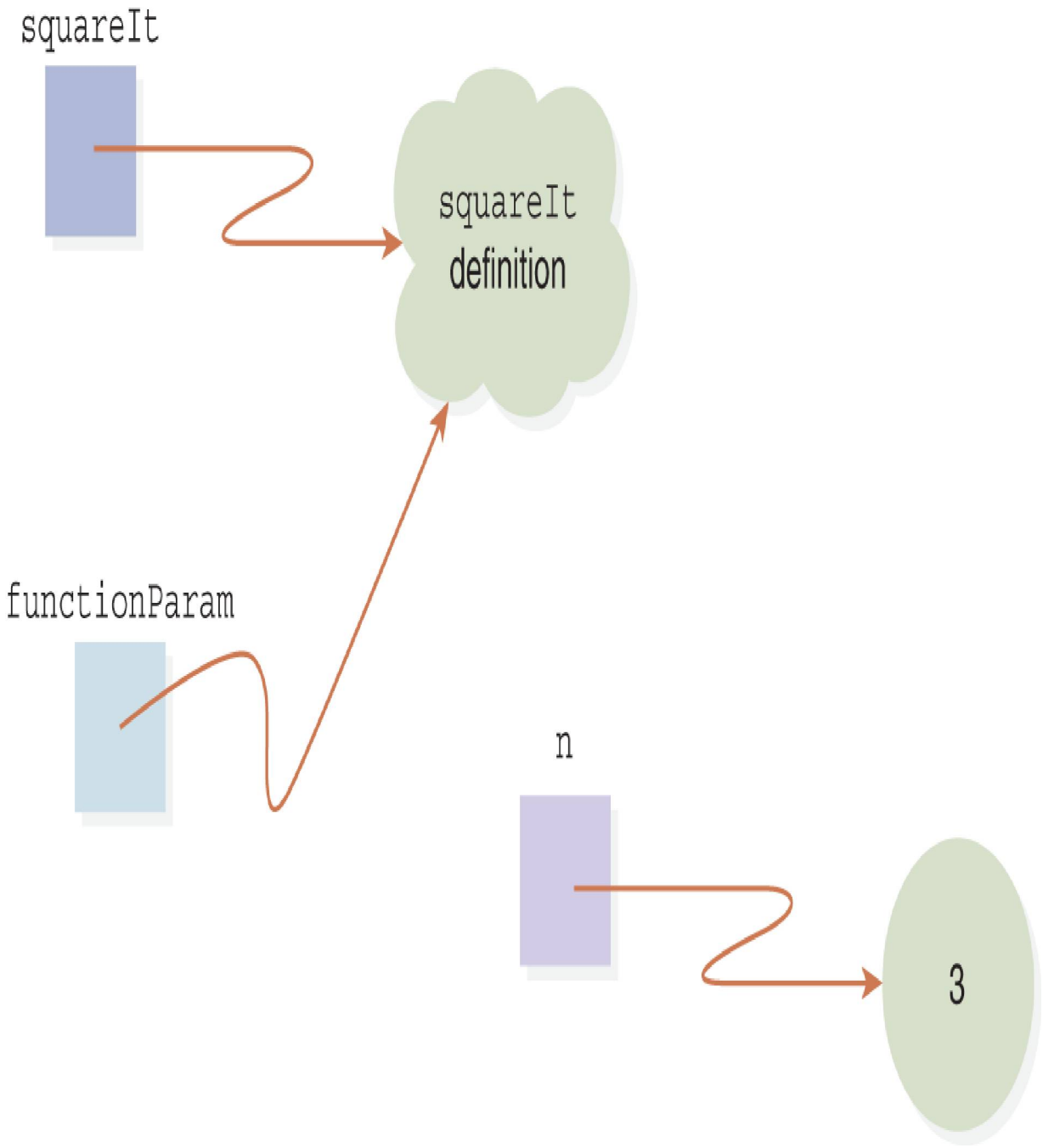


Figure 6.8 Passing the function name and an integer.

Description

The statement `return functionParam(n)` will cause the function referred to by `functionParam` to be invoked on the value of `n`. In this case, since `functionParam` is a reference to `squareIt`, the `squareIt` function

will be invoked with the parameter value of 3. The next example shows the result of passing 5; we get the correct result of 25. However, note the error message that appears after statement 5, when the first parameter is actually a call to the `squareIt` function. The result is that we pass the return value instead of passing the function object.

HEADS UP:
When passing a function name as a parameter, do not include parentheses. This syntax would call the function and pass the return value instead of the function object.

With these mechanics in place, it is now possible to implement our general `pixelMapper` function (see **Listing 6.5**). Recall that this function takes two parameters: a `FileImage` and an RGB function. Lines 3–5 construct a new empty image that is the same size as the original image. On line 10, the `rgbFunction` parameter is applied to each pixel from the original image. Once the nested iteration is complete, we return the new image.

```
1  def pixelMapper(fileImage, rgbFunction):
2
3      width = fileImage.getWidth()
4      height = fileImage.getHeight()
5      newIm = EmptyImage(width, height)
6
7      for row in range(height):
8          for col in range(width):
9              oldPixel = fileImage.getPixel(col, row)
10             newPixel = rgbFunction(oldPixel)
11             newIm.setPixel(col, row, newPixel)
12
13     return newIm
```

Listing 6.5 A general pixel mapping function

Description

We can complete our example by calling the `pixelMapper` function, using one of our RGB functions from the previous sections. **Listing 6.6** shows the main function, `generalTransform`, that sets up the image window and loads the original image. Line 9 invokes `pixelMapper` using the `grayPixel` function. The result is identical to that shown in Figure 6.5.

```
1  def generalTransform(imageFile):
2      oldImage = FileImage(imageFile)
3      width = oldImage.getWidth()
4      height = oldImage.getHeight()
5
6      myImageWindow = ImageWin("Grayscale", width * 2, height)
7      oldImage.draw(myImageWindow)
8
9      newImage = pixelMapper(oldImage, grayPixel)
10
11     newImage.setPosition(oldImage.getWidth() + 1, 0)
12     newImage.draw(myImageWindow)
13     myImageWindow.exitOnClick()
```

Listing 6.6 Calling the general pixel mapping function

Description

Try It Out

6.12 Explain the error message that appears in Session 6.10.

6.13 Use the `generalTransform` and `pixelMapper` functions to create a negative image, using the `negativePixel` function developed earlier.

6.14 Write an RGB function to remove the red component from a pixel. Test this function with `pixelMapper`.

6.15 Write an RGB function to convert a pixel to black and white. Test this function with `pixelMapper`.

6.16 Write an RGB function of your choice. Test this function with `pixelMapper`.

6.17 Sepia tone is a brownish color that was used for photographs in times past. The formula for creating a sepia tone is as follows:

```
newRed = (red × 0.393 + green × 0.769 + blue × 0.189)
newGreen = (red × 0.349 + green × 0.686 + blue × 0.168)
newBlue = (red × 0.272 + green × 0.534 + blue × 0.131)
```

Write an RGB function to convert a pixel to sepia tone. *Hint:* Remember that RGB values must be integers between 0 and 255.

6.4 Parameters, Parameter Passing, and Scope

We have used functions to implement abstraction. We have broken problems down into smaller, more manageable pieces and have implemented functions that we can call again and again. In the previous section, we pushed this idea one step further by passing functions as parameters to other functions. In this section, we explore in more detail the underlying mechanics of functions and parameter passing.

Consider the function shown in **Listing 6.7**, which computes the hypotenuse of a right triangle. Using the Pythagorean theorem, $a^2 + b^2 = c^2$, this function takes in the lengths of the two shorter sides, `a` and `b`, and computes and returns the length of the long side, `c`. In the function definition, the parameters `a` and `b` are known as the **formal parameters** because they are named in the function header. **Session 6.11** shows the function in use.

```
1  import math
2  def hypotenuse(a, b):
3      c = math.sqrt(a**2 + b**2)
4      return c
```

Listing 6.7 A simple function to compute the hypotenuse of a triangle

Description

```
In [1]: from hypotenuse import *
```

```
In [2]: hypotenuse(3, 4)                    #a receives 3, b receives 4
```

```
In [2]: hypotenuse(3, 4)                    #a receives 3, b receives 4
```

```
Out[2]: 5.0
```

```
In [3]: side1 = 3
```

```
In [4]: side2 = 4
```

```
In [6]: hypotenuse(side1, side2)            #a receives 3, b receives 4
```

```
Out[6]: 5.0
```

```
In [7]: hypotenuse(side1 * 2, side2 * 2)    #a receives 6, b receives 8
```

```
Out[7]: 10.0
```

```
In [8]: hypotenuse                          #evaluate the function itself
```

```
Out[8]: <function hypotenuse.hypotenuse(a, b)>
```

Description

In the first example, references to the objects `3` and `4` are passed to the function. These are known as the **actual parameters** because they represent the "actual" data that the function will receive. As we have seen before, the parameter list `(a,b)` receives these object references one at a time, in order, from left to right. So, `a` receives a reference to the object `3`, and `b` receives a reference to the object `4`.

In the second example in Session 6.11, the actual parameters are not literal numbers but are instead names that refer to the objects `3` and `4`. Prior to the function call, Python evaluates the two names `side1` and `side2` to find the objects. Once again, `a` receives a reference to the object `3`, and `b` receives a reference to the object `4`.

6.4.1 Call by Assignment Parameter Passing

In general, the process by which a function's formal parameter receives an actual parameter value is known as **parameter passing**. There are many different ways to pass parameters, and different programming languages have chosen to use a variety of them. In Python, all parameters are passed using a single mechanism known as **call by assignment** parameter passing.

Call by assignment parameter passing uses a simple two-step process to pass data when the function is called—a process known as **invocation**. The first thing that happens is that the actual parameters are evaluated. This evaluation results in an object reference to the value of the parameter. In the first case from Session 6.11, evaluating a literal number simply returns a reference to the number itself. In the second example, evaluating a variable name returns the object reference named by that variable.

Once the evaluation of the actual parameters is complete, the object references are passed to and received by the formal parameters in the function. The formal parameter becomes a new name for the reference that is passed. In a sense, it is as if we executed the assignment statement

```
formal parameter = actual parameter
```

As a final example, consider the third invocation shown in Session 6.11. Here, the actual parameters are expressions that double the lengths of the original sides. Call by assignment parameter passing evaluates these expressions first, and then assigns the references to the resulting objects to the formal parameters `a` and `b`. The `hypotenuse` function has no idea where the references came from or how complicated the original expression might have been. Instead, the references that the function receives are simply the results of the evaluation.

Call by assignment parameter passing has some important ramifications that may not be immediately obvious. Changes to the formal parameter may or may not result in changes to the actual parameter, depending on whether the actual parameter is mutable or immutable. If the actual parameter is immutable, then changes to the formal parameter will have no effect outside the function. If the actual parameter is mutable, then changes to the object referenced by the formal parameter will be visible outside the function.

For example, if the actual parameter is a reference to the integer 3 and the function assigns a reference to the integer 5 to the formal parameter, that would not affect the value of the parameter after the function ends. It would still be 3. If the actual parameter is a reference to a list, however, then any changes the function makes to the contents of the list, including additions or deletions, will affect the list itself. However, if the formal parameter is assigned to a different list, the behavior is consistent with the behavior for integers.

RECAP:

If the actual parameter is immutable (e.g., an integer, string, or tuple), changes made to the formal parameter in the function will not change the original value. If the actual parameter is mutable (e.g., a list), changes made to the formal parameter in the function will directly affect the actual parameter.

6.4.2 Namespaces

In Python, all the names defined in a program, whether for data or for functions, are organized into **namespaces**—collections of the names that are available to be accessed at a particular time during the execution of a Python program. When we start Python, we create two namespaces. The first, called the **builtins namespace** (see **Figure 6.9**), includes all the system-defined names of functions and data types that we regularly use in Python. Names such as the built-in functions `range`, `str`, and `int` are included in this namespace. The second namespace, called the `main` **namespace**, is initially empty. Python calls these two namespaces `__builtins__` and `__main__`.

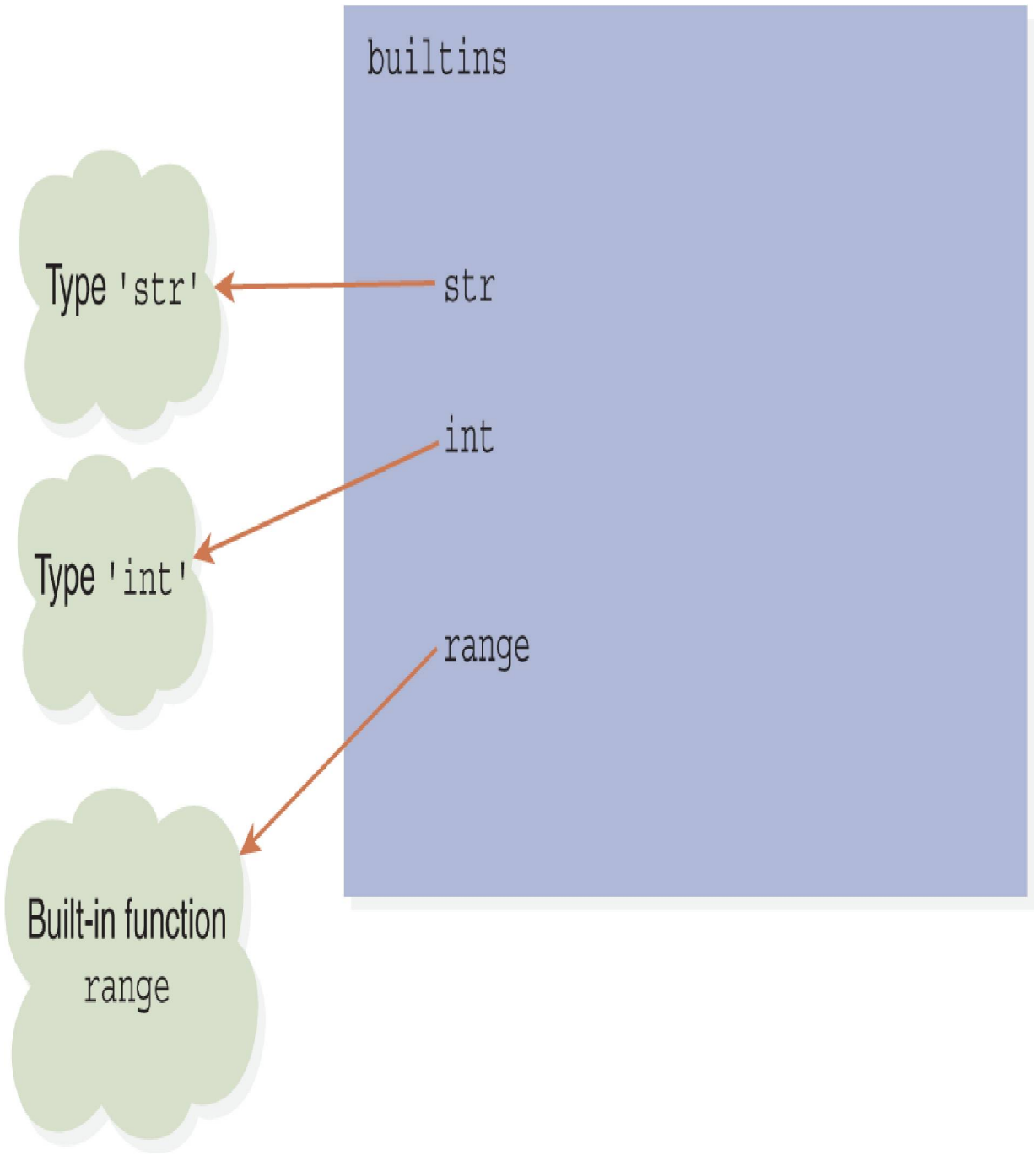
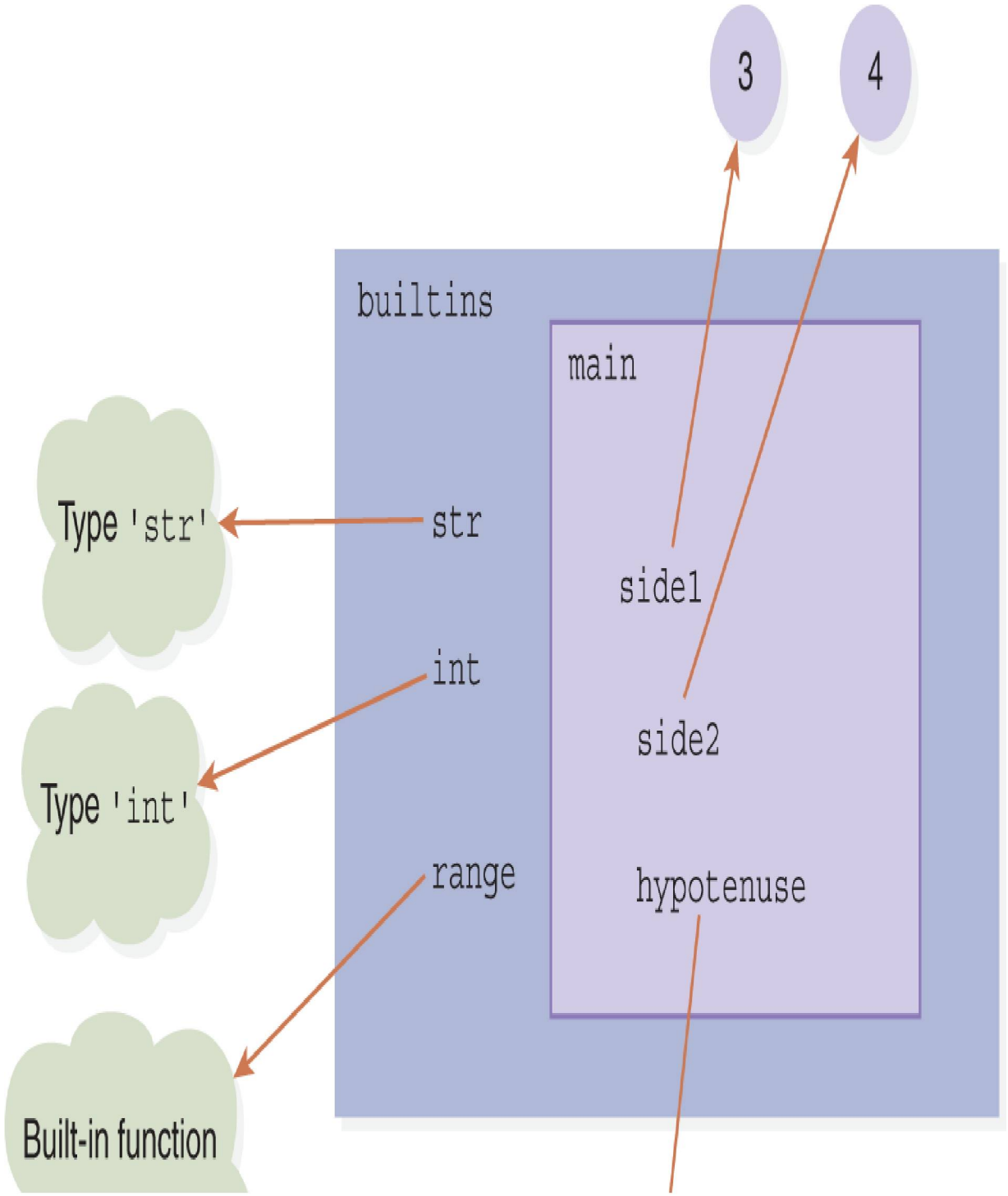


Figure 6.9 The `builtins` namespace.

Description

As we begin to create our own names in a Python session, they are added to the `main` namespace. In the example in Session 6.11, the variable names `side1` and `side2` are added to the `main` namespace. Likewise, the function

name `hypotenuse` is added to the `main` namespace. The names are added as a result of an assignment statement or a function definition; the objects referred to by the names are shown as well. **Figure 6.10** shows the location of the names and the objects they reference. Note that the objects exist outside the namespace.



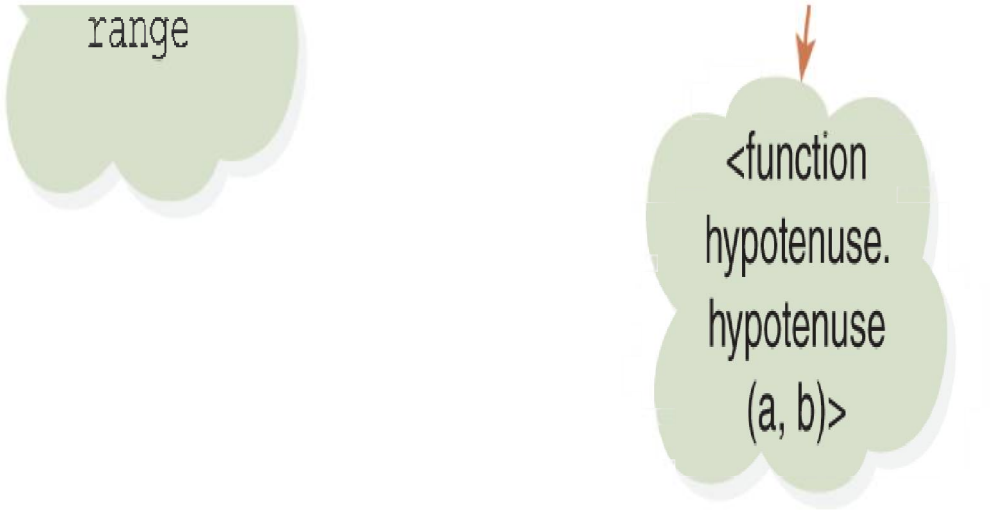


Figure 6.10 The `main` namespace.

Description

Let's investigate the namespace idea interactively. Python provides us with the function `dir`, which allows us to see a list of the names in any namespace. **Session 6.12** shows what happens when we invoke the `dir` function after defining the `hypotenuse` function and assigning values to the `side1` and `side2` variables.

```
In [1]:  1 import math
```

```
In [2]:  1 def hypotenuse(a, b):
         2     c = math.sqrt(a**2 + b**2)
         3     return c
         4
```

In [3]:
```
1 side1 = 3
```

In [4]:
```
1 side2 = 4
```

In [5]:
```
1 __name__
```

Out[5]:
```
'__main__'
```

In [6]:
```
1 print(dir())
```

```
['In', 'Out', '_', '_5', '__', '___', '__builtin__', '__builtins__',
'__doc__', '__loader__', '__name__', '__package__', '__spec__', '_dh', '_i',
'_i1', '_i2', '_i3', '_i4', '_i5', '_i6', '_ih', '_ii', '_iii', '_oh',
'exit', 'get_ipython', 'hypotenuse', 'math', 'quit', 'side1', 'side2']
```

In [7]:
```
1 math
```

Out[7]:
```
<module 'math' (built-in)>
```

```
In [8]: 1 print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb',
'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt',
'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder',
'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

```
In [9]: 1 print(math.__doc__)
```

```
This module provides access to the mathematical functions
defined by the C standard.
```

Session 6.12 Exploring the `__main__` namespace

Description

In addition to the three names that we just created, there are four other names of interest defined in our namespace. The name `__builtins__` refers to the `builtins` namespace we mentioned earlier. It includes the built-in functions, as well as Python's predefined errors. To find the names of the

objects in the `builtins` namespace, you can call the function `dir(__builtins__)`. The `__doc__` name is always available to hold a reference to a string that describes the namespace. The `__main__` namespace does not have documentation, but other namespaces such as `math` may. The `__name__` variable holds the name of the current namespace. In Session 6.12, you can see that we evaluated `__name__` to find that we are in the `__main__` namespace.

One name in the list from `dir` that we have yet to discuss is `math`. This name appears because we imported the `math` module on line 1 of the session. We import the `math` module because it provides the `sqrt` function for calculating a square root. As you can see, the name `math` refers to an object that is a module. A module defines its own namespace. Just as with the `builtins` namespace, you can see the names in the `math` namespace by evaluating `dir(math)`. Note that the `math` namespace also has its own `__name__` and `__doc__` entries.

We now have all the tools we need to understand the difference between the statements `import math` and `from math import *`. **Session 6.13** shows the result of calling the `dir` function after importing the `math` module using the statement `from math import *`. Notice that all the names from the `math` module now appear as part of the `main` namespace. This allows us to call functions such as `sqrt` directly without prefacing the function name with the module name.

```
In [1]: from math import *
```

```
In [2]: print(dir())
```

```
['In', 'Out', '_', '__', '___', '__builtin__', '__builtins__', '__doc__',
'__loader__', '__name__', '__package__', '__spec__', '_dh', '_i', '_i1',
'_i2', '_ih', '_ii', '_iii', '_oh', 'acos', 'acosh', 'asin', 'asinh',
'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh',
'degrees','dist', 'e', 'erf', 'erfc', 'exit', 'exp', 'expm1', 'fabs',
'factorial','floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'get_ipython',
'hypot','inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm',
'ldexp','lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
'nextafter','perm', 'pi', 'pow', 'prod', 'quit', 'radians', 'remainder',
'sin', 'sinh','sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

```
In [4]: sqrt(9)
```

```
Out[4]: 3.0
```

Session 6.13 Importing the `math` names into the `main` namespace

Description

Try It Out

6.18 Try calling the `dir` function on the `__builtins__` object.

6.19 Import the `turtle` module, and see what names it defines.

6.20 Look at the `__doc__` string for the `turtle` module.

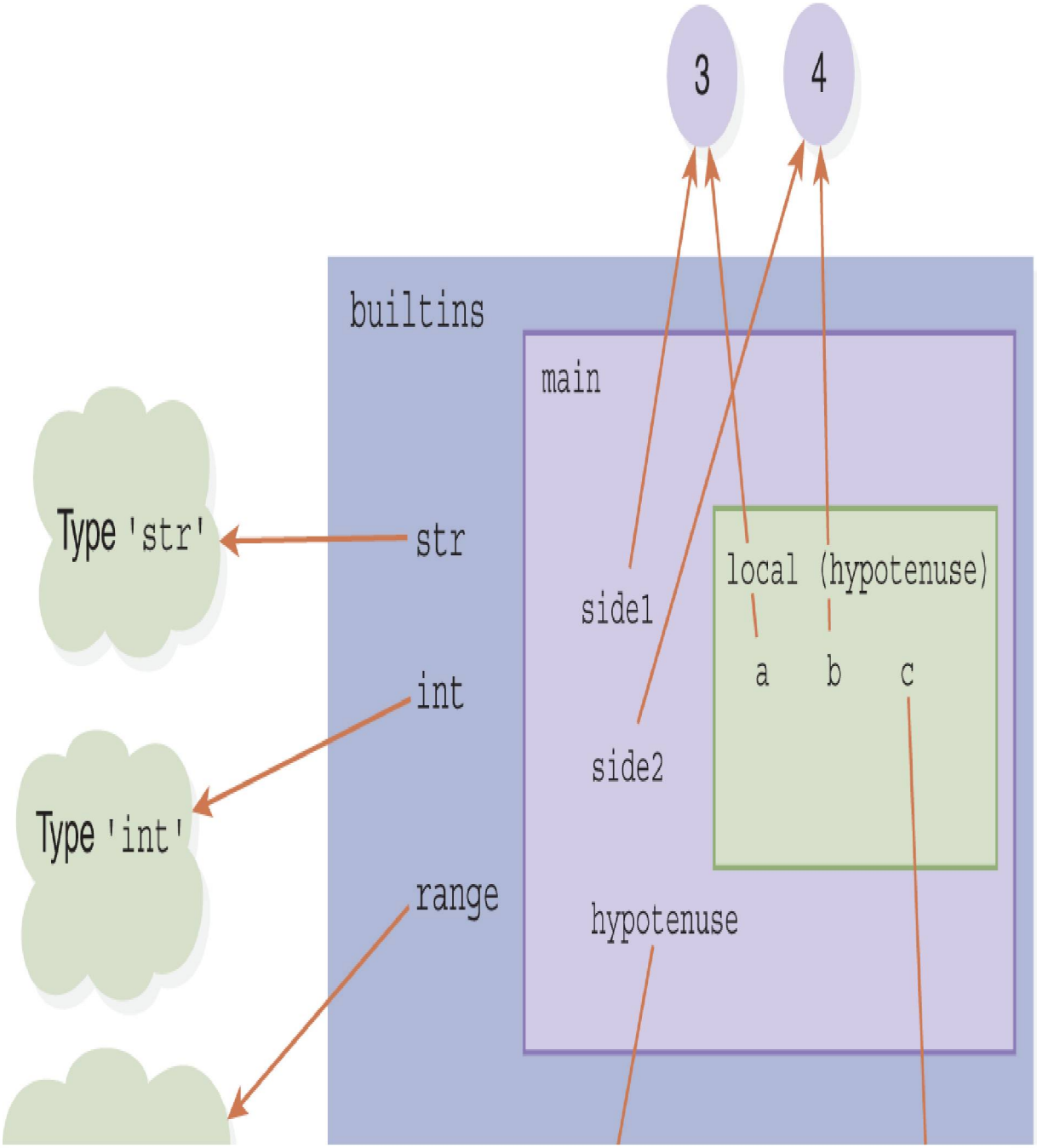
6.21 If you put a string at the beginning of any Python file, that string becomes the `__doc__` string for that module. Try adding a string to the beginning of one of your Python files. Can you import that file and see the string you added? The `help` function also returns this string as part of the documentation for a module.

6.4.3 Calling Functions and Finding Names

When a function is invoked, a new namespace known as the **local namespace** is created corresponding to the function itself. This namespace includes those names that are created inside the function—the formal parameters as well as any names that are created via an assignment statement in the body of the function. These names are referred to as **local variables** because they have been created within the function and are part of the local namespace. When the function completes, either due to a `return` statement or because it runs out of code statements, the local namespace is destroyed. In turn, all the local variables are no longer available for use.

The placement of these namespaces with respect to one another is a key consideration. The `__main__` namespace is placed inside the `__builtins__` namespace. Likewise, local namespaces are placed inside the namespace of the module where they are defined. For programs that you write, the namespaces for your functions will be placed in the `__main__` namespace. The namespaces for functions from modules that you import will be placed in the namespace of the imported module.

Figure 6.11 shows the placement of the local namespace for the `hypotenuse` function when it is called; it is just a sampling of the names in the `builtins` namespace. This figure also provides an illustration of the call by assignment mechanism. Note that the formal parameters `a` and `b` refer to the same objects as the actual parameters `side1` and `side2`, respectively. Note also that after the `hypotenuse` function ends, `a` and `b` will no longer be available, although `side1` and `side2` are still available.



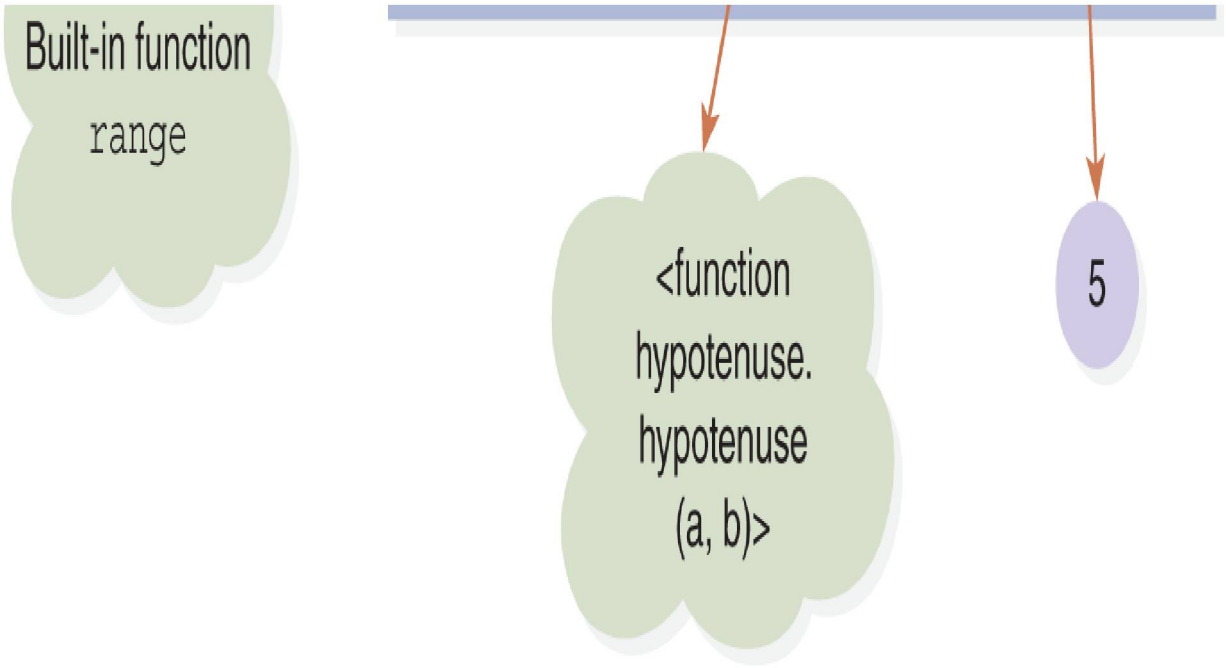


Figure 6.11 A local namespace.

Description

When a name is used in a statement, Python needs a way to locate that name among all of the names that have been introduced up to that point. To find the name, Python uses the following simple rules:

1. Whenever a name is used, except on the left-hand side of an assignment statement, Python searches through the namespaces in the following order:

   a. The current local namespace, if one exists

   b. The `main` or module namespace

   c. The `builtins` namespace

   When Python finds the first occurrence of the name, the search ends. Looking again at Figure 6.11, you may find it helpful to think of this process as searching from the "inside out." If the name is not found, a `NameError` is reported.

2. When you use a name on the left-hand side of an assignment statement, Python searches only the current namespace.

a. If the name is not found, a new name is created in the current namespace.
b. If the name is found, then the old reference is replaced with the object from the right-hand side of the assignment statement.

To show these rules in action, consider the code shown in **Session 6.14**. Here, function `test1` defines one formal parameter value called `a`. It adds 5 to `a` and prints the result. Since `a` is a formal parameter, it becomes part of the local namespace for the function `test1`.

```
In [1]: def test1(a):
            a = a + 5      #refers to a in local namespace
            print(a)
```

```
In [2]: a = 6
```

```
In [3]: test1(a)
```

```
11
```

```
In [4]: a
```

```
Out[4]: 6
```

In [5]:
```
def test2(b):
    print(b)        #refers to b in the local namespace
    print(a)        #refers to a in the main namespace
```

In [6]:
```
test2(14)
```

```
14
6
```

In [7]:
```
a                   #refers to a in main namespace
```

Out[7]:
```
6
```

In [8]:
```
b                   #test2 local namespace has been destroyed
```

```
----------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [8], in <cell line: 1>()
```

```
----> 1 b

NameError: name 'b' is not defined
```

Session 6.14 Showing name lookup rules at work

Description

Next, an assignment statement creates a variable called `a` and sets it to refer to the object `6`. This occurrence of the name `a` is added to the `main` namespace. When we invoke `test1` using `a` as the actual parameter, call by assignment parameter passing will first evaluate `a`. The result is a reference to the object `6`, which is passed and received by the formal parameter `a` in the local namespace. When the assignment statement is executed in the function `test1`, Python must search for the name `a`. The result of the search is that the `a` from the local namespace is used in the statement `a = a + 5`. The `a` in the main namespace is unaffected and still refers to the object `6`.

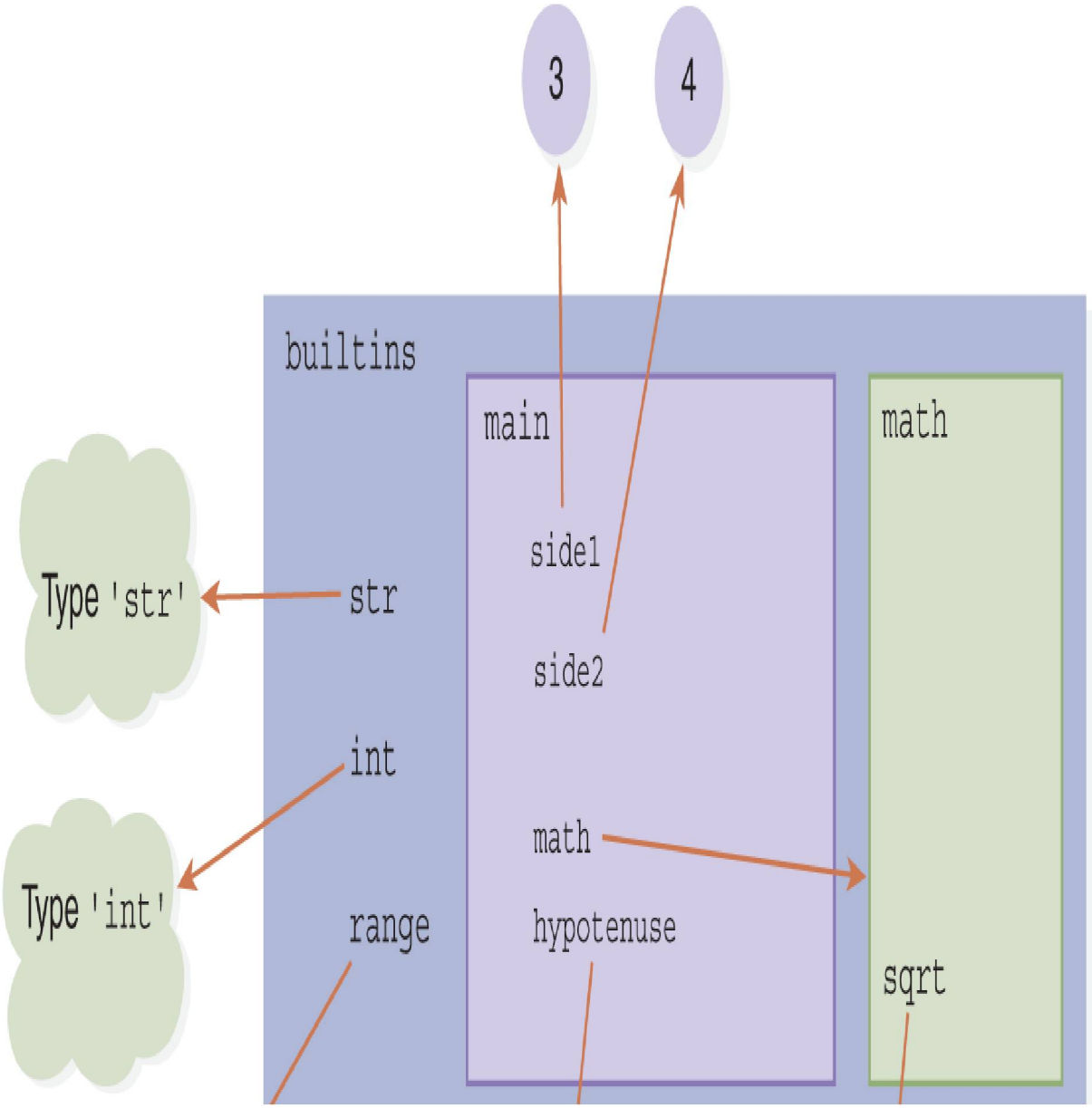
In the second example in Session 6.14, `test2` is defined with a formal parameter called `b`. This function prints `b` and then prints `a`. However, the name `a` is not defined in the local namespace for `test2`. When the `print` statements are executed, we will use the previous rules to locate the names. The result of the search is that `b` is found in the local namespace but `a` is found in the `main` namespace.

In this example, the reference to the object 14 is assigned to the formal parameter `b` in `test2` so that the first `print` statement simply prints the value 14. The second `print` statement tries to find a variable called `a`. Because it cannot be found in the local scope, the search proceeds outward to the `main` namespace, where `a` is found with a value of 6. Therefore, 6 is printed.

Note that after the call to `test2` completes, `a` still has the value 6 because it is part of the `main` namespace. Once the function `test2` completes, its namespace is destroyed and `b` is no longer present. Therefore, `b` does not exist in the `main` namespace, so an error is reported because `b` is not defined.

6.4.4 Modules and Namespaces

The `hypotenuse` function defined earlier uses the `sqrt` function from the `math` module. To access that function, we needed to import the `math` module. In Python, the statement `import math` creates a name in the current namespace with a reference to a new namespace for the module itself. In this case, the name `math` is added to the `main` namespace. The new namespace for the `math` module is placed in the `builtins` namespace, as shown in **Figure 6.12**. The `math` namespace contains names of functions such as `sqrt`. Note that the name `sqrt` refers to a function definition.



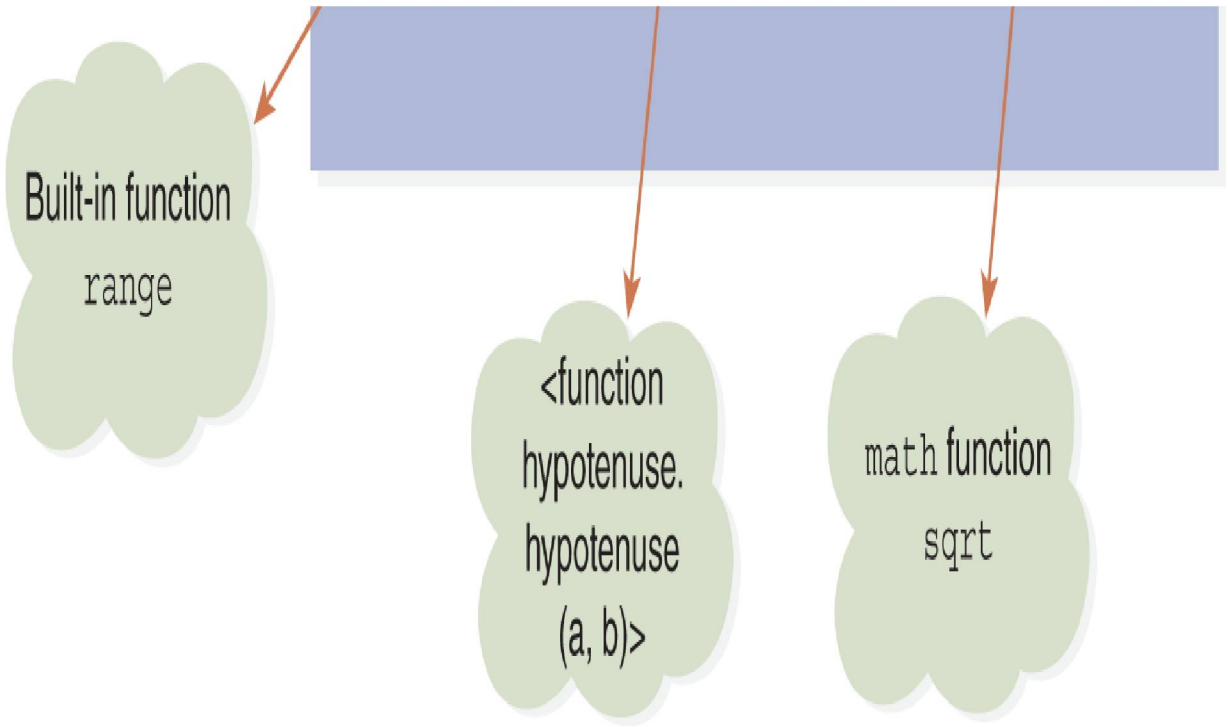


Figure 6.12 Namespaces including imported module.

Description

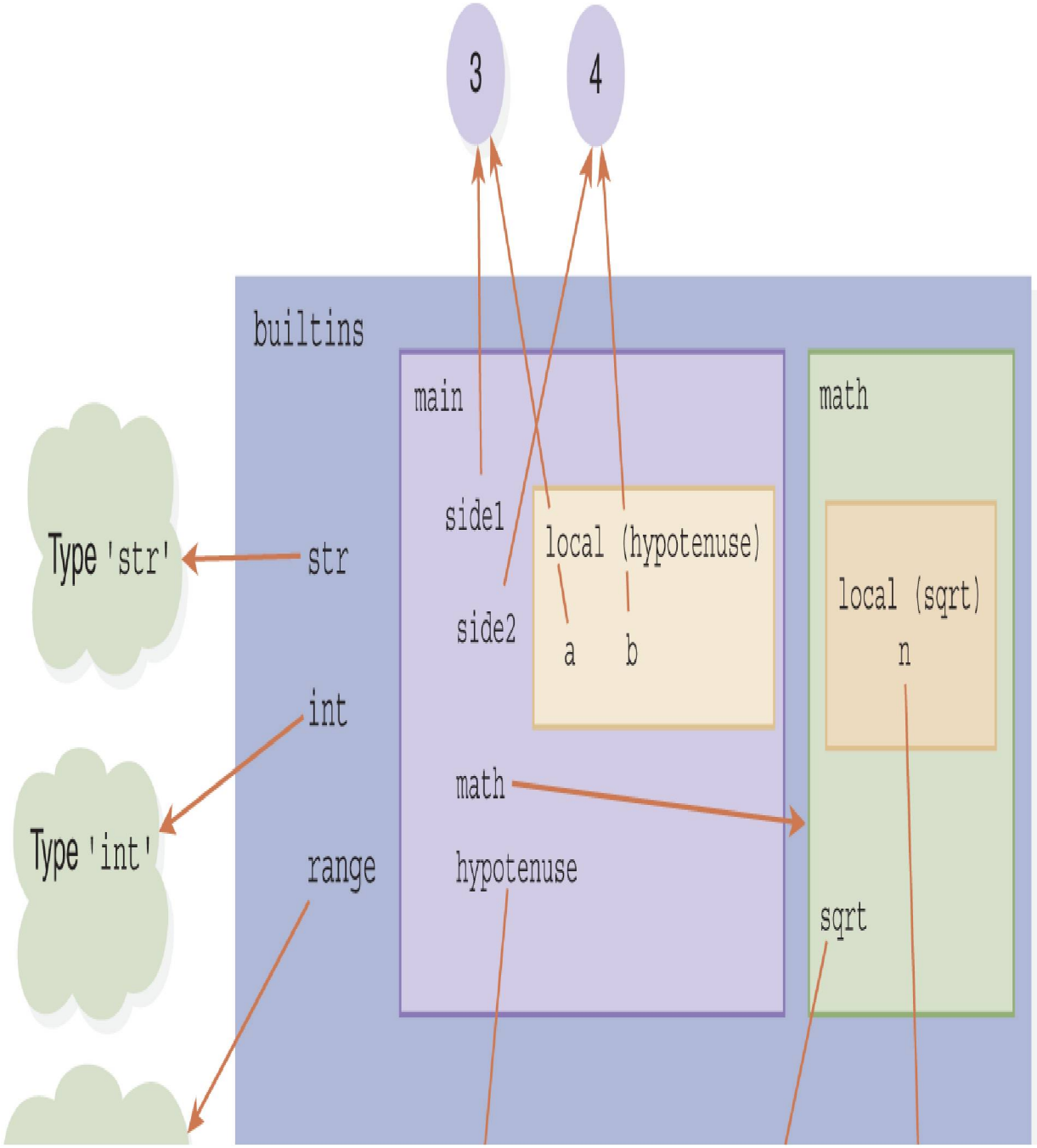
Notice that the namespace for the `math` module was placed in the `builtins` namespace, not in the `main` namespace. The namespaces for all imported modules will be placed at the same level within the `builtins` namespace. The only thing that is placed directly in the `main` namespace is the name of the module with the corresponding reference to the module namespace.

To take this one step further, consider the `hypotenuse` function at the point where the `sqrt` function has been invoked. As before, the namespace for `hypotenuse` has been placed as a local namespace in the `main` namespace. When the `hypotenuse` function calls the `sqrt` function, a new local namespace for `sqrt` is created. Even though the call to `sqrt` was made from the `hypotenuse` namespace, the `sqrt` namespace is placed in the `math` namespace because that is where the `sqrt` function was defined.

The local namespace created when a function is called is always created in the namespace for the module in which the function was defined. In our example, this is true regardless of how the `math` module is imported. Even if

we had imported `math` using `from math import *`, the namespace for `sqrt` would be placed in the `math` namespace.

Figure 6.13 shows all of the namespaces created up to this point in our example. According to the name lookup rules defined earlier, searches for names used in the `sqrt` function will start in the local `sqrt` namespace, proceed outward to the `math` namespace, and finally move to the `builtins` namespace.



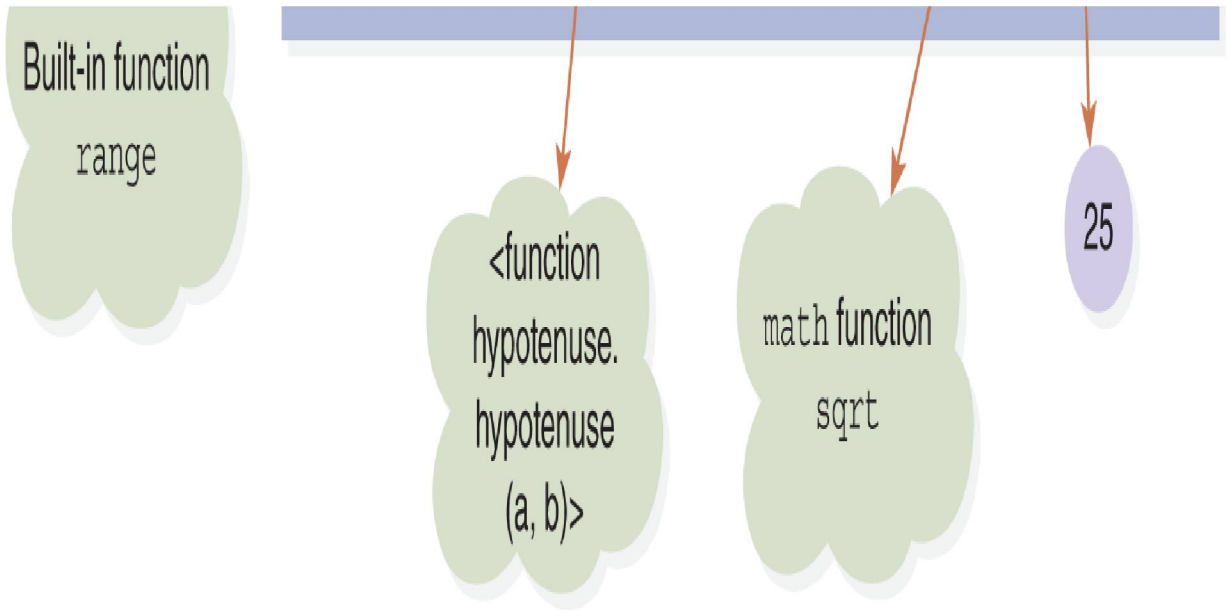


Figure 6.13 Invocation of `sqrt` function from `math`.

Description

6.5 Advanced Image Processing

We now turn our attention to some image processing algorithms that require the manipulation of more than one pixel, either in the original image or in the new image. These techniques will require us to look for additional patterns in the way we process the pixels of the image.

6.5.1 Resizing

One of the manipulations most commonly performed on images is **resizing an image**—increasing or decreasing the dimensions (width and height). In this section, we focus on enlarging an image. In particular, we consider the process of creating a new image that is twice the size of the original.

Figure 6.14 shows the basic idea. The original image is 3 pixels wide by 4 pixels high. When we enlarge the image by a factor of 2, the new image will be 6 pixels wide by 8 pixels high. This presents a problem with respect to the individual pixels within the image.

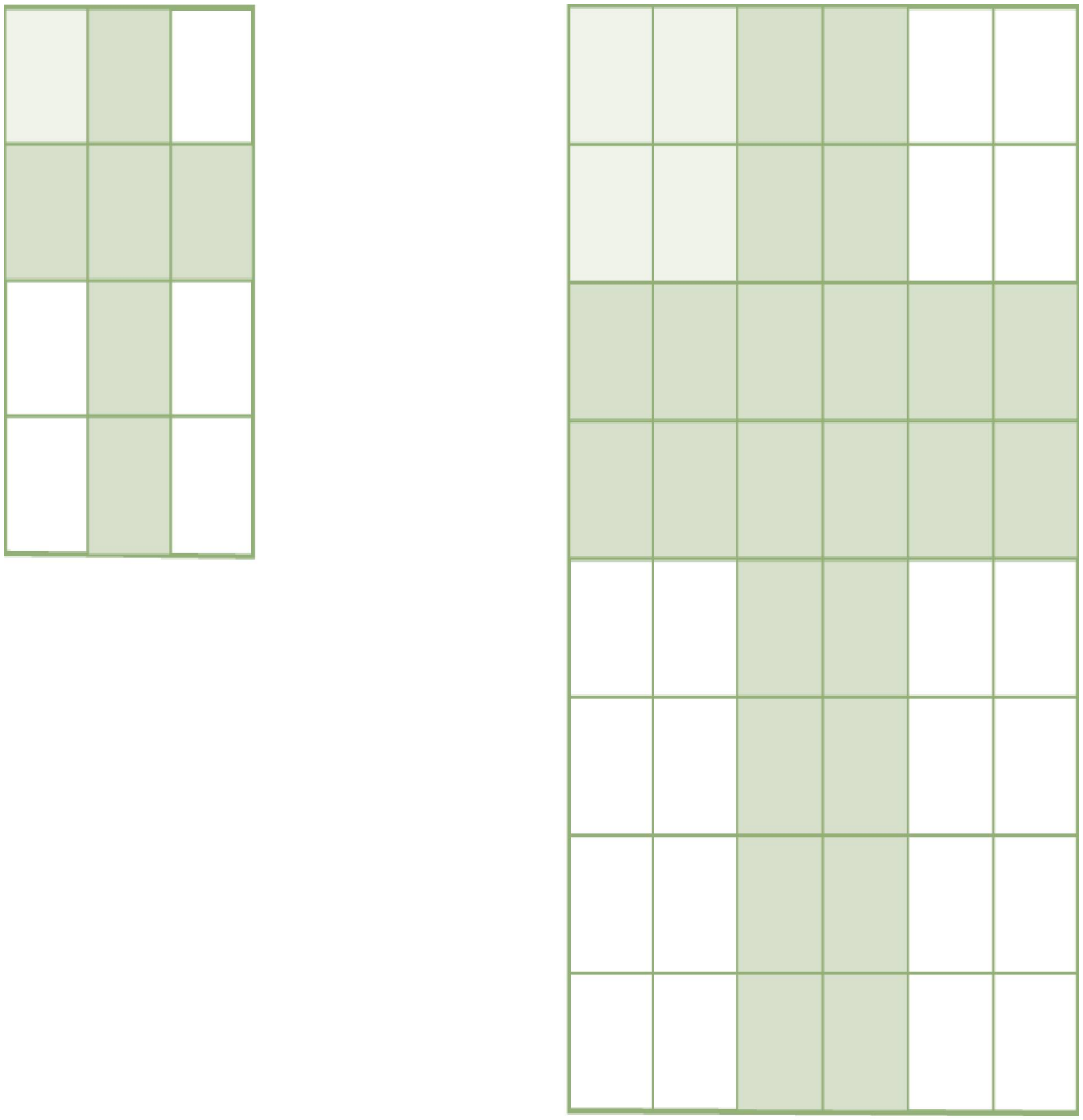


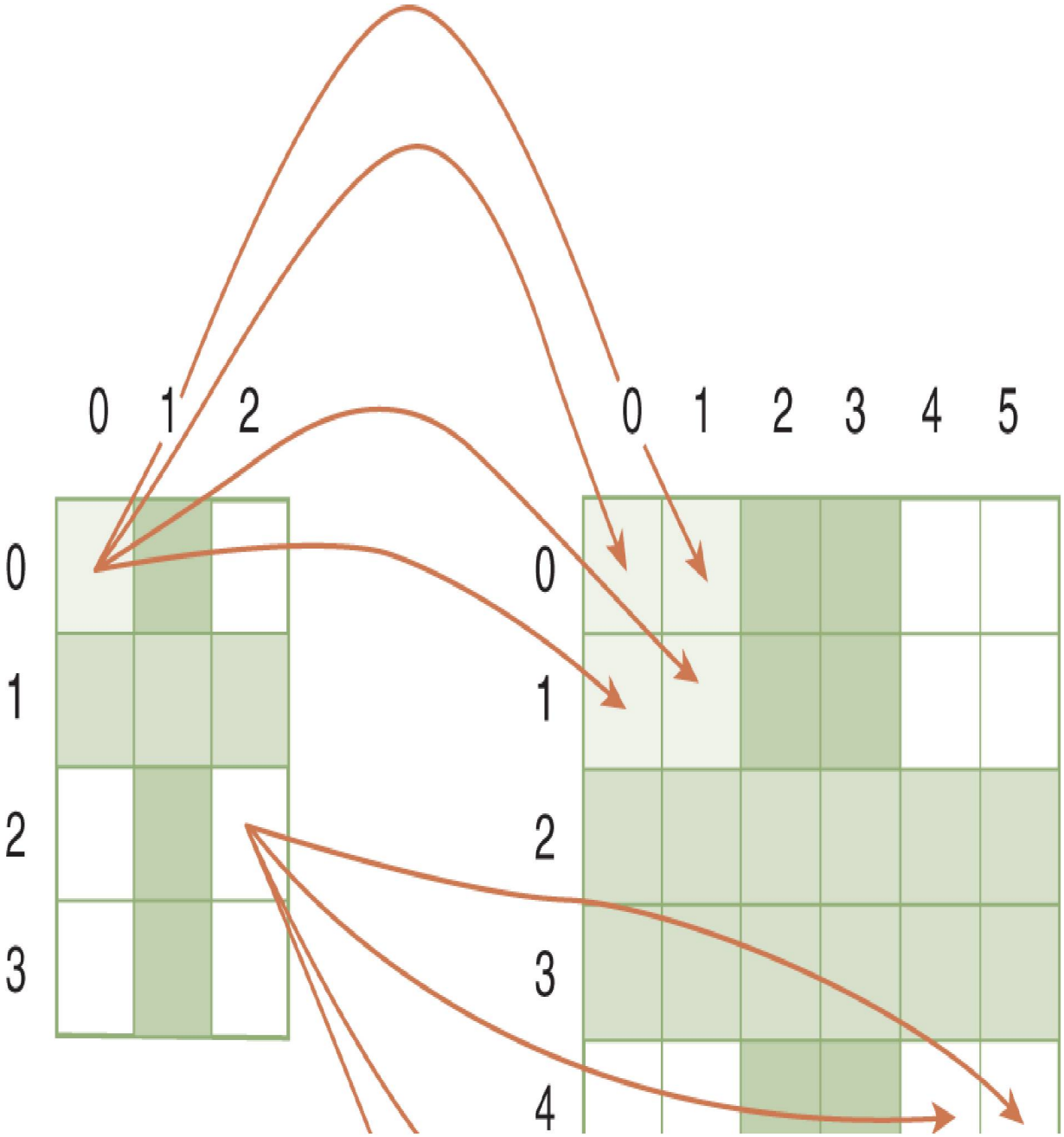
Figure 6.14 Enlarging an image by a factor of 2.

Description

The original image has 12 pixels. No matter what we do, we will not be able to create any new detail in the image. In other words, if we increase the number of pixels to 48 in the new image, 36 of the pixels must use information that is already present in the original. Our problem is to decide

systematically how to “spread” the original detail over the pixels of the new image.

Figure 6.15 shows one possible solution to this problem. Each pixel of the original will be mapped into 4 pixels in the new image. Every 1-by-1 block of pixels in the original image is mapped into a 2-by-2 block in the new image. This results in a 1-to-4 mapping that will be carried out for all the original pixels. Our task is to discover a pattern for mapping a pixel from the original image onto the new image.



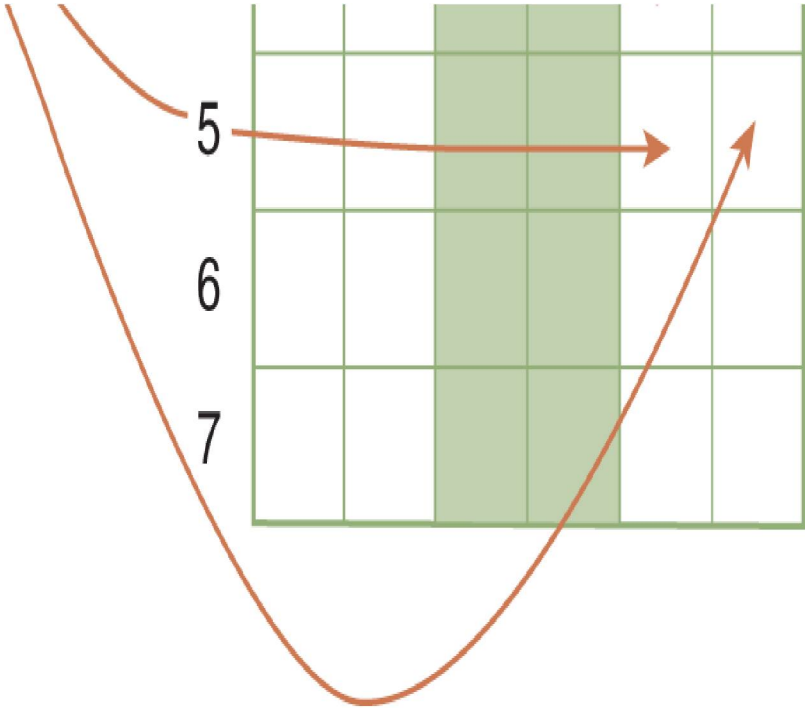


Figure 6.15 Mapping each old pixel to four new pixels.

Description

An example of this mapping process shows that pixel (0, 0) will be mapped to pixels (0, 0), (1, 0), (0, 1), (1, 1). Likewise, pixel (2, 2) maps to pixels (4, 4), (5, 4), (4, 5), (5, 5). Extending this pattern to the general case of a pixel with location `(col, row)` gives the four pixels

```
(2 × col, 2 × row),        (2 × col+1, 2 × row),
(2 × col, 2 × row+1),      (2 × col+1, 2 × row+1)
```

Description

Figure 6.16 illustrates the mapping equations for a particular pixel. You should check your understanding by considering other pixels in the original image.

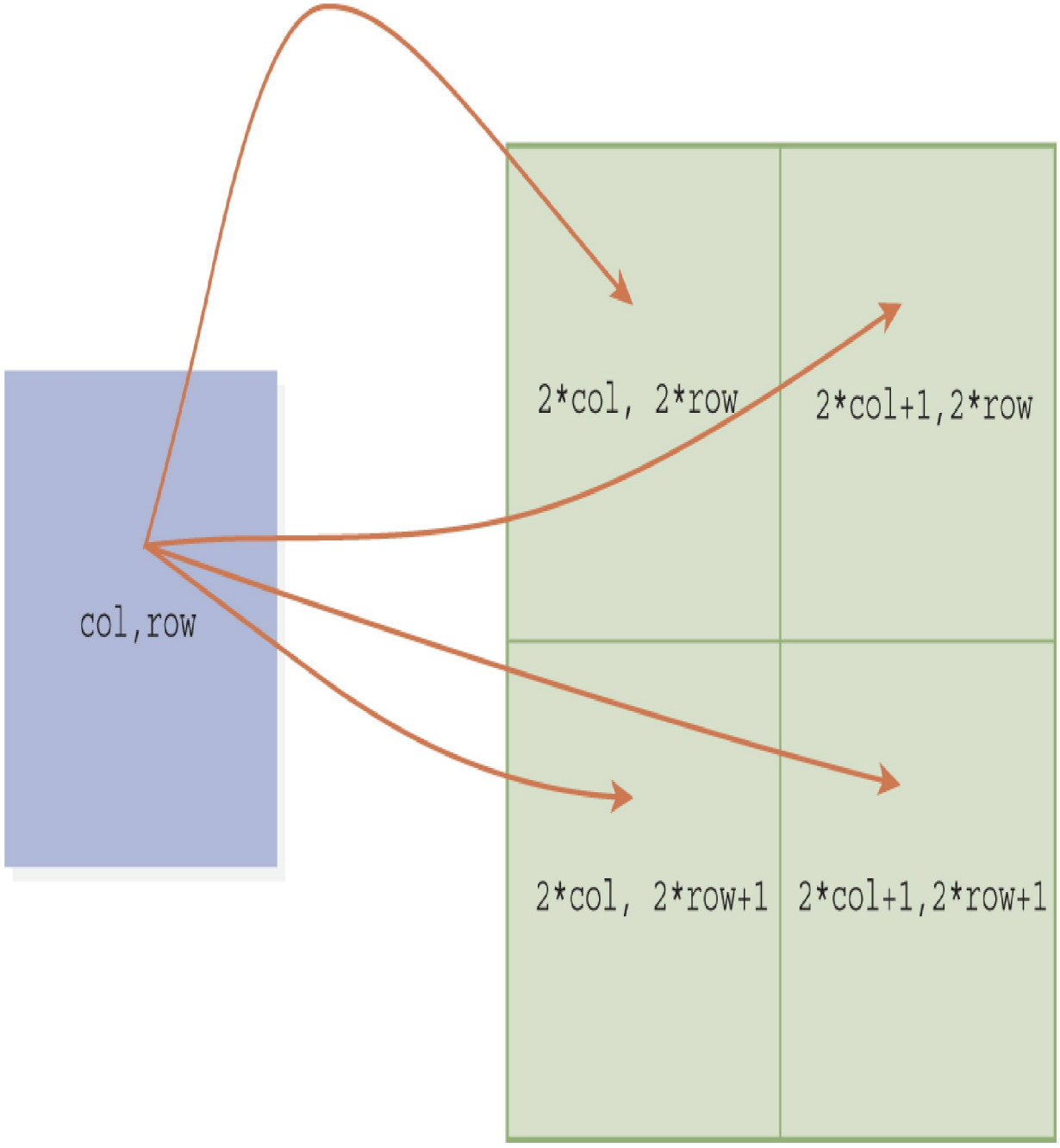


Figure 6.16 Mapping for a pixel at location `(col, row)`.

Description

Listing 6.8 shows the complete function for doubling the size of an image. Because the new image will be twice the size of the old image, it will be necessary to create an empty image with dimensions that are double those of the original (see lines 2–5).

```
1  def doubleImage(oldImage):
2      oldW = oldImage.getWidth()
3      oldH = oldImage.getHeight()
4
5      newIm = EmptyImage(oldW * 2, oldH * 2)
6
7      for row in range(oldH):
8          for col in range(oldW):
9              oldPixel = oldImage.getPixel(col, row)
10
11             newIm.setPixel(2 * col, 2 * row, oldPixel)
12             newIm.setPixel(2 * col + 1, 2 * row, oldPixel)
13             newIm.setPixel(2 * col, 2 * row + 1, oldPixel)
14             newIm.setPixel(2 * col + 1, 2 * row + 1, oldPixel)
15
16     return newIm
```

Listing 6.8 Doubling the size of an image

Description

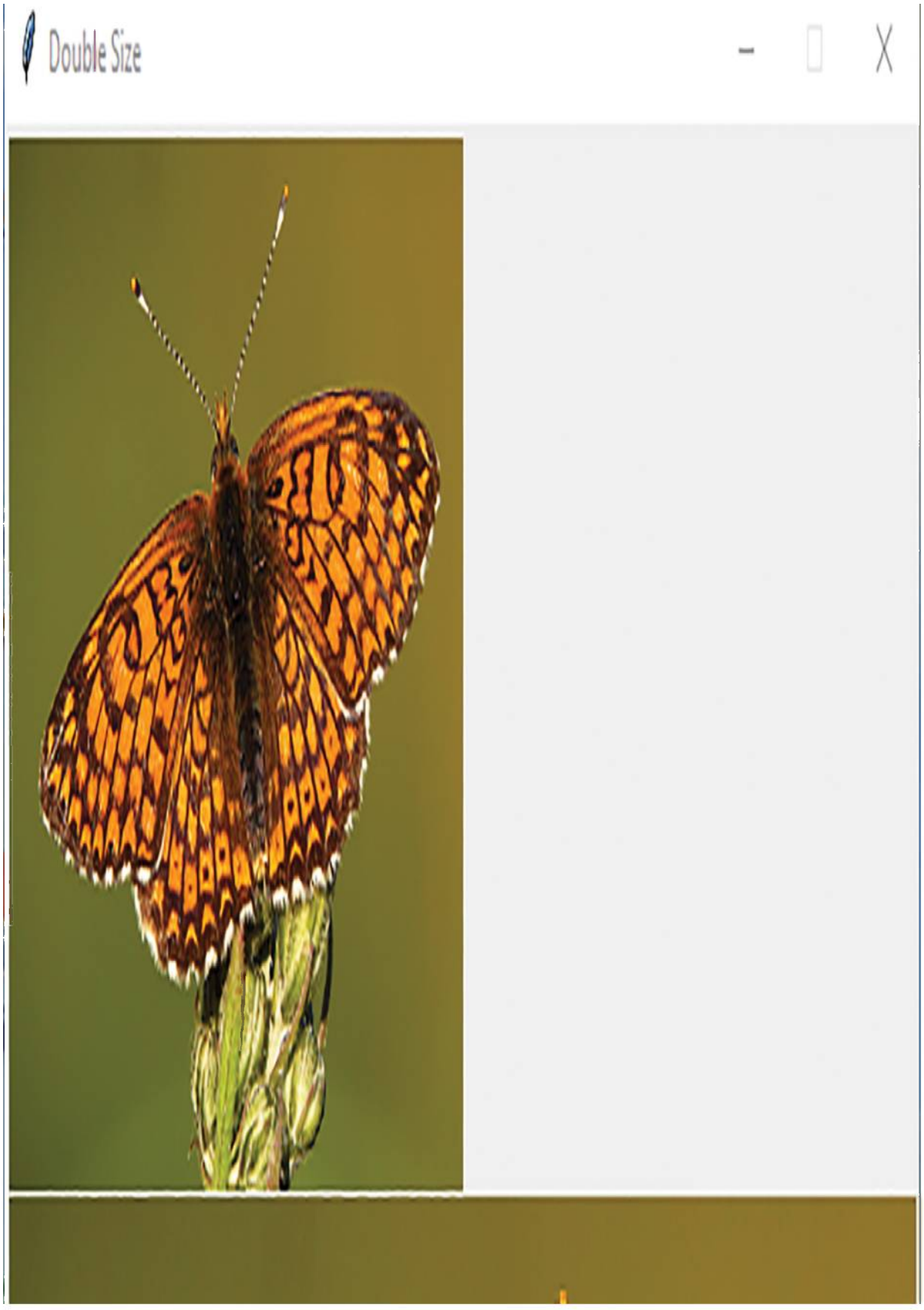
Now we can use nested iteration to process each original pixel. As before, we use two `for` loops—one for the columns and one for the rows—to

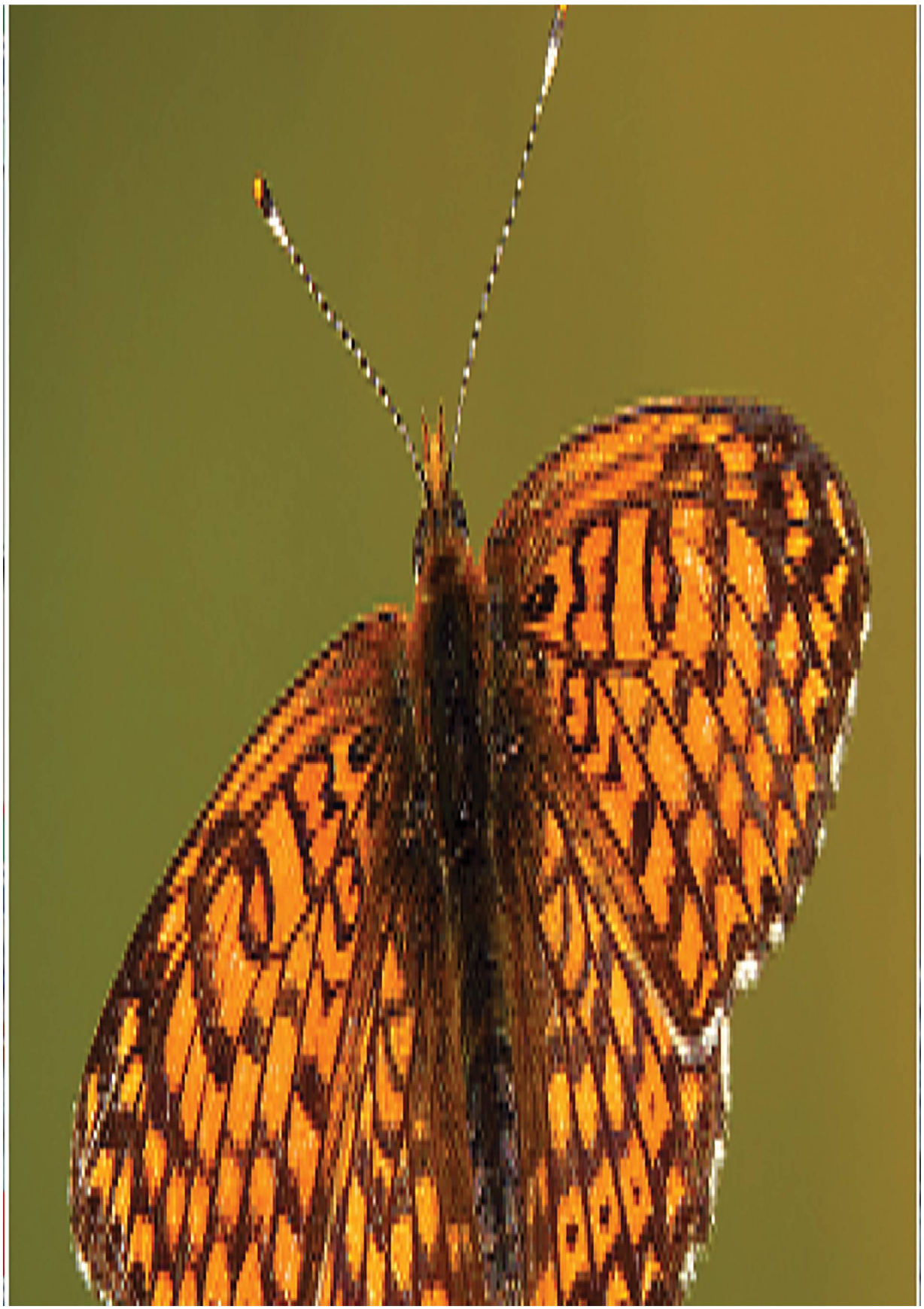
systematically process each pixel. Using the color components from each old pixel, we copy them to the new image. Lines 11–14 use the pattern discussed previously to assign each pixel in the new image. Note that each of the four pixels receives the same color tuple. In **Listing 6.9**, we call the `doubleImage` function and place the newly sized image below the original image. **Figure 6.17** shows the result.

```
1  def makeDoubleImage(imageFile):
2      oldImage = FileImage(imageFile)
3      width = oldImage.getWidth()
4      height = oldImage.getHeight()
5
6      myWin = ImageWin("Double Size", width * 2, height * 3)
7      oldImage.draw(myWin)
8
9      newImage = doubleImage(oldImage)
10     newImage.setPosition(0, oldImage.getHeight() + 1)
11     newImage.draw(myWin)
12
13     myWin.exitOnClick()
```

Listing 6.9 Calling the `doubleImage` function

Description


Double Size



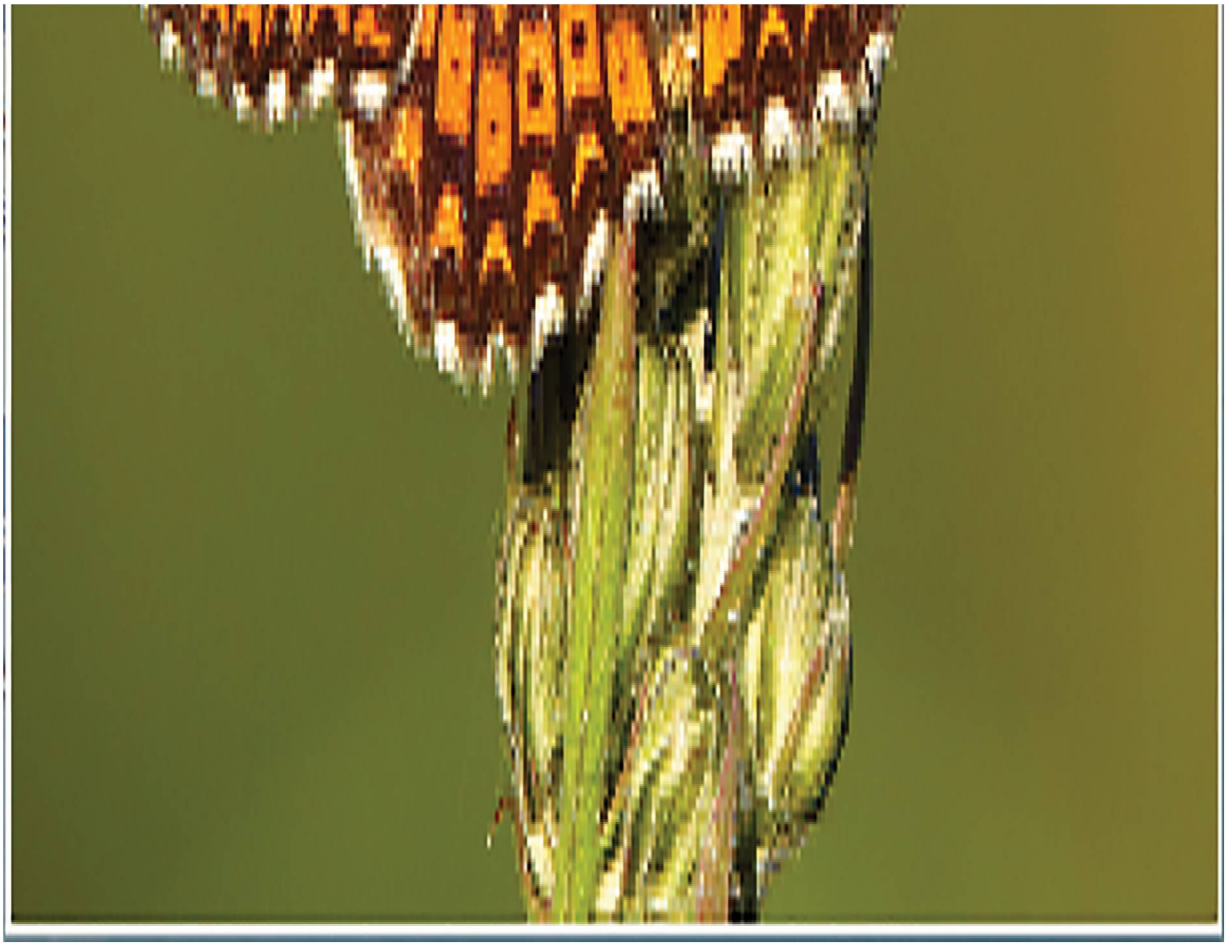


Figure 6.17 Enlarged image.

© 2001–2025 Python Software Foundation. All Rights Reserved. Images A and B © Luka Hercigonja/Shutterstock.

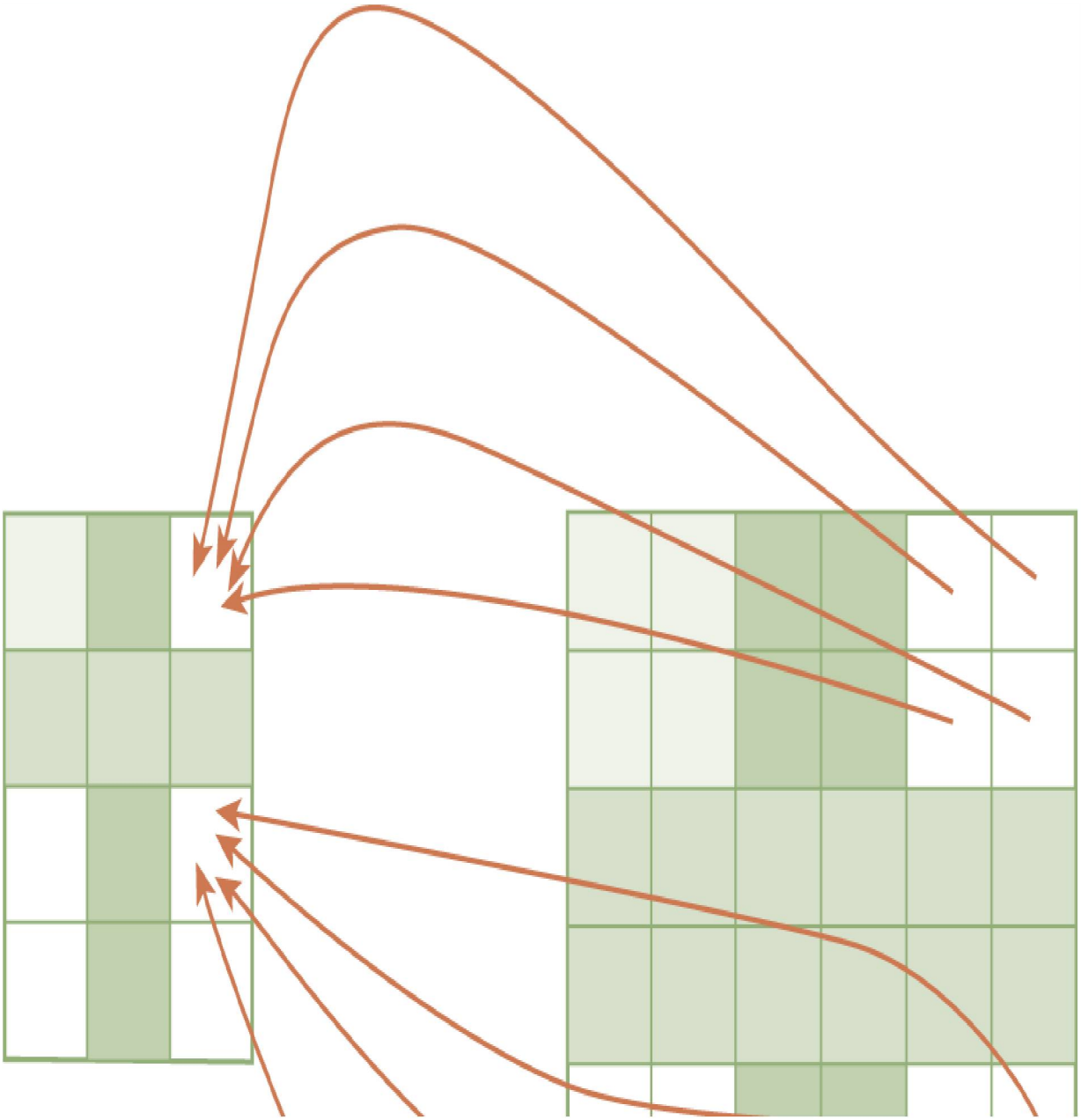
6.5.2 Stretching: A Different Perspective

The algorithm we just developed for enlarging an image requires us to map each pixel in the original image to 4 pixels in the new image. In this section, we consider an alternative solution: constructing a larger image by mapping pixels from the new image to the original image. Viewing problems from many different perspectives can often provide valuable insight. Our alternative solution takes advantage of this insight and leads to a simpler solution.

GOOD PROGRAMMING PRACTICE:

Viewing a problem from a different perspective can lead to a simpler solution.

Figure 6.18 shows the same image, but with the pixel mapping drawn in the reverse direction. More specifically, instead of looking at the problem from the perspective of the original image, we now turn our focus to the pixels of the new image. As we process the pixels of the new image, we need to figure out which pixel in the original image should be used.



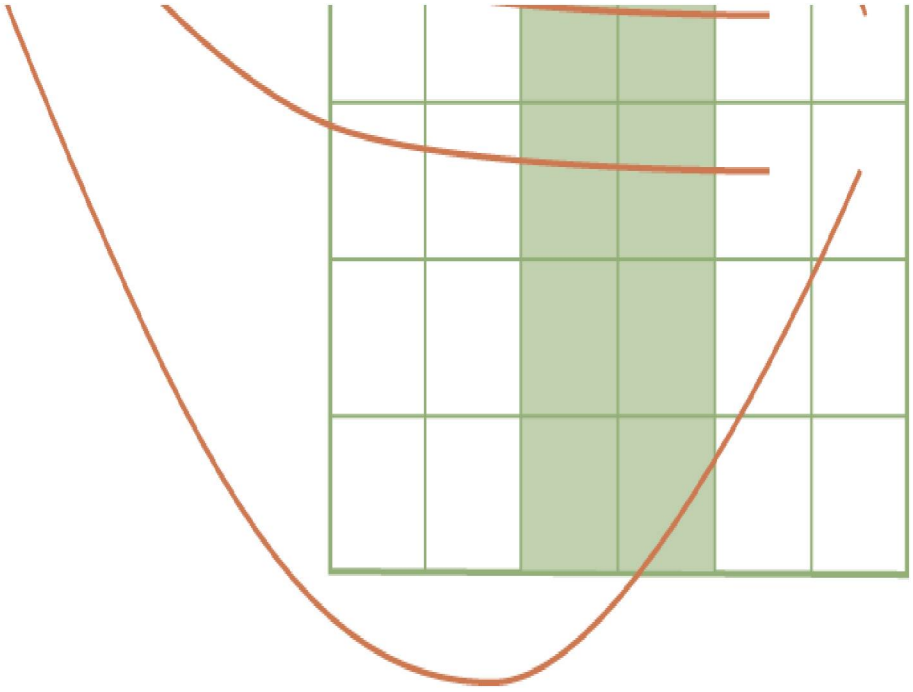


Figure 6.18 Mapping each of four new pixels back to one old pixel.

Description

Listing 6.10 shows the completed code for our new function, which will take an original image as a parameter and return the new, enlarged image. Again, we will need a new empty image that is twice the size of the original image. This time, we write our iteration to process each pixel in the new image. The nested iteration idea will still work, but the bounds will need to be defined in terms of the new image, as can be seen in lines 7–8.

```
1  def doubleImage2(oldImage):
2      oldW = oldImage.getWidth()
3      oldH = oldImage.getHeight()
4
5      newIm = EmptyImage(oldW * 2, oldH * 2)
6
7      for row in range(newIm.getHeight()):
8          for col in range(newIm.getWidth()):
9
10             originalCol = col // 2
11             originalRow = row // 2
12             oldPixel = oldImage.getPixel(originalCol, originalRow)
13
14             newIm.setPixel(col, row, oldPixel)
15
16     return newIm
```

Listing 6.10 Doubling the size of an image: mapping new back to old

Description

We now need to perform the pixel mapping. As we saw in the last section, the pixels at locations (4, 4), (5, 4), (4, 5), (5, 5) will all map back to pixel

(2, 2) in the original. As another example (see Figure 6.18), pixels (4, 0), (5, 0), (4, 1), (5, 1) will all map back to pixel (2, 0). Our task is to find the mapping pattern that will allow us to locate the appropriate pixel in the general case.

Once again, it may appear that we have four cases because four pixels in the new image are associated with a single pixel in the original. However, upon further examination, we realize that is not true. Because we are considering the problem from the perspective of the new image, there is only *one* pixel that is of interest in the original. This suggests that we can use a single operation to map each of the new pixels back to the original. Looking at the pixels in the example, we can see that integer division will perform the operation that we need (see **Figure 6.19**).

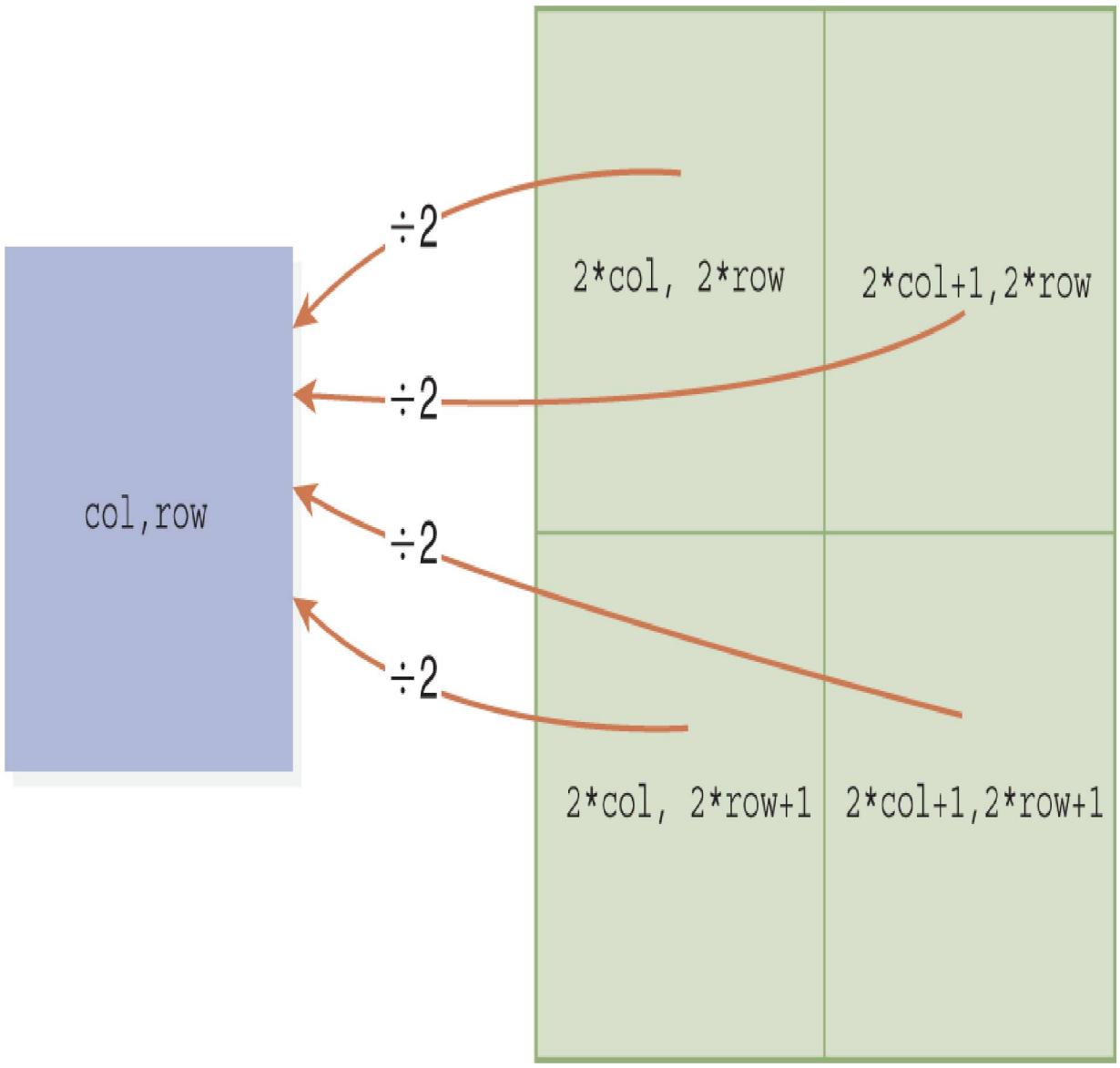


Figure 6.19 Mapping back using integer division.

Description

In our example, we need an operation that can be done to both 4 and 5 and that will result in 2. Also, the same operation on 0 and 1 will need to yield 0. Recall that both `4//2` and `5//2` give a result of 2 because the `//` operator when working on integers gives an integer result, discarding the remainder. Likewise, `0//2` and `1//2` both give 0 as their result.

We can now use this operation to complete the function. Lines 10–11 extract the correct pixel from the original image by using the integer division

operator to compute the corresponding column and row. Once the pixel has been chosen, we can assign it to the location in the new image. To double an image using our new approach, we use the same code as Listing 6.9 except that in line 9, we call the `doubleImage2` function. Of course, the result is the same as that seen in Figure 6.17.

As we stated earlier, enlarging an image provides no new detail. In some cases, the new image may look "grainy" or "blocky" because we are mapping one pixel to many locations in the new image. Although we cannot create any new detail to add to the image, it is possible to "smooth" out some of the hard edges by processing each pixel with respect to those around it.

Try It Out

6.22 Write a function to quadruple the size of an image.

6.23 Write a general function for enlarging an image that accepts a scale parameter for enlarging in the x direction and another parameter for enlarging in the y direction.

6.24 Write a function for reducing the size of an image.

6.25 Write a function that will smooth the enlarged image. *Hint:* You will want to replace each pixel in the enlarged image with the average of itself and its neighbors.

6.26 Write a function to remove noise from an image. You can do this by replacing each pixel with the median of itself and its neighbors.

6.5.3 Flipping an Image

We now consider manipulations that physically transform an image by moving pixels to new locations. In particular, we consider a process known as **flipping**. Flipping an image requires that we decide where the flip will occur. We will assume that flipping happens with respect to a line called the **flip axis**. The basic idea is to place pixels that appear on one side of the flip

axis in a new location on the opposite side of that axis, keeping the distance from the axis the same.

As an example, consider the simple image with 16 pixels in **Figure 6.20** and a flip axis placed vertically at the center of the image. Because we are flipping on the vertical axis, each row will maintain its position relative to every other row. However, within each row, the pixels will move to the opposite side of the axis, as shown by the arrows. The first pixel will be last, and the last pixel will be first.

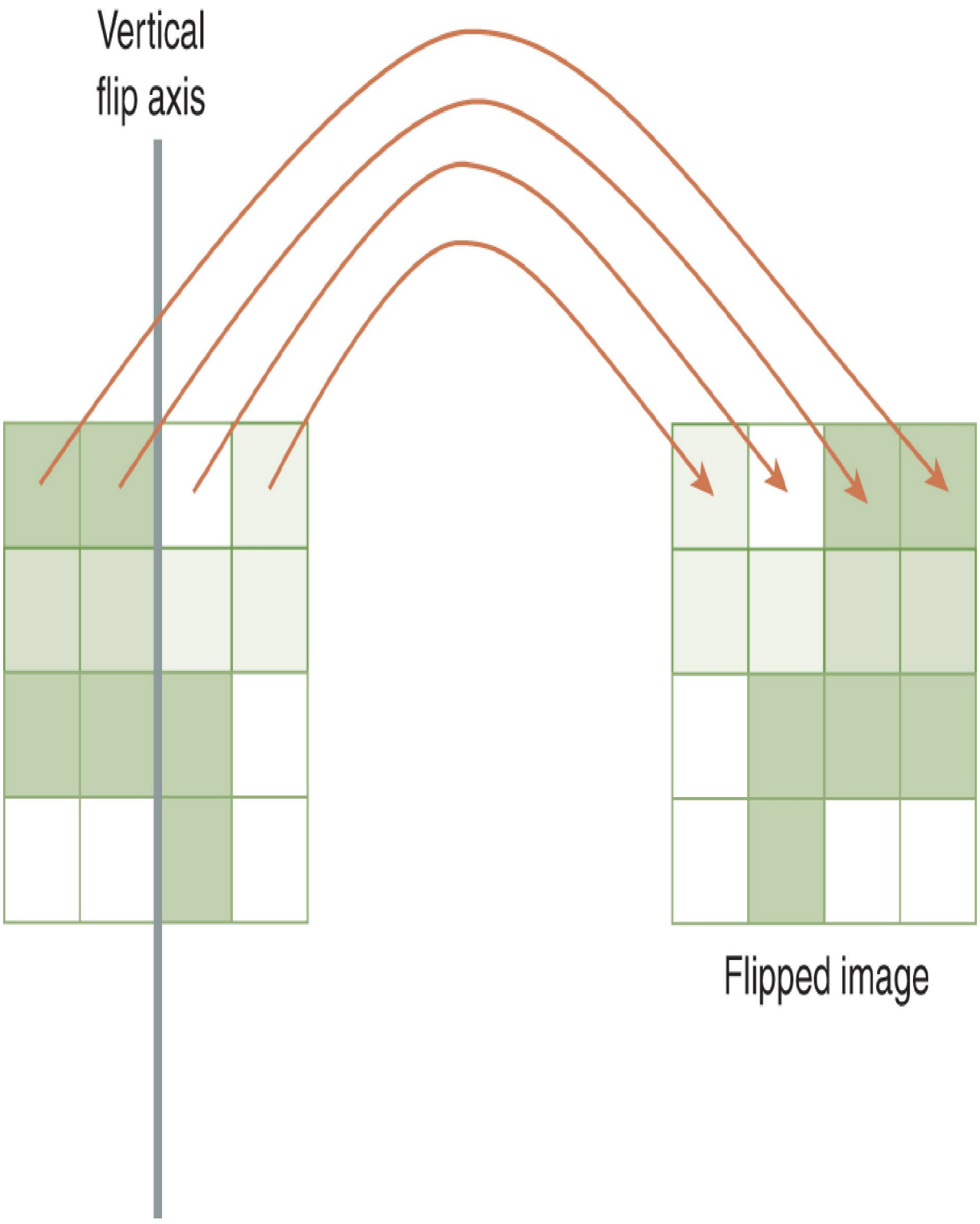


Figure 6.20 Flipping an image on the vertical axis.

Description

The structure of this function is similar to that of the other functions we have written thus far. We build our nested iteration such that the outer iteration will process the rows and the inner iteration will process each pixel within the row. **Listing 6.11** shows the completed function. Because we are flipping the image rather than resizing it, the new image will have the same height and width as the original image.

```
1  def verticalFlip(oldImage):
2      oldW = oldImage.getWidth()
3      oldH = oldImage.getHeight()
4
5      newIm = EmptyImage(oldW, oldH)
6
7      maxP = oldW - 1
8      for row in range(oldH):
9          for col in range(oldW):
10
11             oldPixel = oldImage.getPixel(maxP - col, row)
12
13             newIm.setPixel(col, row, oldPixel)
14
15     return newIm
```

Listing 6.11 Creating the vertical flip of an image

Description

We need to discover a pattern to map each pixel from its original location into a new location with respect to the flip axis. Referring to Figure 6.20, we can see that the following associations are needed in an image that is four pixels wide: Column 0 will map to column 3, column 1 will map to column 2, column 2 will map to column 1, and column 3 will map to column 0. In general, small values map to large values and large values map to small.

The first thing we might try is to use the width and simply subtract the original column to get the new column. If we try this with column 0, we immediately see a problem: $4 - 0 = 4$, which is outside the range of valid column values. This problem occurs because we start counting the columns (as well as the rows) at zero.

To fix this problem, we can base the subtraction on the actual maximum pixel position instead of the width. Because the pixels in this example are named with column 0 through column 3 (width of 4), we can use 3 as our base for the subtraction. In this case, the general mapping equation will be `(width - 1) - column`. Note that `width - 1` is a constant, which means that we can perform this calculation just once, outside the loop, as we do on line 7.

We are performing a flip using a vertical flip axis, so the pixels stay in the same row. Line 11 uses the calculation we identified to extract the proper pixel from the original image, and line 13 places that pixel in its new position in the new image. Note that `row` is used in both `getPixel` and `setPixel`. **Figure 6.21** shows the resulting image.

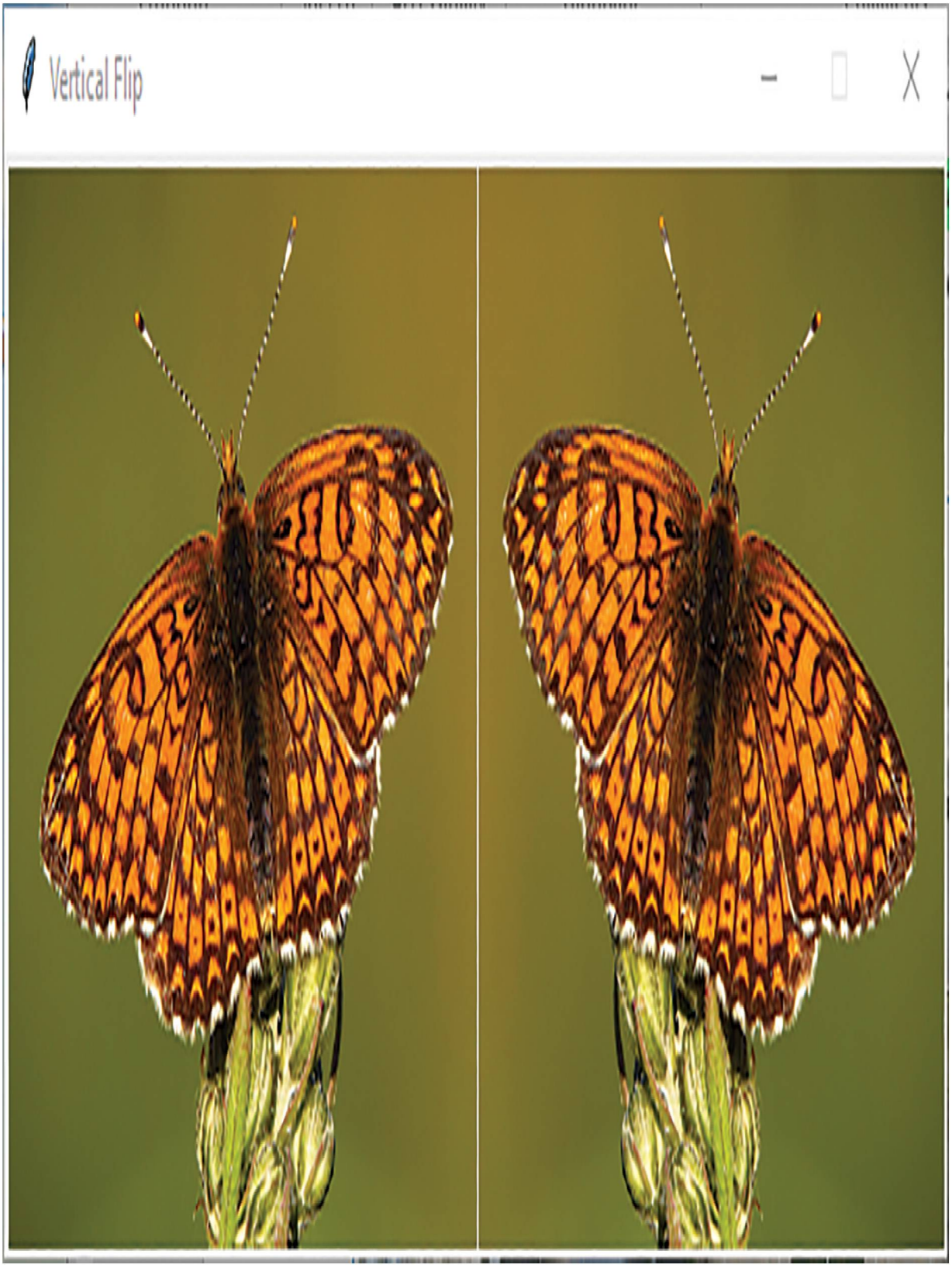


Figure 6.21 A flipped image.

© 2001–2025 Python Software Foundation. All Rights Reserved. Images A and B © Luka Hercigonja/Shutterstock.

Description

Try It Out

6.27 Write the function `horizontalFlip` to flip an image on the horizontal axis.

6.28 Rewrite the `verticalFlip` function so that it flips an image on both axes.

6.29 Mirroring is a manipulation similar to flipping. When producing a mirror, the pixels on one side of the mirror axis are reflected back on the other side. In a mirror operation, half of the pixels are lost. Implement a mirror on the vertical axis.

6.30 Implement a mirror on the horizontal axis.

6.31 Implement a mirror at a specific column or row. Note: This operation will change the image size.

6.32 Write a function `rotateImage90` that takes an image as a parameter and rotates the image by 90 degrees.

6.33 Write a function `rotateImage180` that takes an image as a parameter and rotates it by 180 degrees.

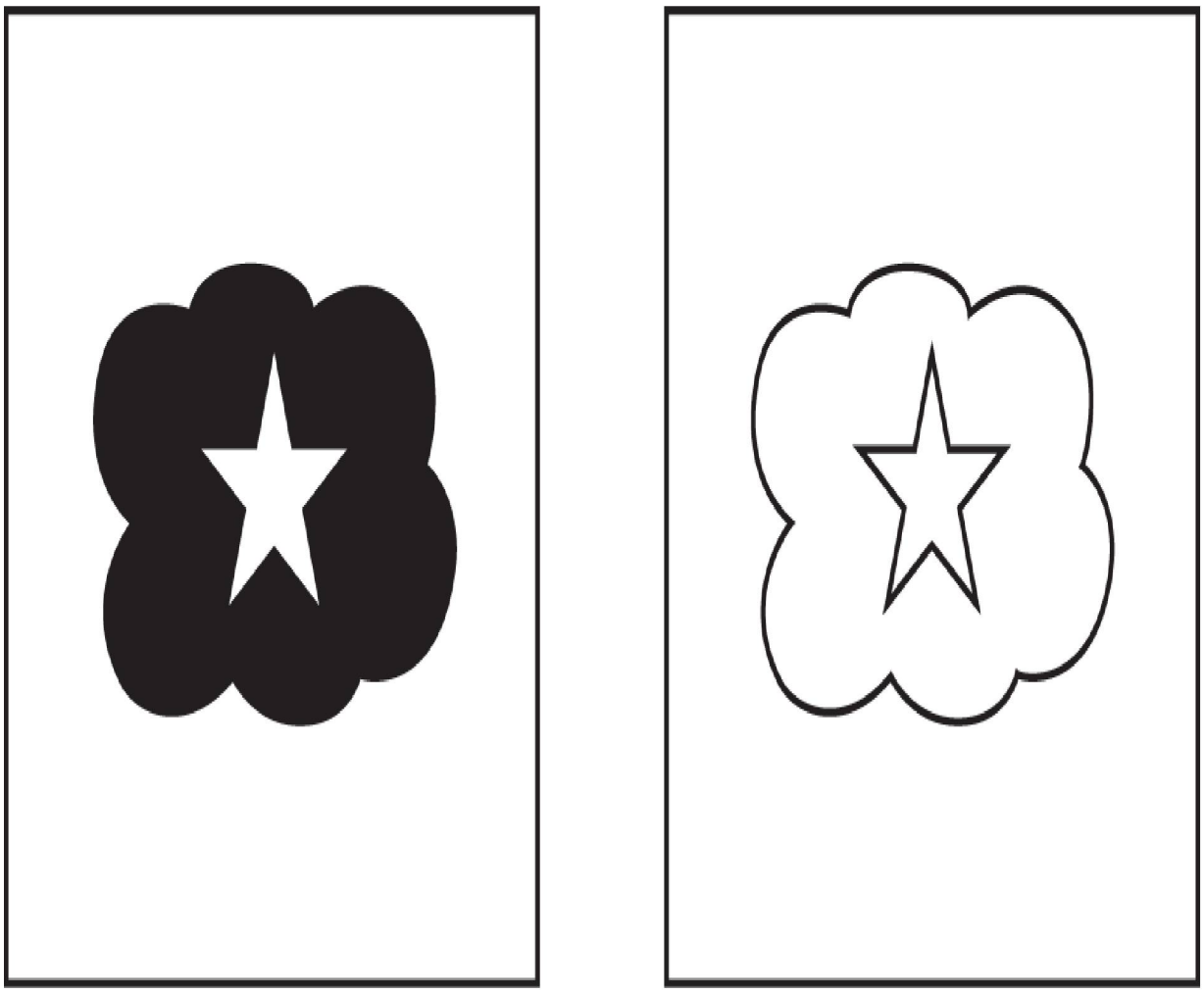
6.34 Write a function `rotate` that takes an image and a number of degrees to rotate the image. Note that this rotation may leave some empty pixels. You will also need to size the new image so it can hold the entire rotated image.

6.5.4 Edge Detection

Our final image processing algorithm in this chapter is called **edge detection**. This image processing technique tries to extract feature information from an image by finding places in the image that have dramatic

changes in color intensity values. For example, assume that we have an image containing two apples, one red and one green, that are placed next to each other. The border between a block of red pixels from the red apple and a block of green pixels from the green apple might constitute an edge representing the distinction between the two objects.

As another example, consider the black and white image shown in **Figure 6.22**. The left image contains three objects: a white square, a cloud, and a star. The right image shows the edges that exist in the image. Each black pixel in the edge image denotes a point where there is a distinct difference in the intensity of the original pixels. Finding these edges helps to differentiate between any features that may exist in the original image.



Original image Edges

Figure 6.22 A simple edge detection.

Description

Edge detection has been studied in great detail, and many different approaches have been developed to find edges within an image. In this section we describe one of the classic algorithms for producing the edges. The mathematics used to derive the algorithm are beyond the scope of this text, but we can easily develop the ideas and techniques necessary to implement the algorithm.

To find an edge, we need to evaluate each pixel in relation to the pixels around it. Since we are looking for places where the intensity on one side of the pixel is greatly different from the intensity on the other side, it will help us to simplify the pixel values. Our first step in discovering edges will be to convert the image to grayscale, so that we can think of the intensity of the pixel as the common color component intensity.

Recall that shades of gray are made from pixels with equal quantities of red, green, and blue. Each pixel can then be thought to have one of 256 intensity values.

As a means of looking for these intensity differences, we use the idea of a **kernel**, also known as a *filter* or a *mask*. Kernels can be used to weight the intensities of the surrounding pixels. For example, consider the 3-by-3 kernels shown in **Figure 6.23**. These "grids" of integer weights are known as **Sobel operators**, named after the scientist Irwin Sobel, who developed them for use in edge detection.

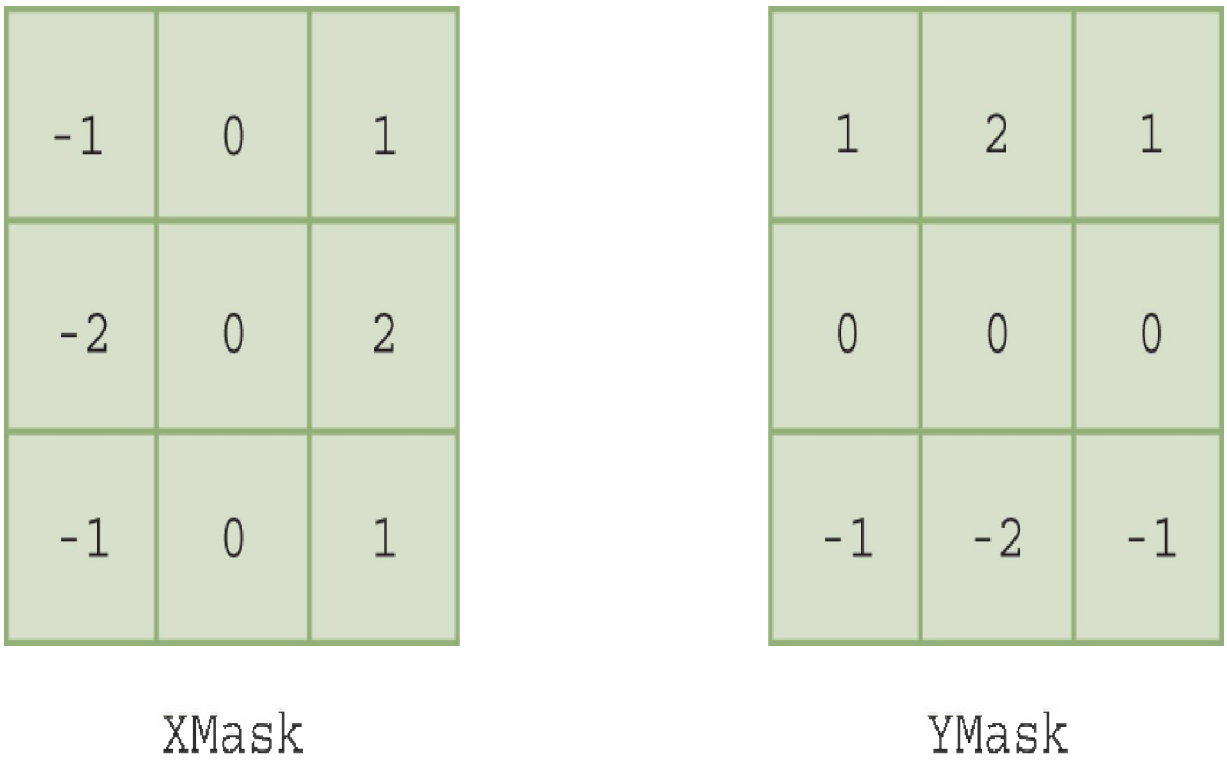


Figure 6.23 Kernel masks for convolving pixels.

Description

We can use the left mask, labeled `Xmask`, to look for intensity differences in the left-to-right direction of the image. You can see that the leftmost column of values is negative and the rightmost column is positive. Likewise, we can use the `Ymask` to look for intensity differences in the up-and-down direction, as can be seen by the location of the positive and negative weights.

The kernels are used during **convolution**—a process in which pixels in the original image are mathematically combined with each mask. The result is then used to decide whether that pixel represents an edge.

Convolution requires both a mask and a specific pixel. The mask is "centered" on the pixel of interest, as shown in **Figure 6.24**. Each weight value in the mask then associates with one of the nine pixel intensities "under" the mask. The convolution process simply computes the sum of nine products, where each product is the weight multiplied by the intensity of the associated pixel. Thus, if you have a large intensity on the left side and a small intensity on the right (indicating an edge), you will get a weighted sum

with a large negative value. If you have a small intensity on the left and a large intensity on the right, you will get a large positive weighted sum. Either way, a large absolute value of the weighted sum will indicate an edge. This same argument applies for the top-to-bottom split of the `Ymask`.

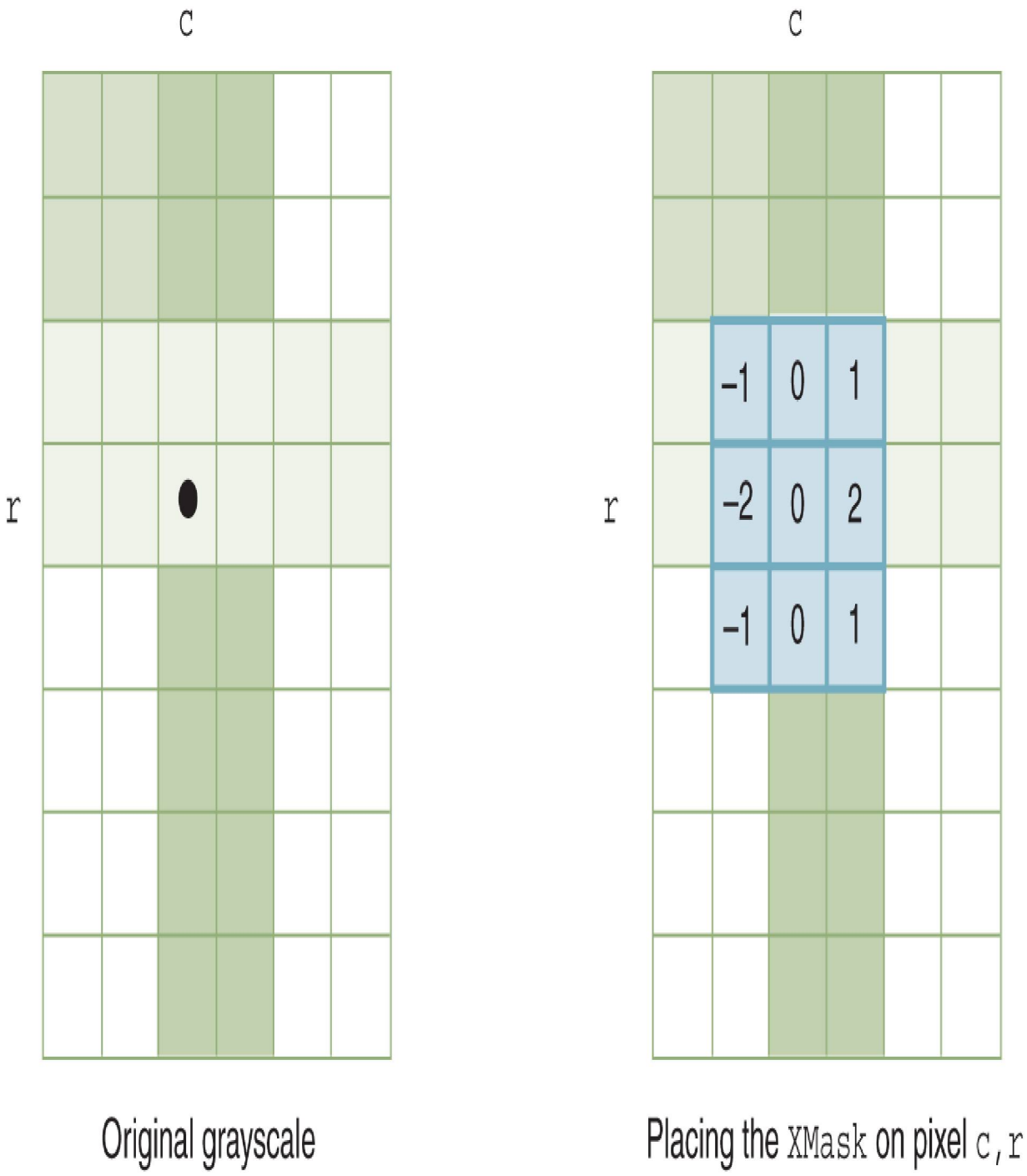


Figure 6.24 Using the `Xmask` to convolve the pixel at `c,r`.

Description

To implement convolution, we will first need to consider a way to store the kernels. Since kernels look very similar to images (that is, they are rows and columns of weights), it makes sense to take advantage of this structure. We will use a **list of lists** implementation for the kernel. For example, the `Xmask` discussed earlier will be implemented as the list

```
[ [-1,0,1],[-2,0,2],[-1,0,1] ]
```

The outer list contains three items, each of which represents a row in the kernel. Each row has three items, one for each column. Similarly, the `Ymask` will be

```
[ [1,2,1],[0,0,0],[-1,-2,-1] ]
```

Accessing a specific weight within a kernel will require two index values: one for the outer list and one for the inner list. Because we are implementing the outer list to be a list of three rows, the first index will be the row value. Once we select a row list, we can use the second index to get the specific column.

For example, `Xmask[1][2]` will access the item in `Xmask` indexed by `1`, which is the middle row of the `Xmask`. The `2` indexes the last item in the list, which corresponds to the last column. This access is for the weight stored in the middle row and last column of `Xmask`.

We can now construct the `convolve` function. As noted earlier, the convolution process requires an image, a specific pixel within the image, and a kernel. The tricky part of this function is to align the kernel and the underlying image. An easy way to do this is to think about a mapping. The kernel row indices will run from 0 to 2, and likewise for the column indices. For a pixel in the image with index `(column, row)`, the row indices for the underlying pixels will run from `row - 1` to `row + 1`; for the columns, the indices will run from `column - 1` to `column + 1`.

We will define the *base index* to be the starting index for the 3 × 3 grid of underlying image pixels. The base index for the columns will be `column - 1` and the base index for the rows will be `row - 1`. As we process the pixels

of the image, the difference between the current image row value and the base index for the rows will be equal to the row index needed to access the correct row in the kernel. We can do the same thing for the columns.

Once we have computed the index into the kernel, we can use that index to compute the product of the weight and the pixel intensity. We will first access the pixel and then extract the red component, which will be its grayscale intensity. Since we have already converted the image to grayscale, we can use either of the red, green, or blue components for the intensity. Finally, that product can be added to a running sum of products for all underlying pixels.

The complete `convolve` function is shown in **Listing 6.12**. Note that the final step is to return the value of the sum.

```
1  def convolve(anImage, pixelRow, pixelCol, kernel):
2      kernelColumnBase = pixelCol - 1
3      kernelRowBase = pixelRow - 1
4
5      sum = 0
6      for row in range(kernelRowBase, kernelRowBase + 3):
7          for col in range(kernelColumnBase, kernelColumnBase + 3):
8              kColIndex = col - kernelColumnBase
9              kRowIndex = row - kernelRowBase
10
11             aPixel = anImage.getPixel(col, row)
12             intensity = aPixel.getRed()
13
14             sum = sum + intensity * kernel[kRowIndex][kColIndex]
15
16     return sum
```

Listing 6.12 Convolution for a specific pixel

Description

Now that we can perform the convolution operation for a specific pixel with a kernel, we can complete the edge detection algorithm. The steps of the

process are as follows:

1. Convert the original image to grayscale.
2. Create an empty image with the same dimensions as the original image.
3. Process each inner pixel of the original image by performing the following:
   a. Convolve the pixel with the `Xmask`; call the result `gX`.
   b. Convolve the pixel with the `Ymask`; call the result `gY`.
   c. Compute the square root of the sum of squares of `gX` and `gY`; call the result `g`.
   d. Based on the value of `g`, assign the corresponding pixel in the new image to be either black or white.

Listing 6.13 shows the Python code that implements these steps. We begin by converting the original image to grayscale using the `pixelMapper()` function developed earlier in the chapter. This will allow for simple intensity levels within each pixel. We also need an empty image that is the same size as the original image. In addition, we will define a few data objects for use later. Because each pixel in the edge detection result will be either black or white, we create black and white `Pixel` objects that can be assigned later in the process. Also, we need the list of lists implementation of the two kernels. These initializations are done on lines 8–9.

```
1 import math
2
3 def edgeDetect(theImage):
4     grayImage = pixelMapper(theImage, grayPixel)
5     newIm = EmptyImage(grayImage.getWidth(), grayImage.getHeight())
6     black = Pixel(0, 0, 0)
7     white = Pixel(255, 255, 255)
```

```
    XMask = [ [-1,-2,-1],[0,0,0],[1,2,1] ]
    YMask = [ [1,0,-1],[2,0,-2],[1,0,-1] ]

    for row in range(1, grayImage.getHeight() - 1):
        for col in range(1, grayImage.getWidth() - 1):
            gX = convolve(grayImage, row, col, XMask)
            gY = convolve(grayImage, row, col, YMask)
            g = math.sqrt(gX**2 + gY**2)

            if g > 175:
                newIm.setPixel(col, row, black)
            else:
                newIm.setPixel(col, row, white)

    return newIm
```

Listing 6.13 The edge detection function

Description

Now it is time to process the original pixels and look for an edge. As each pixel is required to have eight surrounding pixels for the convolution operation, we will not process the first and last pixel on each row and

column. Thus, our nested iteration will start at one, not zero, and it will continue through `height - 2` and `width - 2`, as shown on lines 11–12.

Each pixel will now participate in the convolution process using both kernels. The resulting sums will be squared and summed together, and in the final step we will take the square root (see lines 13–15).

The value of this square root, called `g`, represents a measure of the difference between the pixel of interest and the pixels around it. Deciding whether the pixel should be labeled as an edge is done by comparing `g` to a threshold value. It turns out that when using these kernels, 175 is a good threshold value for considering whether you have found an edge. Using simple selection, we just check whether the value is greater than 175. If it is, we color the pixel black; otherwise, we make it white. **Figure 6.25** shows the result of executing this function.

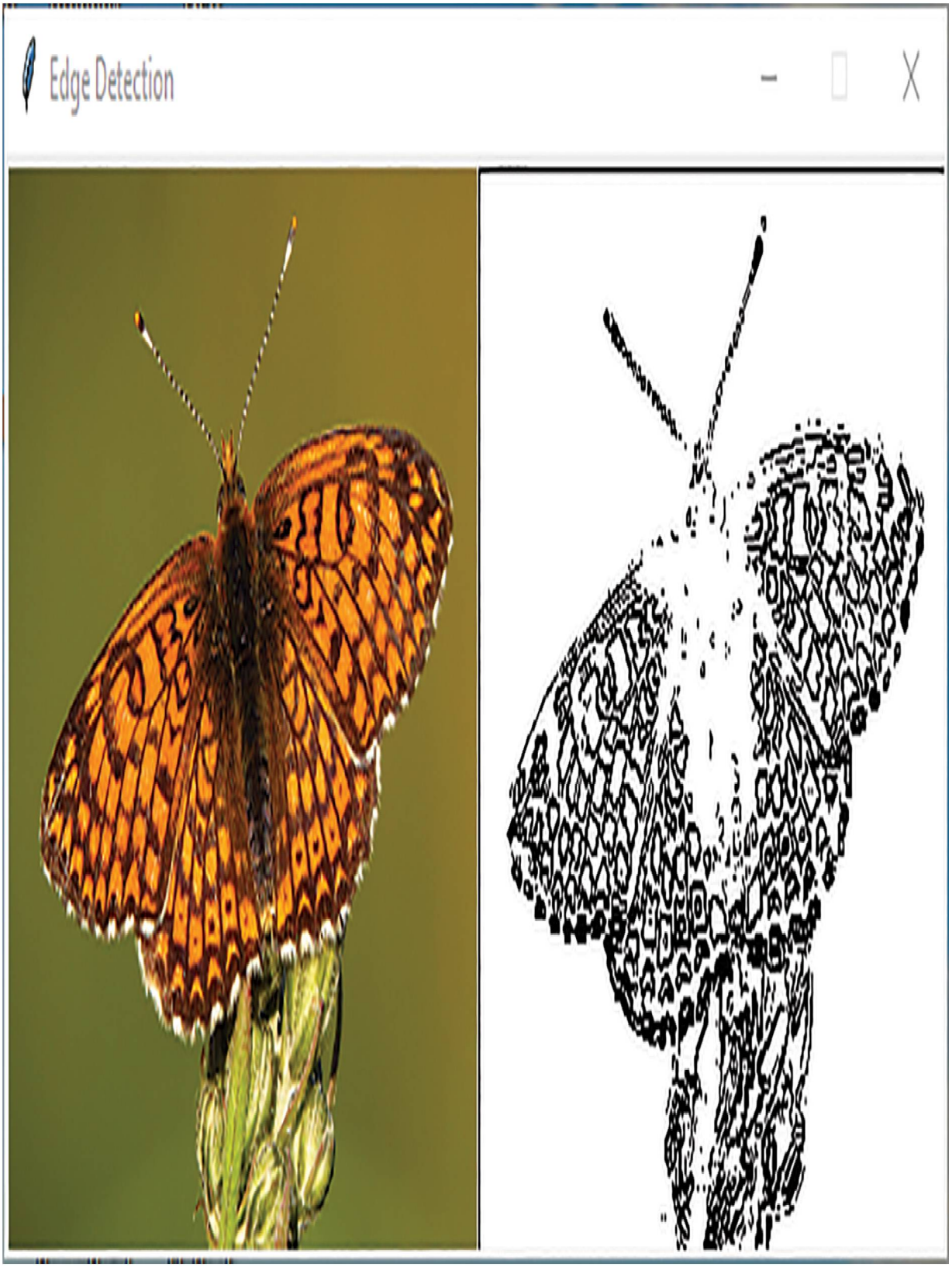


Figure 6.25 Running the edge detection algorithm.

© 2001–2025 Python Software Foundation. All Rights Reserved. Images A and B © Luka Hercigonja/Shutterstock.

Description

Try It Out

6.35 Try several different threshold values in `edgeDetect`. What effect does changing the threshold have on the image? Does a threshold of 175 work best for all images? What would be a way to automatically select a good threshold for an image?

6.36 Modify the `convolve` function so that it applies the kernel to the red, green, and blue components separately and returns a tuple of values as a result.

6.37 Convolution has many uses. For example, a simple convolution kernel is the blurring kernel, which looks like this:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

In this case, we simply apply the kernel and return the weighted average without doing any thresholding. Write a `blur` function that uses the new `convolve` function to blur an image.

6.38 The sharpen kernel looks like this:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

You can sharpen a pixel by emphasizing its value and deemphasizing the pixels around it. Sharpening is the opposite of blurring. Use the sharpen kernel to sharpen an image.

6.39 Write a general function that can take an image and a kernel and return an image with the convolution kernel applied to each pixel.

6.40 Research convolution kernels, and find a new one to try.

6.6 Summary

In this chapter we focused on image processing using `cImage`—a module that contains the following objects for manipulating digital images:

- `ImageWin`
- `EmptyImage`
- `FileImage`
- `Pixel`

To process the pixels of an image, we used a pattern called nested iteration—that is, iteration inside iteration. Nested iteration allowed us to process all the pixels in a given row, column by column, before moving on to the next row. Using this iteration pattern, we implemented the following image processing algorithms:

- Negative images
- Grayscale images
- Resized images
- Flipped images
- Edge detection

In this chapter we also introduced the notion of namespaces—collections of names available at a particular point in time. These namespaces are organized to allow Python to look up names when they are used, thereby making sure that there are no ambiguities. Along with describing namespaces, this chapter provided a more detailed explanation of parameter passing.

Key Terms

actual parameters
`builtins` namespace
call by assignment
color intensity
convolution
digital image
digital image processing
edge detection
flip axis
flipping
formal parameters
grayscale
invocation
kernel
list of lists
local namespace
local variables
`main` namespace
megapixels
namespaces
negative image
nested iteration
parameter passing
pixels

resizing an image
resolution
RGB color model
Sobel operators

Programming Exercises

6.1 Write a program to create a collage. Your program should combine several images with different effects applied to the images.

6.2 Write a program to blend one image with another. You can try different techniques for combining the RGB values for two pixels, each from a different image.

6.3 Take a picture of yourself against a white background. Use the fact that you can "filter" out all the white pixels to place your picture in an interesting scene. This same process is used by weather forecasters on television; the only difference is that they stand in front of a solid blue or solid green background called a chromakey.

6.4 Another way to put yourself in an interesting picture is to take a picture of yourself against a relatively plain background, then take another picture of exactly the same background (use a tripod here with autofocus off) without you in it. Now you can compare the two images and remove the pixels that are exactly the same, or close to the same. Once you have removed those pixels, you can superimpose yourself on any background.

6.5 Using `getMouse` to get the coordinates of a pixel in an image, devise a way to remove the red-eye effect from the area of the image you click on.

6.6 Using `getMouse`, write a program that will allow you to "cut" a rectangular region out of an image and display it as a new image.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 7

Data Mining: Cluster Analysis

Data Mining Techniques Using Clustering

7.1 OBJECTIVES

- To use Python lists as a means of storing data
- To implement a nontrivial data mining application
- To understand and implement cluster analysis
- To use visualization as a means of displaying patterns

7.2 What Is Data Mining?

We know there are ways that statistical techniques can help process and summarize large amounts of data. By computing statistical measures such as the range, mean, standard deviation, and frequency, we can begin to make statements about our data. In this chapter, we will explore this idea further.

Large amounts of data can be overwhelming. Yet, it is likely that important pieces of information are hidden away within the data that may not be obvious when only simple types of descriptive statistics are used. In this kind of situation, we can use **data mining**—the application of automated techniques that attempt to discover underlying patterns. These techniques can be applied to any number of data domains. For example, in business, data mining is often used for marketing purposes to find consumer-related patterns. Once these patterns are identified, they can be used to recommend the products that a customer might likely purchase. In addition, many applications in science and medicine require finding patterns in large amounts of data.

Cluster analysis is a data mining technique that attempts to divide the data into meaningful groups called **clusters**. These clusters represent data values that show some kind of similarity to one another while exhibiting a dissimilar relationship to data values outside of the cluster.

In this chapter, we focus on cluster analysis as a way to find hidden information in a collection of data. Our goal is to implement one version of cluster analysis and to apply tools that allow us to see the results.

7.3 Cluster Analysis: A Simple Example

As a simple example of what can be learned through cluster analysis, consider the data shown in **Figure 7.1**. This data set represents the performance of a class of mathematics students. The *x*-axis gives the homework total for the semester as a percentage of total points available; the *y*-axis shows the performance on exams, again as a percentage of total points.

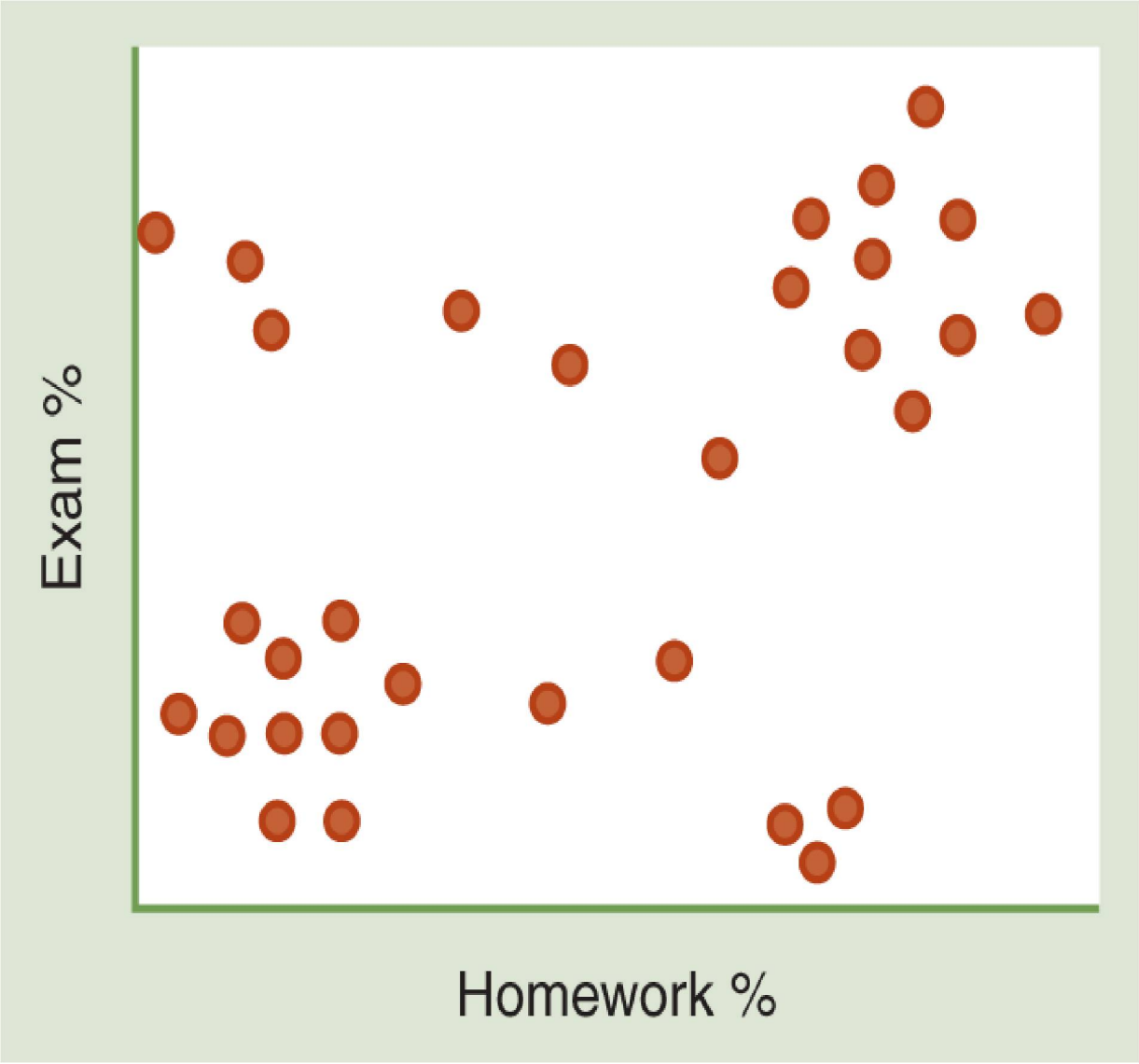


Figure 7.1 Data representing exam scores and homework totals.

Description

At first glance, it may not be apparent that patterns exist within the data. In fact, if we look at the raw data scores in a class that has hundreds of students, it is unlikely that we would notice anything. If we run the data through a cluster analysis routine, however, we may discover that some values tend to group together, as shown in **Figure 7.2**. Our analysis technique has identified three clusters, labeled A, B, and C. All of the values in cluster A are said to have some underlying characteristic in common based on some measure of similarity. The same can be said for clusters B and C. The values that appear in a cluster have characteristics that make them similar to one another but distinguish them from other clusters.

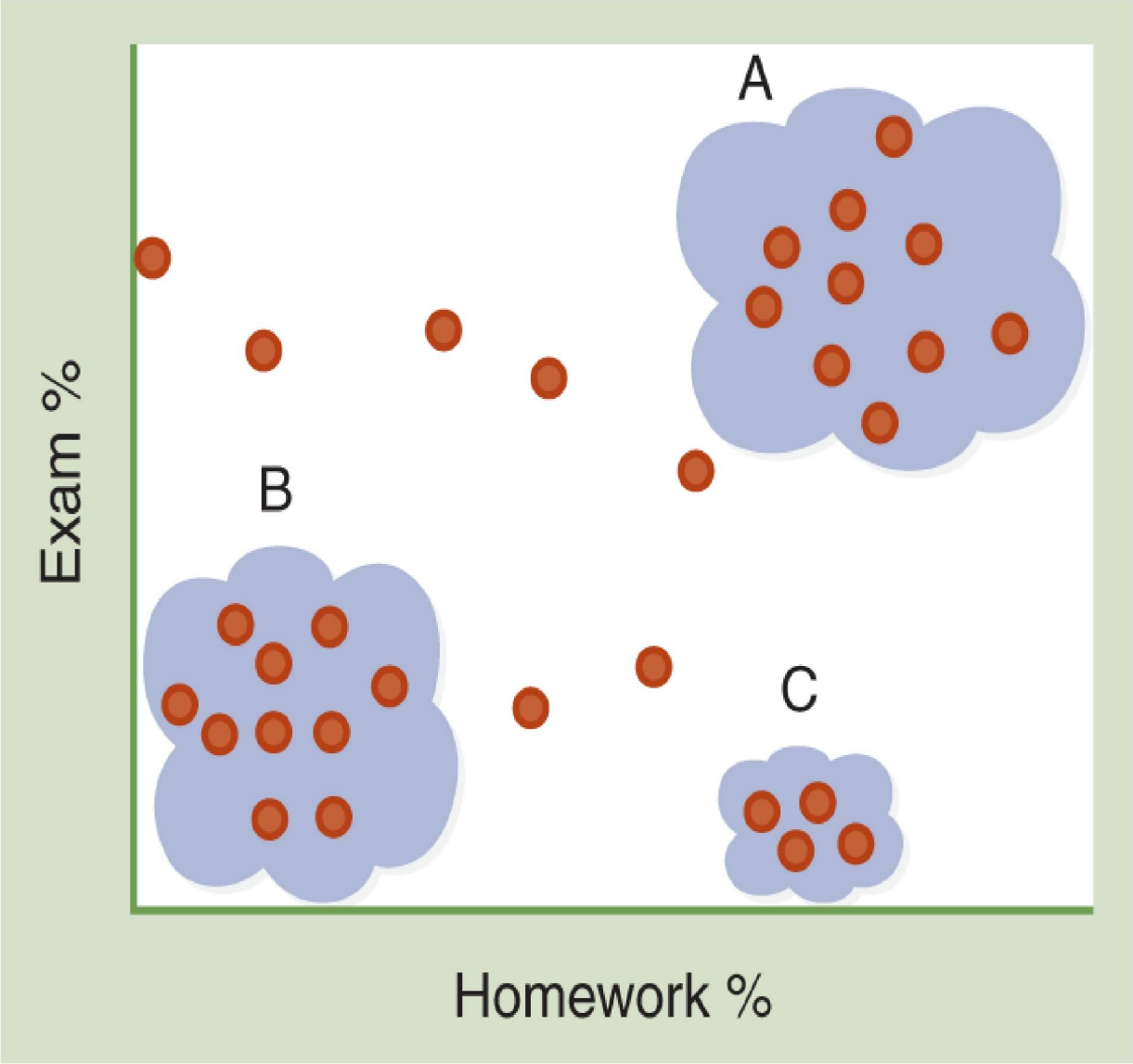


Figure 7.2 Possible clusters within the data.

Description

Notice that some data points do not appear in any of the three clusters. These points have characteristics that are different from the characteristics that define the other clusters. In addition, these points do not have enough similarity to form their own cluster. Some cluster analysis techniques will allow for this possibility, whereas others will require that all points eventually be placed in a cluster.

If we analyze our results further, we might be able to infer that these clusters are actually identifying performance clues for our students. For example, cluster A would seem to suggest that a number of students do well on both exams and homework. This could suggest a causal relationship—a contention that is supported by cluster B, in which students did poorly on both measures. Cluster C could lead us to think that these students have "test anxiety," as they did well with the homework but performed poorly on the exams.

In any case, some natural relationships exist within our data set. Finding these relationships can be critical to understanding underlying processes that are inherent in the data. Using these clusters to describe, predict, or otherwise quantify relationships within data can be a powerful tool.

7.4 Implementing Cluster Analysis on Simple Data

We now turn our attention to the fundamental steps of the cluster analysis technique. To concentrate on the algorithm, we use a very simple data set. Later, we will apply these same ideas to more complex data. Our example data in this section consists of exam scores for a group of 21 students taking an introduction to computer science course. These scores represent the percentage of correct answers. The range of scores is between 0 and 100.

Table 7.1 provides a simple listing of the scores. Our initial observation does not reveal any patterns. We could certainly use our previous work to compute descriptive statistics that might yield a bit more information, but right now we are interested in identifying patterns of similarity among the student scores. Cluster analysis could help us answer that question.

TABLE 7.1 Exam Scores for 21 Students

34	56	12	44	87	45	76
98	25	34	76	12	78	98
78	90	89	45	77	22	11

Description

7.4.1 Distance Between Two Points

One of the most important steps in the cluster analysis algorithm is to classify data points according to their similarity to other data points. To measure this similarity, we need some way to determine whether two points are "close" to each other. This can be done by computing a value that we will call the "distance" between two data points.

We can measure the distance between two data points in many ways. For our purposes, we use a simple measure known as the **Euclidean distance**.

Consider the two data points *A* and *B*, shown in **Figure 7.3**. If we assume that point *A* has location X_1 and point *B* has location X_2, then the distance between them, *d*, will be the simple difference between the two location values: $d = X_2 - X_1$. Because we do not know whether this difference will be positive or negative, however, we should use the absolute value.

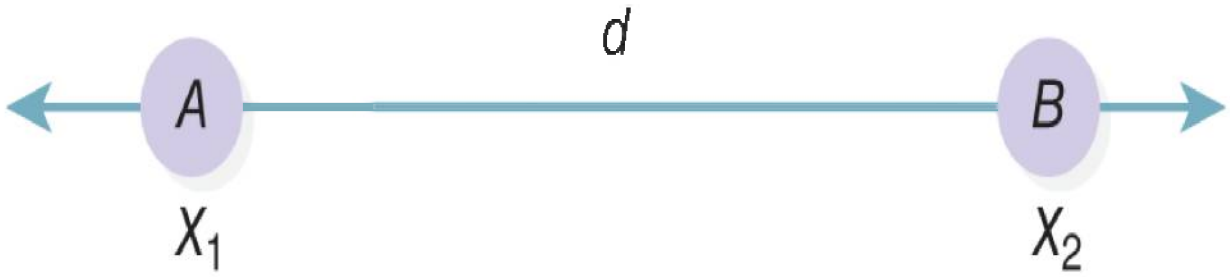


Figure 7.3 Simple distance between two points.

Description

An alternative way to eliminate negative values is to square them, as the squaring operation will always result in a positive number. To get the original positive value, we can then take the square root. Using this approach yields this equation:

$$d = \sqrt{(X_2 - X_1)^2}$$

Although this may seem like extra work, the benefit can be seen if we extend our example to computing the distance between two points in more than one dimension. **Figure 7.4** shows our two points. This time the location of point *A* is given by the pair (X_1, Y_1). Likewise, point *B* has location (X_2, Y_2). To compute the distance, *d*, between the two points now requires that we use the two-dimensional version of the previous equation, which is also known as the Pythagorean theorem:

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

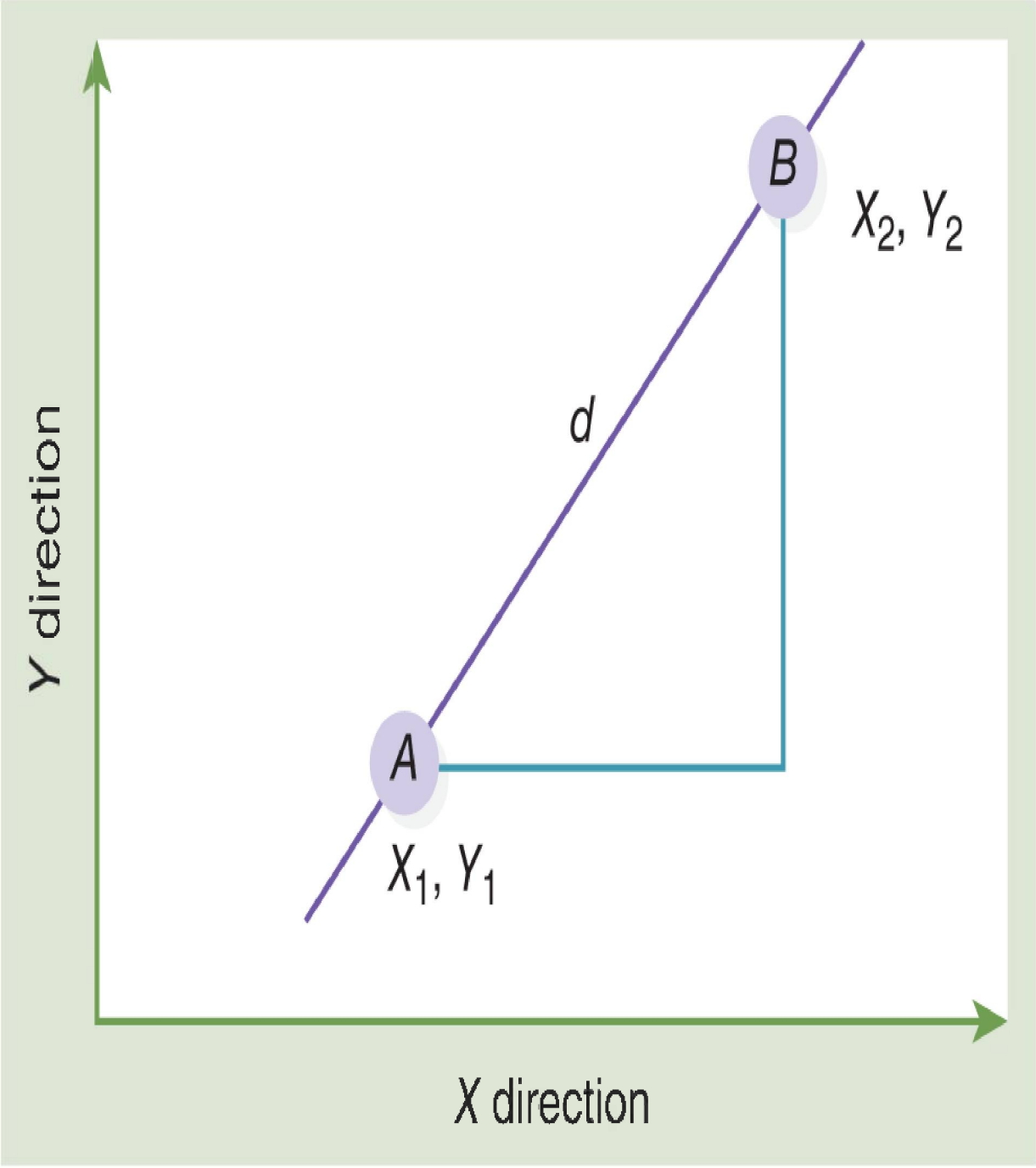


Figure 7.4 Two-dimensional distance between two points.

Description

In general, this calculation can be extended to handle any number of dimensions. The distance between two data points in *n* dimensions can be computed as the square root of the squares of the differences between each of

the n coordinates. For example, consider a point A in five-dimensional space with location (23, 44, 12, 76, 34). If point B has location (67, 55, 85, 23, 24), the distance between point A and point B is given by

$$d = \sqrt{(67-23)^2 + (55-44)^2 + (85-12)^2 + (23-76)^2 + (24-34)^2}$$

RECAP:
The distance between two data points in n dimensions can be computed as the square root of the squares of the differences between each of the n coordinates.

Listing 7.1 shows a function to compute the distance between two points. Each point is defined as a list of n coordinate values. Starting with `total` initialized to 0, we iterate through the coordinate list. If we assume that each point has the same dimensionality, the index range for `point1` will be the same as the index range for `point2`. Line 4 computes the square of the difference, and line 5 adds that value to the running `total`. Finally, the square root is computed and returned. The `euclidD` function assumes that the `math` module has been imported.

```
1 def euclidD(point1, point2): # points are equal-length lists or tuples
2     import math
3     total = 0
4     for index in range(len(point1)):
5         diff = (point1[index] - point2[index]) ** 2
6         total = total + diff
7
8     euclidDistance = math.sqrt(total)
9     return euclidDistance
```

Listing 7.1 Computing the Euclidean distance between two points

Description

7.4.2 Clusters and Centroids

In a cluster, all data points are similar to one another in some characteristic. Another way to think about this similarity is to suggest that all the data points in the cluster are associated with some notion of a center point. This center point can be used to identify a particular cluster.

A **centroid** is defined as the mean of a collection of data points. Each cluster will have a centroid that represents the center of the cluster. Note that the centroid does not need to be an actual point in the cluster: It is simply the "point" that tends to be in the center of all others.

To compute the centroid for a set of points, we calculate the mean value for each dimension. For example, if we have two points in two dimensions, (3, 7) and (1, 5), the centroid will be

$$\left(\frac{3+1}{2}, \frac{7+5}{2}\right)$$

or (2, 6). This pattern can be extended to include any number of points with any number of dimensions.

Figure 7.5 shows three points with two dimensions: (2, 4), (5, 8), (8, 6). The centroid (5, 6), shown by the star, represents the center with respect to the three points. If these points were a cluster, we could use the centroid as its identity.

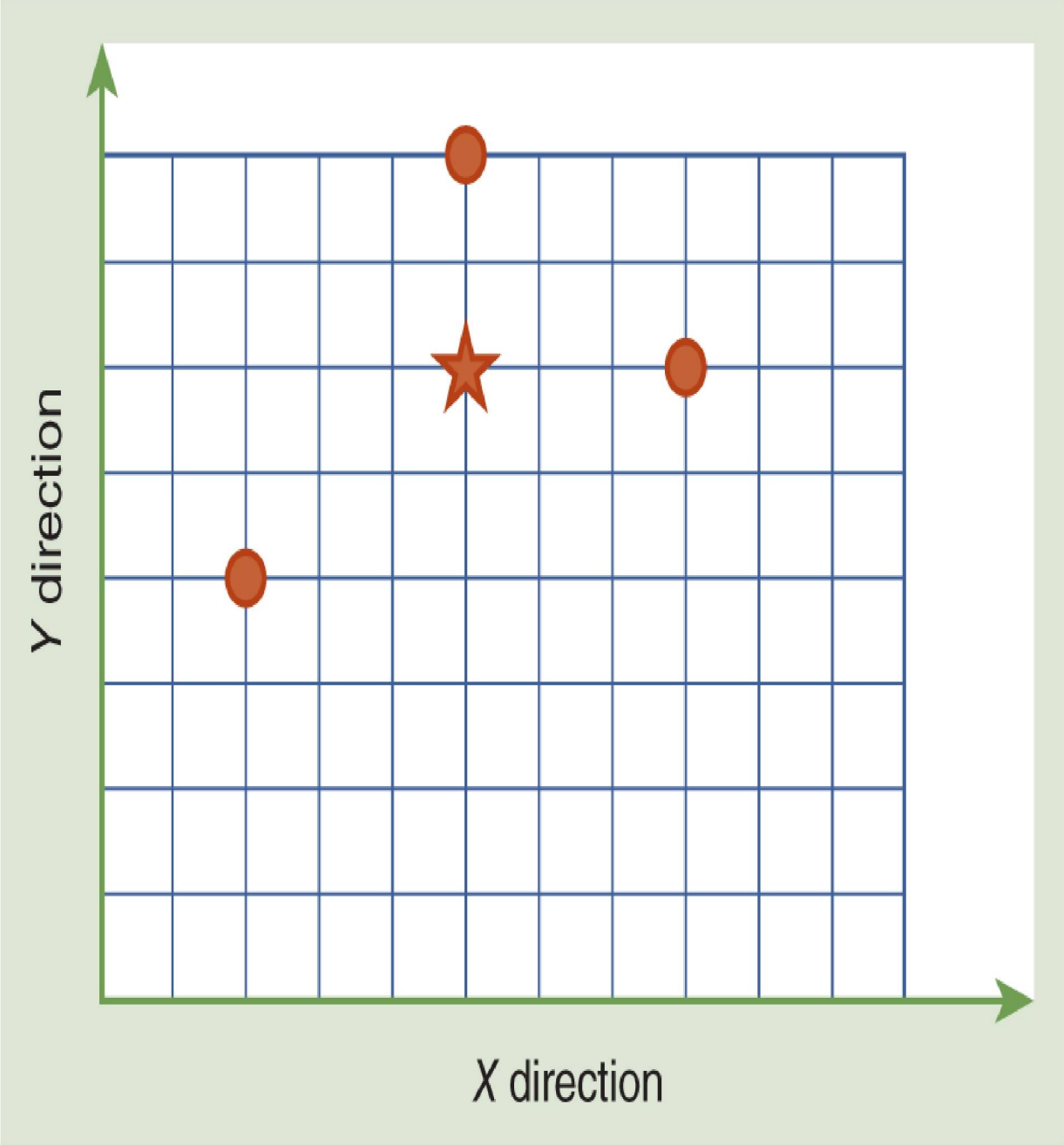


Figure 7.5 The centroid of three points in two dimensions.

Description

7.4.3 The K-Means Cluster Analysis Algorithm

Many clustering algorithms have been developed. Here, we present one of the oldest and easiest-to-understand techniques: the **K-means algorithm**. The

basic steps of this algorithm, which is simple to both describe and implement, are as follows:

1. Decide how many clusters you would like to create; call that number `k`.
2. Randomly choose `k` data points to serve as the initial centroids for the `k` clusters.
3. Repeat the following steps for a specified number of times or until the clusters are stable:
   a. Assign each data point to a cluster corresponding to its closest centroid.
   b. Recompute the centroids for each of the `k` clusters.
4. Show the clusters.

The number of iterations performed in Step 3 can vary. Sometimes, a simple maximum iteration value is used. Alternatively, the step can be repeated until the clusters become stable. Clusters are considered to be stable when the centroids no longer change from iteration to iteration. Under some circumstances, this state may never occur, as the points will tend to oscillate back and forth between different clusters.

Choosing the initial centroids randomly, however, can lead to empty clusters and results that differ from run to run. We will mention more of the shortcomings of this approach in Section 7.6.

HEADS UP:
Choosing random centroids simplifies the process but can lead to empty clusters and results that vary from one run to another.

7.4.4 Implementation of K-Means

We now implement the cluster analysis algorithm on the exam data set. We assume that the data is stored in a file named `cs150exams.txt`, with one score per line. There is no order or additional identifying information in the data file. We need to process the file line by line and extract the exam scores.

The `cs150exams.txt` file (with line numbers indicated in gray) is shown below. The line numbers are not part of the file.

1	34
2	56
3	12
4	44
5	87
6	45
7	76
8	98
9	25
10	34
11	76
12	12
13	78
14	98
15	78
16	90
17	89
18	45
19	77
20	22

Listing 7.2 shows a function that takes a filename as a parameter and returns the list of scores as described earlier. We open the file, create an empty list, and then start accessing each line of the file. Because the only data on the line is the test score, we can simply use the `int` function to convert the score to an integer. Line 8 appends the score to the data list. Note that the score is surrounded by brackets before adding it to the data list. Recall that each data point can be multidimensional. Even though these scores are not multidimensional, the `euclidD` function will expect the data points to be lists of values.

```
1  def readFile(filename):
2      with open(filename, 'r') as dataFile:
3
4          dataList = []                #start with an empty list
5
6          for aLine in dataFile:
7              score = int(aLine)       #convert string to int
8              dataList.append([score]) #add to dataList as a single-element list
9
10     return dataList
```

Listing 7.2 Processing the exam score data file

Description

Session 7.1 shows the data list of lists created by executing the `readFile` function with the `cs150exams.txt` file.

```
In [1]: from readfile import readFile
        readFile("cs150exams.txt")
```

```
Out[1]: [[34],
         [56],
         [12],
         [44],
         [87],
         [45],
         [76],
         [98],
         [25],
         [34],
         [76],
         [12],
         [78],
         [98],
         [78],
```

```
[90],
[89],
[45],
[77],
[22],
[11]]
```

Session 7.1 The data list created by the `readFile` function

Description

The next step is to choose `k`, the number of clusters. Often the number of clusters may be based on some preconceived notion about the data values. In this case, we create five clusters, assuming that one possible reason for clustering the data is to look for groupings that represent five grading categories: `A`, `B`, `C`, `D`, or `F`.

Now we need to pick `k` random data values to be our initial centroids. To implement this step, we use the `randint` function from the `random` module. This random number generator will pick an integer in the range `[a, b]`, including the endpoints. For example, `randint(2, 5)` will return a random integer between 2 and 5 inclusive.

It is important to choose unique centroid values. If we were to simply select `k` random values, we could possibly choose duplicate `k` values. To satisfy this requirement for selecting unique centroid values, we use a `while` loop.

RECAP:

A `while` loop has three parts:

1. Initialization

2. Condition
3. Loop update

To avoid an infinite loop, remember to include a statement in the loop body that updates the loop condition.

Try It Out

7.1 Read the exam scores using a `while` loop. Use the `readline` method to read the file.

7.2 Write a function that asks the user to enter exam scores one at a time until the word `stop` is entered. When `stop` is entered, the program should compute the average of the scores.

7.3 Write a function that takes a string as a parameter and returns `True` if the string is a palindrome, and `False` otherwise. This function should use a `while` loop. Hint: A palindrome is a word that is spelled the same both forward and backward.

7.4.5 Implementation of K-Means, Continued

The function shown in **Listing 7.3** implements the centroid selection process. It takes the number of centroids (called `k`) and the data list as parameters. Line 2 creates an empty list of centroids. Our task is to fill this list with `k` randomly selected unique data points, which we will identify by index. Because we do not want to use a centroid twice, we will store a list of the selected indices and check each randomly selected index against this list. If the index is already in the list, we will not include its data point as a centroid, but instead will randomly pick another index. This process will continue until `k` centroids have been selected. Because we do not know how many random selections will be required, we will use a `while` loop.

```
1  def createCentroids(k, dataList):
2      import random
3      centroids = []          # list of unique values
4      centroidIndices = []    # list of unique indices
5      centroidCount = 0
6
7      while centroidCount < k:
8          rIndex = random.randint(0,len(dataList)-1)
9          if rIndex not in centroidIndices:   # if index not already selected
10             centroids.append(dataList[rIndex]) # add to centroids
11             centroidIndices.append(rIndex) # add index to the selected centroid indices
12             centroidCount = centroidCount + 1
13
14     return centroids
```

Listing 7.3 Choosing `k` random centroids

Description

We keep track of the number of valid centroids that have been found with `centroidCount`. As long as that number is less than `k`, we continue to generate random indices. If an unused index is found, the centroid is added to the list of centroids, the index is added to the list of selected indexes, and the count is incremented (lines 10–12).

Listing 7.4 implements a function to create the clusters. The `createClusters` function takes as parameters the number of clusters (again

called `k`), the previously created centroids, the data list, and the number of repetitions, and it returns a list of the clusters. Each cluster is represented by a list. As we have a collection of clusters, a list of those cluster lists will be the appropriate way to store them. Lines 6–8 create this list of `k` empty clusters.

```
1  def createClusters(k, centroids, dataList, repeats):
2
3      for aPass in range(repeats):
4          print("****PASS", aPass + 1, "****")
5
6          clusters = []    # create list of k empty lists
7          for i in range(k):
8              clusters.append([])
9
10         for dataIndex in range(len(dataList)):  # calculate distance to centroid
11             distances = []
12             for clusterIndex in range(k):
13                 dtoC = euclidD(dataList[dataIndex], centroids[clusterIndex])
14                 # dtoC = math.dist( tuple(dataDict[aKey]), tuple(centroids[clusterIndex]))
15                 distances.append(dtoC)
16
17             minDist = min(distances)                    # find minimum distance
18             closestCluster = distances.index(minDist)   # find index of the min distance
19
```

```
            clusters[closestCluster].append(dataIndex)    # add to cluster

        dimensions = len(dataList[0])  # recompute the clusters
        for clusterIndex in range(k):
            sums = [0] * dimensions     # init sum for each dimension
            for dataIndex in clusters[clusterIndex]:
                dataPoints = dataList[dataIndex]
                for ind in range(len(dataPoints)): # calculate sums
                    sums[ind] = sums[ind] + dataPoints[ind]
            for ind in range(len(sums)):  # calculate average
                clusterLen = len(clusters[clusterIndex])
                if clusterLen != 0:       # do not divide by 0
                    sums[ind] = sums[ind] / clusterLen

            centroids[clusterIndex] = sums # assign avg to centroids

        for c in range(k):    # output the clusters
            print("CLUSTER #" + str(c+1))
            for dataIndex in clusters[c]:
                print(dataList[dataIndex], end = " ")
            print()

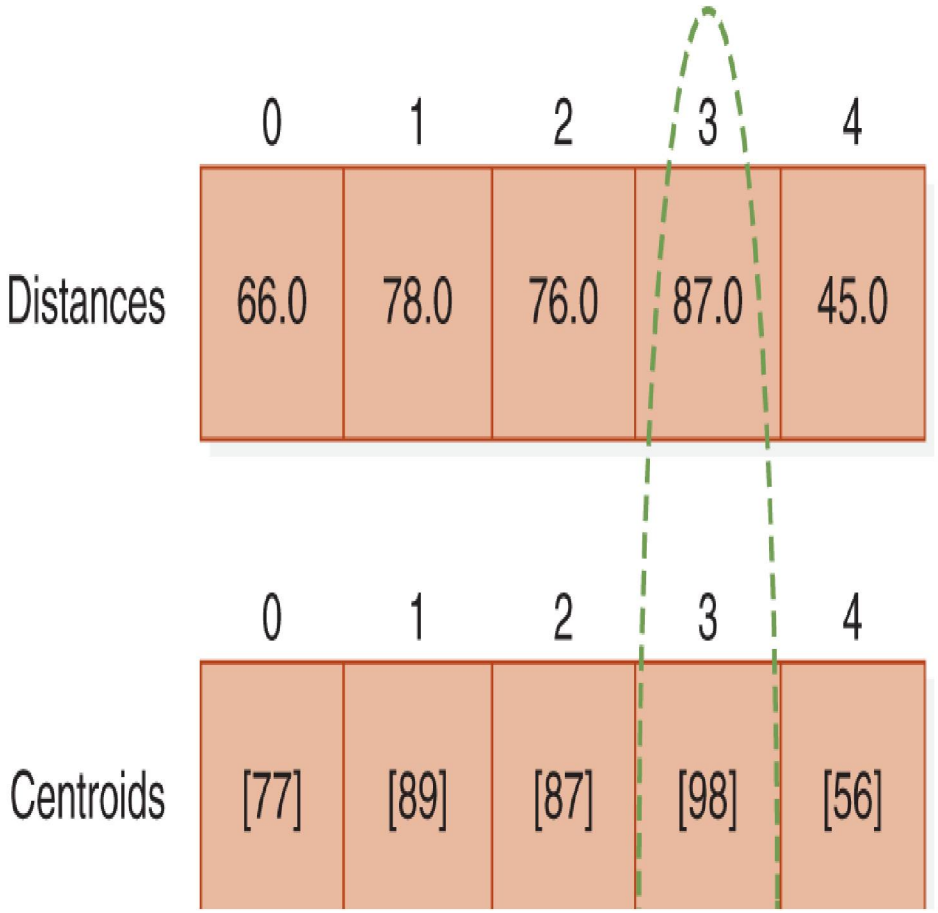
    return clusters
```

Listing 7.4 Creating the clusters

Description

We then go through each item in our data list and assign it to the proper cluster. Recall that we have a list of centroids, one for each cluster. We want to assign each data point to the cluster with the closest centroid. We can do this by using our distance function, `euclidD`, described earlier.

As there are `k` clusters, each data point will have `k` distances associated with it, one from each centroid. Line 11 creates an empty list of distances, and lines 12–15 compute the distance between the data point and each centroid. These distances are placed in the `distances` list. The `distances`, `centroids`, and `clusters` lists are **parallel lists**. In other words, for any index number `i`, `i` refers to data for the same cluster in each list. **Figure 7.6** shows the three parallel lists and the relationships among them.



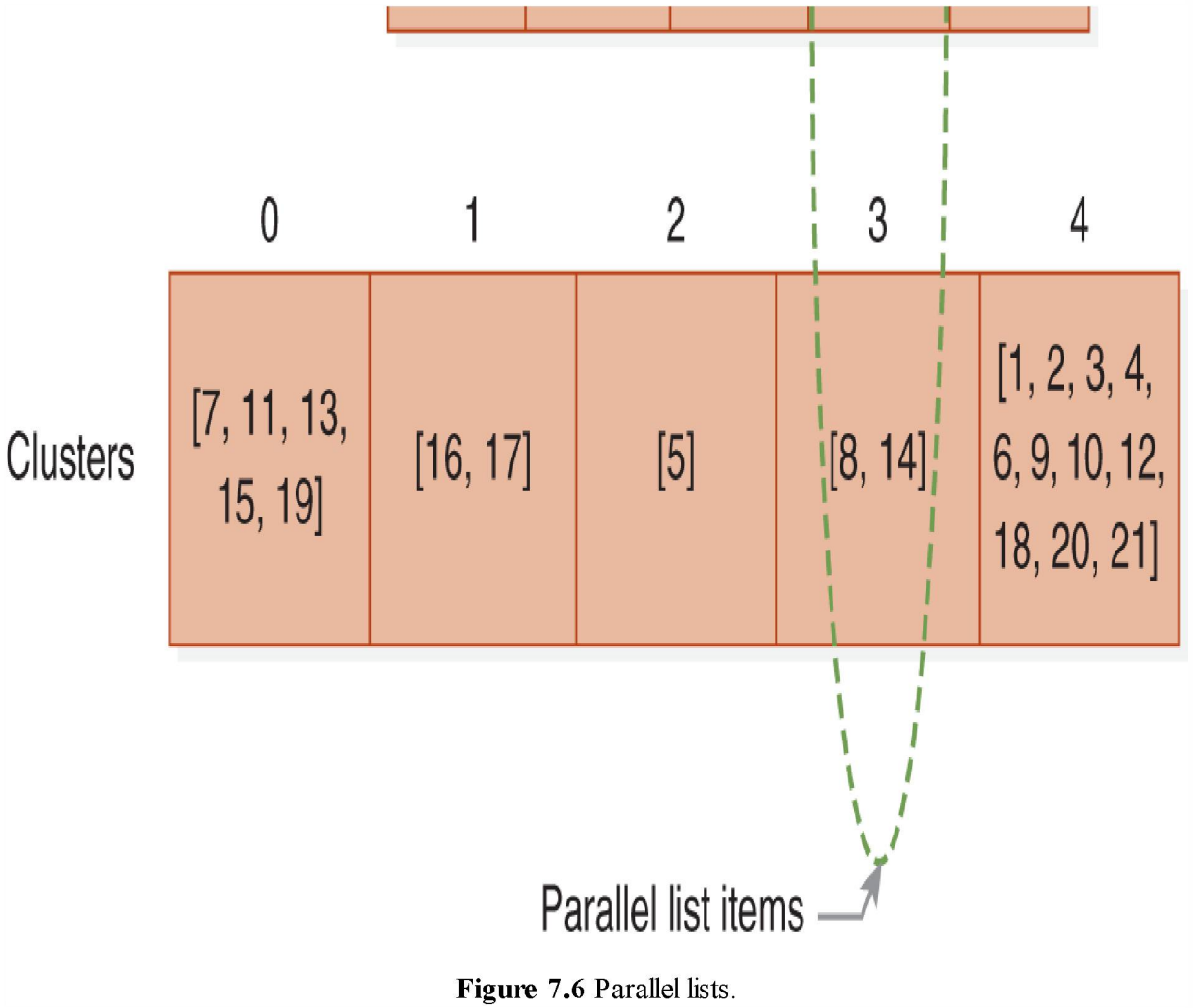


Figure 7.6 Parallel lists.

Description

The `math` module provides a `dist` function that, like our `euclidD` function, also calculates the distance between two points. The `dist` function, shown in **Table 7.2**, accepts its parameters as equal-length tuples. In our `createClusters` function, however, we are defining the parameters to the `euclidD` function as lists. If desired, you can use the `math.dist` function instead of the `euclidD` function by converting the list parameters to tuples. We have included the appropriate code to use the `dist` function in Listing 7.4 on line 14 as a comment.

TABLE 7.2 The `dist` Function in the `math` Module

Function	Use	Explanation
dist	math.dist(p1, p2)	Returns the Euclidean distance between two points, p1 and p2, specified as a tuple.

Once we have computed all of the distances, we can look for the closest centroid. Recall that Python provides a function called `min` that returns the smallest value in a list. We call the `min` function on line 17. Once we know the smallest value, we can use the `list` method called `index` to find where the minimum occurred in the `distances` list (line 18). The index of the minimum tells us the cluster to which the data point should belong. By using that index, we can access the list of clusters and append the index of the current data point to the proper cluster (line 20). Again, it is important to remember that we are storing the *indices* instead of the actual data points.

The final step in the K-means algorithm requires that we recompute the centroids for each cluster. Because the centroid of a cluster is simply the average of all data points in the cluster, we can iterate through the points, create a running sum, and then divide by the number of points.

Lines 22–34 of Listing 7.4 implement the centroid recalculation. Line 22 is important because our data points can be multidimensional. In this line, `dimensions` is the number of dimensions within the data point. For our exam score example, its value will be 1. Recall that to compute the new centroid, we must take the average of the coordinate value in each dimension.

A list of running sums, represented by `sums`, will include a sum for each dimension of the data point. Each sum component is initialized to 0. Lines 24–28 calculate the running sums of the components, and lines 29–32 compute the average. The last statement (line 34) assigns the average to the proper position in the centroids list.

Recall that the clustering process is repeated a number of times. In Listing 7.4, the parameter `repeats` allows the user to decide how many iterations to perform. Listing 7.4 also includes a small fragment of code (lines 36–40) that prints the contents of the clusters after each pass.

Finally, **Listing 7.5** shows a function that will perform the cluster analysis of the `cs150exams.txt` data file. It makes three passes through the data and produces five clusters.

```
1 def clusterAnalysis(dataFile):
2     examDict = readFile(dataFile)
3     examCentroids = createCentroids(5, examDict)
4     examClusters = createClusters(5, examCentroids, examDict, 3)
```

Listing 7.5 Cluster analysis for the exam data set

Description

When we run the program, it produces the output in **Session 7.2**. In the first pass, the exam scores are spread over the five clusters. The second pass shows that some of the scores have moved due to the new centroid values that were computed after the first pass. Finally, the last pass shows a final modification of the clusters. If we were using this analysis to assign grades, we might suggest that the first and second clusters would be the "F"s, the third cluster would be the "A+"s, and so on. Of course, running the program again could give different results because we randomly select the initial centroids.

```
In [1]: from clusteranalysis import clusterAnalysis
        clusterAnalysis("cs150exams.txt")

        ****PASS 1 ****
        CLUSTER #1
        [34] [12] [44] [45] [25] [34] [12] [45] [22] [11]
        CLUSTER #2
        [56]
```

[90]
CLUSTER #3
[98] [98]
CLUSTER #4
[76] [76] [78] [78] [77]
CLUSTER #5
[87] [90] [89]
****PASS 2 ****
CLUSTER #1
[34] [12] [25] [34] [12] [22] [11]
CLUSTER #2
[56] [44] [45] [45]
CLUSTER #3
[98] [98]
CLUSTER #4
[76] [76] [78] [78] [77]
CLUSTER #5
[87] [90] [89]
****PASS 3 ****
CLUSTER #1
[34] [12] [25] [34] [12] [22] [11]

```
[97] [22] [29] [97] [22] [22] [22]
CLUSTER #2
[56] [44] [45] [45]
CLUSTER #3
[98] [98]
CLUSTER #4
[76] [76] [78] [78] [77]
CLUSTER #5
[87] [90] [89]
```

Session 7.2 Clusters of exam scores

Description

Try It Out

7.4 Load and run the `clusterAnalysis` function using the exam score data. Compare your clusters with those shown in Session 7.2. Rerun it several times to see how the initial random centroids affect the end result.

7.5 Run `clusterAnalysis` again, but using different numbers of clusters and passes.

7.6 Modify `createClusters` so that the outer loop uses a `while` loop. The loop should exit when the clusters no longer change.

7.7 The previous exercise could create an infinite loop if the clusters are oscillating. Add another condition to the loop to ensure that no more than `maxRepeats` number of iterations occur.

7.8 Implement a different distance function and use it to cluster the exam data. Do you see any changes in the clusters?

7.5 Implementing Cluster Analysis: Earthquakes

We have real data describing 500 earthquakes that occurred during a month in late 2022. Given the raw data, it might be difficult to see any type of pattern or similarity in this data set. However, if we extend our cluster analysis technique from the previous section, we might discover some interesting results.

7.5.1 File Processing

Our first problem will be to find a way to process and store the data contained in the data file so that we can use it in our clustering algorithm. Recall that in the `earthquakes.csv` file, the first line contains column titles that identify each data item, like this:

```
time,latitude,longitude,depth,mag,magType,nst,gap,dmin,rms,net,id,updated,place
type,horizontalError,depthError,magError,magNst,status,locationSource,magSource
```

Description

Each succeeding line of the file describes one earthquake. The line for the first earthquake was as follows:

```
2022-11-30T22:30:36.483Z,5.5812,127.1411,150.253,4.6,mb,103,113,
2.144,0.54,us,us6000j5w0,2022-12-11T03:42:29.040Z,"Philippine Islands
region",earthquake,6.82,5.018,0.06,81,reviewed,us,us
```

Description

This entry describes an earthquake of magnitude 4.6 that occurred at a depth of 150.253 kilometers in the general region of the Philippine Islands, at

latitude 5.5812° and longitude 127.1411°.

For this example, we deploy our cluster analysis algorithm using the location data for each earthquake. In other words, we would like to see if some earthquakes are clustered in close proximity to one another. To do this, we need to understand how the location data is stored in the file and what it means.

Each earthquake is described by providing its exact location as a pair of values: longitude and latitude. The latitude values run north–south, with zero latitude being located at the equator. The north pole of the globe is +90, and the south pole is −90.

Likewise, the longitude values run west–east, with zero longitude being the prime meridian, an imaginary line that runs north–south through Greenwich, England. For latitudes in the far west, the measurement is −180; the far east measurement is +180, because the globe is assumed to be a full 360-degree circle on the equator.

Our data points now have two dimensions: a latitude and a longitude. Our previous description of the algorithm is still appropriate because, as you will recall, we created the `euclidD` function to work with multidimensional data points.

We can now work on extracting the necessary data from the file. Looking at the example line again shows us that for each earthquake, the data items are separated by commas. We can easily extract the data values we want by using the `csv` module as shown in **Session 7.3**. Recall that the `csv.reader` function interprets the comma-separated values in the file and that the `next` function returns the next line in the file.

```
In [1]: import csv
        with open("earthquakes.csv", "r") as dataFile:
            csvReader = csv.reader(dataFile)        # get iterator
            titles = next(csvReader)                # read titles line
            print("titles:", titles)                # output titles
```

```
    print()
    earthquakeLine = next(csvReader)       # read first earthquake line
    print("earthquake:", earthquakeLine)   # output all data
    print()
    print("latitude:", earthquakeLine[1])  # output latitude
    print("longitude:", earthquakeLine[2])  # output longitude
```

```
titles: ['time', 'latitude', 'longitude', 'depth', 'mag', 'magType', 'nst',
'gap', 'dmin', 'rms', 'net', 'id', 'updated', 'place', 'type',
'horizontalError', 'depthError', 'magError', 'magNst', 'status',
'locationSource', 'magSource']

earthquake: ['2022-11-30T22:30:36.483Z', '5.5812', '127.1411', '150.253',
'4.6', 'mb', '103', '113', '2.144', '0.54', 'us', 'us6000j5w0',
'2022-12-11T03:42:29.040Z', 'Philippine Islands region', 'earthquake',
'6.82', '5.018', '0.06', '81', 'reviewed', 'us', 'us']

latitude: 5.5812
longitude: 127.1411
```

Session 7.3 File processing for earthquake data

Description

We can build the Python code using the framework from Session 7.3. After opening the file, we iterate through the lines and extract the latitude and longitude. **Listing 7.6** shows the `readEarthquakeFile` function, which creates a list of two-dimensional data points.

```
1  def readEarthquakeFile(filename):
2      import csv
3
4      dataList = []
5
6      with open("earthquakes.csv", "r") as dataFile:
7          csvReader = csv.reader(dataFile)
8          titles = next(csvReader)     # read and skip the titles
9
10         for aLine in csvReader:
11             lat = float(aLine[1])    # extract latitude
12             long = float(aLine[2])   # extract longitude
13             dataList.append([long, lat])
14
15     return dataList
```

Listing 7.6 Processing the earthquake data file

Description

We continue to use the key as a way to refer to each unique line of the file. In this case, the longitude–latitude data point is appended to the `dataList` list. Lines 11–12 extract the longitude and latitude data as string values, which we need to convert into floating-point numbers.

Once we have our data points, we can call our `clusterAnalysis` function. Remember that we need to make decisions about how many clusters we want to create and how many iterations should be used. Unfortunately, when we run our program, the output, a fragment of which is shown here, is difficult to understand. The reason for this lack of clarity is that our simple output mechanism displays the contents of each cluster by showing the data points.

```
CLUSTER #1
[55.1789, 26.8456] [55.241, 26.8872] [24.2549, 38.2977] [24.2734, 38.3262]
[31.01, 40.8448] [27.7429, 35.3543] [3.2541, 72.5191] [30.9831, 40.8356]
[14.1232, 36.1568] [26.4358, 35.841] [13.3283, 43.8974] [13.7911, -52.3065]
[-16.4832, -58.8209] [55.2174, 26.9637] [56.0137, 27.7191] [-11.0345, -20.1783]
[26.5301, 34.3958] [3.0727, 36.3127] [56.1092, 27.5471] [-13.3391, -15.6013]
[-13.2234, -15.5834] [-13.1191, -15.5765] [-13.2489, -15.6171]
...
```

Description

7.5.2 Visualization

One more modification will make our results much more interesting. Instead of printing the longitudes and latitudes in long lists, we will use **visualization** to plot the positions of the earthquakes on a map of the world and show the clusters as points on that map. This process of "visualizing" the data can be quite useful, especially if we are looking for relationships that may not be readily apparent from long lists of data. It is common to look for some way to visualize the clusters.

As an example of this type of visualization, consider **Figure 7.7**, where the earthquake data has been processed using six clusters. Each earthquake is shown as a point on the map. In addition, the clusters are colored to distinguish one from the other. We can readily identify some relationships regarding where these earthquakes are taking place (although you likely knew this already).

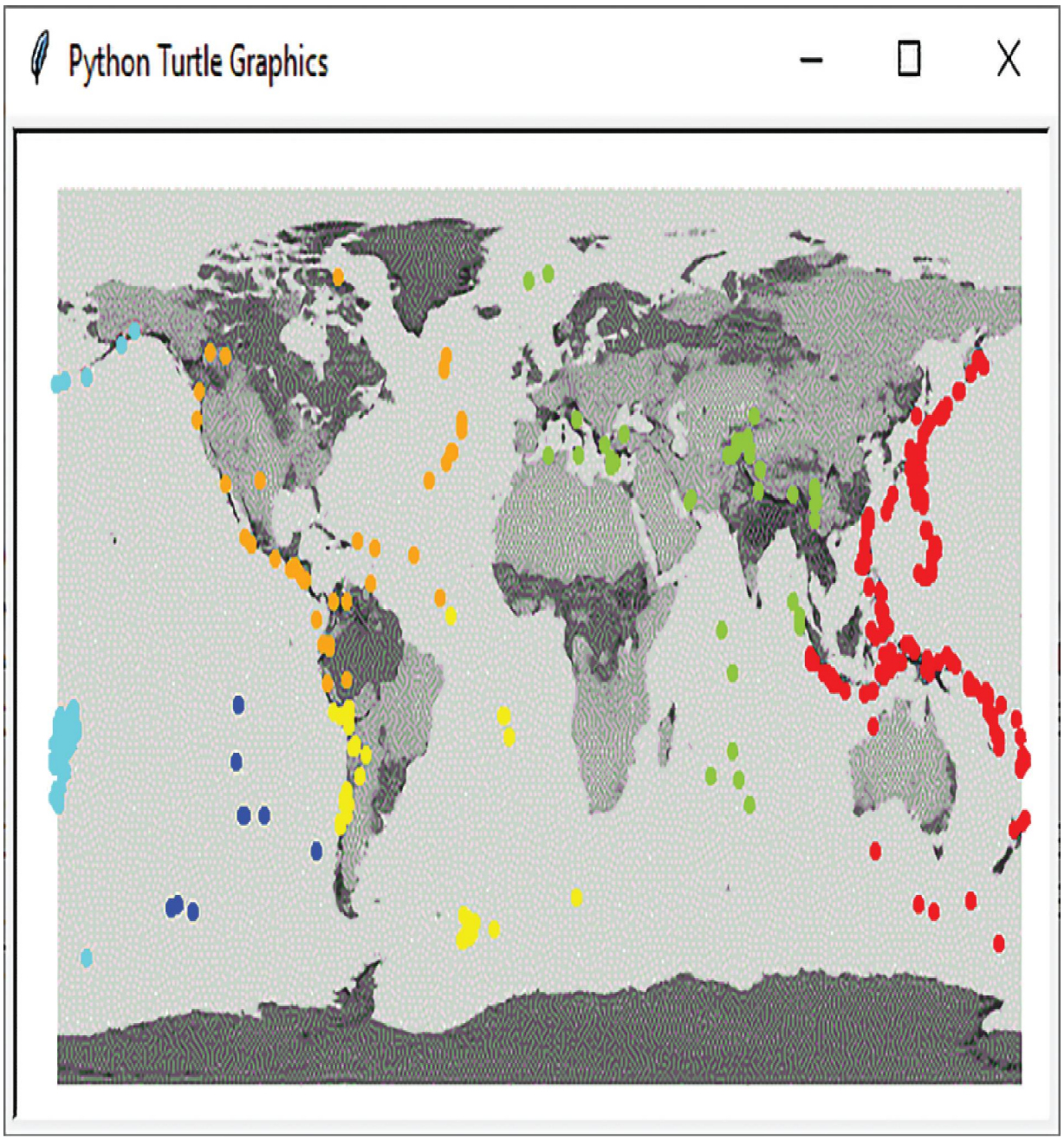


Figure 7.7 Plotting earthquakes to show clusters.

Map credit: NASA's Goddard Space Flight Center. Image credit: Copyright © 2001–2025 Python Software Foundation. All Rights Reserved.

We can easily build this visualization by revisiting our old friend, the `turtle` module. The basic idea is to use the turtle to plot a colored "dot" at each earthquake location given by the longitude and latitude. The color will depend on the cluster to which the earthquake belongs. The challenging part

is setting up the drawing window so that the map and the coordinates are consistent with our longitude and latitude values.

Listing 7.7 shows a function to generate clusters and visualize them. First, we make use of the `readEarthquakeFile`, `createCentroids`, and `createClusters` functions. Once the clusters have been computed, we create a turtle called `quakeT` using the `Turtle` constructor. Looking again at Figure 7.7, you can see that the background of the turtle drawing window contains an image of the world. This image, stored in a file named `worldmap.gif`, is 448 pixels wide by 252 pixels high. By using the `bgpic` method (line 8), we can set this image as the background picture for the drawing window. Because we want the drawing window to include only the area of the map, we can then use the `screensize` method (line 9) to reset the width and height of the drawing window.

```
1  def visualizeQuakes(dataFile):
2      dataList = readEarthquakeFile(dataFile)
3      quakeCentroids = createCentroids(6, dataList)
4      clusters = createClusters(6, quakeCentroids, dataList, 7)
5
6      quakeT = turtle.Turtle()
7      quakeWin = turtle.Screen()
8      quakeWin.bgpic("worldmap.gif")
9      quakeWin.screensize(448, 252)
10
11     wFactor = (quakeWin.screensize()[0]/2)/180
12     hFactor = (quakeWin.screensize()[1]/2)/90
```

```
13
14      quakeT.hideturtle()
15      quakeT.up()
16
17      colorList = ["red","lawngreen","blue","orange","cyan","yellow"]
18
19      for clusterIndex in range(6):
20          quakeT.color(colorList[clusterIndex]) #choose cluster color
21          for aIndex in clusters[clusterIndex]:
22              lon = dataList[aIndex][0]
23              lat = dataList[aIndex][1]
24              quakeT.goto(lon * wFactor, lat * hFactor)
25              quakeT.dot()
26      quakeWin.exitonclick()
```

Listing 7.7 Visualizing earthquake clusters

Description

From our previous discussion of longitude and latitude, we know that the lower-left corner of the map should be location (−180, −90) and the upper-right corner should be (180, 90). We can remap our data by recognizing that the current lower-left corner is (−224, −126) and the upper-right corner is (224, 126). Thus, we simply compute multiplication factors for the width and

the height (`wFactor` and `hFactor` in lines 11 and 12, respectively) and use them when we plot the longitude and latitude (line 24).

Lines 14–15 hide the turtle and raise the turtle's tail so that lines are not drawn as the turtle is moved from place to place. Finally, line 17 creates a list of colors that will be used to distinguish each cluster: `red` for the first cluster, `lawngreen` for the second, and so on.

We can now show the contents of each cluster by iterating through the clusters, processing each earthquake in the cluster. Line 20 sets the tail color by using `clusterIndex` as an index into the `colorList`. For each earthquake in the cluster, we extract the longitude and latitude data from the `dataList` and use those two values as coordinates for the turtle. Once the turtle has been directed to the proper location, the `dot` method plots a point using the current tail color. Note that the number of colors in the `colorList` must be at least as large as the number of clusters being created so that each cluster gets a unique color.

Try It Out

7.9 Load and run the code for the `visualizeQuakes` function. How do your results compare with those in the example?

7.10 Try to change the number of clusters. Be sure to add more colors if you add more clusters.

7.11 Rewrite the `readEarthquakeFile` to use the `pandas` module instead of `csv`. Hint: You will want to keep only the latitude and longitude columns of the DataFrame, and you can extract the values of that columnar data by using the `.values` property of the DataFrame.

7.12 Using the earthquake data, try to cluster the quakes by their depth instead of by latitude and longitude.

7.6 Cluster Analysis Shortcomings and Solutions

The basic K-means cluster analysis algorithm is easy to implement, but it can lead to a number of problems. We briefly describe a few of these problems here and leave the solutions for you as exercises.

Step 2 of our algorithm required that we pick `k` data points to serve as the initial centroids. Our solution was to use a random selection, but this meant that two runs of the program would likely produce different results. It seems intuitive that by choosing the centroids in a more intentional manner, we can guide the way in which the clusters are ultimately constructed. This determination could be based either on user input or on data analysis.

It is possible that clusters may become empty as the iteration process continues. In our implementation, once a cluster becomes empty, there is no way to repopulate it because it no longer has a centroid. We can choose either to leave it empty or to create a new centroid so that data points can be added in the next iteration.

Sometimes a cluster can get too large or can encompass data points that are seemingly not related. This can happen when some data points in the set are clearly different from the rest. Such points are called **outliers**. When an outlier is found, we may want to provide some special processing to create an additional cluster for it, or to exclude it from any cluster, thereby nullifying the impact the outlier might have on the centroid calculations.

Try It Out

7.13 Implement a solution to the empty cluster problem.

7.14 Implement a solution for duplicate data values. This case could happen if the data set includes two exam scores that are identical but belong to different people.

7.15 Implement an alternative to random centroid selection in which the user has some say in the process.

7.16 Implement an alternative to random centroid selection in which some simple data analysis is performed to pick the "best" centroid values.

7.17 Implement a function to find and eliminate outliers in the data set.

7.7 Summary

In this chapter, we implemented the K-means algorithm—a simple cluster analysis technique that is vulnerable to problems in some cases. This approach can be applied to a wide variety of application domains and performs in a fairly efficient manner. There are other techniques for cluster analysis, and we urge you to consider some of them for comparison purposes.

We utilized lists—specifically, parallel lists—and lists of lists (multidimensional lists) as a means of organizing our data. Finally, we created a visualization for our cluster analysis results.

Key Terms

centroid
cluster analysis
clusters
data mining
Euclidean distance
K-means algorithm
outliers
parallel lists
visualization

Programming Exercises

7.1 Find another data set that interests you and use the clustering techniques described in this chapter on that data. Possibilities include sports statistics, weather data, and medical data.

7.2 Research other cluster analysis algorithms and implement them on the data from this chapter. How do the results differ?

Design Credits: Calculator Icon made by Smashicons from www.flaticon.comv



© Yuri Hoyda/Shutterstock.

CHAPTER 8

Cryptanalysis

Advanced Dictionaries and Lists; Regular Expressions

8.1 OBJECTIVES

- To understand more advanced examples of using a dictionary in Python
- To understand more advanced examples of using lists in Python
- To use pattern matching with regular expressions
- To learn how simple programs can help you solve more advanced problems

8.2 Introduction

One of the most secure algorithms for encrypting secret messages today is called RSA, named after its authors—Ron Rivest, Adi Shamir, and Leonard Adleman. Web browsers often use RSA to make secure connections to sites like banks and online stores. Obviously, any encryption technique that is widely used for such sensitive data will be the target of many attacks. If someone breaks RSA—and a few people out there say they have already broken it—a lot of bank account numbers and passwords would be vulnerable to theft.

In this chapter, we focus on the field of code breaking, known as **cryptanalysis**. Of course, the study of breaking codes is as old as the field of making codes. In fact, some of the first computer scientists were deeply involved in cryptanalysis. Charles Babbage, the designer of the Difference Engine, was working on breaking the Vigenère cipher. Alan Turing, creator of the Turing test for artificial intelligence and the Turing Machine, was responsible for breaking the German Enigma cipher during World War II.

Breaking advanced codes by brute force—just trying every possibility until one of them works—is an interesting exercise for people with access to supercomputers or who want to investigate distributed computing. The rail fence cipher can be easily broken using a brute-force approach on any computer. Many beginning computer science students are reluctant to try brute-force approaches to solving problems, convincing themselves that "there must be a better way." In fact, many times a brute-force solution is appropriate simply because it takes advantage of the computer's ability to perform millions of calculations very quickly.

As a problem-solving challenge, we will see that there are more interesting ways to attack codes. In particular, this chapter presents some examples of short Python programs that can enhance our human abilities for problem-solving. Just as we use a calculator to solve complex math problems, short Python programs can help us solve other complex problems.

We start this chapter with a simple brute-force attack on the rail fence cipher. Next, we show how to use frequency analysis to crack the substitution

cipher.

8.3 Cracking the Rail Fence

We can write a generalized version of the transposition cipher called the rail fence cipher. In this section, we will look at a brute-force technique to break a two-rail fence with the odd and even characters placed into separate rails. Let's begin by reviewing the procedure for the rail fence.

The key to the rail fence cipher is the number of rails used. When you know the number of rails, you encrypt the message by filling the rails top to bottom and left to right. For example, let's suppose we want to encrypt the message "new ipad coming tomorrow" using three rails.

rail 1	n		a	c	i		m	r
rail 2	e	i	d	o	n	t	o	o
rail 3	w	p		m	g	o	r	w

Description

Once we have constructed the rails, we create a new string for each rail by concatenating the letters in the rail from left to right. We then concatenate the rails from top to bottom. This gives us the message "n aci mreidontoowp mgorw".

Cryptanalysts intercepting this message may know that the message was encrypted using the rail fence cipher, but they will not know how many rails were used. A brute-force way to decrypt this message is to try decrypting the message using two rails, three rails, four rails, and so on, until we find a message that makes sense.

Assuming that we have written a function `railDecrypt` (which we will do in Section 8.3.3) to decrypt a message, we might use a simple `for` loop to try all possible numbers of rails. **Session 8.1** shows our first attempt at a brute-force decryption. Scanning the results manually, we find the second attempt to be the correct solution.

In [1]:
```
cipherText = "n aci mreidontoowp mgorw"
for i in range(2, len(cipherText) + 1):
    print (railDecrypt(cipherText, i))
```

```
['nn', 'taocoiw', 'pm', 'rmegiodrow']
['new', 'ipad', 'coming', 'tomorrow']
['nmn', 'rtmaeogciooidwr', 'opw']
['nienw', 'itpamdo', 'croom']
['nienwg', 'itpoamdo', 'rcroomw']
['ncmino', 'irdtwma', 'eoopg']
['ncmino', 'o', 'irdtwmra', 'eoopgw']
['naimednow', 'c', 'riotop']
['naimednow', 'c', 'riotopm']
['naimednow', 'g', 'c', 'riotopmo']
['naimednow', 'gr', 'c', 'riotopmow']
['n', 'aci', 'mreidon']
['n', 'aci', 'mreidont']
['n', 'aci', 'mreidonto']
['n', 'aci', 'mreidontoo']
['n', 'aci', 'mreidontoow']
['n', 'aci', 'mreidontoowm']
```

```
['n', 'aci', 'mreidontoowp']
['n', 'aci', 'mreidontoowp', 'm']
['n', 'aci', 'mreidontoowp', 'mg']
['n', 'aci', 'mreidontoowp', 'mgo']
['n', 'aci', 'mreidontoowp', 'mgor']
['n', 'aci', 'mreidontoowp', 'mgorw']
```

Session 8.1 Brute-force decryptions of the rail fence cipher

Description

8.3.1 Checking Our Work with a Dictionary

As you can see from Session 8.1, the problem with the brute-force approach is that if the ciphertext is quite long, we may have to scan through many lines of gibberish before we find the correct message. One way we can improve this situation is to use a dictionary—both literally and in the Python sense. Our `railDecrypt` function returns a list of strings. Suppose we check each string against a dictionary to see if it was gibberish or a real word? A list of strings in which a high percentage appear in the dictionary is likely to represent a real message.

We do not need a dictionary that includes definitions; instead, we just need a file that contains numerous real words. Many of these lists of words are freely available on the Internet. We will refer to one such list stored in a file called `wordlist.txt`. This file is supplied with the source code for this chapter.

The `wordlist.txt` file contains 41,238 English words, with one word on each line. To make looking up words easy, we will load this file into a Python dictionary. **Listing 8.1** is an example of a function that reads a file of words and returns a Python dictionary containing all the words in the file.

```
1 def createWordDict(dName):
2     myDict = {}
3     with open(dName, 'r') as myFile:
4         for line in myFile:
5             myDict[line[:-1]] = True  #set all values to True
6     return myDict
```

Listing 8.1 Loading words from a file into a dictionary

Description

Because we are not storing a useful value along with the key, you might ask, "Why use a dictionary?" The answer is speed. Suppose you rewrote the `createWordDict` function to be `createWordList`, where `createWordList` returns a list of words rather than a dictionary. We can still check whether a word is in our word list with the following Python statement:

```
if w in wordlist:
```

If you checked the amount of time that Python needs to decide whether a word is in the list, you would see that, on average, using a dictionary is many times faster than using a list. One disadvantage of using a dictionary in this scenario is that this approach wastes space, as we have no use for the value associated with each key.

Try It Out

8.1 Write a `createWordList` function to create a list of words from the `wordlist.txt` file.

8.2 Devise an experiment to measure the performance of `createWordList` and `createWordDictionary` in terms of how long it takes to create the list or the dictionary. You can use the `time` module with the function `time.time()` to get the time in milliseconds. Getting the clock time before you start a task and after you complete a task allows you to measure how long the task takes.

The `time.time()` function returns the number of milliseconds since the "epoch." The epoch is January 1, 1970, at midnight, and is an arbitrary time used for computer time functions. This function will return a very large number, but if you subtract the starting `time.time()` result from the ending one, you will get a number of milliseconds, which you can easily convert to seconds by dividing by 1,000.

8.3 Devise an experiment to measure the performance of the dictionary and the list created using `createWordDict` and `createWordList` in terms of how long it takes to find a word in the list or the dictionary. Again, you can use the `time` module with the function `time.time()` to get the current clock time in milliseconds. Getting the clock time before you start a task and after you complete a task allows you to measure how long the task takes.

8.3.2 A Brute-Force Solution

Let's return to the main theme of using brute force to crack the rail fence cipher. We can now count the number of decrypted words that are found in the dictionary. We can improve our algorithm by remembering which rail size had the largest number of recognizable words. After we have tried all possible rail sizes, we can simply look at the message with the most correct words. Note that no list of words is likely to contain every word in our message, especially if the word is a proper noun, such as a place or person's name.

Listing 8.2 shows a complete solution for breaking the rail fence cipher. On line 9, a loop begins to iterate over all the words returned by `railDecrypt`. If a word is in the dictionary of known words, we increment

`goodCount` by 1. By now you will recognize this as the accumulator pattern, where we count values as they are found. On lines 12–14, we use the minmax pattern for finding the max value in a group. When the current decrypted word list contains more known words than any of the previous decryptions, we remember it by setting `maxGoodSoFar` to the number of real words. In addition, we remember the best version of the message so far by assigning it the name `bestGuess`.

```
1  def railBreak(cipherText):
2      wordDict = createWordDict('wordlist.txt')
3      cipherLen = len(cipherText)
4      maxGoodSoFar = 0
5      bestGuess = "No words found in dictionary" #default response
6      for i in range(2, cipherLen + 1):
7          words = railDecrypt(cipherText, i)
8          goodCount = 0     #reset for new list
9          for w in words:
10             if w in wordDict:
11                 goodCount = goodCount + 1
12         if goodCount > maxGoodSoFar:   #if more words in this list
13             maxGoodSoFar = goodCount
14             bestGuess = " ".join(words) #join words in list with space
15     return bestGuess
16
```

Listing 8.2 A brute-force algorithm for breaking the rail fence cipher

Description

Let's take a closer look at the assignment statement on line 14. The expression `"".join(words)` is a very useful combination of strings and lists. In fact, you can think of it as the opposite of the `split` function. In this example, the `join` method glues together the strings in the list `words`, separating the strings with the space character. Note that you can join the words together using any (and as many) characters as you want. If you wanted to separate the words with two dashes, you would simply change the call to `"--".join(words)`. **Table 8.1** summarizes the `join` method.

TABLE 8.1 The String Method `join`

Method	Use	Explanation
`join`	`separator.join(list)`	Returns a string composed of the elements in the list, separated by the string used to call the method.

Session 8.2 shows that we are able to find the correct plaintext for our ciphertext.

```
In [1]: cipherText = "n aci mreidontoowp mgorw"
        railBreak(cipherText)

Out[1]: 'new ipad coming tomorrow'
```

Session 8.2 Running the `railBreak` function

Description

Try It Out

8.4 Given the list of words `['the', 'quick', 'brown', 'fox']`, use the `join` method on the following separator strings: '

', ':', ', ', '--'.

8.5 Run the `railBreak` function from Listing 8.2 on the ciphertext `"w det zhoedhpzorr eia"`.

8.6 Use the rail fence encryption scheme to encrypt your own message. Give the ciphertext to a partner to decrypt.

8.7 Find a different word list on the Internet to use with the `railBreak` function. Compare the values of `maxGoodSoFar` for different word lists on the same ciphertext message.

8.3.3 A Rail Fence Decryption Algorithm

It is now time to write the `railDecrypt` function we have been using in the previous sections. The key to the rail decryption algorithm is to remember how the message was put together in the first place. With that set of steps in mind, the `railDecrypt` function will simply undo the original operation.

Let's go back to our original example and examine the encrypted string ("n aci mreidon toowp mgorw") a bit more carefully. We know that in this case we used three rails. We also know that because the string is 24 characters long, each rail was 24 ÷ 3 = 8 characters. Now we know that characters 0, 0 + 8, and 0 + 2 * 8 are the first characters of each row and, therefore, are the first three letters of the decrypted message. As you can see in **Table 8.2**, character 0 is '`n`', character 8 is '`e`', and character 16 is '`w`'. To determine the next three characters of the message, we shift each of those three positions 1 to the right. Shifting to the right gives us 1, 1 + 8, and 1 + 2 * 8. This gives us the next three characters of `' '`, `'i'`, and `'p'`.

TABLE 8.2 Ciphertext with Character Indices

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
n		a	c	i		m	r	e	i	d	o	n	t	o	o	w	p		m	g	o	r	w	

Description

The method we used for finding the first six characters in the decrypted message is an example of something the computer does for us all the time. That is, it translates the coordinates for data stored in tabular form into linear form. The memory in the computer is a one-dimensional structure. So, if we want to store a two-dimensional table in the computer's memory, we need to map from the "row, column" coordinates corresponding to a cell in the table to a location in memory. There are two ways to do the mapping: **row-major order** and **column-major order**. Row-major order is most often used.

Table 8.3 illustrates how 18 contiguous memory locations would be organized using both row-major and column-major order.

TABLE 8.3 Examples of Row- and Column-Major Order

Row-Major Order					
1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18

Column-Major Order					
1	4	7	10	13	16
2	5	8	11	14	17
3	6	9	12	15	18

Description

Row-major order stores all the values on the first row, one after the other; it then stores all the values on the second row, then the third row, and so on. Column-major order stores all the values in the first column, one after another, followed by the second column, the third column, and so on. The question is: How do you compute the position in memory for any one row–column value?

For row-major order, the answer is a generalization of the pattern we employed when figuring out how to decode our secret message. For a given row and column, we find the value by using the following formula:

```
position = column + (row × rowLength)
```

In **Listing 8.3**, this formula appears on line 6 as part of the `railDecrypt` function.

```
1 def railDecrypt(cipherText, numRails):
2     railLen = len(cipherText) // numRails
3     solution = ''
4     for col in range(railLen):
5         for rail in range(numRails):
6             nextLetter = col + (rail * railLen)
7             solution = solution + cipherText[nextLetter]
8     return solution.split()
```

Listing 8.3 Decrypting the rail fence cipher

Description

Let's look at the entire `railDecrypt` function. The first thing to notice is that the ciphertext and the number of rails are both parameters passed to the `railDecrypt` function. The nested `for` loops in lines 4–7 act as if you are accessing the characters of the message in the original two-dimensional table format. The calculation on line 6 allows us to find the index into the one-dimensional string.

As a final note, if you have not already written the `railEncrypt` function, you might think of this function as a transformation of the original plaintext string, using a column-major order scheme. The equation for finding the memory location in a column-major order scheme is

```
position = row + (column × numRows)
```

The column-major calculation allows you to treat the plaintext string as if it were a table of characters stored in column-major format. Iterating over the string in a column-by-column manner and using the accumulator pattern gives you the ciphertext string you have just decoded.

Try It Out

8.8 Write the `railEncrypt` function using the column-major access pattern just described.

8.9 Now that you have seen the details of the `railDecrypt` function, you can make it smarter in two ways:

a. You do not need to check cases where the number of rails is greater than the message length divided by 2. Can you explain why?

b. You need to check only cases where the number of rails evenly divides the total message length. Can you explain why?

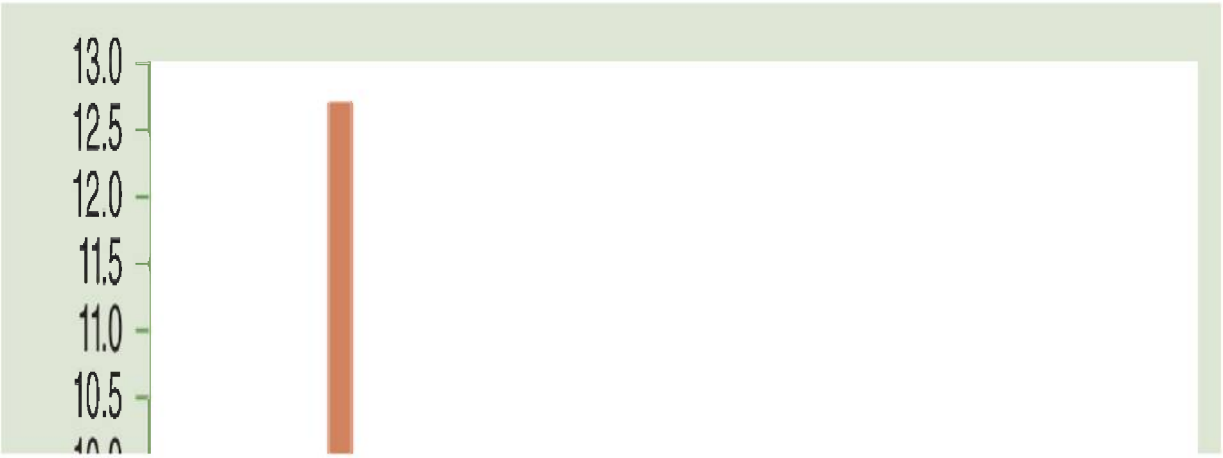
8.4 Cracking the Substitution Cipher

The substitution cipher is much more difficult to crack than the rail fence cipher. As we mentioned previously, 26! or roughly 4×10^{26} different possible arrangements of the alphabet could be used as a key. Clearly, the brute-force strategy for finding the key would take an extremely long time. However, the substitution cipher does have a fatal flaw that we can exploit.

The flaw in the substitution cipher allows us to take advantage of patterns in the English language that can help us deduce letters in the key. The first pattern we will exploit is that some letters in the English language are used more frequently than others. If we count the number of times each letter occurs in a document such as this text, or any English-language text, we would find that the letters *e*, *t*, *a*, *o*, and *i* occur much more frequently than any other letters in the alphabet.

8.4.1 Letter Frequency

We can write our own program to calculate the frequency of letters in a text. To report the letter frequency, we might simply count the total number of occurrences of each letter. However, that total would change depending on the size and nature of the document. A better approach is to report each letter's percentage—that is, the total count for each letter divided by the total number of letters. We will keep our analysis simple. **Figure 8.1** shows a graph of the results we are seeking from our letter frequency program.



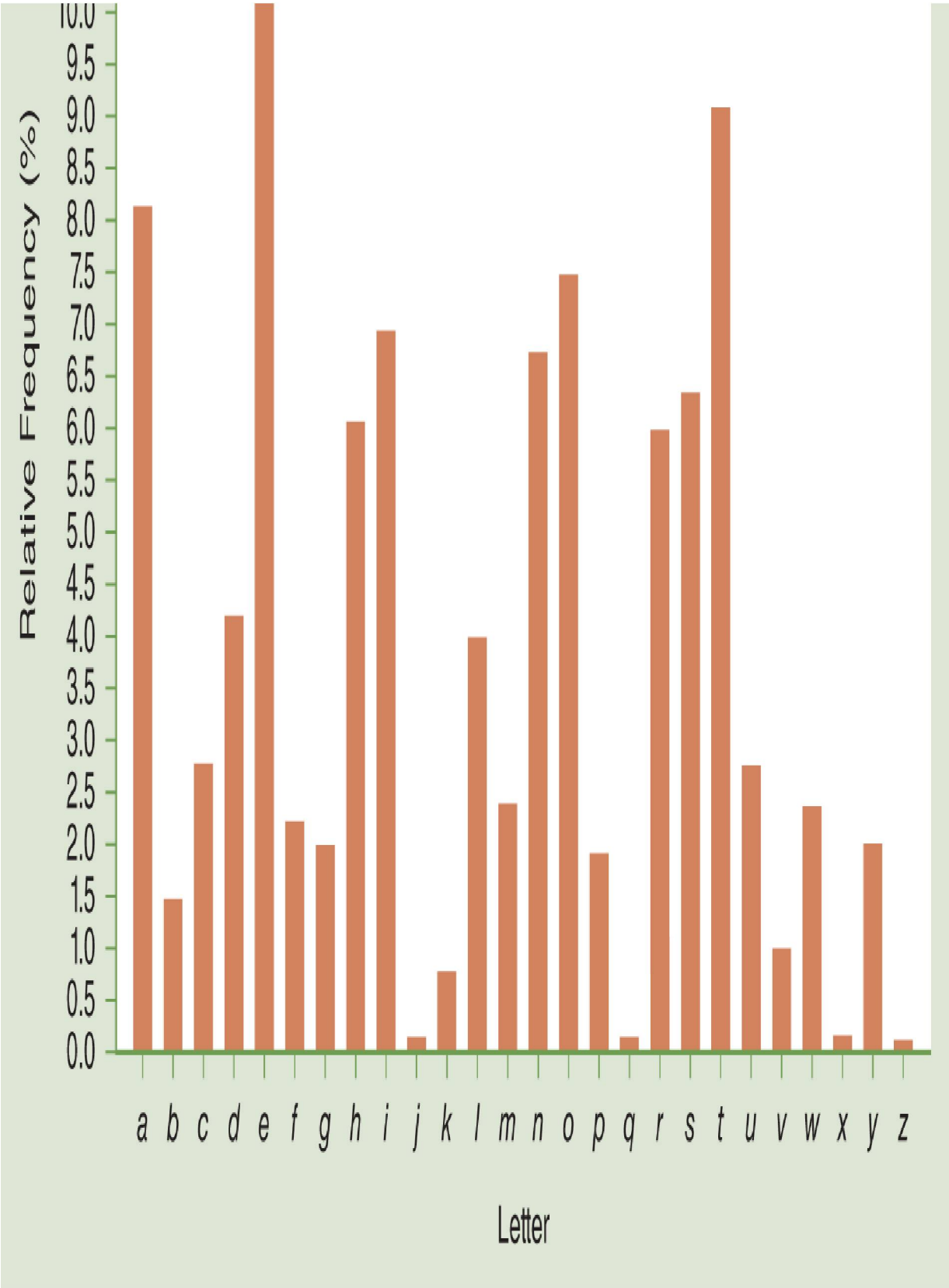


Figure 8.1 Relative letter frequencies.

Description

To generate the data for the graph in Figure 8.1, we will write a Python function that takes a string, `text`, as a parameter. For each letter in the alphabet, the function prints the number of times each letter occurred in `text` and the percentage of the total characters in `text` that the count represents. Counting the letters is easy because we can use the `count` method of the string class. The trickier part of this exercise is that to get an accurate percentage, we need to ignore nonletter characters such as spaces, punctuation marks, and numbers.

One effective way to ignore nonletter characters is to remove them from the string before we start counting. Thus, we need a way to remove all the nonletter characters from `text`. Earlier, we wrote a function that can remove characters from a string, called `removeMatches`. We can write a variation of this function, r`emoveNonletters`, by creating a string composed of all the letters in the alphabet and only adding characters to the new string if we find that character in the alphabet string. **Session 8.3** shows the `removeNonletters` function. The text string at the end of the session is ready for analysis by our counting function.

In [1]:
```
1 def removeNonletters(myString):
2     myString = myString.lower()
3     newStr = ""
4     letters = 'abcdefghijklmnopqrstuvwxyz'
5     for ch in myString:
6         if ch in letters:
7             newStr = newStr + ch
8     return newStr
```

In [2]:
```
1 text = "Are there 25, 26, or 27 non-letters to remove?"
2 removeNonletters(text)
3
```

Out[2]:
```
'arethereornonletterstoremove'
```

Session 8.3 Demonstrating `removeNonletters`

Description

To count all the letters of the alphabet, we will use the dictionary technique to keep track of the frequency of numbers. The key for each entry in the dictionary is the letter. To build our dictionary, we will simply iterate over all the characters in a string. The dictionary will initially keep track of the number of times each character occurs in the string. After all the counts

have been tallied, we will make a second pass through the dictionary and convert the counts into percentages. **Listing 8.4** shows the Python code for the letter-counting function.

```
1  def removeNonletters(myString):
2      myString = myString.lower()
3      newStr = ""
4      letters = 'abcdefghijklmnopqrstuvwxyz'
5      for ch in myString:
6          if ch in letters:
7              newStr = newStr + ch
8      return newStr
9
10 def letterFrequency(text):
11     text = removeNonletters(text)
12     lCount = {}
13     total = len(text)
14     for ch in text:    # count each letter's occurrence
15         lCount[ch] = lCount.get(ch, 0) + 1
16     for ch in lCount:  # calculate percentages
17         lCount[ch] = lCount[ch] / total
18     return lCount
```

Listing 8.4 Counting the frequency of each letter in the alphabet

Description

Let's look at a session where we calculate the text frequencies of the letters in H. G. Wells's *The War of the Worlds*, a novel that is free from copyright and freely downloadable from Project Gutenberg (http://www.gutenberg.org). Project Gutenberg includes this statement with the downloadable copy of the book:

> This eBook is for the use of anyone anywhere in the United States and most other parts of the world at no cost and with almost no restrictions whatsoever. You may copy it, give it away, or reuse it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.org. If you are not located in the United States, you will have to check the laws of the country where you are located before using this eBook.

We will save the `.txt` version of the book as `wells.txt`. You may note when you download this book that the `.txt` file is in UTF-8 format. Thus, to read this file, we need to specify the encoding format in the `open` function using the `encoding` keyword parameter. We use the value `'utf-8'` for the text format.

After opening the file, we use the `read` method to read in the entire file from disk as one string. We then use our `letterFrequency` function to compute the relative frequencies of each letter. **Session 8.4** demonstrates the use of `letterFrequency`.

```
In [1]: with open("wells.txt", 'r', encoding='utf-8') as wells:
            text = wells.read()
```

```
In [2]: lf = letterFrequency(text)
        for letter in 'abcdefghijklmnopqrstuvwxyz':
```

```
    print(letter, lf.get(letter))
```

```
a 0.08320891420222216
b 0.015231738758834941
c 0.023973586498189568
d 0.04755707025750327
e 0.1250988555804197
f 0.022874206052266675
g 0.02420055536444462
h 0.060274419544856496
i 0.06815804125868422
j 0.0009788032357248995
k 0.00767438479024885
l 0.037960542881156974
m 0.0256013788358625
n 0.07199523365380864
o 0.07110509013146463
p 0.018068849587023057
q 0.0006738138216946772
r 0.05987722402891016
```

```
s 0.06015029594612326
t 0.0966780978590452 4
u 0.026817790103448154
v 0.008844693006876448
w 0.023047979090493196
x 0.0014646584650521142
y 0.018104313472375407
z 0.0003794635732701603
```

Session 8.4 Letter frequency for *The War of the Worlds*

Description

Of course, it would be much more interesting to see the information sorted from most frequent letter to least frequent. To do this, we will first extract the information from the dictionary into a list, and then sort the list. We can extract all the key–value pairs from a dictionary using the `items` method, along with the `list` function. For example, the first three elements in the list returned by `items` in our letter frequency dictionary are as follows:

```
In [1]: list(lf.items())[:3]

Out[1]: [('t', 0.0966780978590452 4),
         ('h', 0.060274419544856496),
         ('e', 0.125098855804197)]
```

Description

Note that the order of the items in the dictionary is the same as the order in which the keys were entered. As you can see, the first three letters of the book are "the."

The `sort` function for lists, as shown in **Table 8.4**, is extremely powerful. Let's see how we can use it to sort this list from most to least frequent letter. The first question you might ask is: How does the `sort` function know which value to use when it sorts this list? Remember, a list can contain *any* Python object, so the `sort` function must be able to sort *any* kind of object. In this case, the elements on our list are tuples, so the `sort` function defaults to using the first element of the tuple. Fortunately, the `sort` function takes some optional keyword parameters that make it more flexible.

TABLE 8.4 The `sort` Method for Lists

Method with Defaults	Use	Explanation
`sort(key = None, reverse = False)`	`list.sort(key = sortKey, reverse = True)`	Performs an in-place sort of the list—that is, the list is modified. Keyword Parameters: `key`: specifies a function to use for comparing items; default is `None`, which means to sort on the first element `reverse`: True specifies the list should be sorted in reverse order (high to low); False (default) is low to high

Two keyword parameters for the `sort` method are `key` and `reverse`, as explained in Table 8.4. If we call `sort` with the keyword parameter `reverse` set to `True`, the `sort` function will sort the list from largest to smallest value,

instead of the other way around. **Session 8.5** demonstrates a reverse sort on a simple list of integers.

```
In [1]: xList = [3, 7, 4, 9]
        xList.sort(reverse = True)
        xList

Out[1]: [9, 7, 4, 3]
```

Session 8.5 Sorting a list in reverse

Description

The keyword parameter, `key`, lets us write a function that takes one object as a parameter and returns a value from that object that we should use as the sort key for that object. For example, suppose we had a list of turtles that we wanted to sort according to the *x* coordinate of each turtle. In this scenario, our function would return the *x* value for the turtle. In the case of our dictionary, we want to write a function that returns the frequency number from a tuple. This is a short two-line function, as shown in **Listing 8.5**.

```
1 def getFreq(t):
2     return t[1]   # return second item in the tuple
```

Listing 8.5 Returning the second value in a tuple

Description

Using the `getFreq` function along with the `key` and `reverse` parameters, we can now sort our letter frequency dictionary from highest frequency to

lowest (see **Session 8.6**). The `sort` method implements an **in-place sort**—that is, it modifies the list it is sorting. It does not create a new list, nor does it return any value. A common mistake is to write a statement such as `myList = myList.sort()`. Try this for yourself, and you will see that unfortunately `myList` has been set to `None` rather than being the sorted list.

```
In [1]: lfList = list(lf.items())
        lfList.sort(key = getFreq, reverse = True)
        for entry in lfList:     # each entry is a tuple
            print("{0} {1:.3f}".format(entry[0], entry[1]))
```

```
e 0.125
t 0.097
a 0.083
n 0.072
o 0.071
i 0.068
h 0.060
s 0.060
r 0.060
d 0.048
l 0.038
u 0.027
```

```
u 0.027
m 0.026
g 0.024
c 0.024
w 0.023
f 0.023
y 0.018
p 0.018
b 0.015
v 0.009
k 0.008
x 0.001
j 0.001
q 0.001
z 0.000
```

Session 8.6 Sorted letter frequency from *The War of the Worlds*

Description

RECAP:
The `sort` method sorts in place and does not return a value.

Try It Out

8.10 Evaluate the following statements:

```
myList = [4, 5, 3, 9, 1, 6, 7]
myList.sort()
```

Description

What is the value of `myList` when you are done?

8.11 Instead of writing the new function, `removeNonletters`, perform the same operation by calling the existing `removeMatches` function once to remove all letters in the alphabet and a second time to remove all the nonletter characters that are left.

8.4.2 Ciphertext Frequency Analysis

Letter frequency is critical for breaking a substitution cipher. Because a substitution cipher makes a one-for-one letter substitution from the plaintext to the ciphertext, the most frequently occurring letter in the plaintext will also appear most frequently in the ciphertext. This small amount of information is enough to begin breaking the encryption through **frequency analysis**.

Let's do a bit of cryptanalysis on the following paragraph:

ul ilahvble jkhbevbt hk kul letl cs kul dvk kuhk kul kuvbt uhe ahel sci vkjlzs jkhivbt hk vkj jkihbtl hddlhihbwl hjkcbvjule wuvlszq hk vkj mbmjmhz juhdl hbe wczcmi hbe evazq dliwlvnvbt lnlb kulb jcal lnvelbwl cseljvtb vb vkj hiivnhz kul lhizq acibvbt ohj ocbelismzzq jkvzz hbe kul jmb xmjk wzlhivbt kul dvbl killj kcohiej olqgivetl ohj hzilheq ohia ul eve bck ilalagli ulhivbt hbq gviej kuhk acibvbt kulil ohj wlikhvbzq bc gillrl jkviivbt hbe kul cbzq jcmbej olil kul shvbk

acnlalbkj sica ovkuvb kul wvbeliq wqzvbeli ul ohj hzz hzcbl cb kul wcaacb

The first step in breaking this code is to apply our letter frequency analysis to the encrypted text. **Table 8.5** shows the results.

TABLE 8.5 Relative Frequency of Letters in the Ciphertext

Frequency	Letter	Frequency	Letter
0.134	l	0.024	q
0.095	b	0.022	w
0.086	k	0.022	o
0.086	h	0.018	s
0.077	v	0.018	m
0.070	i	0.013	d
0.059	j	0.011	n
0.053	u	0.009	g
0.051	e	0.002	x
0.046	c	0.002	r
0.042	z	0.000	y
0.031	a	0.000	p
0.029	t	0.000	f

Description

One approach would be to hope that we find an exact match in the frequencies of letters between the plaintext sample and the ciphertext. If this

were the case, we could figure out the key by matching the letters in order of their popularity: *l -> e, b -> t, k -> a, h -> n*, and so on. Using these letter substitutions, the first line of ciphertext would be decrypted as follows: "se ieunoter hantrotm na ase erme df ase yoa asna ase asotm snr unre fdi oahelf haniotm na oah haintme." Clearly, a blind mapping is not the answer.

8.4.3 Letter Pair Analysis

Because the frequency of letters is bound to vary to some extent (particularly in smaller samples), we need to try to match the letters with a bit more flexibility. Let's begin by assuming that the four most popular plaintext letters will match the four most popular ciphertext letters in some order. For example, a ciphertext `'l'` will be either `'e'`, `'t'`, `'n'`, or `'a'`. Likewise, the ciphertext letters `'b'`, `'k'`, and `'h'` will match one of `'e'`, `'t'`, `'n'`, or `'a'`.

Notice that two of the four most popular plaintext letters are vowels and two are consonants. One way that we can statistically tell the difference between a vowel and a consonant is to notice that vowels frequently appear before or after almost any other letter in the alphabet. Consonants typically have a small number of letters that they appear next to. This suggests that we can write a variation of our letter frequency function to figure out how frequently each letter appears next to another letter.

Although this may seem like a daunting task, it is important to understand that if we start with our basic letter frequency function, we have already made a great deal of progress toward our goal. Let's look at the additional problems we need to solve:

1. Rather than count the number of times each letter occurs, we want to keep a list of neighboring letters. This list will contain all the letters that appear before or after the letter in question.
2. The output of the function will be a dictionary that contains a key for each letter of the alphabet and a list of neighboring letters.
3. We need to be careful about how we account for nonletters in our counting. If we simply delete nonletters as we did previously, we will end up creating false neighbors.

Once again, we want to keep track of some information that is indexed by a letter—which again suggests that we will use a dictionary. The difference is that we need to keep track of something more complicated than a simple number. The good news is that dictionaries can store any Python object as a value. To keep track of a character's neighbors, we use that character as the key and store a list of neighboring characters as the value. For example, if our dictionary is called `myNeighbors`, then `myNeighbors['q']` will evaluate to the list of letters that occur next to `'q'`. For example, in one large text, the letter `'q'` appears next to the letters `['u', 'e', 'o', 's', 'n', 'd', 'i', 'h', 'a', 'c']`.

Once we know that we will use a list for keeping track of the neighbors, the next problem is easy to solve. We can use the `len` function to find the length of our list of neighbors after we have processed all the characters in the string.

Thus, our new function will include the following steps:

1. Create an empty dictionary `nbDict`.
2. Loop over all the characters in a string.
   a. If the current character is not in `nbDict`, add an empty list. Otherwise, retrieve the list of neighbors already stored.
   b. Add the next character as a neighbor of the current character.
   c. If the next character is not in `nbDict`, add an empty list. Otherwise, retrieve the list of neighbors already stored.
   d. Add the current character as a neighbor of the next character.
3. Loop over all the keys of the dictionary, and replace the list with the length of the list.
4. Return the dictionary.

A couple of ideas in this overview of the new function are worth considering. First, notice that when we are looking at the character at index `i` in the character string, we add the character at index `i+1` as a neighbor. At the same time, we add the character at index `i` as a neighbor of the character at index `i+1`. In this way, we can ensure that as we move through the list, we

are getting the neighboring characters that occur both before and after each character.

Second, we need to be careful about which characters we add to the list of neighbors. If a letter is already on the neighbor list, we do not want to add it again. In addition, because we do not wish to count spaces, punctuation marks, numbers, or any other nonletter as neighbors, we will ignore those characters. The easiest way to make sure that we are counting only letters is to check whether the character is one of the 26 letters of the alphabet. To do this, we write a little function that takes two parameters: a character and a list. Let's call this function `maybeAdd`. **Listing 8.6** shows the complete function to add a character to a list under the conditions just described.

```
1 def maybeAdd(ch, toList):
2     if ch in 'abcdefghijklmnopqrstuvwxyz' and ch not in toList:
3         toList.append(ch)
```

Listing 8.6 Conditionally adding a character to a list

Description

The `maybeAdd` function is a good example of a function that modifies one of its parameters. Notice that we pass a list as a parameter, but instead of returning the list, the function appends the character directly to the list. This simplifies the process of adding characters to the list item in the dictionary.

HEADS UP:
The `maybeAdd` function modifies the parameter list in place rather than returning the modified list.

Session 8.7 shows the `maybeAdd` function in action.

In [1]:
```
myList = []
maybeAdd('a', myList)   # a is a letter and not in the list
myList
```

Out[1]:
```
['a']
```

In [2]:
```
maybeAdd('-', myList)   # - is not a letter
myList
```

Out[2]:
```
['a']
```

In [3]:
```
maybeAdd('b', myList)   # b is a letter and not in the list
myList
```

Out[3]:
```
['a', 'b']
```

In [4]:
```
maybeAdd('a', myList)   # a is already in the list
myList
```

```
Out[4]: ['a', 'b']
```

Session 8.7 Testing the `maybeAdd` function

Description

Listing 8.7 shows the complete `neighborCount` function. This function allows us to make a few interesting observations. First, we use the statement

```
for i in range(len(text) - 1)
```

```
1  def neighborCount(text):
2      nbDict = {}
3      text = text.lower()
4      for i in range(len(text) - 1):
5          nbList = nbDict.setdefault(text[i], [])
6          maybeAdd(text[i + 1], nbList)
7          nbList = nbDict.setDefault(text[i + 1], [])
8          maybeAdd(text[i], nbList)
9      for key in nbDict.keys():  #replace letters list with count
10         nbDict[key] = len(nbDict.get(key))
11     return nbDict
```

Listing 8.7 Creating a dictionary of neighbors

Description

rather than iterating over each character in the string, because we need to index the current character as well as the next character. If we simply used `len(text)`, we would eventually get an "index out of range" error.

Another statement to look at carefully is

```
nbList = nbDict.setdefault(text[i],[])
```

The `setdefault` method first checks whether the key is in the dictionary. If it is not, it adds the default value and returns a reference to the default value, which is then assigned to the list of neighbors. The `setdefault` method is described in **Table 8.6**.

TABLE 8.6 The `setdefault` Method for Dictionaries

Method	Use	Explanation
`setdefault(key, default)`	`value = nDict.setdefault(dict[i], defaultValue)`	If the key is not in the dictionary, insert `key` with a value of `default` and return `default`. If the key is in the dictionary, return the value for the key.

The statement given in the preceding paragraph is a shortcut for the Python code shown in **Listing 8.8**.

```
1  nbList = nbDict.get(text[1])
2  if nbList == None:
3      nbDict[text[i]] = []
4      nbList = nbDict[text[i]]
```

Listing 8.8 Adding a new default value to a dictionary

Description

This pattern is quite common when using dictionaries. Using the `setdefault` method is a good idea because it reduces the amount of code to write and reduces your chances of making an error.

HEADS UP:
Using the dictionary `setdefault` method requires less code and reduces the chance for error.

Next, we will use the `neighborCount` function to continue our analysis of the ciphertext. **Session 8.8** shows the `neighborCount` function in action. After building the dictionary `freqDict`, we print the number of neighbors for the commonly used letters `'e'`, `'n'`, and `'t'`.

```
In [1]: freqDict = neighborCount(text)
        for i in 'ent':
            print(i, freqDict[i])

        e 26
        n 25
        t 21
```

Session 8.8 Number of neighbors for *e*, *n*, and *t*

Description

As we can see in Session 8.8, *e* appears next to 26 different letters, *n* appears next to 25 different letters, and *t* appears next to 21 different letters.

We were hoping for a result that would allow us to differentiate easily between vowels and consonants. Unfortunately, this is not it. For these counts to be more useful, we need a bit more detail. For example, *e* probably appears frequently next to many different letters, whereas *n* and *t* may appear next to some letters very frequently and next to other letters infrequently. We need to see the frequency with which different letters appear next to each other. What we need to build is a table like that shown in **Table 8.7**.

TABLE 8.7 Letter–Pair Frequency Analysis

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	0	3	5	1	0	4	3	0	0	5	2	8	0	2	0	10	5	8	1	1	6	0	0	0
b	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0	1	0
c	0	0	0	0	4	0	0	1	1	0	0	1	0	2	2	0	0	1	0	0	0	0	0	0	1	0
d	3	0	0	0	9	0	2	0	6	0	0	0	0	9	0	0	0	2	3	0	0	0	0	0	1	0
e	5	1	4	9	4	1	3	17	2	0	0	2	7	7	0	3	0	15	3	0	0	4	2	0	1	2
f	1	0	0	0	1	0	0	0	0	0	0	2	0	0	3	0	0	2	0	0	1	0	0	0	0	0
g	0	0	0	2	3	0	0	0	1	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0
h	4	0	1	0	17	0	0	0	3	0	0	0	0	0	0	0	0	0	2	16	0	0	0	0	0	0
i	3	1	1	6	2	0	1	3	0	0	0	2	1	20	0	2	0	8	2	8	0	4	1	0	0	0
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	5	0	1	0	2	2	0	0	2	0	0	6	1	2	3	0	0	2	0	0	1	0	0	0	7	0
m	2	1	0	0	7	0	0	0	1	0	0	1	2	0	7	0	0	1	0	0	0	0	0	0	0	0
n	8	0	2	9	7	0	11	0	20	0	0	2	0	0	8	0	0	2	0	2	4	0	0	0	1	0
o	0	0	2	0	0	3	0	0	0	0	0	3	7	8	0	0	0	4	2	3	2	1	2	0	0	0
p	2	0	0	0	3	0	0	0	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0

q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	10	2	1	2	15	2	0	0	8	0	0	2	1	2	4	0	0	4	0	3	1	0	0	0	1	0
s	5	0	0	3	3	0	0	2	2	0	0	0	0	0	2	0	0	0	0	12	4	0	0	0	0	0
t	8	0	0	0	0	0	0	16	8	0	0	0	0	2	3	0	0	3	12	0	0	0	0	0	0	0
u	1	0	0	0	0	1	0	0	0	1	0	1	0	4	2	0	0	1	4	0	0	0	0	0	0	0
v	1	0	0	0	4	0	0	0	4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
w	6	0	0	0	2	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
y	0	1	1	1	1	0	0	0	0	0	0	7	0	1	0	0	0	1	0	0	0	0	0	0	0	0
z	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Description

Let's improve `neighborCount` so we can produce a table of letter–pair frequency counts. This may allow us to learn enough to make better guesses about which letters are vowels and which are consonants. The most important change we will make is to replace the list of letters with a dictionary so that we can keep a count for each neighboring letter. In doing so, we will create a dictionary of dictionaries. As an example, `d['a']['x']` tells us the number of times that the letters `'a'` and `'x'` appear next to each other.

The necessary modifications are actually quite simple. In the `neighborCount` function, we need to change only the calls to `setdefault` to create a dictionary instead of an empty list. We do this by changing `[]` to `{}` for the default value and removing the second `for` loop that replaces the letters with a count.

Another change occurs in the `maybeAdd` function, which now updates a count of each neighboring letter in a dictionary rather than simply appending a letter to a list. **Listing 8.9** shows the new version of `neighborCount` and

`maybeAdd`. Once again, the `setdefault` function simplifies the implementation of the `maybeAdd` function.

```
1  def maybeAdd(ch, toDict):
2      if ch in 'abcdefghijklmnopqrstuvwxyz':
3          toDict[ch] = toDict.setdefault(ch, 0) + 1
4
5  def neighborCount(text):
6      nbDict = {}
7      text = text.lower()
8      for i in range(len(text) - 1):
9          nbList = nbDict.setdefault(text[i], {})
10         maybeAdd(text[i + 1], nbList)
11         nbList = nbDict.setdefault(text[i + 1], {})
12         maybeAdd(text[i], nbList)
13     return nbDict
```

Listing 8.9 New `maybeAdd` and `neighborCcunt` functions

Description

Session 8.9 shows the result of calling the `neighborCount` function on the entire book. Let's look at what we can learn about the letters *e*, *n*, and *t*. It seems that *e* appears more frequently next to a very large number of letters. In addition, although both *n* and *t* appear next to many different letters, they do

so much less frequently. Also note that *e* appears least frequently next to other vowels and several consonants such as *z* and *q*.

```
In [1]: d = neighborCount(text)
```

```
In [2]: print(d['e'])
```

```
{'h': 8632, 'j': 144, 'c': 1663, 't': 3143, 'n': 4541, 'b': 1268, 'r': 8255,
'w': 1436, 'l': 2894, 's': 4139, 'd': 4837, 'o': 230, 'v': 2087, 'f': 660,
'a': 1928, 'g': 1010, 'y': 798, 'k': 701, 'p': 1333, 'm': 2552, 'x': 333,
'e': 2604, 'u': 168, 'i': 900, 'q': 26, 'z': 70}
```

```
In [3]: print(d['n'])
```

```
{'e': 4541, 'b': 113, 'a': 5203, 'y': 204, 'o': 4017, 'i': 6684, 'u': 1257,
'd': 4040, 's': 915, 'c': 576, 'l': 226, 't': 1744, 'g': 3185, 'h': 32,
'm': 34, 'w': 292, 'r': 375, 'f': 101, 'v': 65, 'n': 278, 'k': 342, 'q': 7,
'p': 13, 'j': 10, 'x': 6}
```

```
In [4]: print(d['t'])
```

```
{'h': 10434, 'c': 705, 'u': 1712, 'e': 3143, 'i': 4516, 's': 3095, 'a':
3847, 'o': 3163, 'r': 1884, 'n': 1744, 'l': 586, 'f': 282, 'w': 215, 't':
864, 'y': 298, 'm': 80, 'p': 194, 'x': 84, 'b': 34, 'g': 29, 'd': 2}
```

Session 8.9 Testing the new `neighborCount` function

Description

As shown in Session 8.9, the data is quite difficult to interpret. Although we can present the data as shown in Table 8.7, a better way to get an idea of how the letter pairs work out is to create a histogram. **Figure 8.2** shows a histogram that compares the frequencies with which letters *a*, *e*, *n*, and *t* occur with the frequencies for all the other letters of the alphabet. This plot was produced with the `matplotlib` module of Python. This histogram readily shows us that *e* and *a* appear more frequently with more letters.



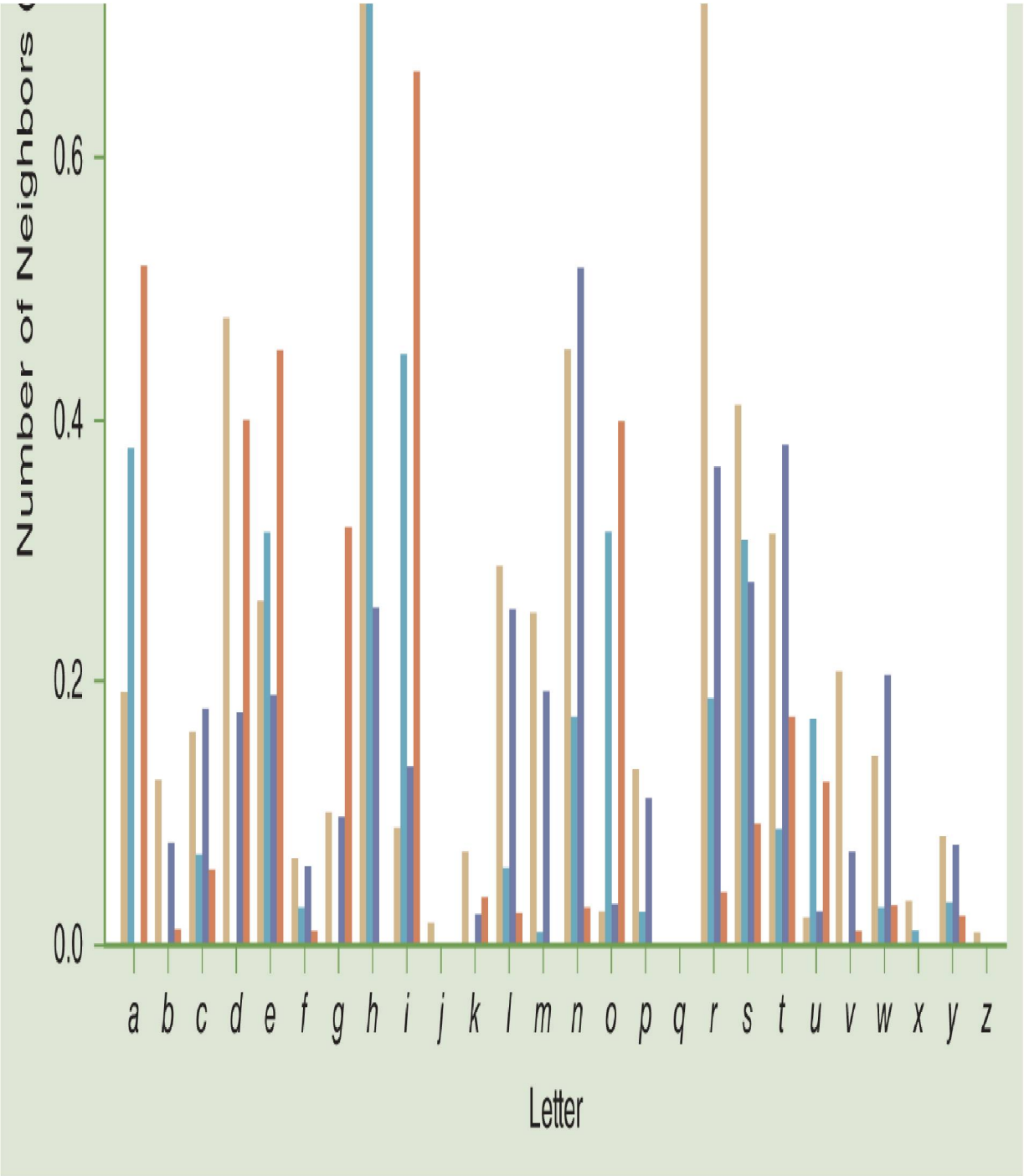
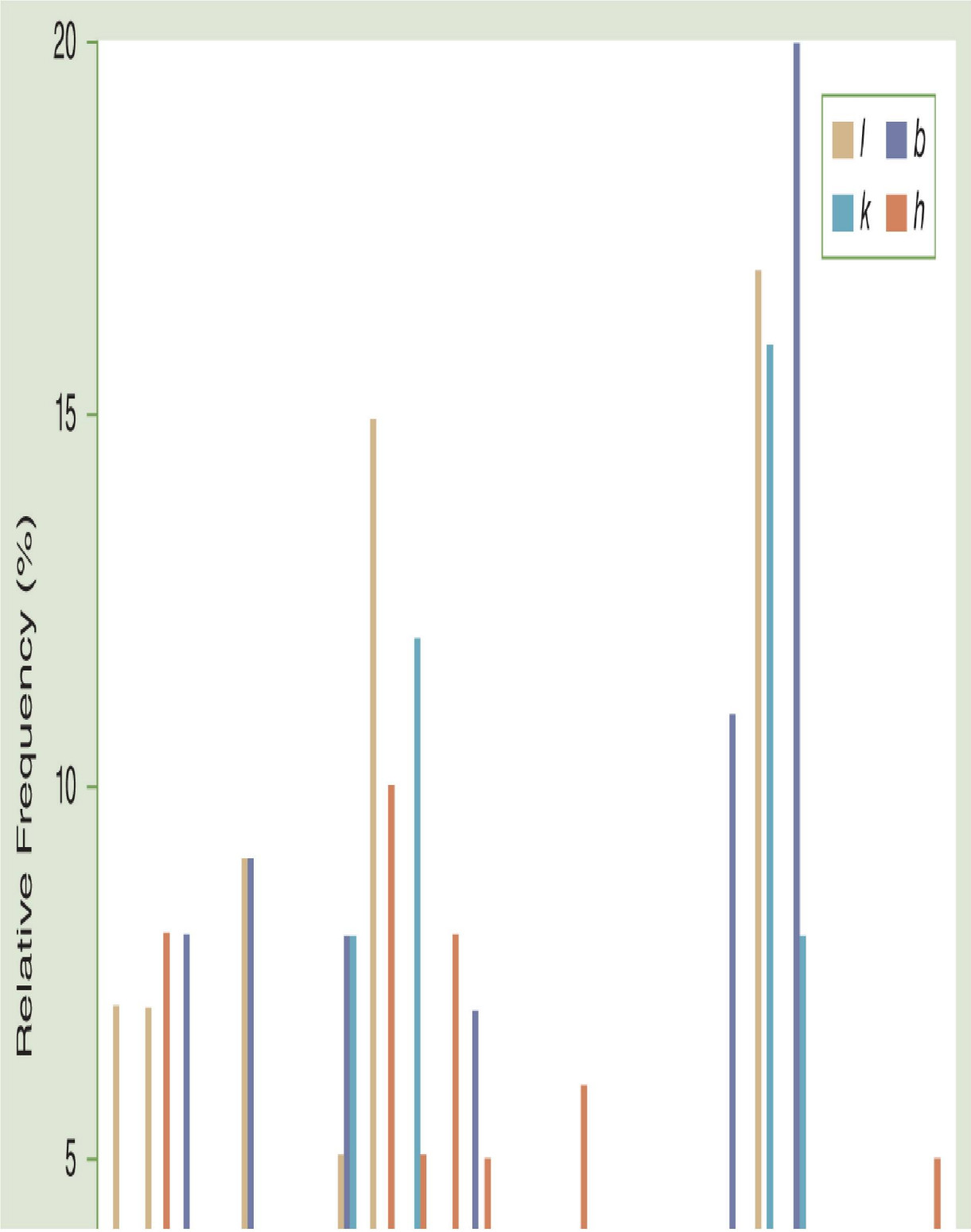


Figure 8.2 Letter–pair frequencies for *a*, *e*, *n*, and *t*.

Description

The histogram in **Figure 8.3** is for the ciphertext paragraph presented earlier in this chapter and corresponds to **Table 8.8**. You can see that *l* and *h*

are vowels because they appear more frequently with more letters. The letters *k* and *b* are consonants. But which of these ciphertext characters is *a* and which is *e*?



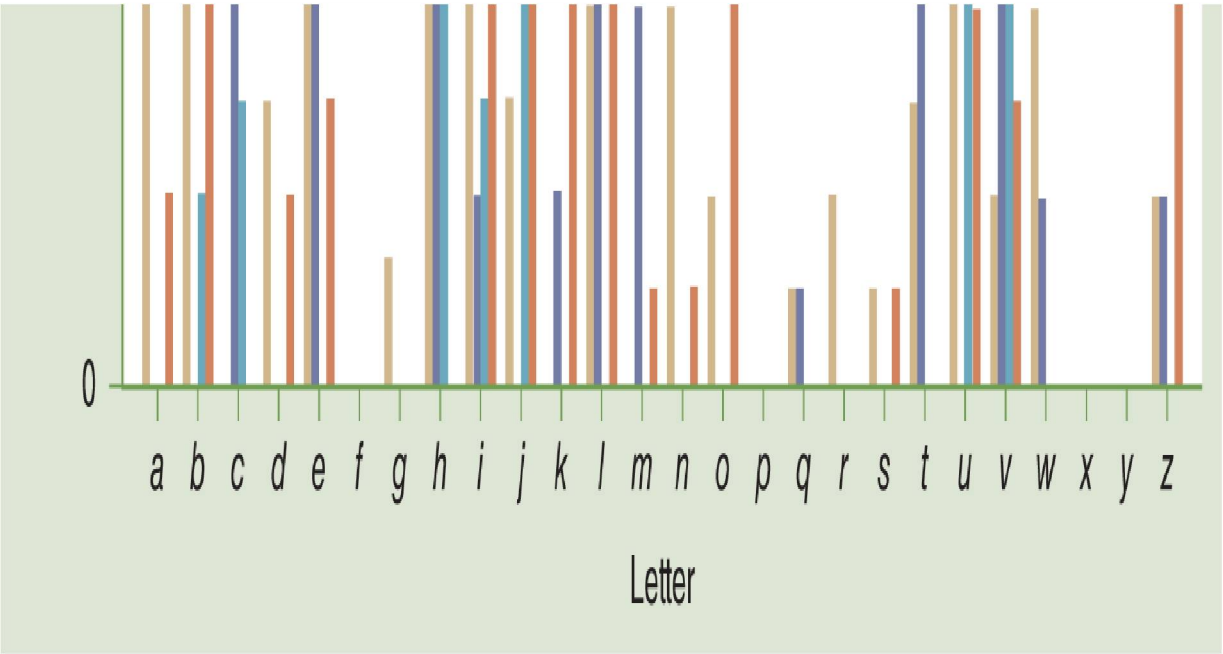


Figure 8.3 Letter frequency in the ciphertext example.

Description

TABLE 8.8 Frequency Pairs for the Four Most Common Ciphertext Letters

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
b	0	0	8	0	9	0	0	8	2	0	2	7	4	0	0	0	1	0	0	11	0	20	2	0	0	2
k	0	2	3	0	0	0	0	8	3	12	0	0	0	0	0	0	0	0	0	0	16	8	0	0	0	0
l	7	7	0	3	9	0	1	5	15	3	0	4	0	4	2	0	1	2	1	3	17	2	4	0	0	2
h	2	8	0	2	3	0	0	0	10	5	8	5	1	1	6	0	0	0	1	0	4	3	0	0	0	5

Description

We have three pieces of evidence that help us make a good guess about the letter *l*. First, it was the most common letter in the ciphertext paragraph.

Second, our letter–pair analysis indicates that *l* is a vowel because it appears relatively frequently with many other letters. Finally, we can see that the letter *l* appears next to itself four times but the letter *h* never appears next to itself. Because it is much more common for *e* to appear next to itself than the letter *a* does (unless you are writing a thesis about aardvarks), we can probably conclude that *l* maps to *e* and *h* maps to *a*.

Now that we have two letter mappings in place, let's plug them into the ciphertext and see if that gives us any clues that will help decrypt the letters *b* and *k*. We can use the Python string `replace` method to help us replace the ciphertext letters with the plaintext letters we have figured out. **Session 8.10** shows how to replace the ciphertext *l* with *E* and *h* with *A*. We will use uppercase for the decoded plaintext letters to make it clear which letters we have decoded and which we still need to work on.

```
In [1]: cipherText = cipherText.replace('l', 'E')
        cipherText = cipherText.replace('h', 'A')
        cipherText
```

```
Out[1]: 'uE iEaAvbEe jkAbevbt Ak kuE EetE cs kuE dvk kuAk kuE kuvbt uAe aAeE sci
vkjEzs jkAivbt Ak vkj jkiAbtE AddEAiAbwE Ajkcbvjuee wuvEszq Ak vkj mbmjmAz
juAdE Abe wczcmi Abe evazq dEiwEvnvbt EnEb kuEb jcaE EnveEbwE cseEjvtb vb
vkj AiivnAz kuE EAizq acibvbt oAj ocbeEismzzq jkvzz Abe kuE jmb xmjk
wzEAivbt kuE dvbE kiEEj kc oAiej oEqgivetE oAj AziEAeq oAia uE eve bck
iEaEagEi uEAivbt Abq gviej kuAk aci bvbt kuEiE oAj wEikAvbzq bc giEErE
jkviivbt Abe kuE cbzq jcmbej oEiE kuE sAvbk acnEaEbkj sica ovkuvb kuE
wvbeEiq wqzvbeEi uE oAj Azz AzcbE cb kuE wcaacb'
```

Session 8.10 First character replacement

Description

Try It Out

8.12 Create a frequency histogram for the ciphertext paragraph using `matplotlib`.

8.13 Produce a table similar to Table 8.7 using a different text file from Project Gutenberg. Compare and contrast the letter–pair frequencies.

8.4.4 Word Frequency Analysis

We have made some progress, but we still have more work to do. Let's employ another cryptanalysis technique to help with the next step. This technique will look at the short words—namely, the words that are one, two, or three letters long.

An easy way to do this is to reorder the words in the ciphertext and sort them by their length. Recall that the `sort` function can accept a key function as a parameter. **Listing 8.10** shows a Python function that accepts one word as a parameter and returns the length of the word. **Session 8.11** illustrates the use of our `sortByLen` function and provides us with some interesting new clues.

```
1  def sortByLen(w):
2      return len(w)
```

Listing 8.10 Function to compare two words by length

Description

```
In [1]: cipherWords = cipherText.split()
        cipherWords.sort(key = sortByLen)
        print(cipherWords)
```

```
['uE', 'Ak', 'cs', 'Ak', 'Ak', 'vb', 'uE', 'bc', 'uE', 'cb', 'kuE', 'kuE',
'dvk', 'kuE', 'uAe', 'sci', 'vkj', 'vkj', 'Abe', 'Abe', 'vkj', 'kuE', 'oAj',
'Abe', 'kuE', 'jmb', 'kuE', 'oAj', 'eve', 'bck', 'Abq', 'oAj', 'Abe', 'kuE',
'kuE', 'kuE', 'oAj', 'Azz', 'kuE', 'EetE', 'kuAk', 'aAeE', 'EnEb', 'kuEb',
'jcaE', 'xmjk', 'dvbE', 'oAia', 'kuAk', 'cbzq', 'oEiE', 'sica', 'kuvbt',
'juAdE', 'evazq', 'EAizq', 'jkvzz', 'kiEEj', 'gviej', 'kuEiE', 'sAvbk',
'AzcbE', 'vkjEzs', 'wczcmi', 'giEErE', 'jcmbej', 'ovkuvb', 'wcaacb',
'jkAivbt', 'jkiAbtE', 'wuvEszq', 'mbmjmAz', 'AiivnAz', 'acibvbt', 'kcoAiej',
'AziEAeq', 'uEAivbt', 'acibvbt', 'wvbeEiq', 'iEaAvbEe', 'jkAbevbt',
'EnveEbwE', 'cseEjvtb', 'wzEAivbt', 'iEaEagEi', 'jkviivbt', 'wqzvbeEi',
'oEqgivetE', 'wEikAvbzq', 'acnEaEbkj', 'AddEAiAbwE', 'Ajkcbvjuee',
'dEiwEvnvbt', 'ocbeEismzzq']
```

Session 8.11 Sorting ciphertext words by length

Description

The new clues from Session 8.11 are that the word *Ak* appears three times, *uE* appears three times, *Abe* appears four times, and *kuE* appears 10 times. So, we might ask these questions:

1. What are some popular three-letter words?
2. Which three-letter words end in "e," since the *E* in *kuE* is in place?
3. Which three-letter words begin with "a," since the *A* in *Abe* is in place?

Try It Out Exercise 8.15 at the end of this section asks you to write a function that reads a string and prints a list of words that are of a given length, ordered from most popular to least. If you run such a function on the text of *The War of the Worlds*, you would produce **Table 8.9**. Now the choices for the letters *k* and *u* become clear. Because the most popular three-letter word is *the* and it is the only such word in the top 10 that ends in *e*, it is a pretty safe assumption that *kuE* corresponds to the word *THE*. In addition, because *and* is the second most popular three-letter word, we can also make the assumption that *b* maps to *N* and *e* maps to *D*. **Session 8.12** shows what happens when we make these four additional substitutions.

TABLE 8.9 Three-Letter Word Frequency Counts

Word	Count
the	4,959
and	2,555
was	851
had	579
for	369
but	295
his	250
out	231
all	225
not	212

```
In [1]: cipherText = cipherText.replace('b', 'N')
        cipherText = cipherText.replace('e', 'D')
        cipherText = cipherText.replace('k', 'T')
        cipherText = cipherText.replace('u', 'H')
        cipherText
```

```
Out[1]: 'HE iEaAvNED jTANDvNt AT THE EDtE cs THE dvT THAT THE THvNt HAD aADE sci
vTjEzs jTAivNt AT vTj jTiANtE AddEAiANwE AjTcNvjHED wHvEszq AT vTj mNmjmAz
jHAdE AND wczcmi AND Dvazq dEiwEvnvNt EnEN THEN jcaE EnvDENwE csDEjvtN vN
vTj AiivnAz THE EAizq aciNvNt oAj ocNDEismzzq jTvzz AND THE jmN xmjT
wzEAivNt THE dvNE TiEEj TcoAiDj oEqgivDtE oAj AziEADq oAia HE DvD NcT
iEaEagEi HEAivNt ANq gviDj THAT aciNvNt THEiE oAj wEiTAvNzq Nc giEErE
jTviivNt AND THE cNzq jcmNDj oEiE THE sAvNT acnEaENTj sica ovTHvN THE
wvNDEiq wqzvNDEi HE oAj Azz AzcNE cN THE wcaacN'
```

Session 8.12 Second replacement, using word frequency

Description

As a bonus, in addition to *and* and *the*, other words such as *that*, *at*, *had*, and *then* also now appear. Considering that we have mapped only six letters, this result is promising. In fact, you can probably make some additional guesses based on what you see.

Referring back to the three-letter frequency counts in Table 8.9, let's try to decrypt a few more letters. First, we might notice that another popular three-letter word appearing in the partially decrypted ciphertext is the word *oAj*. Looking at our list of popular three-letter words, we see that *was* is the third most popular word. In addition, *was* has an *a* in the middle position. Scanning through the rest of the ciphertext, we notice that many words end in *j*. Of course, *s* is a very popular letter at the end of words because it makes a word plural. Once again, you can write a function to find out which letters appear most frequently at the end of a word. It is highly likely you would find that *s* appears most frequently at the end of a word. If we use the new mappings, *o* maps to *W* and *j* maps to *S*, we now find that our ciphertext looks like the following:

HE iEaAvNED STANDvNt AT THE EDtE cs THE dvT THAT THE THvNt HAD aADE sci vTSEzs STAivNt AT vTS STiANtE AddEAiANwE ASTcNvSHED wHvEszq AT vTS mNmSmAz SHAdE AND wczcmi AND Dvazq dEiwEvnvNt EnEN THEN ScaE EnvDENwE cs DESvtN vN vTS AiivnAz THE EAizq aciNvNt WAS WcNDEismzzq STvzz AND THE SmN xmST wzEAivNt THE dvNE TiEES TcWAiDS WEqgivDtE WAS AziEADq WAiaHE DvD NcT iEaEagEi HEAivNt ANq gviDS THAT aciNvNt THEiE WAS wEiTAvNzq Nc giEErE STviivNt AND THE cNzq ScmNDS WEiE THE sAvNT acnEaENTS sica WvTHvN THE wvNDEiq wqzvNDEi HE WAS Azz AzcNE cN THE wcaacN

We can get two more letters with another observation. Look at the last three letters of each word and notice that several end in *vNt*. In fact, 10 words end in the pattern *vNt*. A common English language suffix for words is *ing*. This gives us the vowel *i*, which means that we now know the mappings for the vowels *a*, *e*, and *i*.

At this point, we can also make an educated guess that the word *cs* in the ciphertext is probably *of*. The pair frequency profile of *c* fits that of a vowel, and the word *of* makes the most sense in the context of the phrase "STANDING AT THE EDGE cs THE dIT."

Finally, let's make one more substitution before looking at the whole text again. There is another common letter in the ciphertext that we have not

decoded yet: *z*. Notice that we have partially decoded the three-letter word *Azz*. The only three-letter word in our top 10 that has a double letter is the word *all*, so it is a good bet that *z* maps to *L*.

HE iEaAINED STANDING AT THE EDGE OF THE dIT THAT THE THING HAD aADE FOi ITSELF STAiING AT ITS STiANGE AddEAiANwE ASTONISHED wHIEFLq AT ITS mNmSmAL SHAdE AND wOLOmi AND DIaLq dEiwEInING EnEN THEN SOaE EnIDENwE OF DESIGN IN ITS AiiInAL THE EAiLq aOiNING WAS WONDEiFmLLq STILL AND THE SmN xmST wLEAiING THE dINE TiEES TOWAiDS WEqgiIDGE WAS ALiEADq WAiaHE DID NOT iEaEagEi HEAiING ANq gIiDS THAT aOiNING THEiE WAS wEiTAINLq NO giEErE STIiiING AND THE ONLq SOmNDS WEiE THE FAINT aOnEaENTS FiOa WITHIN THE wINDEiq wqLINDEi HE WAS ALL ALONE ON THE wOaaON

We have made good progress, finding the mappings for 12 letters with only 14 more letters to fill in. **Table 8.10** shows the letters we have identified and those letters we have yet to decode. We could continue to make some deductions about which letters go where. But let's look at another way we can use Python to help us by doing some automated pattern matching—a common application for computers.

TABLE 8.10 Progress on Mapping the Letters

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	h			e	l	s	t	u	v			z		b	c				j	k						

Description

Try It Out

8.14 Write a function `wordPop` that accepts a text and a length *N* and returns the list of all the words that are *N* letters long.

8.15 Write a function that returns the most popular ending letter for words.

8.16 Write a function that finds the most popular suffixes for words. You may want to try this function for two- and three-letter suffixes.

8.4.5 Pattern Matching with Partial Words

We would like to be able to select a word from the partially decoded ciphertext and ask the program to find a word from the dictionary that is the same length and matches the letters that we have decoded so far. It turns out that humans are pretty good at the task of **pattern matching**. For example, given the template *aADE*, you can probably come up with the words *MADE*, *FADE*, and *WADE*. Some of those would work for us; others would not. For example, *FADE* would not work for us because *F* has already been decoded. Ideally, we would like to give the function the set of letters that are legal to use when matching the coded parts of the words. If the computer finds only one match, we can then be fairly confident that we have decoded some additional letters.

Most modern programming languages provide a pattern-matching library that uses **regular expressions**. In Python, regular expressions are available through the `re` module, which we need to import to use regular expressions. Regular expressions are extremely useful and powerful programming tools. We will just barely scratch the surface of their application in this section, but even a simple use of regular expressions can make the final decryption step easier.

Regular expressions allow us to see if two strings match, much as when we use the `==` operator, except that in regular expressions we can use wildcard characters as part of the match. When using regular expressions, the `.` character is a *wildcard* that matches any character. For example, we can test whether the string `.ADE` matched `FADE`. The regular expression function that we use to test whether two strings match is called `match`. The `match` function takes two parameters: a regular expression and the string we want to

match against. **Session 8.13** illustrates the use of the `match` function, as well as some simple pattern matching.

```
In [1]: import re
```

```
In [2]: re.match('.ADE', 'FADE')
```

```
Out[2]: <re.Match object; span=(0, 4), match='FADE'>
```

```
In [3]: re.match('.ADE', 'FADER') # matches FADE, but not FADER
```

```
Out[3]: <re.Match object; span=(0, 4), match='FADE'>
```

```
In [4]: re.match('.ADE$', 'FADER') # no match, $ matches ADE at the end
```

```
In [5]: re.match('.ADE', 'ADE') # no match
```

```
In [6]: re.match('.ADE', 'FUDE') # no match
```

Description

If two strings match, the `match` function returns a Match object. At this point, we can just think of that object as being our indication of success. As you learn to use more advanced regular expressions, you will use the Match object for more interesting purposes. If the two strings do not match, the `match` function returns `None`. This allows us to use `re.match` as a condition in an `if` statement.

Because we want to be more specific in our matches at this point, we should learn some additional regular expression syntax. Within a regular expression, the square brackets (`[]`) allow us to define a set of characters, any one of which may match. For example, `[abc]` will match *a* or *b* or *c*, but not any other character. In addition, we can use the `+` character to match multiple instances of a character. For example, `[abc]+` will match *a* or *b* or *c* or *abc* or *aaaaaaa* or *bbbb* or *aaabbbccc* or any other combination of the letters *abc*. You read the `+` character to mean "one or more," so we say that the regular expression `[abc]+` means "match one or more instances of the characters *a*, *b*, or *c*."

You may also have noticed that `('.ADE', 'FADER')` matched `'FADE'`. This is because *FADE* appears enclosed within *FADER*. The regular expression matcher starts at the beginning and tries to match the whole regular expression. If we want the pattern to be applied at the end of the string, we end our pattern with a `$`. In the third example in Session 8.13, the `$` in the pattern means that *ADE* must come at the end of the string to be a match.

Table 8.10 showed that we have not decoded *B*, *C*, *J*, *K*, *M*, *P*, *Q*, *R*, *U*, *V*, *W*, *X*, *Y*, or *Z*. When we check for matches in the dictionary, we can use the regular expression `[BCJKMPQRUVWXYZ]` to limit our choices for a particular letter position to match only those characters. The call to `match` for the example we have been using would be

```
re.match('[BCJKMPQRUVWXYZ]ADE', 'FADE')
```

Because *F* is not one of the characters inside the square brackets, this call would fail and return `None`. However,

```
re.match('[BCJKMPQRUVWXYZ]ADE', 'MADE')
```

would be successful and return a Match object because *M* is one of the characters in the brackets.

Let's write a Python function that uses the regular expression module to create a list of all the words in the dictionary file that match a given regular expression. The basic form of our function is quite simple: We use a loop to read through all the words in the dictionary, trying to match each word against the regular expression. If a word matches, we add it to the list; otherwise we ignore it. When we have tested all the words, we return the list that we constructed. The code for this matching function, called `checkWord`, is shown in **Listing 8.11**.

```
1 def checkWord(regex):
2     resList = []
3     with open('wordlist.txt', 'r') as wordFile:
4         for line in wordFile:
5             if re.match(regex, line[:-1]):
6                 resList.append(line[:-1])
7     return resList
```

Listing 8.11 Matching words from the dictionary against patterns

Description

Session 8.14 uses the `checkWord` function to see if we can identify more letters. Our partially decoded message includes the string *aOiNING*. Using the pattern `'.o.ning'`, the `checkWord` function matches the word *morning* from the dictionary. The second example uses a pattern that explicitly restricts the possible matches to our list of unmapped characters. The rest of Session 8.14 finds matches for some additional words.

```
In [1]: checkWord('.o.ning')
```

```
Out[1]: ['morning']
```

```
In [2]: # match only unmapped characters
        checkWord('[bcjkmpqruvwxyz]o[bcjkmpqruvwxyz]ning')
```

```
Out[2]: ['morning']
```

```
In [3]: # match a, any two characters, i, any one character, al
        # stop finding characters when pattern ends
        checkWord('a..i.al$')
```

```
Out[3]: ['admiral', 'arrival']
```

```
In [4]: # match only unmapped characters, stop when pattern ends
        checkWord('a[bcjkmpqruvwxyz][bcjkmpqruvwxyz]i[bcjkmpqruvwxyz]al$')
```

```
Out[4]: ['arrival']
```

Session 8.14 Using `checkWord` to find matches

Description

Constructing the pattern string for the `checkWord` parameter is quite tedious, but we can reduce the amount of work necessary by making the `checkWord` function more intelligent. To do so, let's pass two parameters to the `checkWord` function: a string of unused letters and the word from the partially decoded ciphertext. As a consequence, letters to keep in place will (initially) be capitalized, and letters that we can match from the unused group will be in lowercase.

We can use the `re` module's replace function (called `re.sub`) to substitute for all of the lowercase characters using the group of letters we want to match. Line 5 of **Listing 8.12** shows the use of the `re.sub` function. Notice that we can use a simplified pattern to represent all the lowercase characters. The pattern `[a-z]` is the same as typing `[abcdefghijklmnopqrstuvwxyz]`.

```
1  def checkWord(unused, pattern):
2      resList = []
3      with open('wordlist.txt', 'r') as wordFile:
4          rePat = '['+unused+']'
5          regex = re.sub('[a-z]', rePat, pattern) + '$'
6          regex = regex.lower()
7          print('matching', regex)
8          for line in wordFile:
9              if re.match(regex, line[:-1]):
10                 resList.append(line[:-1])
11     return resList
```

Listing 8.12 `checkWord` construction of a regular expression

Description

The syntax for the `re.sub` function is as follows:

```
re.sub(pattern, replacement string, target string)
```

The `re.sub` function finds each instance of `pattern` in `target string` and replaces it with `replacement string`. We have added a `print` statement to the `checkWord` function so that you can see the final version of the regular expression constructed by `checkWord`.

Session 8.15 illustrates the new `checkWord` function. But as you can see, we could make one more improvement: `checkWord` could tell us the

ciphertext to plaintext mapping for the letters that complete the match. In fact, regular expressions allow us to do this very nicely by using Match objects and capture groups, which help us learn which characters in the target word matched letters in different parts of our pattern. To create a **capture group**, you surround some part of the regular expression with left and right parentheses. For the example, the regular expression `'F(..)L(..)$'` will match any word that starts with an *F* followed by any two letters, followed by an *L* followed by two more letters. Thus, words like *FUELED*, *FOILED,* and *FOOLER* would all match.

```
In [1]: checkWord('bcjkmpqruvwxyz', 'WONDEiFmLLq')

        matching wonde[bcjkmpqruvwxyz]f[bcjkmpqruvwxyz]ll[bcjkmpqruvwxyz]$

Out[1]: ['wonderfully']

In [2]: checkWord('bcjkmpqruvwxyz', 'AiiInAL')

        matching a[bcjkmpqruvwxyz][bcjkmpqruvwxyz]i[bcjkmpqruvwxyz]al$

Out[2]: ['arrival']

In [3]: checkWord('bcjkmpqruvwxyz', 'mNmSmAL')

        matching [bcjkmpqruvwxyz]n[bcjkmpqruvwxyz]s[bcjkmpqruvwxyz]al$

Out[3]: ['unusual']
```

Session 8.15 Using the new `checkWord` to find matches

Description

Session 8.16 illustrates the use of capture groups in regular expressions. Let's suppose that we matched the word *FOILED*. The first capture group corresponds to the first set of parentheses, so the letters in the capture group would be *OI*. The second capture group would contain the letters *ED*. The Match object allows us to get capture groups by number, using the `group` method, or to get all the capture groups as a list, using the `groups` method.

```
In [1]: import re
        cg = re.match('F(..)L(..)', 'FOILED')
        cg

Out[1]: <re.Match object; span=(0, 6), match='FOILED'>

In [2]: cg.group(1)

Out[2]: 'OI'

In [3]: cg.group(2)

Out[3]: 'ED'

In [4]: cg = re.match('F(..)L(..)', 'FOOLER')
        cg.groups()

Out[4]: ('OO', 'ER')
```

Session 8.16 Demonstrating capture groups

Description

Now that you have some idea of how capture groups work with regular expressions, we will use capture groups to create a list of character maps that show us which characters we can substitute for the ciphertext characters. We begin with the `checkWord` function and add this new feature. There are three key differences between the new version and the old one:

1. We create a list of the lowercase ciphertext letters in our original pattern.
2. We add capture groups to the regular expression.
3. When we get a matching word, we save the matching letters from the capture groups along with their ciphertext equivalents.

Listing 8.13 shows the entire new version of `checkWord`, now called `findLetters`. Let's look at the lines that differ significantly from the old `checkWord` function. First, we want to create a list of all the lowercase ciphertext letters in the partially decrypted word (`pattern`). You could write a loop to do this and build this list yourself by checking one letter at a time. However, the `re` module has a `findall` function that does exactly this job for us. It returns a list of all the substrings of a string that match a particular regular expression. **Session 8.17** demonstrates the `findall` function. Line 4 in Listing 8.13 uses the `findall` function to create the list of ciphertext letters.

```
1  def findLetters(unused, pattern):
2      resList = []
3      with open('wordlist.txt', 'r') as wordFile:
4          ctLetters = re.findall('[a-z]', pattern)
5          print(ctLetters)
6          rePat = '(['+unused+'])'
7          regex = re.sub('[a-z]', rePat, pattern) + '$'
8          regex = regex.lower()
9          for line in wordFile:
10             myMatch = re.match(regex, line[:-1])
11             if myMatch:
12                 matchingLetters = myMatch.groups()
13                 matchList = []
14                 for l in matchingLetters:
15                     matchList.append(l.upper())
16                 resList.append(line[:-1])
17                 resList.append(list(zip(ctLetters, matchList)))
18     return resList
```

Listing 8.13 Showing the matching letters

Description

```
In [1]: import re
        re.findall('[123]', '1,234')  # find single characters

Out[1]: ['1', '2', '3']

In [2]: re.findall('[1234]+', '1,234')  # find multiple characters

Out[2]: ['1', '234']

In [3]: re.findall('[A-Z]', 'Hello World')  # find capital letters

Out[3]: ['H', 'W']
```

Session 8.17 Finding all occurrences of a regular expression

Description

We next make a minor change to line 6 to add parentheses around our regular expression that matches the undecoded ciphertext letters. This change ensures that each instance of a lowercase letter in the pattern will end up in its own capture group.

Finally, lines 11–17 make use of the Match object and the `groups` method. When a match is made against a word from the word list file, the call to `myMatch.groups()` gives us the letters that were used to make the matching

word. At this point, we have two lists that contain important information: The `ctLetters` list contains the original ciphertext letters and the `matchingLetters` list contains the plaintext letters that complete the word.

We must now match the letters from `ctLetters` with the corresponding letters from `matchingLetters`. It is easy to know which letter from `ctLetters` corresponds to a letter in `matchingLetters` because they are parallel lists. The first letter in `ctLetters` corresponds to the first letter in `matchingLetters`, and so on. Although we could write our own function that accepts two lists and puts the letters together into a new list by moving through the lists one item at a time, Python has a handy `zip` function that takes care of this step for us. The `zip` function takes two lists and "zips" them together, matching the first item from `list1` with the first item from `list2`, and so on. **Table 8.11** explains the `zip` built-in function, and **Session 8.18** illustrates how `zip` works.

TABLE 8.11 The `zip` Built-in Function

Function Use	Explanation
`zip(list1, list2, …)`	Iterator that returns tuples composed of parallel elements of each list.
`tList = list(zip(listA, listB))`	

```
In [1]: list(zip([1,2,3], [4,5,6]))

Out[1]: [(1, 4), (2, 5), (3, 6)]

In [2]: list(zip(['a','b','c'], ['Z','Y','X']))

Out[2]: [('a', 'Z'), ('b', 'Y'), ('c', 'X')]

In [3]: list(zip(['a','b','c'], ['Z','Y','X'], [1,2,3]))

Out[3]: [('a', 'Z', 1), ('b', 'Y', 2), ('c', 'X', 3)]
```

Session 8.18 Using the `zip` function to create a list of tuples

Description

Session 8.19 shows how the `findLetters` function can help us finish decoding the ciphertext. On the first line, `findLetters` searches for the mappings that use the partially decrypted word `'AiiInAL'`. We discover that `'arrival'` is a good match for `'AiiInAL'`. The third line of the session shows us that the ciphertext letter *i* maps to the plaintext letter *R* and that *n* maps to the plaintext letter *V*.

```
In [1]: findLetters('bcjkmpqruvwxyz', 'AiiInAL') # i -> R, n -> V
```

```
['i', 'i', 'n']
```

Out[1]:
```
['arrival', [('i', 'R'), ('i', 'R'), ('n', 'V')]]
```

In [2]:
```
findLetters('bcjkmpqruvwxyz', 'ALiEADq') # q -> Y
```

```
['i', 'q']
```

Out[2]:
```
['already', [('i', 'R'), ('q', 'Y')]]
```

In [3]:
```
findLetters('bcjkmpqruvwxyz', 'giEErE') # g -> B, r -> Z
```

```
['g', 'i', 'r']
```

Out[3]:
```
['breeze', [('g', 'B'), ('i', 'R'), ('r', 'Z')]]
```

In [4]:
```
findLetters('bcjkmpqruvwxyz', 'mNmSmAL') # m -> U
```

```
['m', 'm', 'm']
```

- -

```
Out[4]: ['unusual', [('m', 'U'), ('m', 'U'), ('m', 'U')]]
```

```
In [5]: cipherText = cipherText.replace('i', 'R')
        cipherText = cipherText.replace('n', 'V')
        cipherText = cipherText.replace('q', 'Y')
        cipherText = cipherText.replace('g', 'B')
        cipherText = cipherText.replace('r', 'Z')
        cipherText = cipherText.replace('m', 'U')
```

Session 8.19 Demonstrating the `findLetters` function

Description

Additional calls to `findLetters` give us the mappings for *m*, *g*, *i*, *r*, *q*, and *n*. Now a message is really starting to show. With just a few more substitutions we will have the entire message decoded.

HE REaAINED STANDING AT THE EDGE OF THE dIT THAT THE THING HAD aADE FOR ITSELF STARING AT ITS STRANGE AddEARANwE ASTONISHED wHIEFLY AT ITS UNUSUAL SHAdE AND wOLOUR AND DIaLY dERwEIVING EVEN THEN SOaE EVIDENwE OF DESIGN IN ITS ARRIVAL THE EARLY aORNING WAS WONDERFULLY STILL AND THE SUN xUST wLEARING THE dINE TREES TOWARDS WEYBRIDGE WAS ALREADY WARaHE DID NOT REaEaBER HEARING ANY BIRDS THAT aORNING THERE WAS wERTAINLY NO BREEZE STIRRING AND THE ONLY SOUNDS WERE THE FAINT aOVEaENTS FROa WITHIN THE wINDERY wYLINDER HE WAS ALL ALONE ON THE wOaaON

From here we need only a few more substitutions to complete the message, as shown in **Session 8.20**. In the first call to `findLetters`, we find matches in *common* and *pompon*. Since *common* is more likely to be a word in the ciphertext, we choose to substitute *C* for *w*. If we later find this is not correct, we could substitute *P* for *w*. In both words, we find that *a* maps to *M*. This makes for a high confidence level for that substitution.

HE REMAINED STANDING AT THE EDGE OF THE PIT THAT THE THING HAD MADE FOR ITSELF STARING AT ITS STRANGE APPEARANCE ASTONISHED CHIEFLY AT ITS UNUSUAL SHAPE AND COLOUR AND DIMLY PERCEIVING EVEN THEN SOME EVIDENCE OF DESIGN IN ITS ARRIVAL THE EARLY MORNING WAS WONDERFULLY STILL AND THE SUN JUST CLEARING THE PINE TREES TOWARDS WEYBRIDGE WAS ALREADY WARM HE DID NOT REMEMBER HEARING ANY BIRDS THAT MORNING THERE WAS CERTAINLY NO BREEZE STIRRING AND THE ONLY SOUNDS WERE THE FAINT MOVEMENTS FROM WITHIN THE CINDERY CYLINDER HE WAS ALL ALONE ON THE COMMON

```
In [1]: findLetters('cjkmpqwx', 'wOaaON') # w -> C?, w -> P?, a -> M

        ['w', 'a', 'a']

Out[1]: ['common',
         [('w', 'C'), ('a', 'M'), ('a', 'M')],
         'pompon',
         [('w', 'P'), ('a', 'M'), ('a', 'P')]]
```

In [2]:
```
cipherText = cipherText.replace('w', 'C')
cipherText = cipherText.replace('a', 'M')
```

In [3]:
```
findLetters('jkpqx', 'dIT')  # no definitive substitution
```

```
['d']
```

Out[3]:
```
['kit', [('d', 'K')], 'pit', [('d', 'P')]]
```

In [4]:
```
findLetters('jkpqx', 'AddEARANCE') # d -> P
```

```
['d', 'd']
```

Out[4]:
```
['appearance', [('d', 'P'), ('d', 'P')]]
```

In [5]:
```
cipherText = cipherText.replace('d', 'P')
```

In [6]:
```
findLetters('jkqx', 'xUST') # x -> J
```

```
        ['x']

Out[6]: ['just', [('x', 'J')]]

In [7]: cipherText = cipherText.replace('x', 'J')
```

Session 8.20 The last substitutions

Description

Table 8.12 shows the final key that maps the ciphertext letters to the plaintext letters. Notice that the `keyGen` function we wrote earlier was used to create the key from the password `hgwells`.

TABLE 8.12 Final Mapping of the Letters

Plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Ciphertext	H	g	w	e	l	s	t	u	v	x	y	z	a	b	c	d	f	i	j	k	m	n	o	p	q	r

Description

8.4.6 Regular Expression Summary

We conclude this section with a summary of the regular expression syntax (**Table 8.13**) and the functions we used from the `re` module (**Table 8.14**).

TABLE 8.13 Simple Regular Expression Syntax

Expression	Meaning
.	Match any character.
[abc]	Match a or b or c.
[^abc]	Match any character other than a or b or c at the beginning of the string.
[abc]+	Match one or more occurrences of any of the characters abc—for example, b or aba or ccba.
[abc]*	Match zero or more occurrences of any of the characters abc—for example, b or aab or ccbba.
(regex)	Create a capture group.
$	Match at the end of the string.

TABLE 8.14 Regular Expression Module Functions and Methods

Function or Method	Use	Explanation
match(pattern, string)	re.match('[abc]XY.', 'myString')	Matches any string in myString that starts with a, b, or c followed by XY followed by any character. Returns a Match object on success or None.
sub(pattern, replacement, string)	re.sub('[tv]', 'X', 'vxyzbgtt')	Returns 'XxyzbgXX'. Like replace, except uses regular expression matching.
findall(pattern,string)	re.findall('[bc]+','abcdefedcba')	Returns ['bc','cb']. Returns a list of all substrings matching the regular expression.
groups()	matchObj.groups()	Returns a tuple of all capture groups matched. matchObj is created by a call to match.
group(i)	matchObj.group(2)	Returns a single capture group at index i. matchObj is created by a call to match.

Try It Out

8.17 Write a regular expression pattern to match all words ending in *s*.

8.18 Write a regular expression pattern to match all words ending in *ing*.

8.19 Write a regular expression pattern to match all words with *ss* anywhere in the word.

8.20 Write a regular expression pattern to match all words beginning and ending with the letter *a*.

8.21 Write a regular expression pattern to match all the words that start with *st*.

8.22 Write a regular expression to match all the four-letter words where the middle two letters are vowels.

8.23 Write a function that can extract the host name from a URL. The host name is the part of the URL that comes after http:// but before the next /.

8.24 Write a function that can take a filename and return the name and the suffix as two separate strings. Note: To match the . character, use \. in the regular expression.

8.5 Summary

In this chapter, we explored cryptanalysis by implementing techniques for reading encrypted text. We utilized many previously introduced features of Python, including lists and dictionaries, but we extended these ideas by using lists and dictionaries as containers for other lists and dictionaries. We used a brute-force solution to systematically solve a challenging problem. Part of that solution used a pattern called “minmax” for systematically finding a “better” answer given a “best” answer so far. Finally, we introduced the notion of a regular expression and showed how this powerful pattern-matching tool can help us locate missing letters in partially decrypted text.

Key Terms

capture group
column-major order
cryptanalysis
frequency analysis
in-place sort
pattern matching
regular expressions
row-major order

Programming Exercises

8.1 Decode the following paragraph:

```
jyn fg jggtwj djtfcn stf sjyn edcyjnc ia zy stes
fjqtye z wzdn owcff gstf gsq sjyn edcyjnc gsjg
mtgs tg gszi xjqcfg owzm gstyc cycxtcf gz gtyq
otgf ty gsq xcdhq jyn gsc wzdn ntn edty jyn aczawc
ntn lcjfg iazy gsc wjxof jyn fwzgsf jyn hjda jyn
jyhszktcf jyn zdjyeigjyf jyn odcjvljfg hcdcjwf jyn
lditg ojgf jyn gsc wzdn fajvc fjqtye ltdfg fsjwg
gszi gjvc zig gsc szwq aty gscy fsjwg gszi hziyg
gz gsdcc yz xzdc yz wcff gsdcc fsjwg oc gsc yixocd
gszi fsjwg hziyg jyn gsc yixocd zl gsc hziygtye
fsjwg oc gsdcc lzid fsjwg gszi yzg hziyg yctgscd
hziyg gszi gmz cphcagtye gsjg gszi gscy adzhccn gz
gsdcc ltkc tf dtesg zig zyhc gsc yixocd gsdcc
octye gsc gstdn yixocd oc dcjhscn gscy wzoocfg
gszi gsq szwq sjyn edcyjnc zl jygtzhs gzmjdnf gszi
lzc msz octye yjiesgq ty xq ftesg fsjww fyill tg
```

8.2 Working with a partner, pick two paragraphs of text from a news article. Encrypt the paragraphs separately and trade the resulting ciphertext. Use the techniques in this chapter to decode the message from your partner.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 9

Fractals: The Geometry of Nature

Recursion, Grammars, and Production Rules

9.1 OBJECTIVES

- To practice writing recursive functions
- To introduce the `match` statement for selection
- To introduce grammars and production rules

9.2 Introduction

Take a look at a tree. Notice that you could choose any of the major branches of that tree and cut it off, and the branch would bear a strong resemblance to the entire tree. Furthermore, you could cut one of the branches off that branch, and it would resemble the entire tree. In fact, you could continue to prune the branch down to the size of small twigs, and you would see that even the twigs have the same basic shape and structure as the entire tree. This self-similarity at smaller and smaller scales is the essence of a **fractal**.

RECAP:
A fractal is a geometric figure in which the parts are similar at different scales.

Now take a look at the tree in **Figure 9.1**. Not only is it reminiscent of a real tree, but it is even more obviously self-similar. In fact, this tree was drawn using a turtle with the following simple instructions:

1. Draw a trunk that is *n* units long.
2. Turn to the right 30 degrees, and draw another tree with a trunk that is *n* − 15 units long.
3. Turn to the left 60 degrees, and draw another tree with a trunk that is *n* − 15 units long.



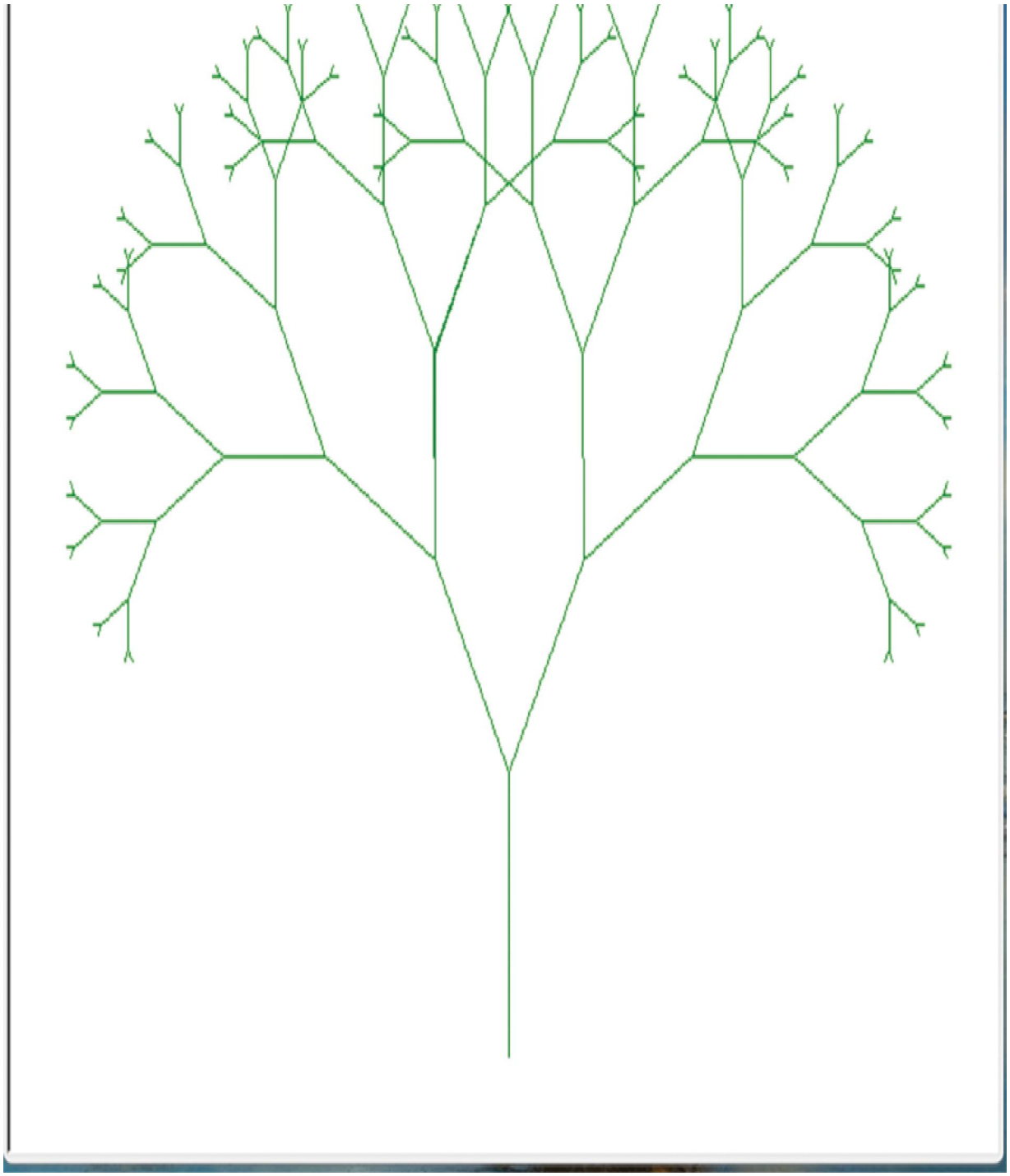


Figure 9.1 A fractal tree drawn with a turtle.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

At first glance, you may think that all this would do is draw a Y shape. But remember that each time we draw a tree, we follow the same set of steps. Those steps include drawing a trunk and then a smaller tree to the right and a smaller tree to the left. What we have just described is a **recursive process** for drawing a tree.

9.3 Recursive Programs

Before we look at the program used to draw the tree, we must first learn a little more about **recursion**. In computer science and mathematics, a recursive function is a function that calls itself.

RECAP:
A recursive function is a function that calls itself.

A very simple, but erroneous, illustration of a recursive function is shown in **Listing 9.1**. The `hello` function prints the message `"Hello World"` and then calls itself again. If you ran this function in Python, the program would continue to call itself and print `"Hello World"` until Python crashed. Clearly, there is more to a successful recursive function than simply calling itself.

```
1 def hello():
2     print("Hello World")
3     hello()
```

Listing 9.1 An erroneous recursive function

Description

A more useful way of thinking about recursive functions is as follows:

1. Are we done yet? Have we found a problem that is small enough to solve trivially? If so, solve it without any further work and return the answer, if one is required.

2. If not, simplify the problem, and solve the simpler problem. Combine the result of solving the simplified problem with what you know to solve the original. Return the combined result, if required.

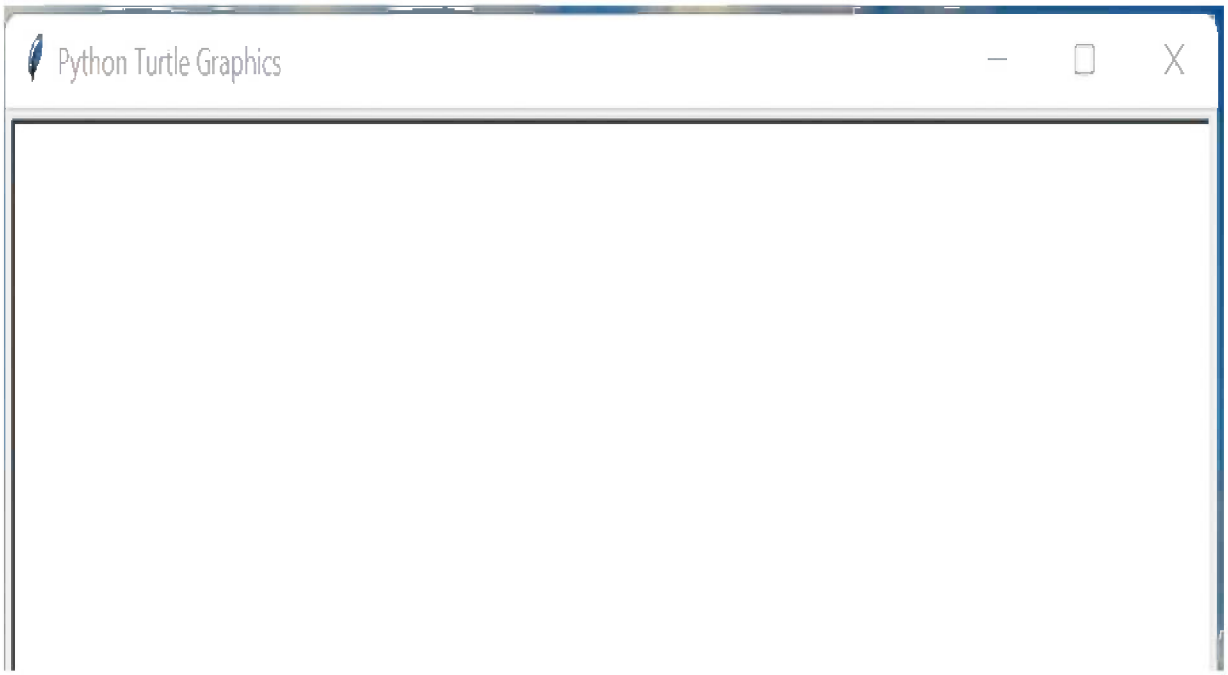
The first step is called the **base case**. Every recursive program must have a base case so that it knows when to stop. Checking for the base case is what's missing from the program in Listing 9.1.

The second step is often called the **recursive step**. The recursive step simplifies the problem in a way that moves it closer to the base case. Sometimes you will need to combine the result returned by the recursive call with some data you saved when you simplified the problem. In other cases, the problem is solved just by making the simplifying step.

HEADS UP:
Omitting the base case can lead to repeated execution until Python crashes. Handling the base case is key to eventually ending the code.

9.3.1 Recursive Squares

Let's look at a problem of drawing the nested boxes shown in **Figure 9.2**.



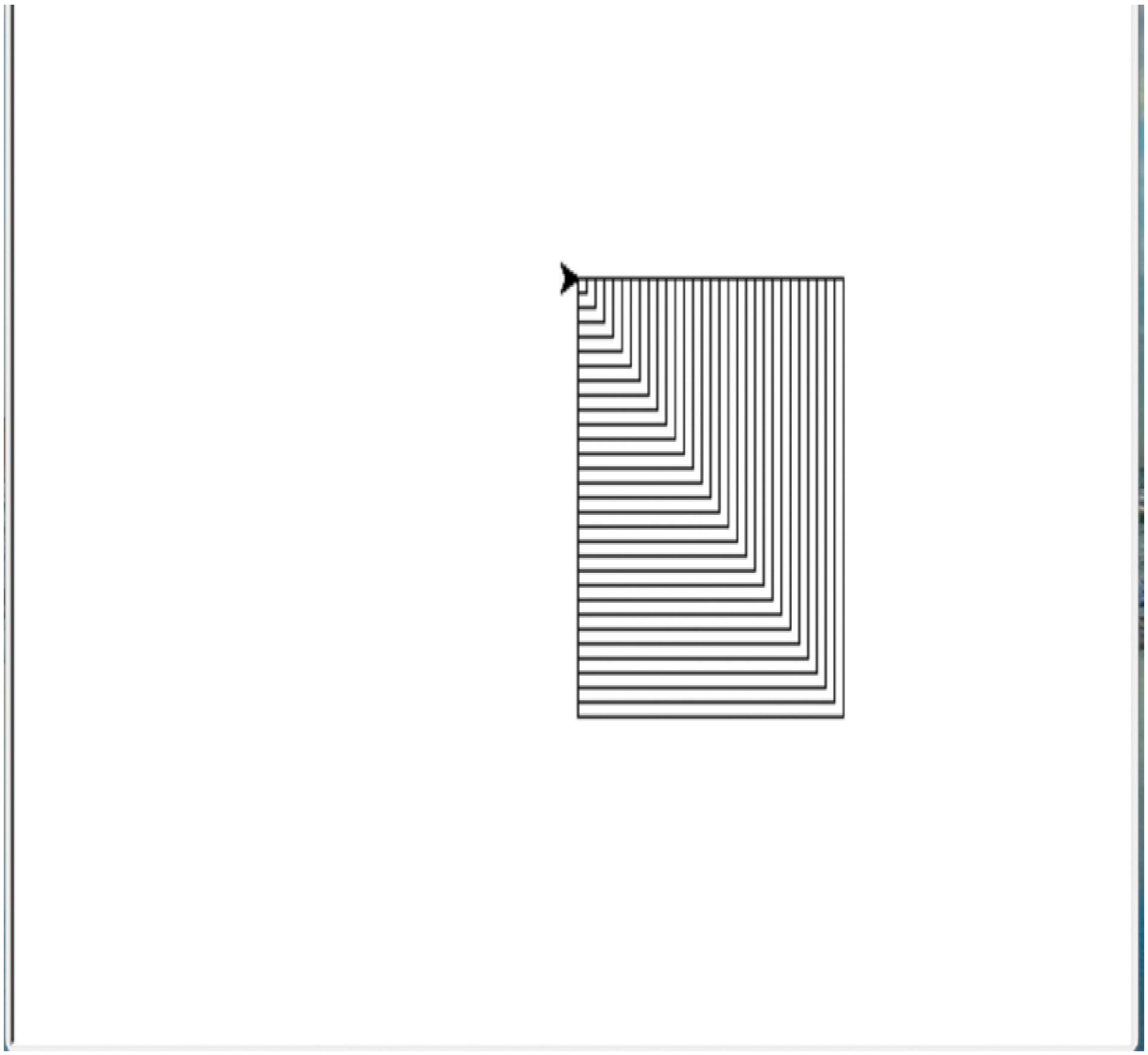


Figure 9.2 Nested boxes.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

In this case, we need to draw a series of squares. We know how to draw a single square, but how do we draw the entire picture? Let's start by identifying the base case. Because drawing a square that has a side length of 1 unit is the smallest possible square, we will identify the base case as the side length being less than 1. The simplification step is to recursively draw smaller and smaller squares as long as the side length is greater than or equal to 1.

Listing 9.2 shows the code for the nested boxes. The `drawSquare` function should look familiar to you: It simply draws a square of a specified size. The `nestedBox` function is a recursive process that follows the two important steps outlined previously. First, line 7 checks for the base case. The expression `side >= 1` will evaluate to `False` as soon as the `side` parameter is less than 1. Because there is no `else` clause, notice that as soon as the base case is reached (`side` is less than 1), the `nestedBox` simply returns without drawing anything.

```
1 def drawSquare(aTurtle, side):
2     for i in range(4):
3         aTurtle.forward(side)
4         aTurtle.right(90)
5
6 def nestedBox(aTurtle, side):
7     if side >= 1:               #check for base case: side < 1
8         drawSquare(aTurtle, side)
9         nestedBox(aTurtle, side - 5)
```

Listing 9.2 Recursively drawing boxes

Description

If the value of `side` is greater than or equal to 1, we draw a square that has a length of `side`. After the square is drawn, the next step is to call `nestedBox` recursively, reducing the side length by 5. By reducing the value of `side` by 5, we are moving toward the base case.

Session 9.1 shows the code that draws the squares in Figure 9.2. We first create a turtle, then call our recursive `nestedBox` function with an initial value of 200.

```
In [1]:
1 from drawSquares import *  #import drawing functions
2 import turtle
3 t = turtle.Turtle()        #create a turtle
4 nestedBox(t, 150)          #initial side 150
5 turtle.done()              #stop the turtle
```

Session 9.1 Calling the recursive `nestedBcx` function

Description

Try It Out

9.1 Create a turtle. Run the `nestedBox` function with an initial side of 200, and then modify `nestedBox` by reversing lines 8 and 9. Can you explain the different behavior?

9.2 Write a recursive function to draw concentric nested boxes, where each box is centered at the same point. You may want to write a helper function called `drawCenteredSquare` that accepts parameters for a turtle, the center *x* and *y* positions, and the length of the side.

9.3.2 Classic Recursive Functions

It is also often possible to use recursion to express mathematical functions in an elegant way. The following exercises allow you to explore simple recursion by writing some nongraphical functions. The key to handling lists recursively is to process one item from the parameter list and then call the function again with that item removed from the list, continuing this pattern until the parameter list is empty or contains only one item, depending on the problem. The approach for strings is similar.

For example, **Listing 9.3** shows a recursive function that counts the number of items in a list. The base case occurs when the list is empty. If the list is not empty, then we count the first item (add 1) and recursively call the function with the remaining items in the list.

```
1 def countList(aList):
2     if aList == []:   #base case, list is empty
3         return 0
4     else:
5         return 1 + countList(aList[1:])
```

Listing 9.3 Recursively counting items

Description

RECAP:
To process a list recursively, process one item in the list and then call the function again with that item removed. The base case is that the list is empty.

Session 9.2 calls the `countList` recursive function with three different parameters: a multi-element list, an empty list, and a list with just one

element. This ensures that the `countList` function correctly handles all these test conditions.

```
In [1]: from recursiveCountList import *
```

```
In [2]: print (countList([1, 20, 10, 30]))  #multiple elements
```

```
4
```

```
In [3]: print (countList([]))               #empty list
```

```
0
```

```
In [4]: print (countList([20]))             #list with 1 element
```

```
1
```

Session 9.2 Testing the recursive `ccuntList` function

Description

HEADS UP:

To test a recursive function that takes a list as a parameter, be sure to run the function with an empty list, a list with one element, and a list with multiple elements.

Try It Out

9.3 The factorial function is defined as follows:

$$\text{fact}(n) = \begin{cases} n \cdot \text{fact}(n-1) & n > 0 \\ 1 & \text{otherwise} \end{cases}$$

Write a recursive function `fact` that computes the factorial of any positive integer.

9.4 Write a recursive function to compute the sum of all the numbers in a list.

9.5 Write a recursive function to find the minimum number in a list.

9.6 Write a recursive function to find the maximum number in a list.

9.7 Write a recursive function to reverse the characters in a string.

9.8 Write a recursive function to decide whether a given string is a palindrome.

9.3.3 Drawing a Recursive Tree

Let's return to the problem of drawing the tree in Figure 9.1. Recall the instructions for drawing the tree:

1. Draw a trunk that is *n* units long.
2. Turn to the right 30 degrees, and draw another tree with a trunk that is *n* − 15 units long.
3. Turn to the left 60 degrees, and draw another tree with a trunk that is *n* − 15 units long.

Now let's apply the rules for recursion to implement a program that follows the tree-drawing instructions.

First, we must identify the base case. When drawing a tree, the base case is a tree where the trunk length is less than some predefined value. We could say that a trunk length of 1 is the smallest possible tree we could draw, but in fact we can pick any number to use as the smallest trunk length. If the trunk length is less than or equal to our predefined minimum, we simply stop drawing new trees and return.

There are actually two recursive steps in drawing the tree. Steps 2 and 3 both specify that we need to "draw another tree." Notice that not only are we drawing another tree (recursively), but we are also making progress toward the base case by drawing a trunk that is smaller than the current trunk length. **Listing 9.4** shows our recursive Python function that takes a turtle and a starting trunk length.

```
1  def tree(t, trunkLength):
2      if trunkLength < 5:          #check for base case
3          return
4      else:
5          t.forward(trunkLength)
6          t.right(30)
7          tree(t, trunkLength - 15)
8          t.left(60)
9          tree(t, trunkLength - 15)
10         t.right(30)
11         t.backward(trunkLength)
```

Listing 9.4 A recursive tree function

Description

The amazing thing about the `tree` function is its length—only 11 lines. Recursion often offers us the chance to capture a complex process elegantly. Let's look at the code in Listing 9.4 carefully and see how it compares with the rules we defined. First, notice that we check for the base case on line 2. We have written the program to be slightly longer than is strictly necessary to make it clear that we are checking for the base case. We could remove the explicit return and the `else` clause if we changed the conditional to `trunkLength >= 5`.

If we have not reached the base case, the turtle moves forward and turns to the right 30 degrees. On line 7, we make the first recursive call to draw a smaller tree on the right side of the trunk. When this side of the trunk is done, then the function returns and turns 60 degrees to the left, and the left side of the trunk is drawn (line 9).

Session 9.3 shows the preparation and function call. Notice that the turtle needs to be moved to the lower part of the window and turned 90 degrees to face up before making the initial call to the function because we want the tree to grow up. After the tree has been drawn, we hide the turtle. Run this program and carefully watch the order in which the branches are drawn.

In [1]:
```
1 from recursiveTree import *
2 import turtle
```

In [2]:
```
1 t = turtle.Turtle()
```

In [3]:
```
1 t.up()
2 t.goto(0, -225)
3 t.down()
4 t.left(90)    #face up
5 t.color("green", "green")
```

In [4]:
```
1 tree(t, 110)   #draw the tree
2 t.hideturtle()
3 turtle.done()
```

Session 9.3 Calling the recursive `tree` function

Description

Try It Out

9.9 Rewrite the `tree` function using the conditional `trunkLength >= 5` to check for the base case.

9.10 Swap the rules for the tree so that it draws the left side of the tree before the right side.

9.11 You can create a more interesting and realistic-looking tree by randomizing the angle at which the turtle turns. Rather than always using a 30-degree angle, select an angle between 15 and 45 degrees.

9.12 You can add even more realism by randomizing how much the branches shrink each time you make a recursive call. Instead of always subtracting 15, try subtracting a random amount between 5 and 25.

9.13 You can add color to the tree by making the large branches brown and the small branches green. Choose a threshold value for the length of the trunk and set the color accordingly.

9.3.4 The Sierpinski Triangle

Let's look at another simple fractal called the **Sierpinski triangle**. Imagine that you have three triangles that are all the same size. If you take the first two triangles, put them side by side, and then balance the third triangle on the peaks of the first two triangles, you will get a new, larger triangle composed of those three triangles. Further suppose that you constructed each of the three original triangles from three smaller triangles and each of the three smaller triangles with three smaller triangles still. **Figure 9.3** gives you an idea about what is involved. The idea of self-similarity at increasingly smaller scales is key to the concept of a fractal, and it is key to understanding recursion.

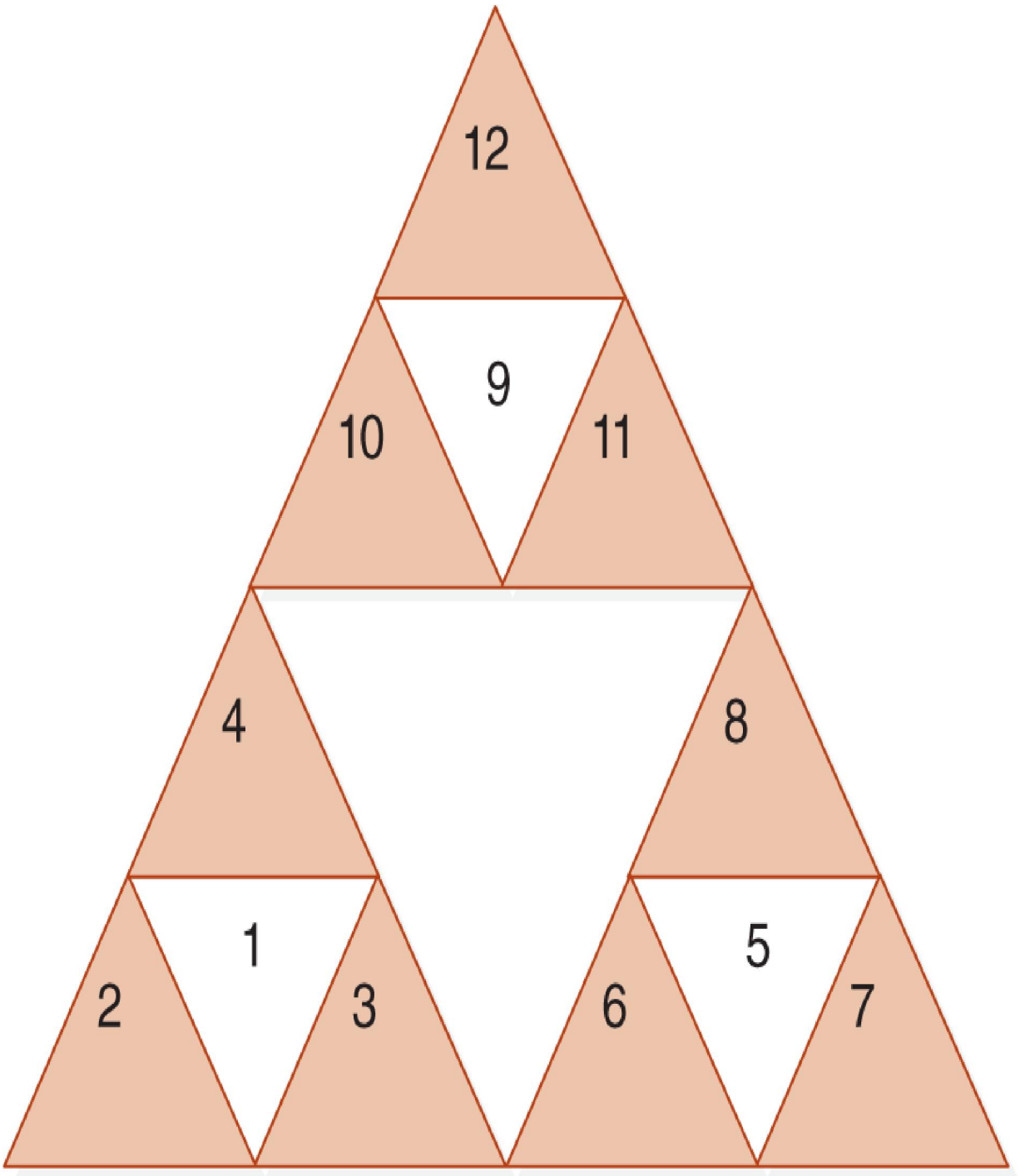


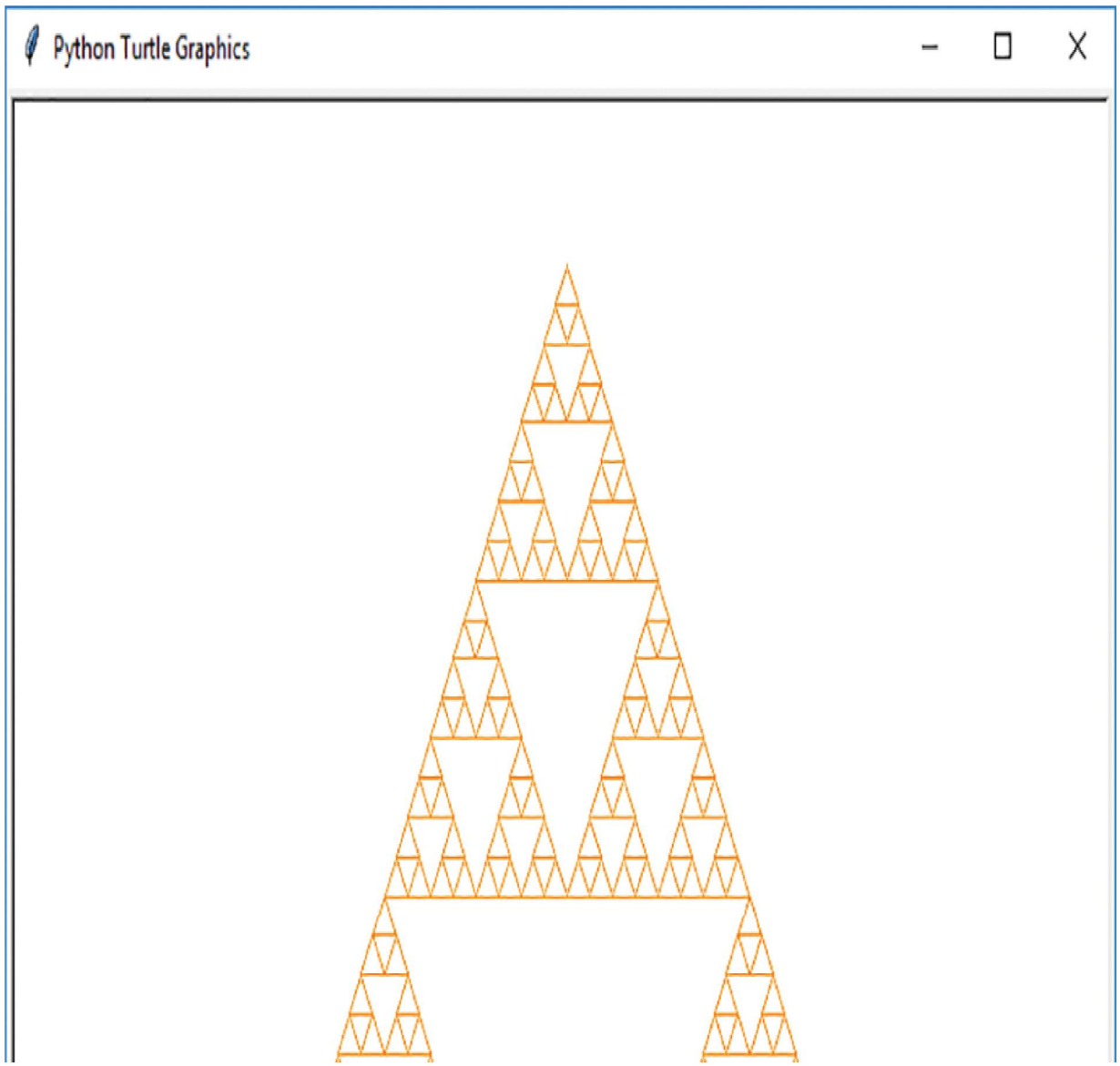
Figure 9.3 The Sierpinski triangle.

Description

Writing a program to draw the Sierpinski triangle is more complex than writing a program to draw a fractal tree, but it is still surprisingly simple. We

will use the *depth*, which represents the number of times we subdivide the original triangle, as the movement toward the base case. When we reach a sufficient depth, we will then draw a triangle of the appropriate size for that depth. In fact, we start the depth with a positive number, and each time we recursively divide a triangle, we subtract 1 from the depth. When we reach a depth of zero, we finally draw a very small triangle.

The Sierpinski triangle shown in Figure 9.3 has a depth of 2. Notice that there are nine shaded triangles, which represent the triangles drawn by the base case. The triangles numbered 1, 5, and 9 were not drawn but are artifacts of the three surrounding triangles. **Figure 9.4** shows a Sierpinski triangle of depth 5.



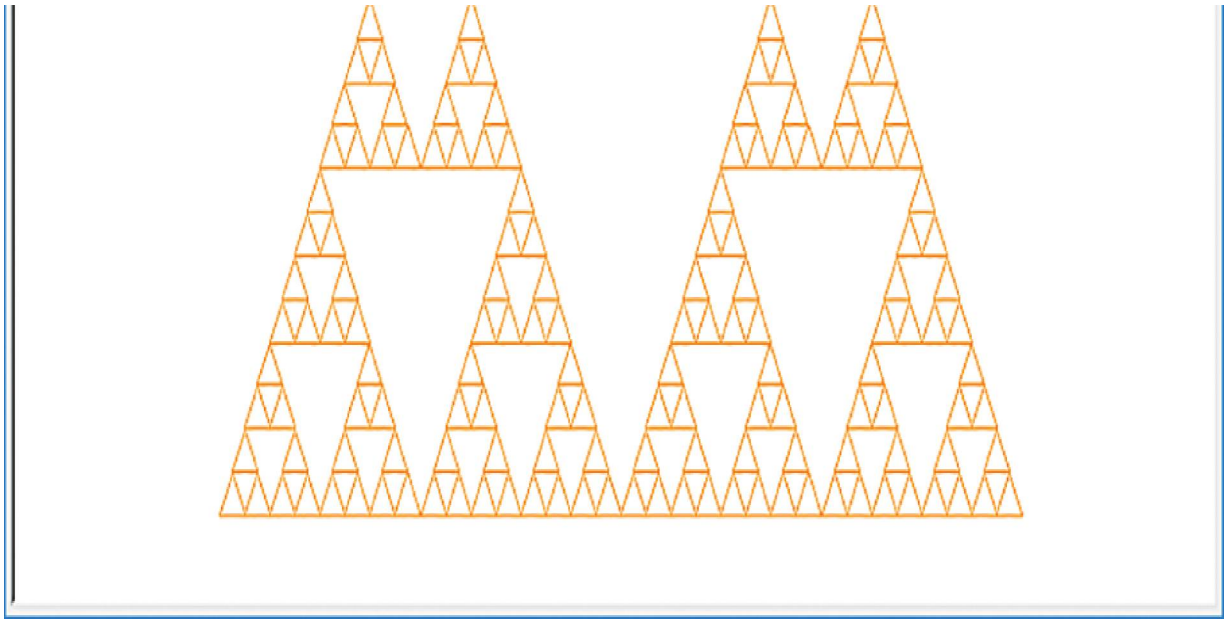


Figure 9.4 A Sierpinski triangle of depth 5.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

The `sierpinski` function will accept three points as parameters; these points define the three corners of the triangle. To subdivide a large triangle into three smaller triangles, we use the following rule:

> For each corner of the large triangle, create a small triangle using the given corner and the points that are halfway between the given corner and the other two corners.

For example, consider triangle ABC in **Figure 9.5**. The three new triangles we can construct are triangles A-1-3, B-2-1, and C-3-2.

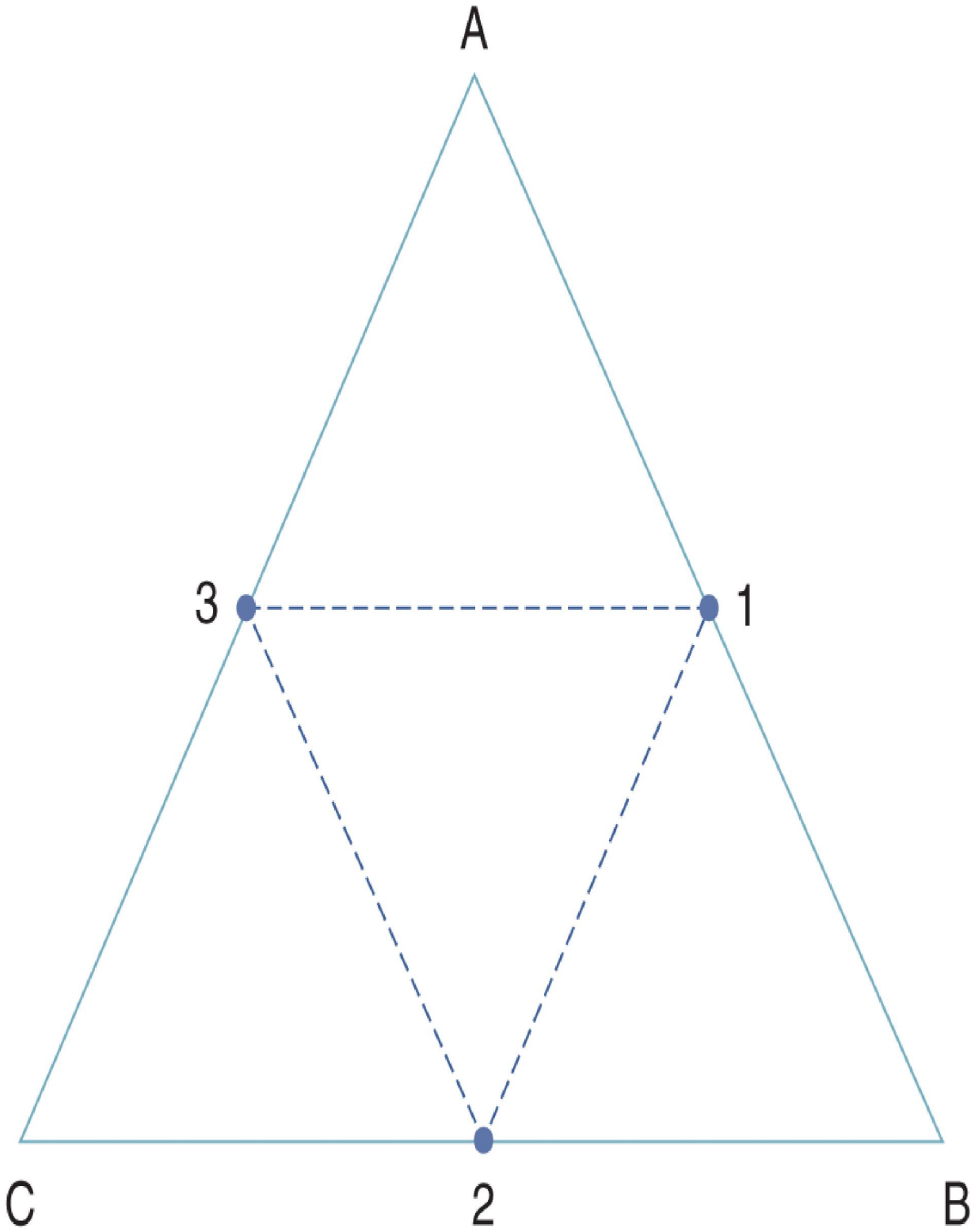


Figure 9.5 Splitting a triangle equally.

Description

To calculate points 1, 2, and 3, we find the midpoints of the line segments AB, BC, and CA, respectively. Recall that we can calculate the midpoint of a line segment (m_x, m_y) by using the equations

$$m_x = \frac{x_1 + x_2}{2} \qquad \text{and} \qquad m_y = \frac{y_1 + y_2}{2}$$

Drawing the smaller triangles is the recursive step in our function. It will involve three recursive calls to the `sierpinski` function using the vertices calculated by the method just described.

We simplify our recursive function by writing two helper functions: a function to draw a triangle and a function to calculate the coordinates of a point halfway between two other points. The `sierpinski` function and the two helper functions are shown in **Listing 9.5**.

```
1  def drawTriangle(t, p1, p2, p3):
2      t.up()
3      t.goto(p1)
4      t.down()
5      t.goto(p2)
6      t.goto(p3)
7      t.goto(p1)
8
9  def midPoint(p1, p2):
10     return ((p1[0] + p2[0]) / 2.0, (p1[1] + p2[1])  / 2.0)
11
12 def sierpinski(t, p1, p2, p3, depth):
13     if depth > 0:
14         sierpinski(t, p1, midPoint(p1, p2), midPoint(p1, p3), depth - 1)
15         sierpinski(t, p2, midPoint(p2, p3), midPoint(p2, p1), depth - 1)
16         sierpinski(t, p3, midPoint(p3, p1), midPoint(p3, p2), depth - 1)
17     else:   #base case
18         drawTriangle(t, p1, p2, p3)
```

Listing 9.5 Drawing a Sierpinski triangle

Description

The `drawTriangle` helper function takes a turtle and three points as parameters. `Turtle`'s `goto` method is used to connect the three points. The `midPoint` helper function encodes the equation for the midpoint of the segment and returns a tuple giving the *x* and *y* coordinates of the midpoint.

Note that the final version of the function has a parameter for the turtle as well as a parameter for the depth. The first line of `sierpinski` checks whether the depth is greater than zero. If that is the case, we make three recursive calls to `sierpinski`. Each recursive call represents one of the three subtriangles. When we make the recursive call, we subtract 1 from the depth to move toward the base case. When the function reaches a depth of zero, no more recursion is required. At this point, a triangle is drawn and the function simply returns. The larger the depth specified originally, the smaller this triangle will be.

Session 9.4 shows the code used to create the Sierpinski triangle in Figure 9.4. As noted earlier, the depth of that triangle is 5.

```
In [1]:
1 from sierpinski import *
2 import turtle
```

```
In [2]:
1 t = turtle.Turtle()
2 t.color('darkorange')
```

```
In [3]:
1 sierpinski(t, [-225, -250], [225, -250], [0, 225], 5)
2 t.hideturtle()
3 turtle.done()
```

Session 9.4 Creating the Sierpinski triangle

Description

Try It Out

9.14 Try calling `sierpinski` with different-shaped triangles.

9.15 Modify the `sierpinski` and `drawTriangle` functions to add color. *Hint:* Make each recursive call draw in a different color.

9.16 Modify the `sierpinski` function so that it unconditionally draws a triangle at each depth rather than just at a depth of zero.

9.17 Change the order of the recursive calls inside the `sierpinski` function.

9.3.5 Call Tree for a Sierpinski Triangle

By now, you have probably tried to run the `sierpinski` function on your own a few times, and it should be clear that the triangle is drawn in thirds. After one-third of the larger triangle has been completed, then the next third is drawn, and finally the last third is drawn. Within each smaller third, the same behavior holds true.

We can graph this behavior using a **call tree**, which enables us to order the recursive calls to the `sierpinski` function to explain why the triangle is drawn using this sequence. The call tree for a Sierpinski triangle of depth 2 is shown in **Figure 9.6**. Notice that the labels on the arrows on the call tree match the triangle numbers shown in Figure 9.3.

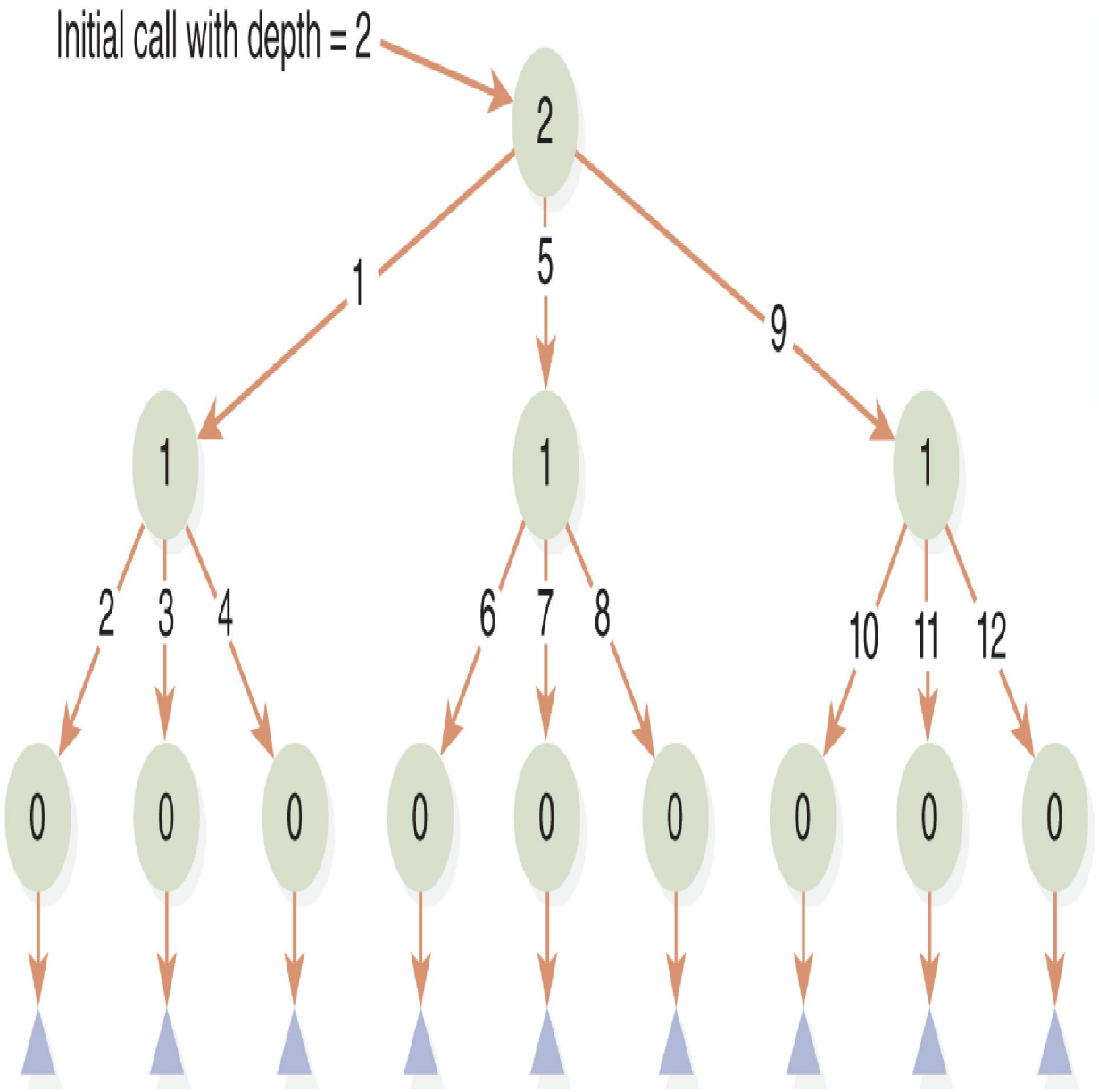


Figure 9.6 The call tree for a Sierpinski triangle of depth 2.

Description

Try It Out

9.18 Make a call tree for a Sierpinski triangle of depth 3.

9.19 You can generate a Sierpinski square by subdividing a square recursively. Take a square, and divide it into 3-by-3 squares. Apply

the division algorithm to every square except the center one. When you get to a sufficiently small square, stop subdividing and draw the square.

9.4 Snowflakes, Lindenmayer, and Grammars

It is said that no two snowflakes are exactly alike. Snowflakes are another item in nature that can be imitated by using fractal techniques. The photo in **Figure 9.7** shows a snowflake. The snowflake in **Figure 9.8** was produced using the **Koch snowflake curve**, a fractal algorithm developed by Helge von Koch in 1906. One difference between real snowflakes and the Koch snowflake curve is our code can create multiple Koch snowflakes that are exactly alike.

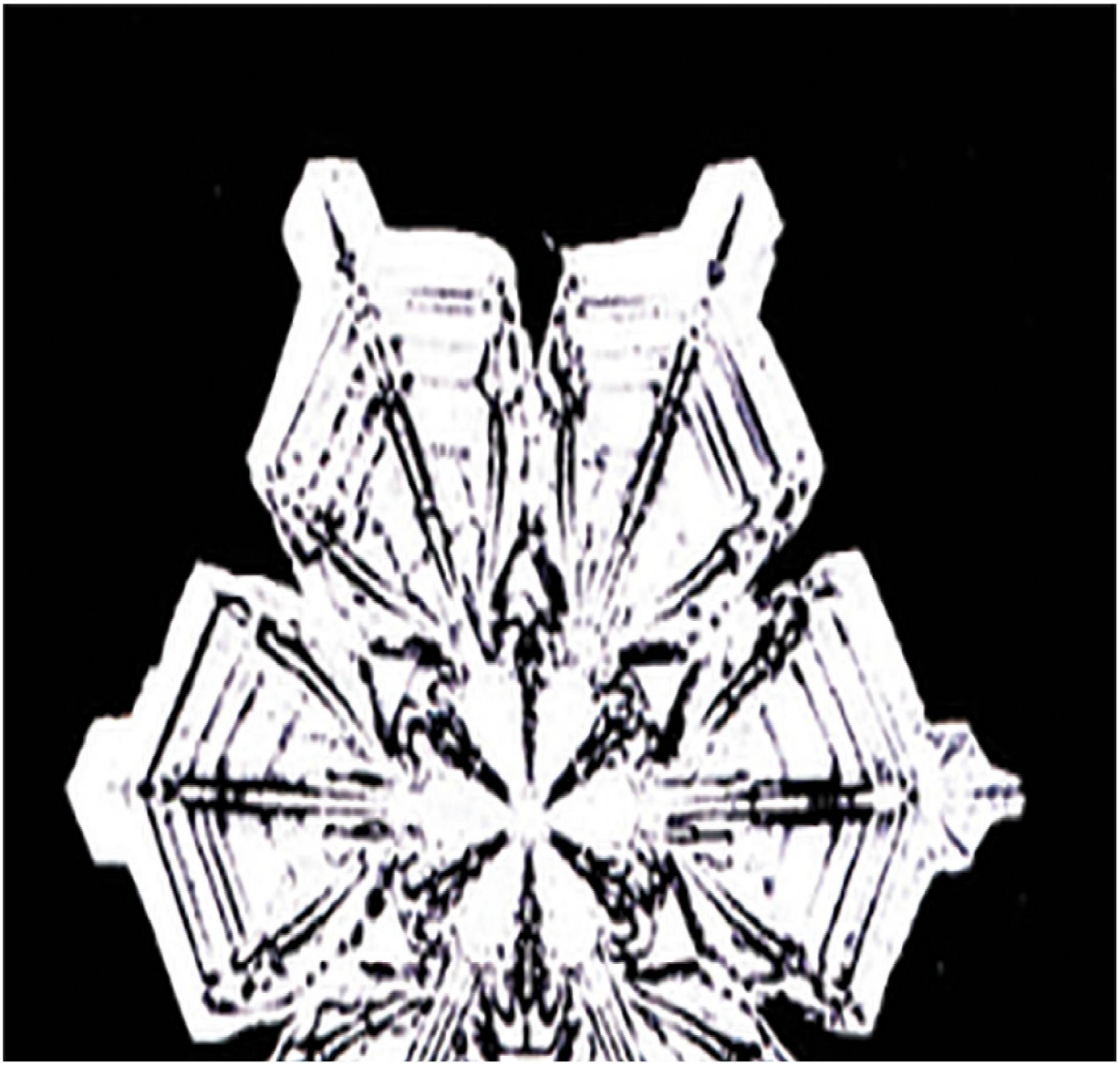
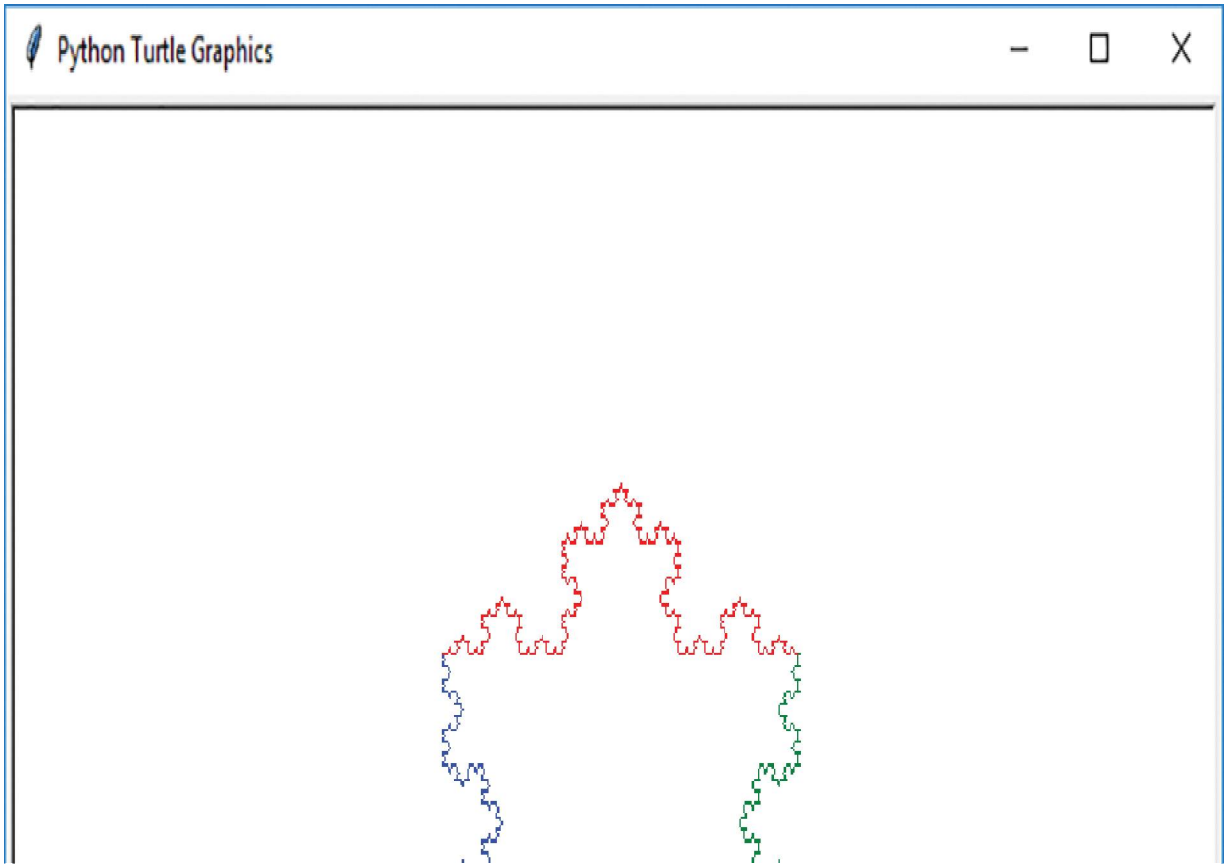




Figure 9.7 Photo of a snowflake.

Wilson Bentley. Courtesy of National Oceanic and Atmospheric Administration/Department of Commerce.

Description



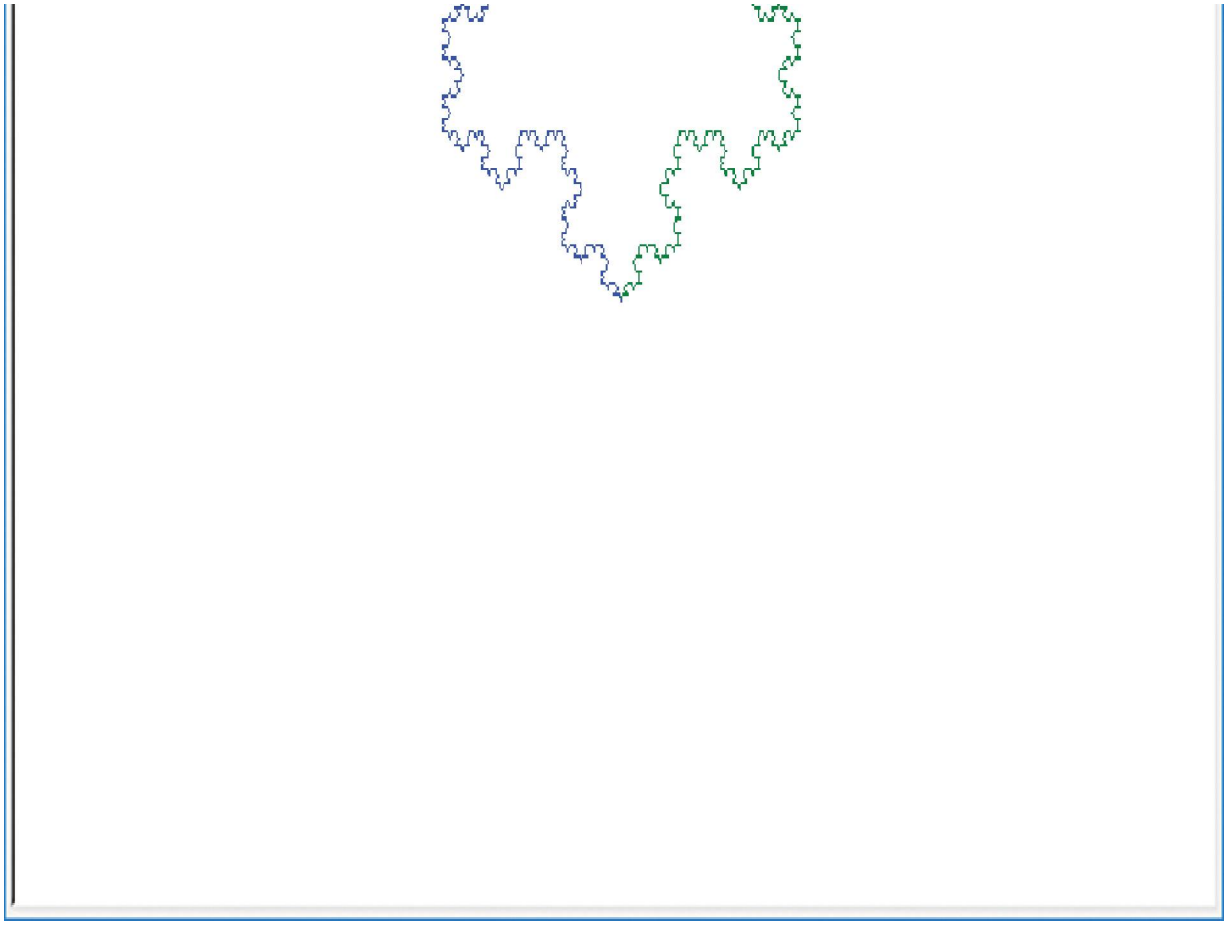


Figure 9.8 A snowflake produced using a fractal algorithm developed by Helge von Koch.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

A Koch snowflake is constructed from several Koch curves. Let's look at Koch's algorithm for drawing a curve in more detail. The simplest possible curve is actually a straight line of length *n*, the base case. The next easiest curve can be described in terms of the following set of turtle instructions:

```
forward(n)
left(60)
forward(n)
right(60)
right(60)
forward(n)
left(60)
forward(n)
```

Description

If you think recursively about the curve, you can imagine that each straight line can be replaced by a set of shorter lines following this set of instructions. **Figure 9.9** illustrates the Koch curve at several levels of detail.

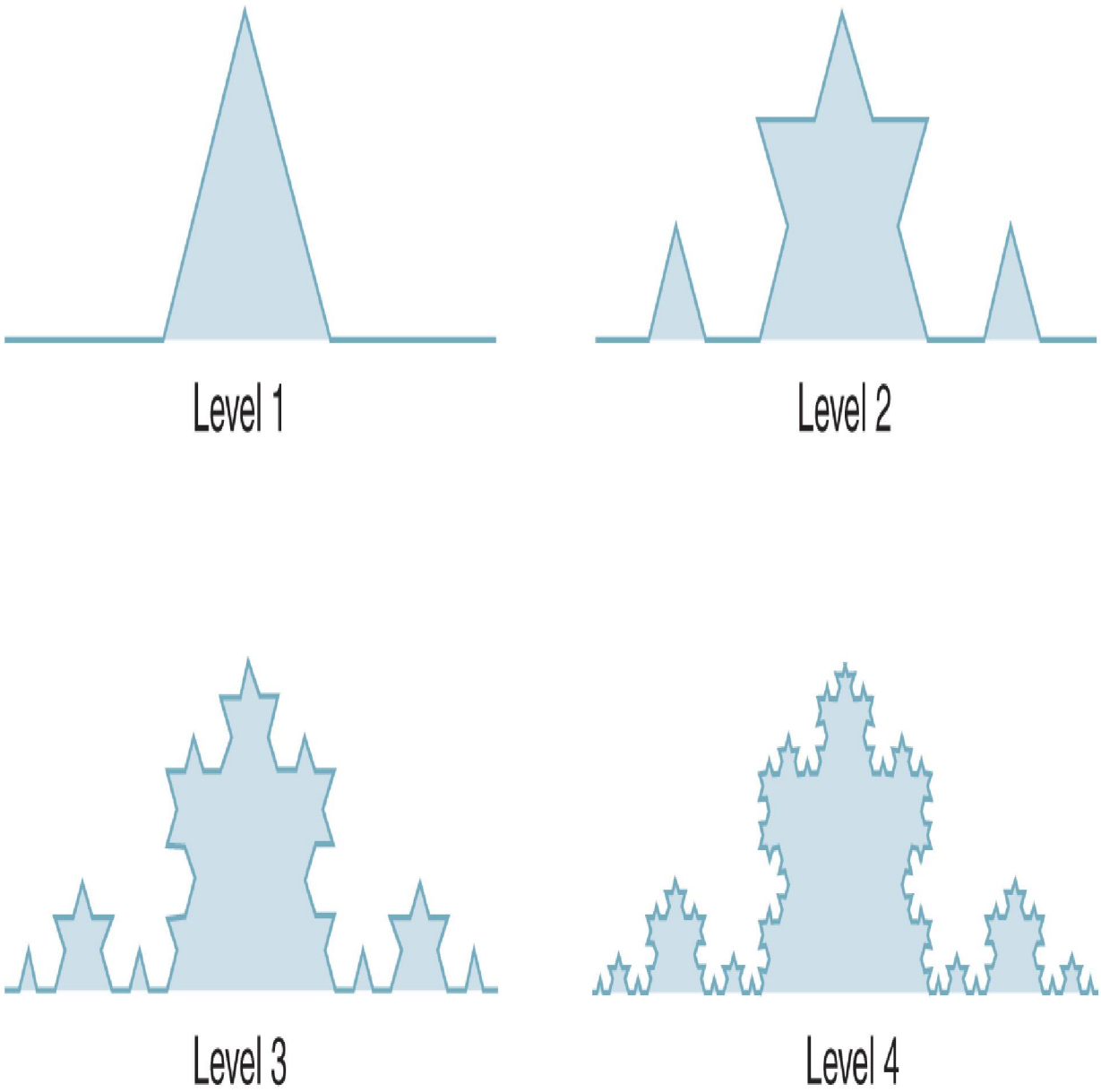


Figure 9.9 Four levels of a Koch curve.

Description

At this point you could write a simple recursive function to plot a Koch curve. (In fact, Try It Out Exercise 9.23 asks you to do so.) But let's think a bit differently about fractals and recursion.

9.4.1 L-Systems and the `match` Statement

The biologist Aristid Lindenmayer invented the **L-system** in 1968. The L-system is a formal mathematical theory designed to model the growth of biological systems. This construct can be used to specify the rules for all kinds of fractals.

As it turns out, L-systems are based on a formal idea from computer science called a **grammar**, which may be used for many purposes, including the specification of programming languages. A grammar consists of a set of symbols and one or more **production rules**. These rules specify how one symbol can be replaced by one or more other symbols. A production rule has two parts:

1. The left side, which specifies a symbol
2. The right side, which specifies the symbols that can replace the symbol on the left

Let's look at a simple example of a grammar:

Axiom	A
Rules (1)	$A \rightarrow B$
(2)	$B \rightarrow AB$

We can interpret this grammar as follows, beginning with the **axiom**, the starting point. Axiom A is the simplest string you can produce in the grammar. You may also think of the axiom in terms of a base case for the grammar. The rules tell us how we can construct more complicated strings that are part of the grammar. For example, rule 1 tells us that we can substitute the string B for the string A, and rule 2 tells us that we can change the string B into the string AB.

When you apply the rules to a string, you must work from left to right, applying the rule where the left side matches the current symbol. If a rule matches, then you *must* apply the rule. Let's look at an example:

A	Axiom
B	(rule 1 applied to A)
AB	(rule 2 applied to B)
BAB	(rule 1 applied to A, then rule 2 applied to B)
ABBAB	(rule 2 applied to B, then rule 1 applied to A, then rule 2 applied to B)
BABABBAB	(rules applied: 1, 2, 2, 1, 2)

Try to apply the rules and produce a few more lines on your own. If you are applying the rules correctly, you will notice that the length of each string follows the Fibonacci sequence of numbers: 1, 1, 2, 3, 5, 8, 13, 21, ...

RECAP:
An L-system consists of a starting point called an axiom and a set of rules for substituting one symbol for another.

To make the transition from these simple L-systems to an L-system that a turtle can use to draw pictures, consider the interpretation of the symbols used to define the L-system. Suppose that rather than using the symbols A and B, we used the symbols F, B, +, and −.

- F indicates that the turtle should move forward while drawing a line.
- B indicates the turtle should move backward while drawing a line.
- The + symbol indicates that the turtle should turn right.
- The − symbol indicates that the turtle should turn left.

Let's return to the Koch curve and consider the set of steps we outlined to draw a simple curve. We can specify the following set of actions as an L-system:

Axiom	F
Rule	$F \rightarrow F - F + + F - F$

Notice that the axiom gives us a straight line, which is the simplest Koch curve. But if we apply the production rule one time, we get the string `F-F++F-F`. This gives us the next simplest Koch curve.

Applying the production rule to the string `F-F++F-F` gives us the following string:

```
F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F
```

Given a string of instructions, it is straightforward to write a Python function that could interpret the string and make the appropriate calls to turtle methods. To successfully draw a Koch curve, we would need to know the distance to have the turtle go forward, along with the angle for the turtle's turn.

Our function will simply iterate over each character in the instruction string. When it encounters an "`F`" or "`B`," the turtle is instructed to move forward or backward `distance` units. When a "`+`" or "`-`" is encountered, the turtle is instructed to turn right or left by `angle` degrees. If any other character is encountered, the function prints an error message.

Certainly, we could compare each element in the pattern using an `if-elif-else` statement in this function. But Python has a `match` command designed especially for comparing an expression (the subject) to a set of values to determine which action to take. The `match` command looks like this:

```
match ‹subject›:
    case value1:
        #statements to execute if subject == value1
    case value2:
        #statements to execute if subject == value2
    ...
    case _:  #optional
        #statements to execute if subject does not match any case values
```

Description

The subject of a `match` statement can be an element in a sequence or some other object. The `match` statement compares the subject to values in `case` statements. Each `case` value is compared to the subject in the order in which the `case` statements appear. If the subject equals a value in a `case` statement, the statements under that `case`(called the **case block**) are executed and the `match` statement ends. Optionally, the last case can be a **wildcard** (_) where you provide statements to execute if the subject does not equal any of the `case` values. Although the same logic could be written in an `if-elif-else` statement, the structure of the `match` statement makes the logic of the code easier to see. We call this code attribute **readability**. Code that is readable is also easier to maintain.

GOOD PROGRAMMING PRACTICE:

The `match` statement can make your code more readable, meaning that the logic is easy to see. Readability can make code easier to understand and to modify when changes are required.

HEADS UP:
Although `match` and `case` have special meanings in the `match` statement, they are not Python keywords, so `match` and `case` can still be used elsewhere as object names.

Listing 9.6 shows the code for the function `drawLS` using a `match` statement to implement the L-system pattern. If the L-system pattern contains an unrecognized character, we print an error message (lines 12–13).

```
1  def drawLS(t, instructions, angle, distance):
2      for cmd in instructions:
3          match cmd:
4              case 'F':
5                  t.forward(distance)
6              case 'B':
7                  t.backward(distance)
8              case '+':
9                  t.right(angle)
10             case '-':
11                 t.left(angle)
12             case _:                  #wildcard matches any other value
13                 print( "Error", cmd, "is an unknown command.")
```

Listing 9.6 A simple function to follow an L-system string

Description

Try It Out

9.20 Use the `drawLS` function to draw the string `F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F`. Use an angle of 60 degrees and a distance of 20.

9.21 Apply the production rule for the Koch curve one more time, and draw the results using `drawLS`.

9.22 Experiment with the `drawLS` function using different angles and distances.

9.23 Write a simple recursive function to draw a Koch curve.

9.24 Using the function for a Koch curve, draw a snowflake with six sides.

9.25 One of Lindenmayer's first L-systems was a set of rules to model the growth of algae. The rules are as follows:

Axiom	A
Rules	A → AB
	B → A

Apply these rules to construct five new strings. The length of each string represents the number of algae cells.

9.26 Here is another simple set of rules that generate `Cantor dust`.

Axiom	A
Rules	A → ABA
	B → BBB

Apply the rules to produce five new strings.

9.4.2 Automatically Expanding Production Rules

Applying the production rules in an L-system can be tedious, so let's write a function to automate the process. We first need to determine "how production rules can be represented in Python." The solution to this problem is to use a Python dictionary.

A dictionary can represent any number of production rules, where the left side of the rule is the key and the right side of the rule is the value. For example, the following set of production rules generates the Fibonacci numbers:

Axiom	A
Rules	$A \rightarrow B$
	$B \rightarrow AB$

The Python dictionary that corresponds to these rules can be stored as follows:

```
productionRules = {'A': 'B',
                   'B', 'AB'}
```

Description

Now let's write a Python function to take an initial string and apply a set of production rules a specified number of times. **Listing 9.7** shows the Python function `applyProduction`.

```
1 def applyProduction(axiom, rules, n):
2     for i in range(n):
3         newString = ""
4         for ch in axiom:
5             newString = newString + rules.get(ch, ch)
6         axiom = newString
7     return axiom
```

Listing 9.7 Automatically applying production rules

Description

Everything in the `applyProduction` function should look familiar. The outer loop allows us to apply the production rules to the string `n` times. The inner loop is a simple accumulator pattern that allows us to construct a new string by applying the production rules one character at a time. Notice the statement `rules.get(ch,ch)` on line 5. This statement elegantly handles the case in which we encounter a character but do not have a production rule for expanding the character further. Recall that the `get` method allows us to specify a default value for the case when a key is not found in the dictionary. In this instance, when character `ch` has no production rule, we simply leave the character in place.

Session 9.5 shows the `applyProduction` function in action using the Fibonacci production rules.

```
In [1]: 1 from applyProductionRules import *
```

In [2]:
```
1 axiom = 'A'
2 myRules = {'A': 'B', 'B': 'AB'}
```

In [3]:
```
1 for i in range(10):
2     res = applyProduction(axiom, myRules, i)
3     print("{0:2d} {1}".format(len(res), res))
4
```

```
 1 A
 1 B
 2 AB
 3 BAB
 5 ABBAB
 8 BABABBAB
13 ABBABBABABBAB
21 BABABBABABBABBABABBAB
34 ABBABBABABBABBABABBABABBABBABABBAB
55 BABABBABABBABBABABBABABBABBABABBABABBABABBABBABABBAB
```

Session 9.5 Using the production rule expander

Description

Try It Out

9.27 Use the `applyProduction` and `drawLS` functions to draw a Koch curve of level 5.

9.28 We can easily draw a Koch snowflake using L-system rules by starting with a more complex axiom. Use the `applyProduction` and `drawLS` functions to draw a fractal using the axiom `F++F++F` and the production rule `F → F-F++F-F`.

9.29 Using an angle of 90 degrees and the following L-system, draw the resulting figure for several different levels of expansion.

Axiom	F-F-F-F
Rules	F → F – F+F+FF – F – F+F

This figure is known as the quadratic Koch island.

9.4.3 More Advanced L-Systems

To use L-systems to model plant growth, we need to add one more feature to our grammar, the ability to save and restore the state of the turtle. We will use the character "`[`" to save the turtle's state and "`]`" to restore the turtle's state. Whenever the function encounters a `[` in the string, the position and heading of the turtle are saved. When it encounters a `]`, the position and heading from the last save are restored.

Using these new characters, we can now use an L-system to specify the tree we drew using Listing 9.4. The grammar for this new L-system is as follows:

Axiom	X
Rules	X → F[–X]+X
	F → FF

Starting with the axiom `X` and applying the production rules twice gives us the following string:

```
FF[-F[-X]+X]+F[-X]+X
```

Suppose that we use an angle of 60 degrees and a distance of 20 units. This string instructs the turtle to take the following actions:

1. Go forward a total of 40 units.
2. Save the position and heading; call this state W.
3. Turn left 60 degrees.
4. Go forward 20 units.
5. Save the position and heading; call this state Z.
6. Turn left 60 degrees.
7. Do nothing. X is not a recognized turtle command.
8. Restore the turtle position from the saved state called Z.
9. Turn right 60 degrees.
10. Do nothing.
11. Restore the position and heading from the saved state called W.
12. Turn right 60 degrees.
13. Go forward 20 units.

At this point, we have succeeded in drawing a Y. But notice the difference between this approach and the code in Listing 9.4: We no longer need to have the turtle back up. By using the save and restore operations, we can simply jump the turtle back to any previous position.

Let's modify the `drawLS` function to support the save and restore operations. **Listing 9.8** shows the new function, again using the `match` command. In this version, we have removed the final wildcard `case` statement and its corresponding error message: This allows the function to silently ignore any commands it does not recognize. The `stateSaver` list allows us to append new states that we want to save to the end of the list. Using the `pop` method, we can restore the most recently saved state.

```
def drawLS(aTurtle, instructions, angle, distance):
    stateSaver = []     #List to save turtle state
    for cmd in instructions:
        match cmd:
            case 'F':
                aTurtle.forward(distance)
            case 'B':
                aTurtle.backward(distance)
            case '+':
                aTurtle.right(angle)
            case '-':
                aTurtle.left(angle)
            case '[':
                pos = aTurtle.position()   #get turtle's position
                head = aTurtle.heading()   #and heading
                stateSaver.append((pos, head))  #save as tuple
            case ']':
                pos, head = stateSaver.pop() #get first items on list
                aTurtle.up()
                aTurtle.setposition(pos)    #restore position
                aTurtle.setheading(head)    #and heading
```

```
21                aTurtle.setheading(head)    #and heading
22                aTurtle.down()
```

Listing 9.8 An improved L-system drawing function

Description

Finally, let's write a simple function that combines all our work to apply the production rules and draw the resulting `L-system`. This `lSystem` function takes seven parameters:

`axiom`	The initial axiom
`rules`	A dictionary specifying the production rules
`depth`	The number of times to expand the production rules
`startPos`	The initial position for the turtle
`heading`	The initial heading for the turtle
`angle`	The angle to turn for the "+" and "–" operations
`length`	The distance to move forward or backward for an "F" or "B," respectively

The code for the `lSystem` function is shown in **Listing 9.9**. In **Session 9.6**, we use this function to draw a simple tree.

```
1  def lSystem(axiom, rules, depth, startPos, heading, angle, length):
2      import turtle
3      aTurtle = turtle.Turtle()
4      aTurtle.color('GREEN')
5      win = turtle.Screen()
6      aTurtle.up()
7      aTurtle.setposition(startPos)
8      aTurtle.down()
9      aTurtle.setheading(heading)
10     newRules = applyProduction(axiom, rules, depth)
11     drawLS(aTurtle, newRules, angle, length)
12     turtle.done()
```

Listing 9.9 A function to expand and draw an L-system

Description

```
In [1]:  1 from drawlSystem import *
         2 import turtle
         3 myRules = {'X': 'F[-X]+X', 'F': 'FF'}
         4 axiom = 'X'
         5 lSystem(axiom, myRules, 7, (0, -200), 90, 30, 2)
```

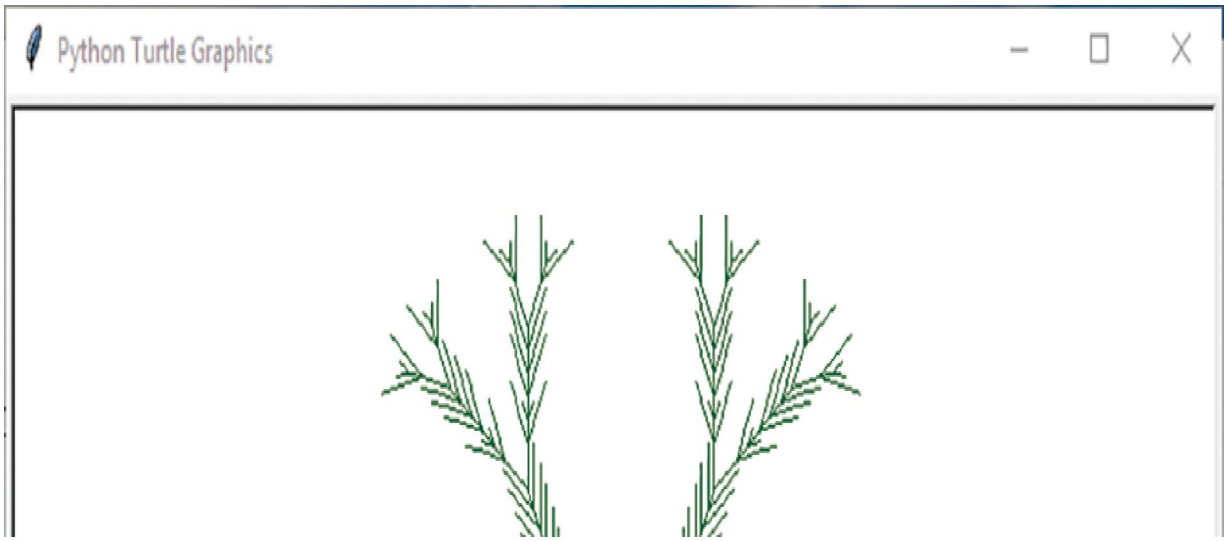
Session 9.6 Drawing a tree using an L-system

Description

Here are the rules for drawing something that looks like a sprig of rosemary:

Axiom	H
Rules	H → HFX[+H][– H]
	X → X[– FFF][+FFF]FX

Using a level of 5, an angle of 25.7 degrees, and a length of 8, we get the image shown in **Figure 9.10**. We also modified `lSystem` to set the color. In this case, we used `'GREEN'`.



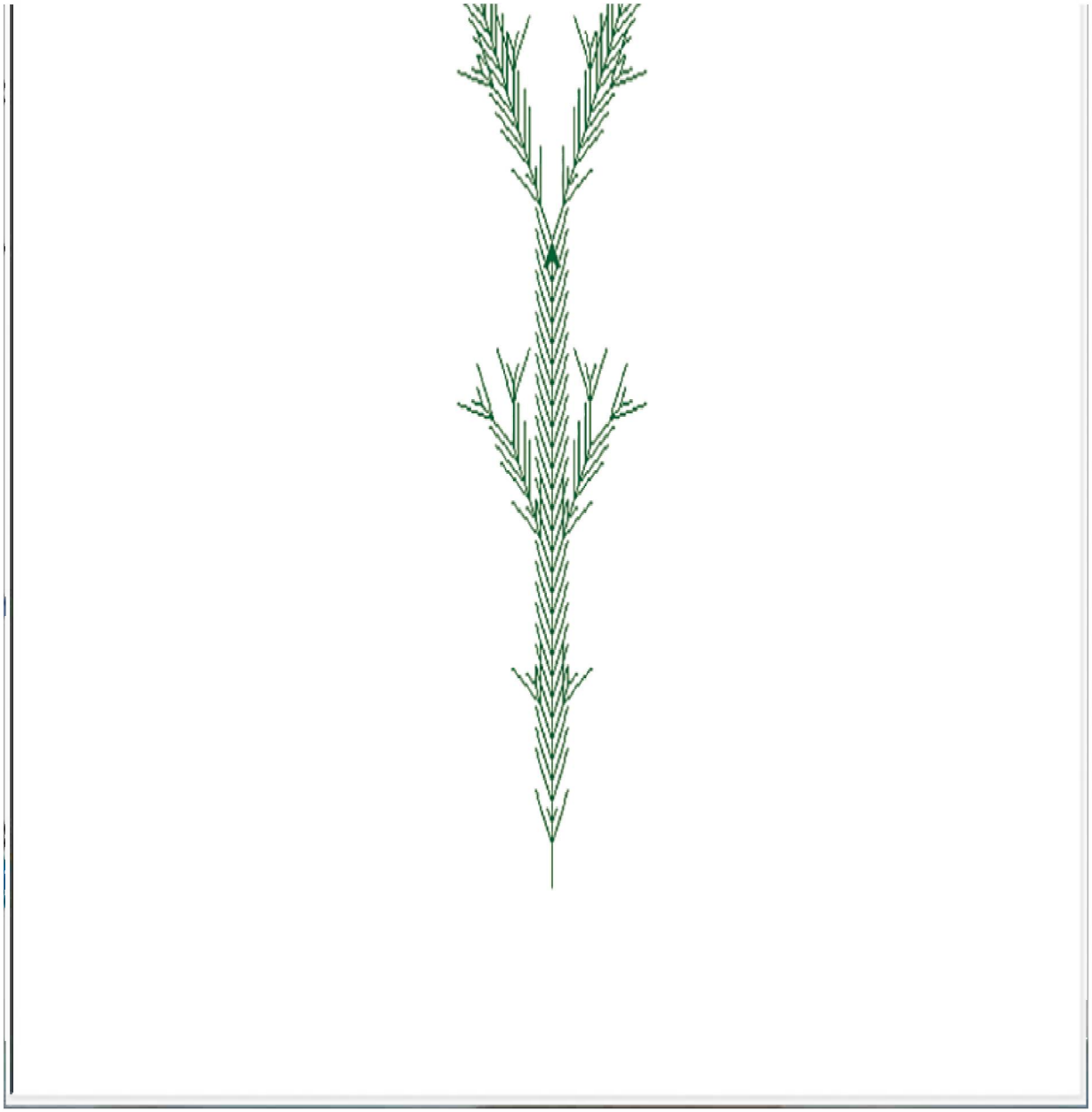


Figure 9.10 A sprig of rosemary drawn by a turtle.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

From here on, you are on your own to explore the world of L-systems. The following exercises have some examples to get you started, but you can find many more L-systems by doing some research.

Try It Out

9.30 Modify the `drawLS` function to support the "G" operation, which allows the turtle to go forward without drawing a line.

9.31 Using your modified `drawLS` function, try the following L-system:

Axiom	F-F-F
Rules	F → F-F-F-GG
	G → GG

9.32 Implement the following L-system using an angle of 25:

Axiom	F
Rules	F → F[-F]F[+F]F

9.33 Implement the following L-system using an angle of 25:

Axiom	F
Rules	F → FF+[+F-F-F]-[-F+F+F]

9.34 Implement the following L-system using an angle of 45:

Axiom	FX
Rules	F →
	Y → +FX-FY+
	X → -FX++FY-

9.5 Summary

In this chapter, we introduced recursion, a powerful programming technique that allows us to elegantly solve problems by taking advantage of "self-reference." Fractals are graphically self-referential—that is, they contain smaller versions of themselves. We can implement recursion by simply letting a Python function call itself. Defining the nonrecursive case, called the base case, is important to make sure that the recursive calls work correctly. We explored L-systems, which formally denote fractal behavior using a set of grammar rules. A string of symbols can be produced by repeatedly applying these rules. By interpreting this result string as a series of turtle commands, we can draw a fractal. We also introduced the `match` statement, which can be a more readable replacement for an `if-elif-else` statement.

Key Terms

axiom
base case
call tree
case block
fractal
grammar
Koch snowflake curve
L-system
production rules
readability
recursion
recursive process
recursive step
Sierpinski triangle
wildcard

Programming Exercises

9.1 Modify the `lSystem` function so that it recursively expands the rules. This will allow you to add a vertical bar (`|`) operator that means "go forward," but with the distance to go forward being scaled according to the depth of the recursion.

9.2 Research L-systems and find a different set of production rules to implement.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 10

Planet Objects

Creating Classes: Writing Constructors, Accessors, Mutators, and Special Methods

10.1 OBJECTIVES

- To explore classes and objects further
- To understand how to construct a class
- To write constructors, accessor methods, mutator methods, and special methods
- To understand the concept of `self`
- To explore instance data
- To implement a graphical simulation using objects

10.2 Introduction

Have you ever witnessed a lunar eclipse? The resulting images of a shadowed moon with changes in color and brightness cause us to ask questions about the details of this uncommon occurrence. We learned in science class that this fascinating celestial event is caused by the relative motion of the Earth, sun, and moon as they orbit around one another. Early astronomers, such as Ptolemy and Copernicus, built elaborate models of the planets to show their relative positions or sizes. These models were constructed so that the motion of the planets in the model matched the motion of the planets in the night sky.

In this chapter, we will build a different type of planetary model: a **software model**. Our model will consist of planets and a sun, and it will also be capable of showing relative movement. Along the way, we examine ways that modern programming languages and techniques allow real situations to be represented and manipulated as programming constructs.

10.2.1 Focusing on Data

Many people assume that computer science is all about programming. As we have seen, computer science is actually much more. It is the study of problems, the problem-solving process, and the solutions to those problems. These solutions, also known as algorithms, provide a starting point for programming.

Programming is the process of encoding an algorithm into a notation that the computer can execute. These notation systems are referred to as programming languages. In this text, we are using the Python programming language to represent our solutions.

Programming is an important part of what a computer scientist does. It is through programming that we create a representation of our solution. However, the solutions that we achieve are often affected by the process and language that we choose.

Algorithms describe the solution to a problem in terms of the data needed to represent the problem and a set of steps necessary to produce the intended

result. Programming languages must then provide a way to represent both the process and the data required by the solution. In other words, programming languages such as Python must provide control structures and data types.

We have already seen some of these basic control structures and data types in action. We know that our programs process statements in sequence. In addition, special statements such as `if` and `for` enable the flow-of-control mechanisms to allow selection and iteration of groups of statements. We can also use abstraction to bundle statements into functions that can accept parameters, thereby enabling them to do their work. In some cases, the functions return a result.

In this chapter, we focus on data. Of course, we will still need control structures to represent our algorithms. However, as we will see, specific requirements for solving a problem can often be included as part of the **data model** that we build.

10.2.2 Object-Oriented Programming

Python supports the **object-oriented programming** paradigm, which implies that Python considers data important to the problem-solving process. In fact, in many ways data becomes our focus when confronted with a new problem to solve. Object-oriented programming is about objects—the individual data items that are manipulated to solve a problem. In Python, every data item is an **object**.

To understand objects, you must also understand the idea of a **class**. Each object is an **instance** of a class. Classes are used in object-oriented programming to describe what an object will "look" like, what an object knows about itself (its **instance data**), and what an object can do (its **methods**). Classes are often described as templates for objects in that each object that belongs to a specific class will have the same instance data and methods. However, because the specific values of these data items will be different for each object instance, the objects will likely behave differently when asked to perform their methods.

You are already familiar with some of these ideas from your work with the `Turtle` class. When we create a turtle using the statement `t = turtle.Turtle()`, we are actually creating an instance of the `Turtle` class. We can create more than one turtle, and each turtle has its own instance data.

For example, each turtle has a position, a heading, a color, and so on. Each turtle also has a number of methods—for example, `forward`, `left`, and `up`. The behavior of a specific turtle depends on the values of its own instance data. Thus, if a turtle has a red tail, it will draw a red line when the `forward` method is called; in contrast, a turtle with a blue tail will draw a blue line when its `forward` method is called.

10.2.3 Python Classes

We have already explored a number of classes provided by Python. Some of these, such as `int`, `bool`, and `float`, are called primitive classes because they represent only a single value. For example, the integer (object) 5 is an instance of the class `int`. Likewise, the object `True` is an instance of the class `bool`. The `type` built-in function, described in **Table 10.1**, allows us to query an object to learn its data type. For example, in **Session 10.1**, we define various kinds of variables and call the `type` function on each.

TABLE 10.1 The `type` Built-in Function

The `type` Function	Explanation
`type(object)`	Returns the data type of an object

```
In [1]:  1 count = 1
         2 type(count)

Out[1]: int
```

```
In [2]:  1 radius = 1.57
```

```
2 type(radius)
```

Out[2]: float

In [3]:
```
1 isRaining = False
2 type(isRaining)
```

Out[3]: bool

In [4]:
```
1 myList = [23, 66, True]
2 type(myList)
```

Out[4]: list

In [5]:
```
1 city = "Kingston"
2 type(city)
```

Out[5]: str

Session 10.1 Python types

Description

Similarly, we have explored classes such as `str` and `list` that describe string and list objects. These collection classes provide a structure that enables us to group objects together. Continuing in Session 10.1, we then define a list as a collection of three objects: two integers and a Boolean. Finally, we define a string.

Although it may not be obvious, even integers use methods to perform basic arithmetic. **Session 10.2** shows an integer variable called `count` with an initial value of 1. The next statement performs an addition using the accumulator pattern that we have seen many times before. As we would expect, the value of `count` is now 2. The next statement looks a bit strange, however. Although it is the accumulator pattern once again, the right side of the assignment uses a special method called `__add__()` to perform the addition. Note that two underscores appear before and after the method name, `add`. This method, which is defined in the `int` class, returns the result of adding the value of the parameter—in this case, 1—to the value of the object. The indexing operator for lists, also shown in Session 10.2, is a special method called `__getitem__()`, where `myList.__getitem__(2)` is equivalent to `myList[2]`. We will talk more about these special methods later in the chapter.

In [1]:
```
count = 1
count = count + 1
count
```

Out[1]: 2

In [2]:
```
count = count.__add__(1)    #equivalent to count + 1
count
```

Out[2]: 3

In [3]:
```
myList = [23, 66, True]
myList[2]                   #get item at index 2
```

Out[3]: True

In [5]:
```
myList.__getitem__(2)       #equivalent to myList[2]
```

Out[5]: True

Session 10.2 Integer addition and list indexing methods

Description

10.3 Designing and Implementing a `Planet` Class

We now turn our attention to solving the problem of building a software model of the solar system. To do so, we need to consider the specific data used in the model. Even with the rich set of built-in classes provided by Python, it is often preferable to describe our problem and solution in terms of classes that are specifically designed to represent the objects in the problem. We begin by building a simple representation of a planet; we will then design and implement a `Planet` class.

To design a class to represent the idea of a planet, we need to consider the data the planet objects will need to know about themselves—that is, their instance data. The values of the instance data will help to differentiate the individual planet objects. To begin, we assume that each planet has a name. Each planet also has size information, such as its radius and its mass. We also want to know how far the planet is from the sun.

In addition to data, the class will provide methods that a planet can perform. Some of these might be simple, such as returning the planet's name. Other methods may require more computation. In Python, the general format for defining a class begins with the keyword `class` followed by the name of the class and a colon. The methods for the class are indented under the class heading and look just like function definitions.

```
class Classname:

    def method1():
        ...
    def method2():
        ...
    ...
```

Description

10.3.1 Constructor Method

The first method that all classes should provide is a **constructor**, which initializes the data objects, assigning the instance its initial state. To create a `Planet` object, we will need the previously listed four pieces of data as parameters: (1) name, (2) radius, (3) mass, and (4) distance. The constructor will then assign values to the **instance variables**. Each instance variable holds a reference to its object.

In Python, the constructor is always called `__init__`. (Note the two underscores that appear before and after `init`.) Methods in a class definition are written in the same way as the other functions we have seen thus far. For the constructor, we provide the name of the method and a collection of parameters that usually provide values for the instance variables. These parameters will be filled in with values when the constructor method is invoked.

Listing 10.1 shows the constructor for the `Planet` class. Notice that even though we said that four pieces of information would be necessary to construct a planet, this method has five formal parameters. The extra parameter, `self`, is a special parameter that will always refer to the object being constructed. It is required and must always be the first parameter in the list. Python automatically adds an actual parameter corresponding to `self` when you call the constructor, so you should not explicitly pass a parameter corresponding to `self`.

```
1 class Planet:
2     def __init__(self, iName, iRad, iM, iDist):
3         self.name = iName
4         self.radius = iRad
5         self.mass = iM
6         self.distance = iDist
```

Listing 10.1 `Planet` class with a constructor

Description

RECAP:
When defining a method in a class, list `self` as the first parameter. When calling the method, however, do not pass a value for `self`. Python supplies the value for the `self` parameter, which refers to the object.

Line 3 of Listing 10.1 defines an instance variable called `name`, which is attached to the special parameter `self` by a dot. The notation `self.name`, when it appears on the left side of an assignment statement in the constructor, defines an instance variable. Because `self` refers to the object being constructed, `self.name` refers to a variable in the object. Lines 4–6 introduce and assign values to the other three instance variables.

We use the constructor to create an instance of the class by using the name of the class (we do not call `__init__` directly). Just as `__add__` is called automatically when the `+` operator is used, `__init__` is called automatically when the class name is followed by the `()` function call operator.

HEADS UP:
Do not call the `__init__` method directly. The `__init__` method is called automatically when the class name is followed by parentheses.

Assuming that we have saved the definition of the `Planet` class in the file `planetclass.py`, **Session 10.3** shows the creation of a `Planet` object named `myPlanet`. Note that the statement is similar to creating an instance of the `Turtle` class. As stated earlier, the call to the `Planet` constructor requires only four parameters, even though it is defined to have five. The first parameter, `self`, never receives an explicit value, as it always refers implicitly back to the object being constructed. Evaluating the reference `myPlanet` shows that it is an instance of the `Planet` class and the value `0x21b7e2179a0` is actually the value of `self`, which is the address in memory where this object is stored. When you run Session 10.3, your `Planet` object's address will almost certainly be different. When we create our own classes, we are creating a new type. This becomes apparent when we call the `type` function on the `myPlanet` object.

```
In [1]:  1 from planetclass import *
```

```
In [2]:  1 myPlanet = Planet("X25", 45, 198, 1000)
         2 myPlanet
```

```
Out[2]: <planetclass.Planet at 0x21b7e2179a0>
```

```
In [3]:  1 type(myPlanet)
```

```
Out[3]: planetclass.Planet
```

```
In [4]: 1 myPlanet.name          #retrieve the values of the instance variables
```

```
Out[4]: 'X25'
```

```
In [5]: 1 myPlanet.radius
```

```
Out[5]: 45
```

```
In [6]: 1 myPlanet.mass
```

```
Out[6]: 198
```

```
In [7]: 1 myPlanet.distance
```

```
Out[7]: 1000
```

```
In [8]: 1 myPlanet.mass = 250   #change the mass
        2 myPlanet.mass

Out[8]: 250
```

Session 10.3 Creating a `Planet` object

Description

Also note that the instance variables are accessible through the object name—which is not a good thing. As Session 10.3 shows, the instance variable values can be accessed, and they can also be changed. This violates a basic design principle of classes, known as **encapsulation**: Only the class methods should be able to directly change the value of an object's instance variables.

To remedy this exposure of the instance variables, we can preface each instance variable name with two underscores (_ _). Instance variables whose names begin with _ _ are hidden from access outside the class. The new constructor for the `Planet` class is shown in **Listing 10.2**.

```
1 class Planet:
2     def __init__(self, iName, iRad, iM, iDist):
3         self.__name = iName
4         self.__radius = iRad
5         self.__mass = iM
6         self.__distance = iDist
```

Listing 10.2 The revised `Planet` class with hidden instance variables

Description

Now when we try to access an instance variable, Python reports an error, as shown in **Session 10.4**. But don't worry; we will soon write methods that allow retrieving and changing the instance variable values. Another advantage of prefacing instance variables' names with two underscores is that instance variables are easily distinguished from other local or global variables.

```
In [1]:  1 from planetclass2 import *
```

```
In [2]:  1 myPlanet = Planet("X25", 45, 198, 1000)
         2 myPlanet._name
```

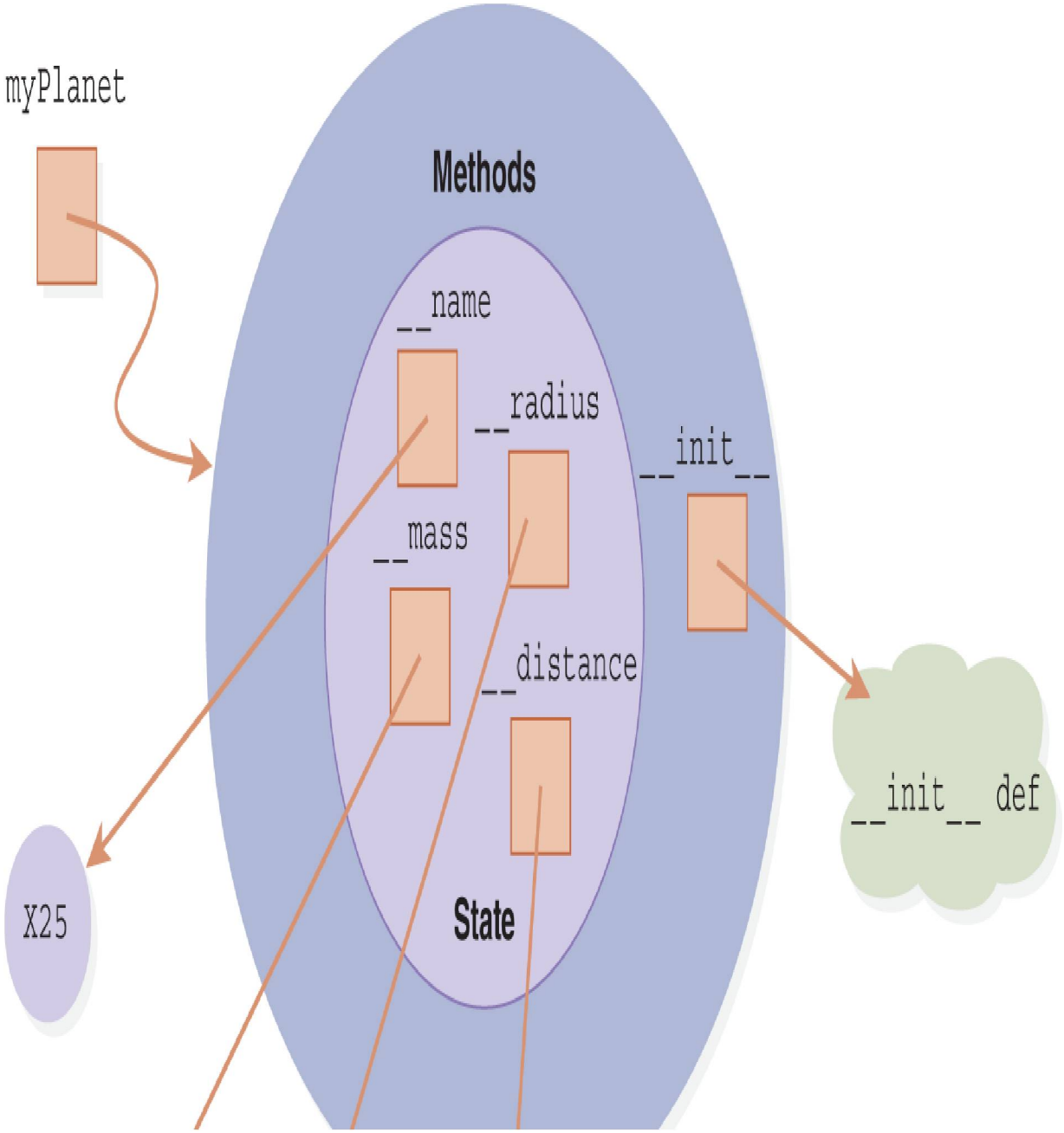
```
AttributeError                            Traceback (most recent call last)
Input In [2], in <cell line: 2>()
      1 myPlanet = Planet("X25", 45, 198, 1000)
----> 2 myPlanet._name

AttributeError: 'Planet' object has no attribute '_name'
```

Session 10.4 Instance variables are now hidden

Description

The newly created object `myPlanet` is an instance of the `Planet` class with a name of `"X25"`, a radius of 45, a mass of 198, and a distance of 1000. **Figure 10.1** shows a logical view of this object. Notice that we have separated the object into two distinct layers. The inner layer, or **object state**, contains the instance variable names. The outer layer contains the names of the methods. In both cases, the names are simply references to the actual objects. We draw objects in this fashion to suggest the strong relationship between the methods of an object and its instance data.



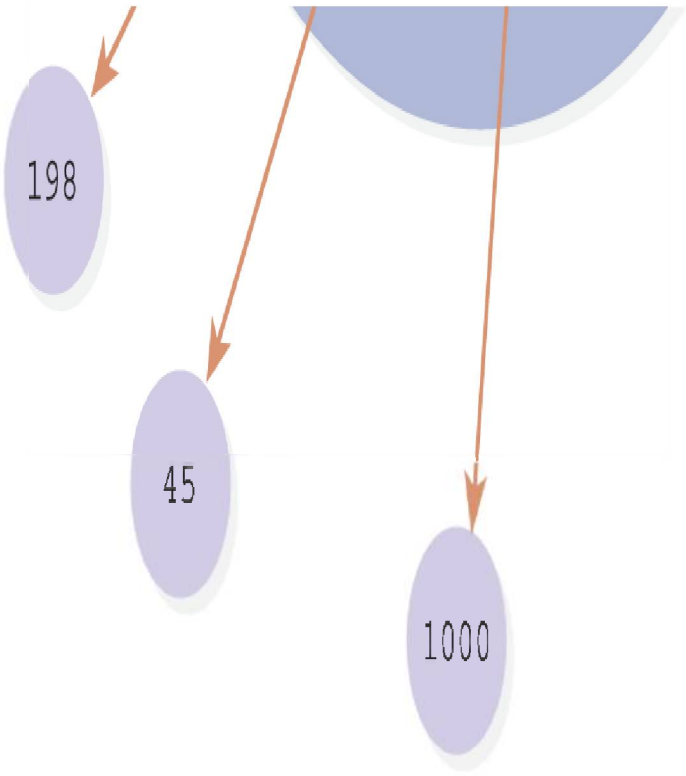


Figure 10.1 A `Planet` object.

Description

10.3.2 Accessor Methods

If no code outside the class can access the data of an object, then you might wonder how the data of an object can ever be known or changed. The answer is that classes typically provide **accessor methods**, which allow code outside the class to access the current values of the instance variables. These methods are also referred to as "**getter methods**" because the word "get" often appears in the name. Each instance variable usually has an associated accessor method. Another advantage of using accessor methods to retrieve the object data is that in a large software project, it is common for the internal representation of an instance variable to change. An accessor method hides those internal changes from the user—a practice known as **information hiding**.

Listing 10.3 shows the four accessor methods associated with the `Planet` instance variables. For example, the `getName` method returns the string

referenced by the `_ _name` instance variable. In the `getName` method, we refer to the `_ _name` instance variable as `self._ _name`, where `self` is a synonym for `myPlanet` (`self` and `myPlanet` both reference the same object).

```
 9  def getName(self):
10      return self.__name
11
12  def getRadius(self):
13      return self.__radius
14
15  def getMass(self):
16      return self.__mass
17
18  def getDistance(self):
19      return self.__distance
20
21  def getVolume(self):
22      v = 4/3 * math.pi * self.__radius**2
23      return v
24
25  def getSurfaceArea(self):
26      sa = 4 * math.pi * self.__radius**2
```

```
26          sa = 4 * math.pi * self.__radius**2
27          return sa
28
29      def getDensity(self):
30          d = self.__mass / self.getVolume()
31          return d
```

Listing 10.3 Accessor methods in the `Planet` class

Description

GOOD PROGRAMMING PRACTICE:
Preface the names of instance variables with two underscores so that they are private to the class. Provide accessor methods so that code outside the class can access the values of the instance variables.

Listing 10.3 also shows accessor methods that return computed results based on values of instance variables. For example, we assume that a planet is a sphere, so we can write an accessor method that returns the volume of the planet—the radius is already an instance variable. We also include methods that return the surface area as well as the density of the planet. Each of these values (volume, surface area, and density) can be calculated using the instance data, but volume, surface area, and density are not instance data. **Session 10.5** shows these methods in action.

```
In [1]:  1 from planet import *
         2 myPlanet = Planet('X25', 45, 198, 1000)
```

In [2]:
```
1 myPlanet.getName()
```

Out[2]: 'X25'

In [3]:
```
1 myPlanet.getRadius()
```

Out[3]: 45

In [4]:
```
1 myPlanet.getMass()
```

Out[4]: 198

In [5]:
```
1 myPlanet.getDistance()
```

Out[5]: 1000

In [6]:
```
1 myPlanet.getVolume()
```

Out[6]: 8482.30016469244

```
In [7]:  1 myPlanet.getSurfaceArea()

Out[7]: 25446.900494077323

In [8]:  1 myPlanet.getDensity()

Out[8]: 0.023342724986811 32
```

Session 10.5 Calling accessor methods

Description

Two additional details: In the methods to compute volume and surface area, we used the value of `pi` from the `math` module. So, we `import math` at the top of the `planet.py` file containing the class definition (not shown here). Also, the computation for density, which requires dividing the mass by the volume, calls the `getVolume` method. This illustrates that a class method can call another class method. To call the `getVolume` method, however, we must use `self.getVolume` because `self` is a reference to the `Planet` object. Note that `self` is a reference to the object invoking the `getDensity` method, which in turn can be asked to invoke the `getVolume` method. If we had simply tried to call `getVolume` directly, we would have gotten an error stating that `getVolume` was not found.

Try It Out

10.1 Modify the `Planet` class constructor to add a new instance variable called `_ _numMoons`.

10.2 Write a new accessor method to return the number of moons that references the `_ _numMoons` instance variable created in the previous exercise.

10.3 Write a new accessor method for `Planet` called `getCircumference`. Test your method by creating a `Planet` object and calling the new method.

10.4 Create a new class called `Sentence`. The constructor should accept a single parameter that is a string. Create an instance variable that stores the sentence as a string. Assume the sentence has no punctuation.

10.5 Write the following accessor methods for the sentence class created in the previous exercise:

a. `getSentence`: Returns the sentence as a string.

b. `getWords`: Returns a list of the words in the sentence.

c. `getLength`: Returns the number of characters in the sentence.

d. `getNumWords`: Returns the number of words in the sentence.

10.6 Create a variation of the `Sentence` class, again called `Sentence`. The constructor should accept a single parameter, a string. This time, create an instance variable that stores the sentence as a list of words.

10.7 Write the following accessor methods for the new class created in the previous exercise:

a. `getSentence`: Returns the sentence as a string.

b. `getWords`: Returns the list of words in the sentence.

c. `getLength`: Returns the number of characters in the sentence.

d. `getNumWords`: Returns the number of words in the sentence.

10.3.3 Mutator Methods

Mutator methods change (or mutate) an object in some way. Changes to the object, in turn, involve changes to one or more of the instance variables. These methods are also referred to as "**setter methods**" because the word "set" often appears in the name. Recall that each object from a particular class has the same instance variables, but the values of those variables are different, which allows an object to "behave" differently when asked to perform methods. To change the values of those variables, we provide mutator methods.

A mutator method will allow us to change the name of a planet in our example. Instead of using a cryptic name, such as "X25," we can change the name to something more meaningful. To do so, we need a method that takes the new name as a parameter and modifies the value of the `_ _name` instance variable. **Listing 10.4** shows the `setName` method. As before, the first parameter is `self`. In addition, we define a second parameter that will pass the new name (`newName`). In the body of the method, the value of `newName` is assigned to the instance variable `_ _name`. Notice that this method does not return anything—a pattern that you will often encounter. Mutator methods modify the state of an object but usually do not return any value to the caller. Mutator methods for the other instance variables follow this same pattern.

```
33  def setName(self, newName):
34      self.__name = newName
```

Listing 10.4 Mutator method in the `Planet` class

Description

RECAP:

- Typically, accessor methods return the value of an instance variable or the result of a calculation involving the instance variables.

- Typically, mutator methods take a parameter value to modify the state of an object and do not return a value.

When we use the `setName` method, as shown in **Session 10.6**, we provide one parameter value. Although the method definition has two parameters, remember that the first parameter, `self`, will receive a reference to the object implicitly. As the state of the object has now changed, it responds with a different value when asked to perform the `getName` method.

```
In [1]:  1 from planet import *
         2 myPlanet = Planet('X25', 45, 198, 1000)
```

```
In [2]:  1 myPlanet.getName()
```

```
Out[2]: 'X25'
```

```
In [3]:  1 myPlanet.setName("Gamma Hydra")
```

```
In [4]:  1 myPlanet.getName()
```

```
Out[4]: 'Gamma Hydra'
```

Session 10.6 Using a mutator method

Description

Try It Out

10.8 Write a mutator method called `setMoons` that changes the number of moons around our planet.

10.9 Add an instance variable called `_ _moonList` and a corresponding mutator method called `addMoon` that adds the name of a moon to the `_ _moonList`.

10.10 Write a mutator method for the `Sentence` class that allows you to capitalize all the letters in a sentence.

10.11 Write a mutator method for the `Sentence` class that allows you to add a punctuation mark to the end of the sentence.

10.12 Write an accessor method that allows you to translate a sentence into "pirate speak" or any other language. Note you could add a dictionary as an instance variable of the class. Refer to the Programming Exercises in the *Introducing the Python Collections chapter* for details on translating English to pirate speak.

10.3.4 Special Methods

As mentioned earlier, Python provides some special class methods, such as integer addition and item selection in a list. We can also define our own special methods within a class. **Table 10.2** shows some of the special methods that can be defined for a class. These special methods are sometimes referred to as **magic methods**, because they "magically" perform the work of an operator. Also, because these methods start and end with double underscores, they are often referred to as **dunder methods** ("dunder" stands for "double under").

TABLE 10.2 Some of Python's Special Methods

Method	Operation	Example	Use
`__getitem__`	Indexing	`obj1.__getitem__(x)`	`obj1[x]`
`__add__`	Addition	`obj1.__add__(x)`	`obj1 = obj1 + x`
`__contains__`	Search	`obj1. __contains__(str)`	`str in obj1`
`__str__`	Provide object as a string	`obj1. __str__ ()`	`print(obj1)`
`__eq__`	Equality	`obj1.__eq__(obj2)`	`obj1 == obj2`
`__ne__`	Inequality	`obj1.__ne__(obj2)`	`obj1 != obj2`
`__lt__`	Less than	`obj1.__lt__(obj2)`	`obj1 < obj2`
`__le__`	Less than or equal to	`obj1.__le__(obj2)`	`obj1 <= obj2`
`__gt__`	Greater than	`obj1.__gt__(obj2)`	`obj1 > obj2`
`__ge__`	Greater than or equal to	`obj1.__ge__(obj2)`	`obj1 >= obj2`
`__len__`	length	`obj1. __len__ ()`	`returns length or number of elements`

Description

One of these dunder methods, `_ _str_ _`, is used to provide a string representation for an object.

Session 10.7 shows the result of printing a `Planet` object. The `print` function automatically tries to convert the object into a string representation, but the `Planet` object does not know how to respond to this request. The result is the default representation, which is certainly not very helpful.

```
In [1]:  1 from planet import *
         2 myPlanet = Planet('X25', 45, 198, 1000)

In [2]:  1 print(myPlanet)

<planet.Planet object at 0x00000207DF077A00>
```

Session 10.7 Printing a `Planet` object

Description

We can provide better printing capability for the `Planet` class by defining our own version of the special method `__str__`. We have already seen that the default implementation for `__str__` returns a string containing the class name and object address. What we need to do is provide an alternative implementation for the `__str__` method that provides more user-friendly information.

To do this, we define a method with the name `__str__` to return a string that we choose to represent the planet, as shown in **Listing 10.5**. The modified method does not require any information other than the `self` parameter. It returns the string that we have chosen to represent the object—in this case, the planet's name. Note that the designer of a class decides which instance variables will be returned by the `__str__` method. The only restriction is that the return value should be a string. **Session 10.8** shows the result of printing the object. It also demonstrates that when you call either the `print` function or the `str` function on an object, that object's `__str__` method is automatically invoked.

```
36  def __str__(self):
37      return self.__name
```

Listing 10.5 The `_ _str_ _` method in the `Planet` class

Description

```
In [1]:  1 from planet import *
         2 myHome = Planet('Earth', 6371, 5.97e24, 152097701)
```

```
In [2]:  1 print(myHome)
```

```
Earth
```

```
In [3]:  1 myHome._str_()
```

```
Out[3]: 'Earth'
```

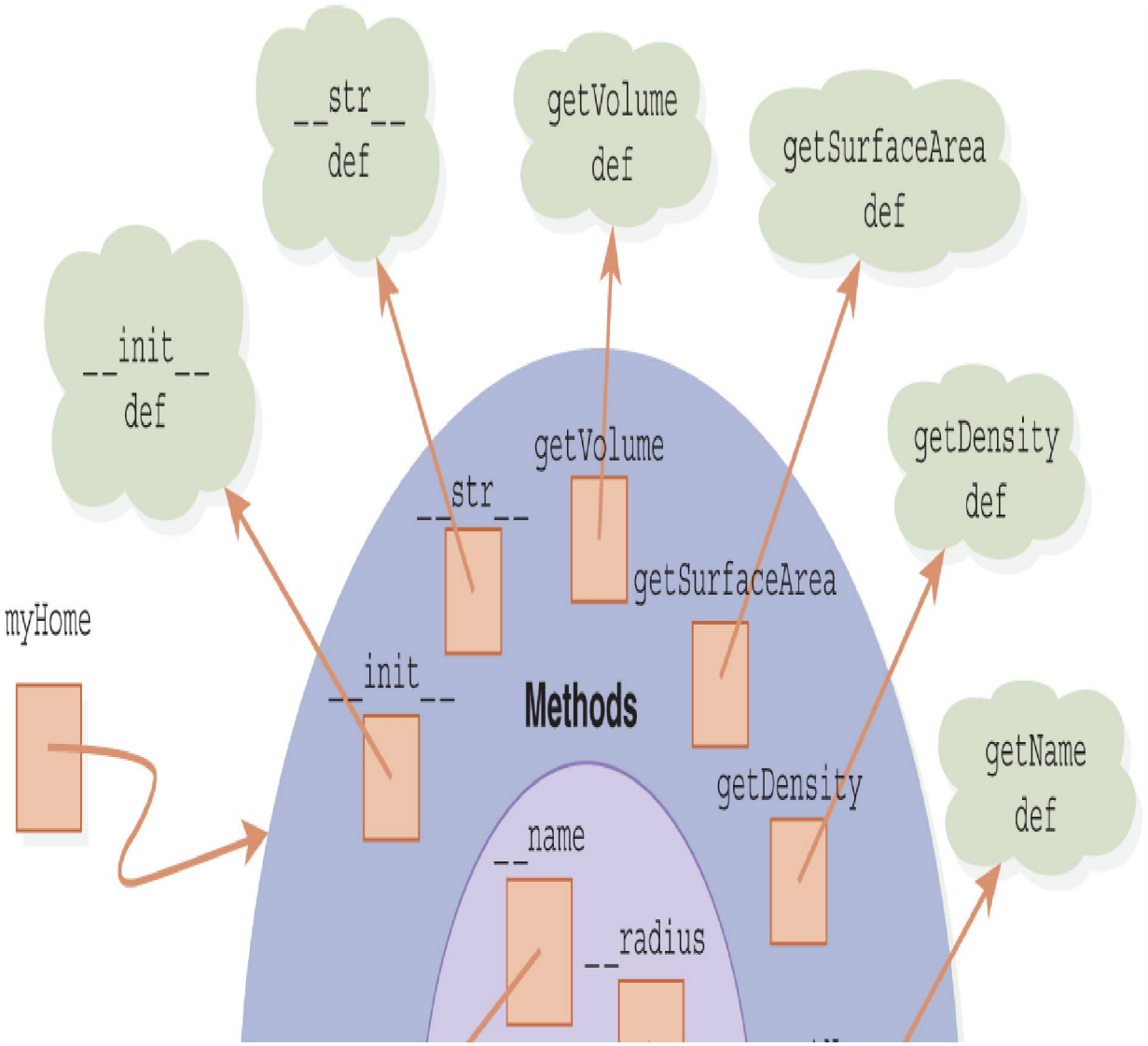
```
In [4]:  1 str(myHome)
```

```
Out[4]: 'Earth'
```

Description

GOOD PROGRAMMING PRACTICE:
It is good practice to provide a `__str__` method so that objects of your class can be printed.

Figure 10.2 shows the complete reference diagram for the `myHome` planet object.



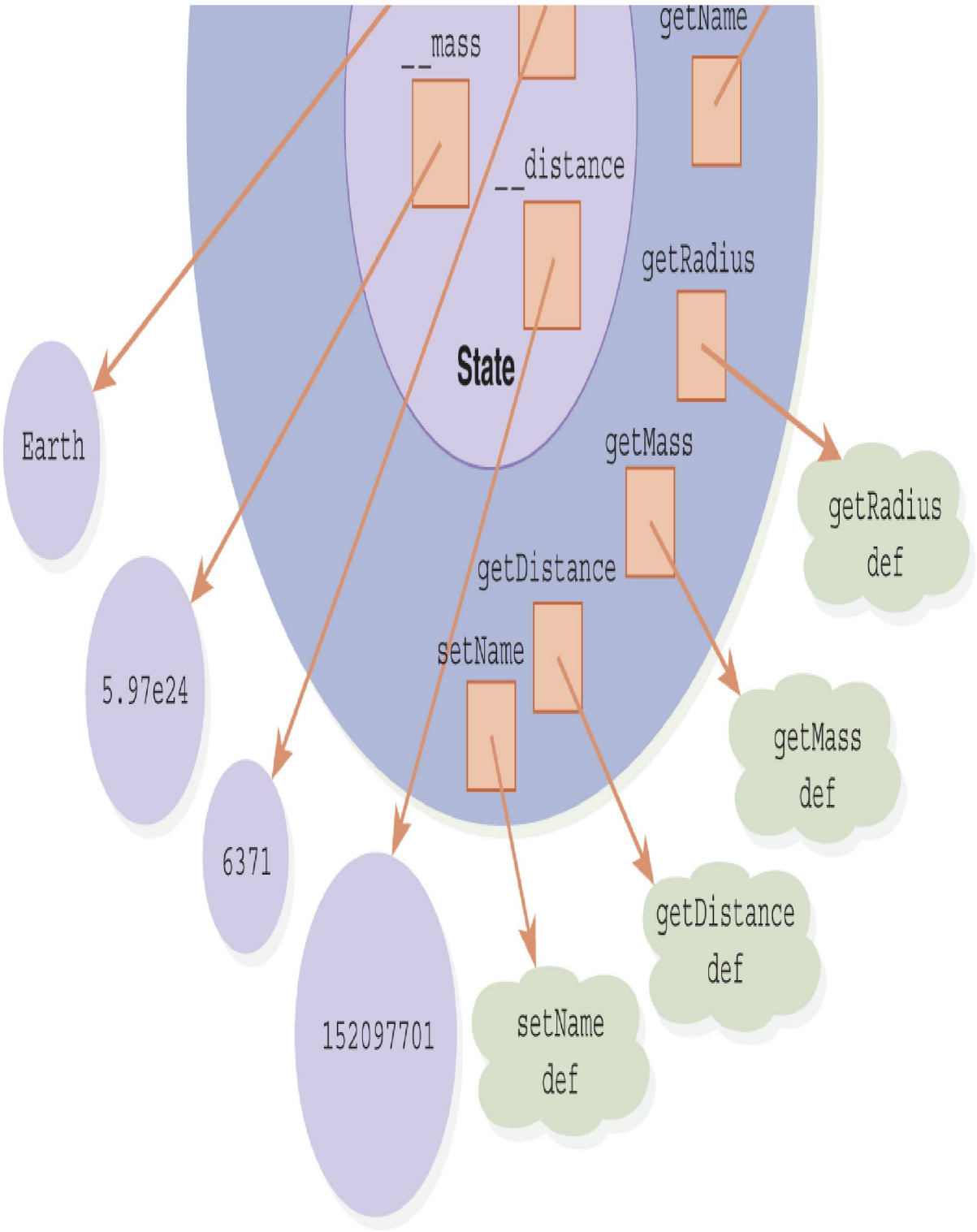


Figure 10.2 Logical view of a `Planet` object.

Description

Whether your class should implement versions of the other special methods depends on the class and its data. Like the return value for the `__str__` method, the special methods that you implement are a design decision. For example, for the `Planet` class, we do not need the `__add__` method because adding a planet to another planet doesn't make much sense. For the comparison methods, usually we choose an instance variable as the value to be compared between objects. To illustrate this concept, we could decide that a planet is "less than" another planet if it is closer to the sun. Then we could define the `__gt__` and `__lt__` special methods as shown in **Listing 10.6**. **Session 10.9** shows these methods in use.

```
39  def __lt__(self, otherPlanet):
40      return self._distance < otherPlanet._distance
41
42  def __gt__(self, otherPlanet):
43      return self._distance > otherPlanet._distance
```

Listing 10.6 The `__lt__` and `__gt__` methods in the `Planet` class

Description

```
In [1]:  1 from planet import *
         2 earth = Planet('Earth', 50, 60, 31)
         3 mercury = Planet('Mercury', 19, 10, 25)
         4 mars = Planet('Mars', 47, 50, 35)
```

```
In [2]:  1 mars < earth       #call the __lt__ method
```

```
Out[2]: False
```

```
In [3]:  1 earth > mercury    #call the __gt__ method
```

```
Out[3]: True
```

Session 10.9 Using the `__lt__` and `__gt__` methods

Description

Try It Out

10.13 Add a `__str__` method to the `Sentence` class that you started in Try It Out Exercise 10.4 or 10.6.

10.14 Implement the `__getitem__` method for the `Sentence` class. This method allows you to use the index operator notation to access a word in the sentence.

10.15 Implement the `__len__` method so that you can use the `len` function to return the number of words.

10.16 Implement the `__contains__` method so that you can ask whether a word is `in` a sentence.

10.17 Implement the `__add__` method, which allows you to concatenate two sentences together. Make sure your `__add__` method returns a new instance of `Sentence`.

10.18 Design a class to represent a die. The constructor should allow you to specify how many sides the die has. Add a `roll` method that simulates rolling a die. Implement the `__str__` method to display the roll.

10.19 Implement the `__ge__`, `__gt__`, `__le__`, `__lt__`, `__eq__`, and `__ne__` methods for the `Die` class. These methods allow you to compare the values of the current rolls of two dice.

10.20 Design a class to implement a playing card. The card should remember its rank and suit. Implement the `__ge__`, `__gt__`, `__le__`, `__lt__`, `__eq__`, and `__ne__` methods for the card using the rule that the 2 is the lowest card in a suit and the ace is the highest. When comparing cards of equal rank from two different suits, use the rule that clubs are the lowest suit, followed by diamonds, hearts, and spades.

10.21 Design a class to represent a deck of cards. The `Deck` class should provide methods for shuffling and drawing.

10.22 Using the `Deck` class, implement a simple blackjack game.

10.23 Design a class to represent a bank account. Provide methods for checking the balance, and for depositing and withdrawing money. The constructor should allow you to provide a customer ID for the account. Initialize the beginning balance to zero.

10.24 Extend the bank account class in the previous exercise so that it has a `transfer` method. The `transfer` method should take an amount and an instance of an account to transfer the money to.

10.25 Design a class to represent a fraction. The constructor should accept two parameters: a numerator and a denominator. Implement

the special methods for addition, subtraction, multiplication, division, comparison, and conversion to string format.

10.26 Extend the `Fraction` class so that fractions are always simplified to their lowest terms.

10.3.5 Methods and `self`

When methods are invoked, they create local namespaces in the namespace where they are defined. The difference between methods and functions is that each local namespace also has a reference to the object for which the method was invoked. The name we have been using for this reference is `self`.

Consider again the `Planet` class described earlier. When we create a new instance (see **Session 10.10**), we add the variable `myPlanet` to the `main` namespace with a reference to the object being constructed. Now when we invoke a method such as `getName`, a local namespace is created and placed in the `main` namespace. The formal parameter, `self`, implicitly receives a reference to the object on which the method was invoked (in this case `myPlanet`). The result can be seen in **Figure 10.3**. Although we do not have to call the first formal parameter `self`, that is a common convention that we will continue to use.

```
In [1]:  1 __name__

Out[1]: '__main__'
```

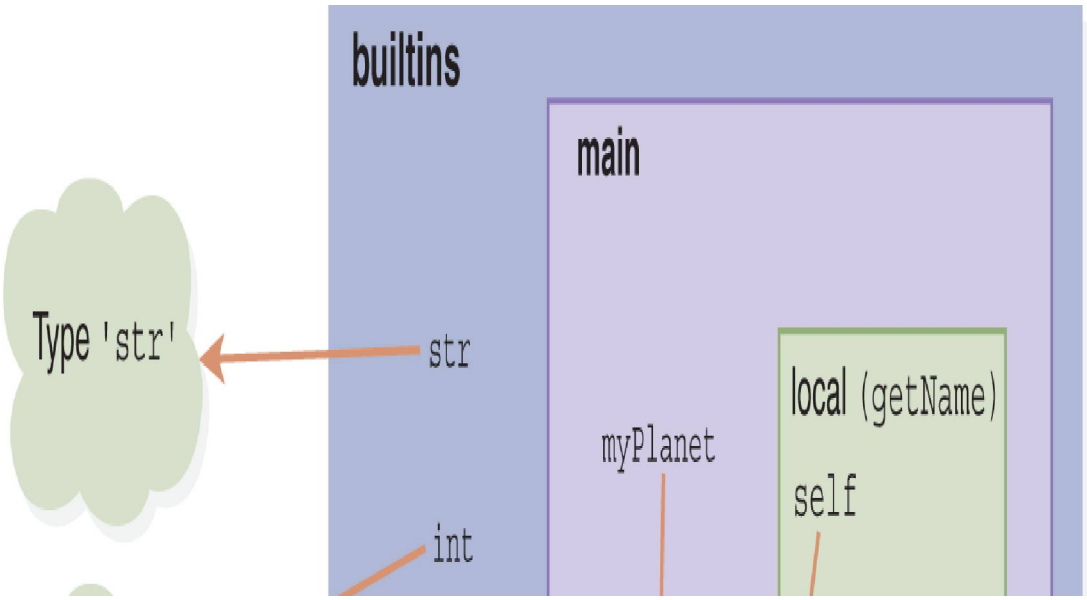
```
In [2]:  1 from planet import *
         2 myPlanet = Planet('X25', 45, 198, 1000)
```

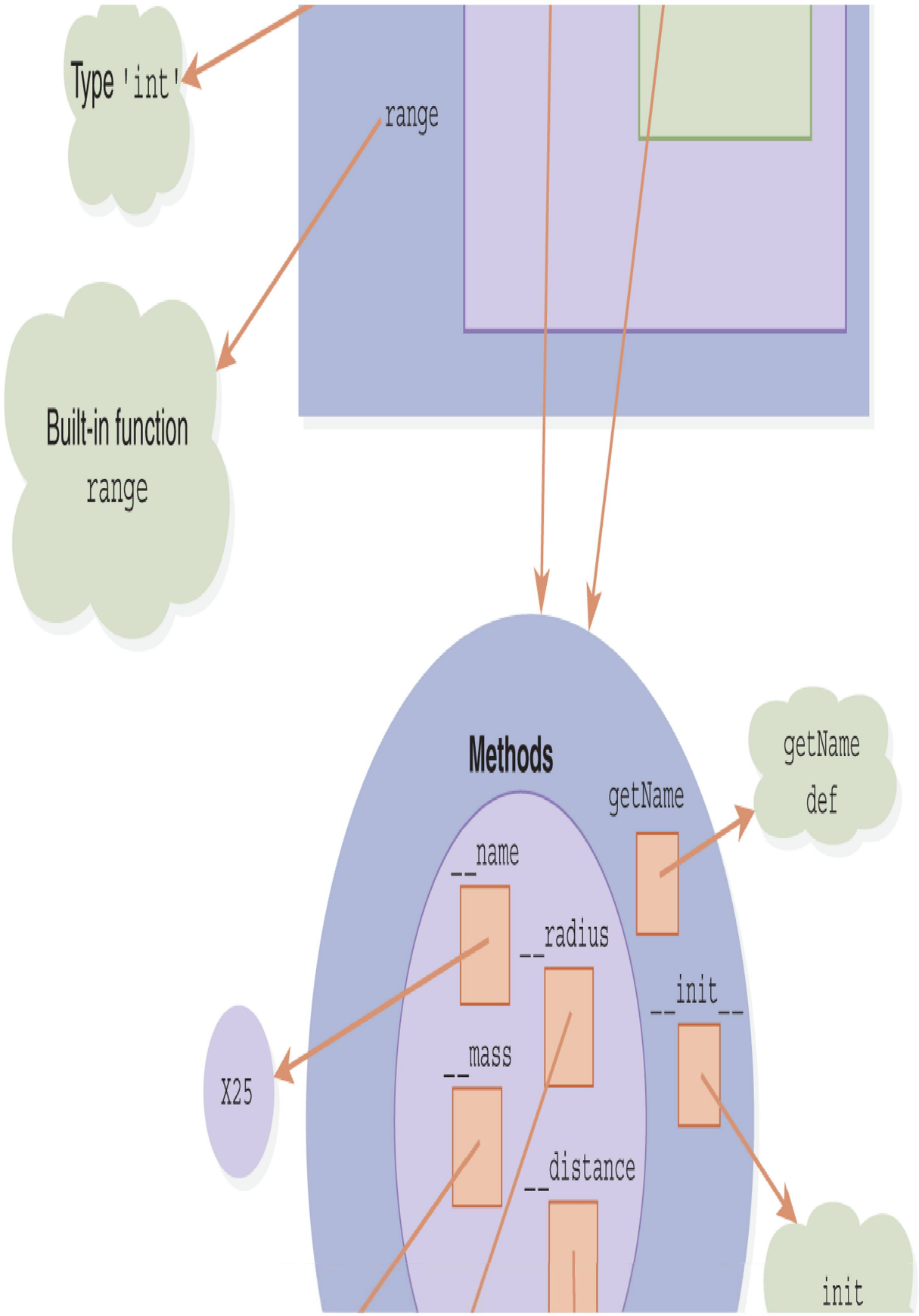
```
In [3]:  1 myPlanet.getName()

Out[3]: 'X25'
```

Session 10.10 Showing scope with methods

Description




Type 'int'
range
Built-in function
range
Methods
getName
def
getName
__name
__radius
__init__
__mass
__distance
X25
init

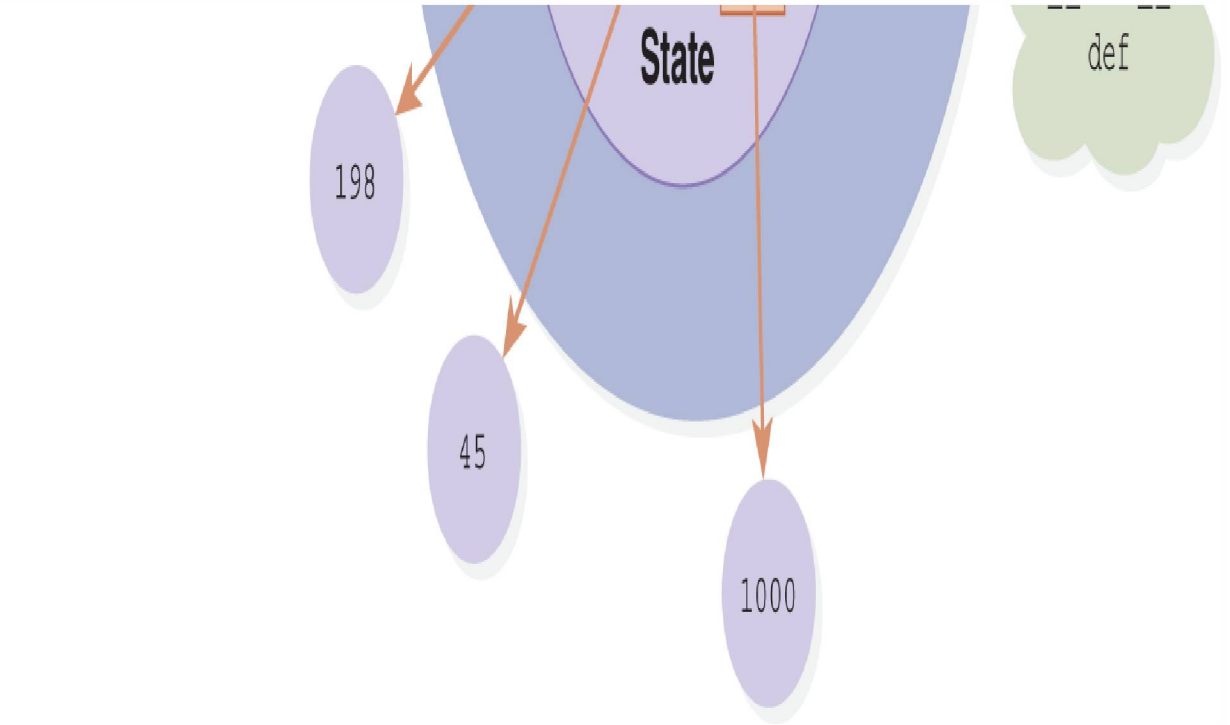


Figure 10.3 A local namespace for a method.

Description

When we look at the `getName` method, the only statement is `return self._ _name`. When Python evaluates `self._ _name`, it first evaluates `self`, which returns a reference to the object on which the method was invoked. Once we have the reference to the object, we continue to search for `_ _name` in the instance variables of the object. Because `self` refers to the `myPlanet` object, the value of the `_ _name` instance variable in `myPlanet` is returned. This process explains why each object, when invoking the `getName` method, will behave differently: These objects will likely have a different value for the `_ _name` instance variable.

Try It Out

10.27 Turtles have many instance variables, including `x`, `y`, `color`, and `heading`. Draw a reference diagram showing the namespaces

for the following session at the point immediately after the `forward` method has been invoked.

```
1 import turtle
2 myturtle = turtle.Turtle()
3 myturtle.forward(10)
```

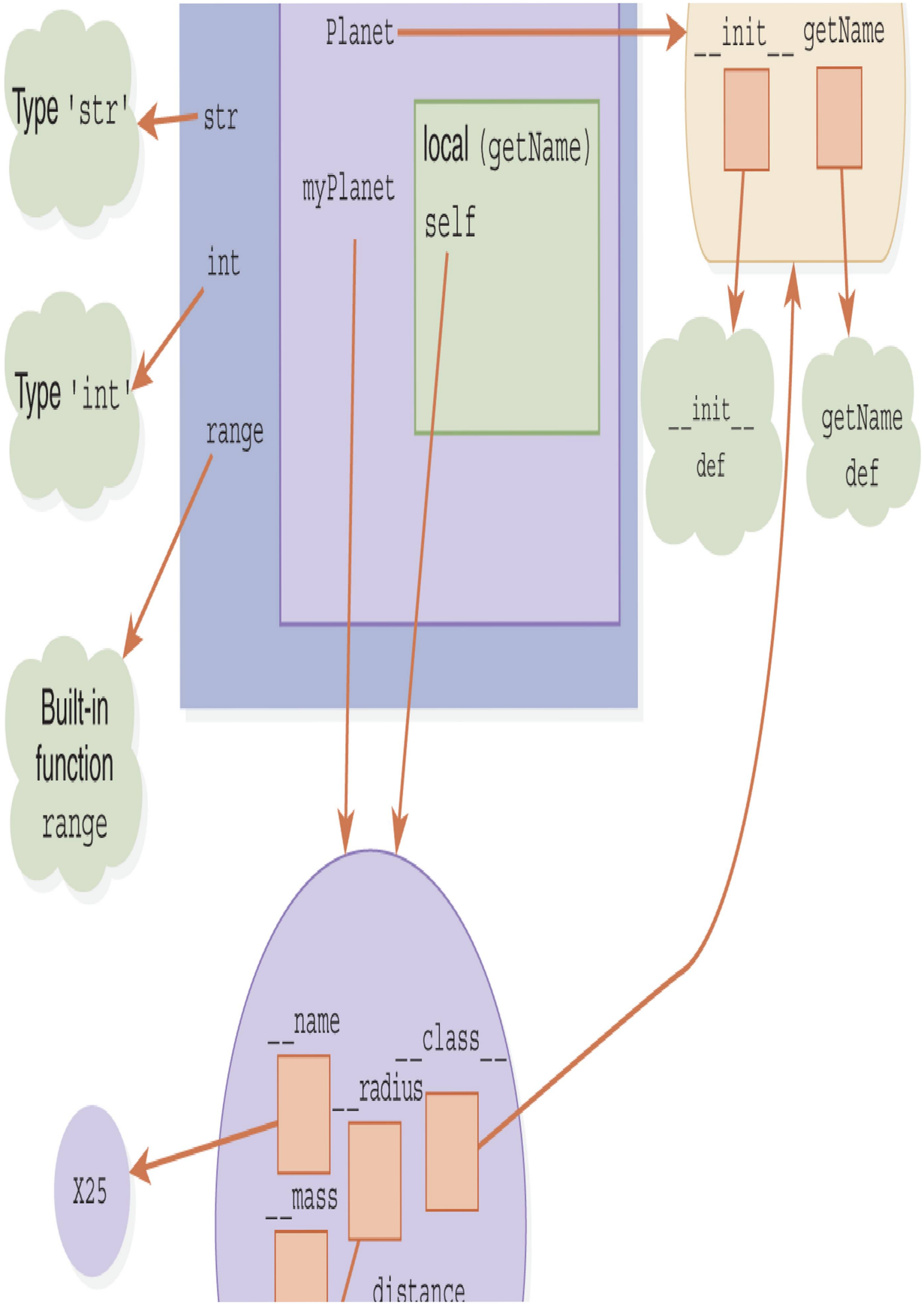
Description

10.3.6 Details of Method Storage and Lookup

Before moving on, it is important to understand the difference between the descriptive view of objects that we are using and the storage and lookup mechanism that Python uses. We have been representing objects as containing both instance variables and methods. This is an accurate representation for the instance variables, as each object needs its own copies. However, because methods are shared by all instances of a class, methods are stored in a slightly different fashion.

To see this, consider **Figure 10.4**. When a class is defined, the name of the class is added to the current namespace with a reference to a class definition object. In this case, we have added the name `Planet` to the `main` namespace with a reference to a corresponding class definition object. Class definition objects store method names that refer to method definitions that are implemented by the class. Figure 10.4 shows only two of the many methods that have been implemented by `Planet`.




Planet
__init__ getName
Type 'str'
str
local (getName)
self
myPlanet
int
Type 'int'
range
__init__
def
getName
def
Built-in
function
range
__name
__class__
__radius
__mass
X25
distance

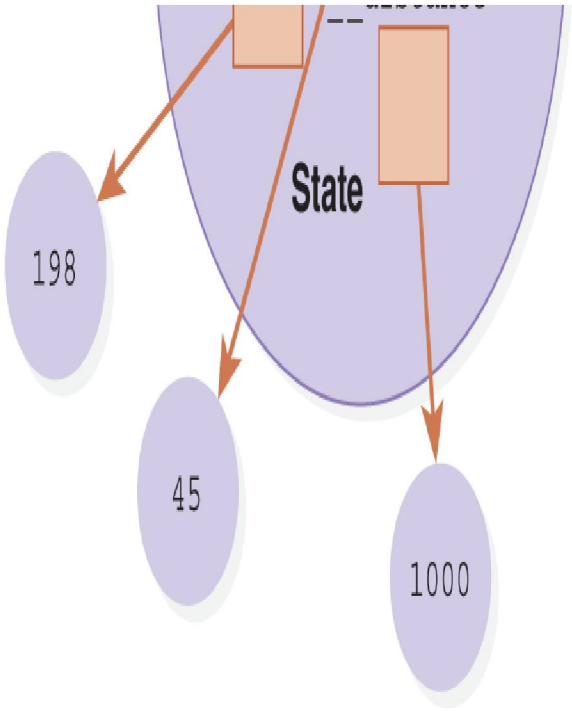


Figure 10.4 Actual storage of methods.

Description

When an instance of the `Planet` class is created, as was done using

```
myPlanet = Planet('X25', 45, 198, 1000)
```

Description

the `Planet` reference is followed to gain access to the `__init__` method definition. This constructor is executed, and an object is created with the given instance variable values. This object is referenced from the `main` namespace using the name `myPlanet`.

Figure 10.4 shows one additional variable. The `__class__` instance variable is used to provide each object with a reference to the class definition object to which it belongs. Now, when a method is called as

```
myPlanet.getName()
```

the lookup sequence can proceed as follows. First, `myPlanet` is dereferenced to gain access to the object. Next, the `__class__` reference is followed to find the class definition object. Then the method name, `getName`, is used to access the method definition. When this method is invoked, a namespace is created and placed as usual.

With this architecture, only one copy of the method definition is needed—no matter how many objects exist for a particular class. Because each object will have a `__class__` reference, the method definitions can always be located. Furthermore, because each method has a formal parameter, `self`, that references the object making the invocation, no confusion will occur when different objects call the same method. Instead, each object will get its own local namespace with an appropriate value for `self`.

10.4 Designing and Implementing a `Sun` Class

Our task in this chapter is to construct a software model of a planetary system. To do so, we need to consider not just the planets that might be present, but also the most important member of the system—the sun. We will consider the sun to be similar to a planet in that it is a large, round, celestial body. It will certainly have some of the same characteristics as planets, including a name, mass, and radius. However, because the sun is at the center of the solar system, it will not have any distance measure. In addition, the sun provides heat and light that can be characterized by the temperature on the surface.

Given that description, we can create a `Sun` class using the same pattern that we followed for the `Planet` class. Our constructor will require values for the name, mass, radius, and temperature. **Listing 10.7** shows a partial implementation of the class, which includes the `__init__`, `getMass`, and `__str__` methods. We leave development of the rest of the methods as an exercise.

```
1  class Sun:
2      def __init__(self, iName, iRad, iM, iTemp):
3          self._name = iName
4          self._radius = iRad
5          self._mass = iM
6          self._temp = iTemp
7
8      def getMass(self):
9          return self._mass
10
11     def __str__(self):
12         return self._name
```

Listing 10.7 Partial definition of the `Sun` class

Description

Try It Out

10.28 Complete the implementation of the `Sun` class by writing the following methods:

a. `getRadius`

b. `getTemperature`

c. `getVolume`

d. `getSurfaceArea`

e. `getDensity`

f. `setName`

g. `setRadius`

10.29 Write a function to print a table of the radius, volume, and surface area for a `Sun` object for values of the radius between 10 and 500, using increments of 10.

10.5 Designing and Implementing a Solar System

Now we are ready to build our solar system, which will consist of a sun and a collection of planets, with each planet defined to be some distance away from the sun. We will assume that the sun resides at the center of the solar system. We will implement the `SolarSystem` class in the same way that we have implemented the other classes seen so far: by providing a constructor that will be responsible for defining the instance variables and by defining appropriate accessor and mutator methods.

The complete `SolarSystem` class is shown in **Listing 10.8**. Our constructor (lines 2–4) assumes that a basic `SolarSystem` object must have a `Sun` object at its center and a list of planets. As a consequence, the constructor expects to receive a `Sun` object as a parameter, but it assumes an empty collection of planets.

```
1  class SolarSystem:
2      def __init__(self, aSun):
3          self.__theSun = aSun
4          self.__planets = []
5
6      def addPlanet(self, aPlanet):
7          self.__planets.append(aPlanet)
8
9      def showPlanets(self):
10         for aPlanet in self.__planets:
11             print(aPlanet)
```

Listing 10.8 The `SolarSystem` class

Description

To add a `Planet` to the `SolarSystem`, we use a mutator method called `addPlanet` that can modify the collection of planets. This method (lines 6–7) receives a `Planet` object as a parameter and adds the object to the collection of planets. Because the collection is a list, we simply use the `append` method.

Finally, the simple accessor method called `showPlanets` (lines 9–11) shows all the planets in the solar system. It iterates through the list of planets and prints each one in turn. Recall that the `Planet` class implements the `__str__` method that returns the name of the planet for use in the `print` method.

To use the three classes we have implemented, we can save them as Python files. The classes `Sun`, `Planet`, and `SolarSystem` will be saved in the files `sun.py`, `planet.py`, and `solarsystem.py`, respectively. When we store a class in a file in this way, we have created a module that can be used by other programs—a module that is accessed and used in much the same way as the `turtle`, `math`, and `cImage` modules. To use our `Planet`, `Sun`, or `SolarSystem` classes, we simply import the module that contains the class definition.

Session 10.11 shows the statements that import the modules, and it creates a solar system with four planets. After importing the modules, we create a `Sun` object called `sun` and a `SolarSystem` object called `ss` using `sun` as its center. Next, we create a new `Planet` object `p` with a planet name of `Mercury` and add it as the first planet in the solar system. We then create and add three other planets in the same way. For simplicity, we reuse the object name `p`. Finally, we call the `showPlanets` method, which prints the names of the planets in our solar system. **Figure 10.5** shows all of the objects and references present at this point.

```
In [1]:  1 from sun import *
         2 from planet import *
         3 from solarSystem import *
```

```
In [2]:  1 sun = Sun('Sun', 5000, 1000, 5800)
         2 ss = SolarSystem(sun)
```

```
In [3]:  1 p = Planet('Mercury', 19, 10, 25)
         2 ss.addPlanet(p)
```

```
2  ss.addPlanet(p)
```

```
In [4]:  1  p = Planet('Earth', 50, 60, 30)
         2  ss.addPlanet(p)
```

```
In [5]:  1  p = Planet('Mars', 47, 50, 35)
         2  ss.addPlanet(p)
```

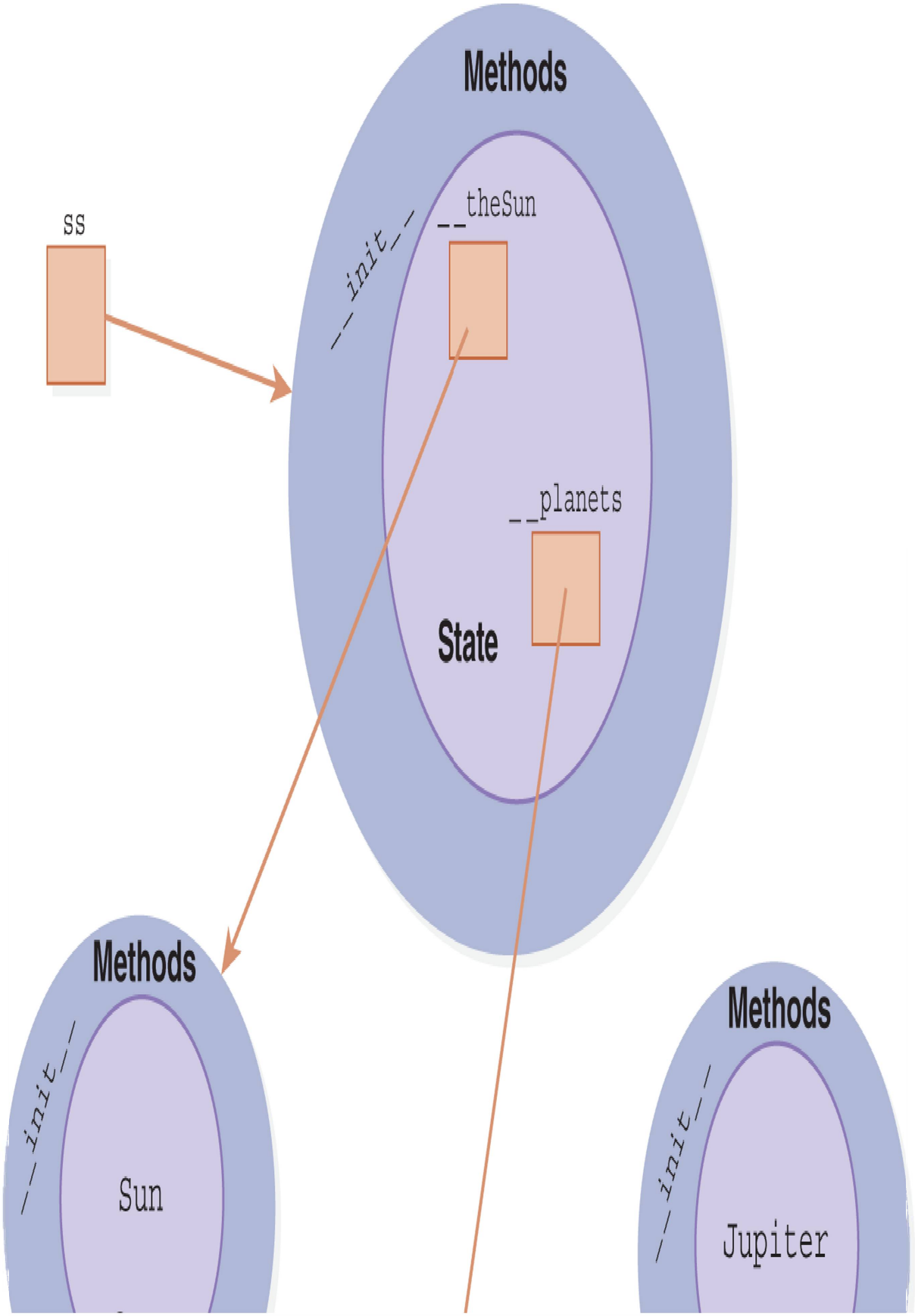
```
In [6]:  1  p = Planet('Jupiter', 75, 100, 50)
         2  ss.addPlanet(p)
```

```
In [7]:  1  ss.showPlanets()
```

```
Mercury
Earth
Mars
Jupiter
```

Session 10.11 Creating and showing a solar system with a sun and four planets

Description


Methods
__init__
__theSun
ss
__planets
State
Methods
__init__
Sun
Methods
__init__
Jupiter

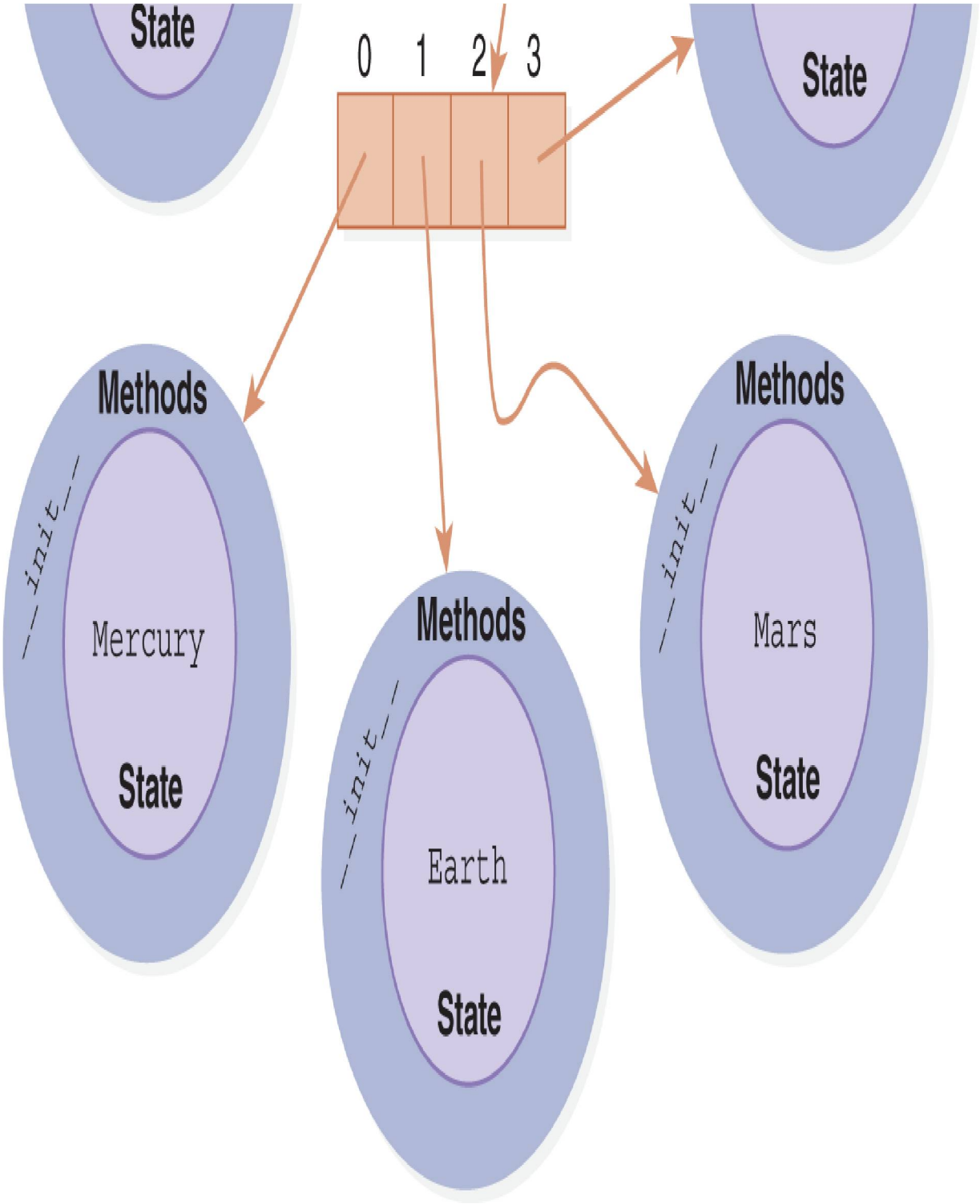


Figure 10.5 A solar system with a sun and four planets.

Description

Try It Out

10.30 Write a method called `numPlanets` that returns the number of planets in the solar system.

10.31 Write a method called `totalMass` that returns the total mass of the solar system. The total mass should include all the planets and the sun.

10.32 Given Pluto's demotion from a planet to a dwarf planet, write a `removePlanet` method for the `SolarSystem` class that you could use to remove a demoted planet.

10.33 Write two methods, called `getNearest` and `getFarthest`, that will return the planet closest or farthest from the sun, respectively.

10.34 Write a `__str__` method that prints the name of the sun and the names of all the planets, in order, from closest to farthest from the sun.

10.6 Animating the Solar System

Now that we can create a solar system, it is a natural next step to consider whether we can draw the solar system and then place the planets in motion. Thus, we will be using our software model to create a **simulation** of the solar system. To do that, we construct an animation based on the laws of motion discovered by Sir Isaac Newton. Although we will not attempt to handle every possible variable in nature's complex interaction of planetary objects, the resulting simulation will still rely on the basic laws of physics.

Understanding planetary movement requires an understanding of Newton's basic laws of motion. In particular, we must understand how objects that are in motion interact with one another and how those interactions affect their movements. Because this is not a course in physics, we will resist the temptation to describe all of the derivations required. It turns out that by using a few simple equations, we can create a fairly accurate simulation.

10.6.1 Using Turtles

To draw our `SolarSystem`, `Sun`, and `Planet` objects, we will use the `Turtle` class. Recall that a `Turtle` is a graphical object that has instance variables, including position and color. A turtle can define a coordinate system; it can move to any position within that coordinate window; and if its tail is in the down position, it can draw a line. In this application, we will also take advantage of the `turtle` module's feature that allows many turtles to exist simultaneously in the same window. We will exploit this capability to provide each planet with a `Turtle` object that graphically represents the planet.

Our first modification involves the `SolarSystem` class. The solar system represents the bounds of our animation, so we will use a `Turtle` to provide a user-defined planetary coordinate system. When a solar system is created, we will provide a width and a height that can, in turn, be used to define the upper and lower bounds on the *x*- and *y*-axes within that coordinate system.

Listing 10.9 shows the new `SolarSystem` class. The constructor now has two parameters: `width` and `height`. The sun is no longer a parameter, however—it will be added later. Instead, the instance variable `_theSun` is initially set to `None`. By doing things in this order, we allow the planets and sun to take advantage of the graphics setup work to be done by the `SolarSystem`.

```
1  class SolarSystem:
2      def __init__(self, width, height):
3          self._theSun = None
4          self._planets = []
5          self._ssTurtle = turtle.Turtle()
6          self._ssTurtle.hideturtle()
7          self._ssScreen = turtle.Screen()
8          self._ssScreen.setworldcoordinates( -width/2.0, -height/2.0,
9                                              width/2.0,  height/2.0)
10
11     def addPlanet(self, aPlanet):
12         self._planets.append(aPlanet)
13
14     def addSun(self, aSun):
15         self._theSun = aSun
16
```

```
17      def showPlanets(self):
18          for aPlanet in self._planets:
19              print(aPlanet)
20
21      def freeze(self):
22          self._ssScreen.exitonclick()
```

Listing 10.9 The `SolarSystem` class with a hidden turtle

Description

Lines 5–9 create instance variables called `_ _ssTurtle` and `_ _ssScreen` that will provide graphical functionality to the solar system. Because the turtle will not actually draw anything, we hide it. Using the `width` and `height` as parameters, we call the `setworldcoordinates` method to create a coordinate system that is equally distributed around the position (0, 0).

Two additional methods have been added to the `SolarSystem` class. Lines 14–15 define a method called `addSun` that allows us to add a sun to the solar system. The `freeze` method (lines 21–22) simply lets us "freeze" the screen after the animation has finished, which keeps the window from automatically closing when the animation ends.

We next modify the `Sun` class (shown in **Listing 10.10**) by creating two new instance variables, `self._ _x` and `self._ _y`, to keep track of the coordinate position for the sun. Because we assume that the sun is at the center of the solar system, initializing these values to zero provides a good starting point.

```
1  class Sun:
```

```
    def __init__(self, iName, iRad, iM, iTemp):
        self.__name = iName
        self.__radius = iRad
        self.__mass = iM
        self.__temp = iTemp

        self.__x = 0
        self.__y = 0
        self.__sTurtle = turtle.Turtle()
        self.__sTurtle.shape('circle')
        self.__sTurtle.color('yellow')

    def getXPos(self):
        return self.__x

    def getYPos(self):
        return self.__y

#other methods as before
```

Listing 10.10 The `Sun` class with visualization

Description

By adding `self._ _sTurtle` as a third instance variable of the `Sun` class, we ensure that each `Sun` object will contain a `Turtle` object to do the graphical work. We define the color of the `Turtle` to be `"yellow"`. In addition, we change the shape of the turtle from the default triangle to a circle. The majority of the `Sun` class methods remain unchanged in this new implementation. Two new accessor methods, `getXPos` and `getYPos`, allow us to retrieve the `x` and `y` coordinates of the sun.

Finally, we modify the `Planet` class to include a `Turtle` (see **Listing 10.11**). Like the instances of the `Sun` class, each planet will keep track of its position using `x` and `y` coordinate values. The initial `x` value will be the distance from the sun, whereas the initial `y` value will be zero. This means that all planets are initially lined up on the *x*-axis. We use the color of the turtle to distinguish each `Planet`. When a `Planet` is created, a color is provided through the constructor. The shape of the turtle is also set to `circle`.

```
1  class Planet:
2      def __init__(self, iName, iRad, iM, iDist, iC):
3          self.__name = iName
4          self.__radius = iRad
5          self.__mass = iM
6          self.__distance = iDist
7
8          self.__x = self.__distance
9          self.__y = 0
10         self.__color = iC
```

```
10          self.__color = ic
11          self._pTurtle = turtle.Turtle()
12          self._pTurtle.color(self._color)
13          self._pTurtle.shape("circle")
14          self._pTurtle.up()
15          self._pTurtle.goto(self._x, self._y)
16          self._pTurtle.down()
17
18      def getXPos(self):
19          return self._x
20
21      def getYPos(self):
22          return self._y
23
24      #other methods as before
```

Listing 10.11 The `Planet` class with visualization

Description

Lines 14–16 move the planet into its initial position. Before invoking the `goto` method, we raise the turtle's tail so that it will not leave a line. Of course, we need to lower the tail again once the planet arrives at its destination.

Session 10.12 shows how these classes work together to create an initial visual representation of a solar system. First, an instance of `SolarSystem` is created with a width and height of two units. Next, a `Sun` object is created and added to the solar system. Finally, four planets with different names, colors, and distances from the sun are added. Because the sun and planets have associated `Turtles` in their constructors, they are automatically shown when the objects are created. The resulting image is shown in **Figure 10.6**.

```
In [1]:
1 from solarSystem import *
2 from planet import *
3 from sun import *
```

```
In [2]:
1 ss = SolarSystem(2, 2)
```

```
In [5]:
1 sun = Sun('Sun', 5000, 10, 5800)
2 ss.addSun(sun)
```

```
In [7]:
1 m = Planet('Mercury', 19.5, 1000, 0.25, 'blue')
2 ss.addPlanet(m)
```

```
In [8]:
1 m = Planet('Earth', 47.5, 5000, 0.3, 'green')
```

```
2 ss.addPlanet(m)
```

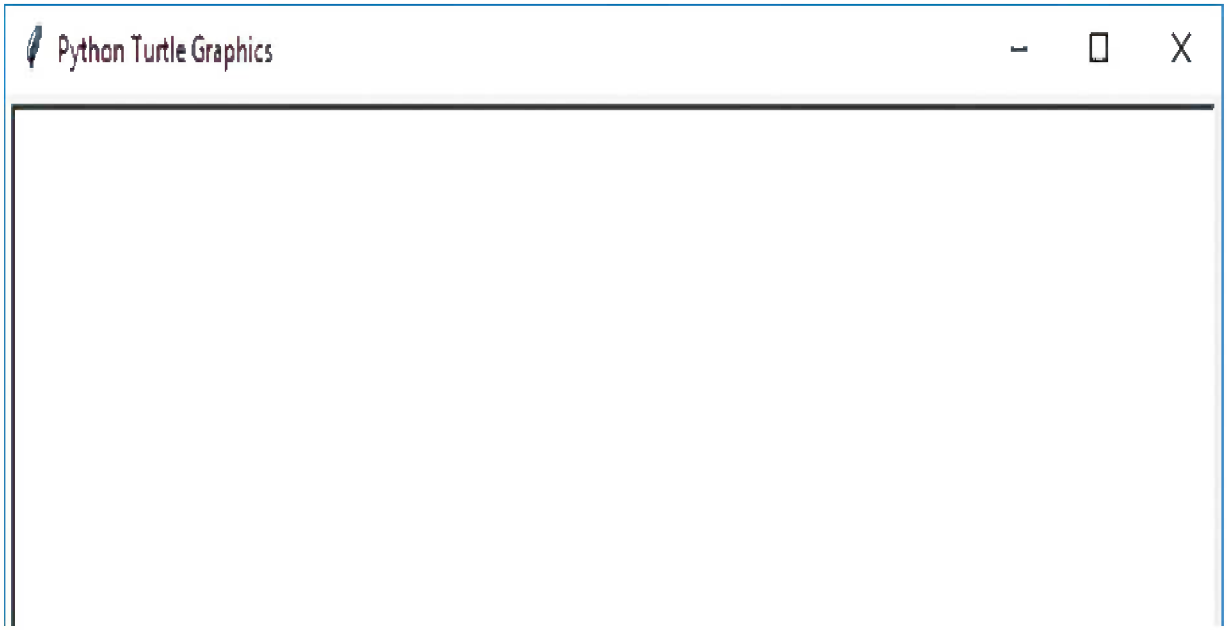
```
In [9]:  1 m = Planet('Mars', 50, 9000, 0.5, 'red')
         2 ss.addPlanet(m)
```

```
In [10]: 1 m = Planet('Jupiter', 100, 49000, 0.7, 'black')
         2 ss.addPlanet(m)
```

```
In [11]: 1 ss.freeze()
```

Session 10.12 Creating and showing a sun and four planets

Description



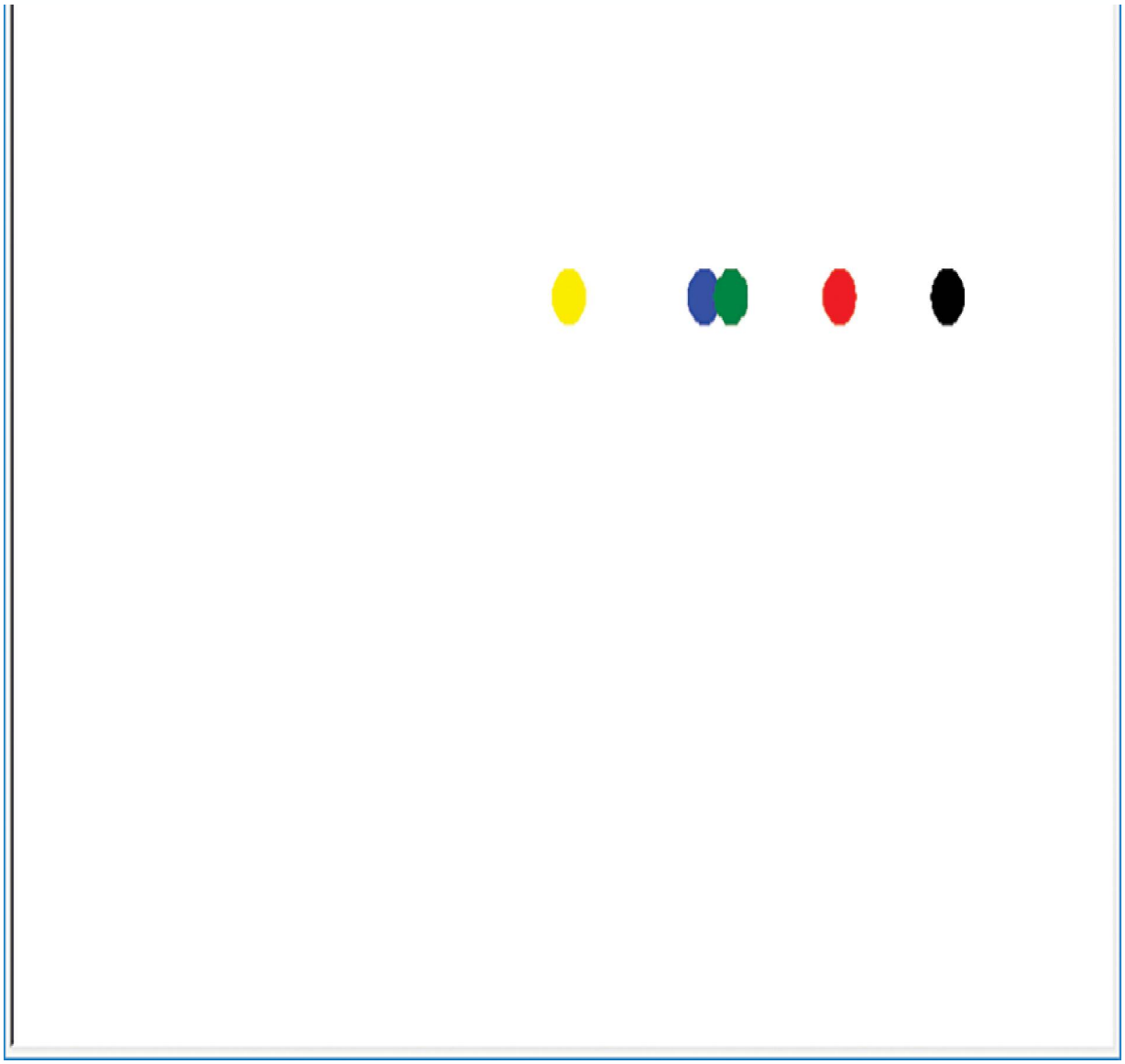


Figure 10.6 Visualizing the sun and planets.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

Try It Out

10.35 Load and run the code shown in Session 10.12.

10.36 Add a planet called Saturn to the solar system created in the previous exercise.

10.37 Modify the `Planet` class so that each planet can be a different size. *Hint:* You will want to look up the `resizemode` and `shapesize` methods in the `turtle` documentation.

10.6.2 Planetary Orbits

Planetary orbits are shaped by a number of variables and related equations. We begin by defining some of the necessary vocabulary and then move on to describe the equations that will be used to calculate the actual orbits. To simplify our work, we will continue to assume that our universe exists in only two dimensions. As we go through the explanations, however, notice that extending our system to three dimensions would not be difficult.

Distance Between Objects

To start, assume that a planet is located at some position in the solar system—call it (x, y). The behavior of the planet depends on its distance from other objects in the solar system. As we are working in two dimensions, we will compute the distance using the notion of Euclidean distance.

To review, for an object located at position (a, b), the distance between it and the planet just described would be $\sqrt{(x-a)^2+(y-b)^2}$. If we assume that the sun is at position (0, 0), then the distance between any planet and the sun is simply $\sqrt{x^2 \quad y^2}$. This distance is often referred to as `r`. For planet interaction, the closer two objects are to each other (the smaller the `r`), the more they interact. Thus, two objects that are separated by a large distance will have very little interaction.

Velocity

Our planet is not stationary in the solar system, but rather is continually moving. The basic description of how it moves is given by its velocity. Velocity is defined as the change in distance divided by the change in time. For example, if we are walking in a straight line with a constant velocity of 5 km per hour, we can say that 1 hour from now we will be 5 km from our current location, after 2 hours we will be 10 km from here, and so on.

Velocity in two dimensions is slightly more complex to compute. **Figure 10.7** shows that velocity, *V*, is actually a combination of two components. The *x*-component, called *Vx*, is the change in the *x* direction for that given amount of time. Similarly, the *y*-component, called *Vy*, is the change in the *y* direction. This idea, known as a vector in physics, will be important when we calculate planetary movement in two dimensions.

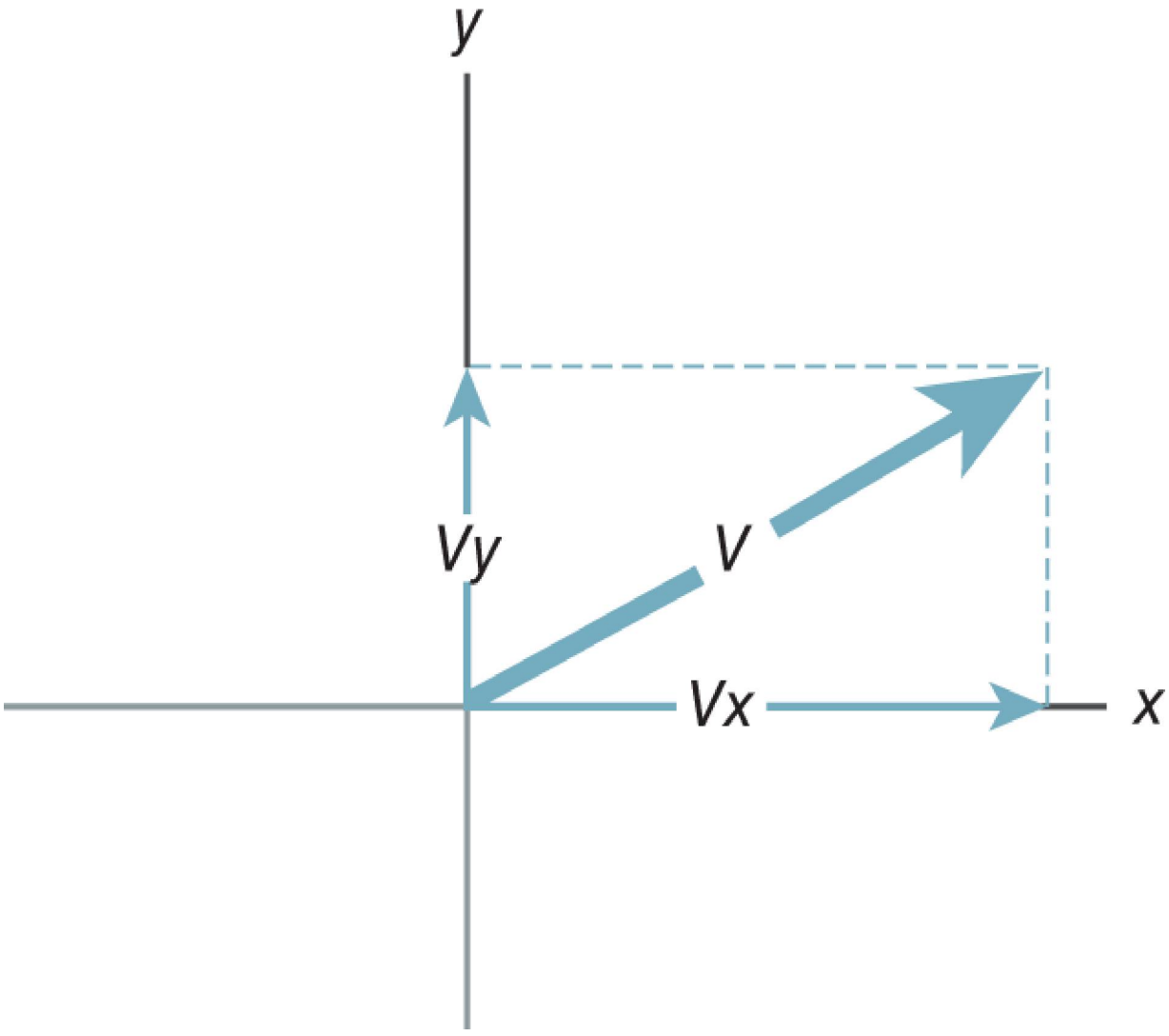


Figure 10.7 Understanding velocity in two dimensions.

Description

Acceleration

The velocity of our planet will not be constant; instead, it will change over some period of time. This change is known as acceleration. Like velocity, acceleration in two dimensions is defined by a vector with two components, *Ax* and *Ay*. Acceleration can cause an object to change direction by altering one or both of the velocity components.

In this example, the acceleration of the planet will change due to interactions with other objects in the solar system. This will, in turn, cause the velocity to change, thereby moving the planet to a new position in the two-dimensional space. We describe these interactions in a later section.

Calculating Distance from Velocity and Velocity from Acceleration

The two previous sections provide relationships for velocity and acceleration that use time in the denominator. If we know that a certain amount of time has elapsed, then a bit of algebra allows us to calculate important values related to the movement of an object. For example, because velocity is the distance traveled divided by the time, it follows that the distance traveled can be computed from the velocity and the time:

$$\left(v = \frac{d}{t} \text{ becomes } d = v \cdot t\right)$$

If an object is traveling at a certain velocity for a period of time, we can use this equation to determine how far it has moved.

We noted earlier that the velocity is actually a vector with both an *x*-component and a *y*-component. As a consequence, the calculation for distance must be carried out twice, once for each component. The distance moved in the *x* direction, *Dx*, is the *x*-component of the velocity, *Vx*, multiplied by the time, and likewise for the movement in the *y* direction.

Because acceleration is the change in velocity over a given period of time,

$$a = \frac{v}{t}$$

the same analysis can be done to compute the velocity components: $Vx = Ax \cdot t$ and $Vy = Ay \cdot t$.

Mass and Gravity

As mentioned earlier, the distance between two objects affects the amount of interaction that occurs between them. Other factors also need to be considered when assessing this relationship. First, the mass of the planet will dictate how much effect it has on other objects in its vicinity. A planet with a large mass will exert more force on other planets of smaller mass at the same distance.

We are all familiar with gravity—the "pull" that keeps us on Earth. Gravity is what causes us to come back down if we jump in the air. In general, gravity is a force that massive objects exert on other objects. The gravity that Earth exerts on a person standing on the planet's surface is what we usually consider to be our weight. Because objects with more mass will be "pulled" to a greater extent given the same gravity, weight and mass can be thought of as similar.

Interaction of the Planets

To write a program that will plot the path of planets, we need to use some basic equations that compute the effect of planet interaction. The first important equation, known as Newton's law of universal gravitation, states that the force or attraction, F, between two objects can be computed as

$$F = G\frac{m_1 m_2}{r^2}$$

where G is a constant, m_1 and m_2 are the masses of the two objects, and r is the distance between the two objects. G is often referred to as the gravitational constant.

Newton's second law of motion is $F = ma$, where F is the force exerted by an object, m is its mass, and a is its acceleration. By setting these two equations equal to each other and solving for acceleration, we can calculate the acceleration of one object as a result of the force exerted on it by another object,

$$a_1 = G\frac{m_2}{r^2}$$

In other words, the acceleration experienced by the first object due to the force of gravity from the second object can be calculated by multiplying the

gravitational constant by the mass of the second object and dividing by the square of the distance between the two objects.

Figure 10.8 can be used to compute the acceleration components for a planet that is interacting with the sun. Notice the two similar triangles. The large triangle has sides x, y, and r. The smaller triangle, representing the force that is "pulling" on the planet, has sides Fx, Fy, and F. Since these two triangles are similar triangles, the ratios of their corresponding sides must be equal. From this relationship, we can derive the following sequence of equations for the x-component of acceleration:

1. $\frac{Fx}{F} = \frac{x}{r}$
2. Multiplying by F gives $Fx = \frac{F \cdot x}{r}$
3. $Fx = \frac{G \cdot m_s \cdot m_p \cdot x}{r}$ since $F = \frac{G \cdot m_1 m_2}{r^2}$
4. $Ax = \frac{G \cdot m_s \cdot x}{r}$ since $Fx = m_p \cdot Ax$

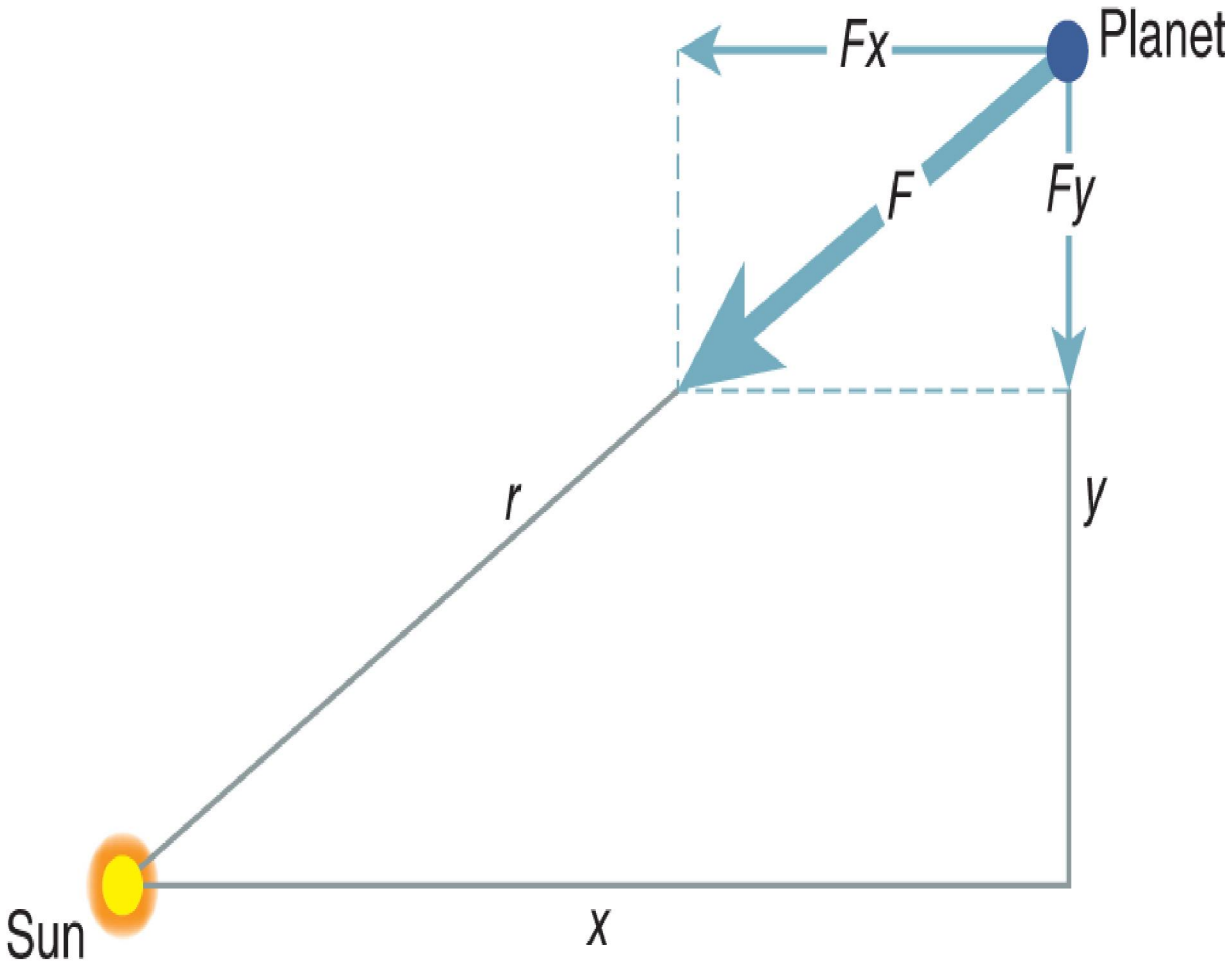


Figure 10.8 Using similar triangles to compute the acceleration components.

Description

A similar derivation can be used to obtain the *y*-component.

10.6.3 Implementing the Simulation

It is now time to implement these equations so that we can compute the positions of the planets as they revolve around the sun. The basic algorithm calculates the planet's movement and change in velocity over a fixed period of time. Each time the planet moves, the gravitational force changes, because the distance between it and the sun will have changed. The change in force will produce a new velocity, which in turn will cause the next change in

distance. These steps will be repeated for a specified number of time periods.

For each time step, do the following (assume that the value of the time period and the value of the gravitational constant have been set):

1. Using the current velocity and elapsed time, move the planet to a new location by computing the *x* distance and the *y* distance from the velocity components.
2. Compute the new distance from the planet to the sun.
3. Use Newton's gravitational equations to compute the new acceleration components for the planet.
4. Use this new acceleration to adjust the velocity components.

To implement these steps, we need to modify our `Planet` class so that each planet can remember the additional information. Each planet will now need to know its current velocity and will need to be given an initial velocity as a parameter to the constructor. Because velocity is a vector, we will track the *x*-component and the *y*-component separately. **Listing 10.12** shows the needed modifications to the `Planet` class. The only parameters that have been added are for the two velocity components.

```
1 class Planet:
2     def __init__(self, iName, iRad, iM, iDist, iVx, iVy, iC):
3
4         #other instance variables as before
5
6         self._velX = iVx
7         self._velY = iVy
```

Listing 10.12 The `Planet` class with animation

Description

We have added some velocity attributes, so we will need to provide corresponding accessor and mutator methods for those attributes (see **Listing 10.13**). Moreover, we have added a `moveTo` method that will modify the *x* and *y* location of the planet; it moves the turtle to that location, leaving a line behind (tracing the orbit).

```
def moveTo(self, newX, newY):
    self.__x = newX
    self.__y = newY
    self.__pTurtle.goto(self.__x, self.__y)

def getXVel(self):
    return self.__velX

def getYVel(self):
    return self.__velY

def setXVel(self, newVelX):
    self.__velX = newVelX

def setYVel(self, newVelY):
    self.__velY = newVelY
```

Listing 10.13 Animation methods in the `Planet` class

Description

Most of the animation will occur within the `SolarSystem` class. Recall that a `SolarSystem` consists of a sun and a collection of planets. Each

instance of `Planet` now has the information necessary to allow the `SolarSystem` object to compute its movement and position according to the equations described earlier. **Listing 10.14** shows the method, called `movePlanets`, that moves each planet in the solar system in one time increment. Each time this method is called, all the planets move to their next position.

```
17  def movePlanets(self):
18      G = .1
19      dt = .001
20
21      for p in self._planets:
22          p.moveto(p.getXPos() + dt * p.getXVel(),
23                   p.getYPos() + dt * p.getYVel())
24
25          rX = self._theSun.getXPos() - p.getXPos()
26          rY = self._theSun.getYPos() - p.getYPos()
27          r = math.sqrt(rX**2 + rY**2)
28
29          accX = G * self._theSun.getMass() * rX/r**3
30          accY = G * self._theSun.getMass() * rY/r**3
31
32          p.setXVel(p.getXVel() + dt * accX)
33          p.setYVel(p.getYVel() + dt * accY)
```

Listing 10.14 Animation method in the `SolarSystem` class

Description

Let's take a look at each line of this method so that we can understand how the equations have been implemented in Python. First, lines 18–19 set the two important constants needed to run the animation. `G` is Newton's gravitational constant. For simplicity in our example, we will use 0.1 as its value, but in the exercises at the end of the section, you will explore using the actual value. The `dt` constant represents the time step; in this example, we will compute the movement that takes place in 0.001 second.

Line 21 sets up the iteration that will process each planet in the solar system. The steps to compute the movement for each planet have already been described. Lines 22–23 move the planet based on its current velocity. Note that the new x location is computed from the x-component of the velocity, and the new y location is computed from the y-component. Once the planet has been moved to a new location, we then need to recompute the distance between the planet and the sun (lines 25–27).

Lines 29–30 compute the new acceleration components based on this new distance. The acceleration is not stored as part of the `Planet` object, but rather is computed and then used to modify the velocity (lines 32–33). The two velocity components are set individually based on the old velocity and the change due to the acceleration. If necessary, review the earlier equations so that you understand each calculation.

Now we can finish the animation by creating a solar system and allowing it to move the planets. **Listing 10.15** shows a function that creates a simple solar system with a sun and four planets. Although we have used the names of real planets, the values for mass, distance, and initial velocities are not accurate. However, they will work nicely to create a stable system of planetary orbits.

```
1  def createSSandAnimate():
2      ss = SolarSystem(2, 2)
3
4      sun = Sun('Sun', 5000, 10, 5800)
5      ss.addSun(sun)
```

```
6
7   m = Planet('Mercury', 19.5, 1000, 0.25, 0, 2, 'blue')
8   ss.addPlanet(m)
9
10  m = Planet('Earth', 47.5, 5000, 0.3, 0, 2.0, 'green')
11  ss.addPlanet(m)
12
13  m = Planet('Mars', 50, 9000, 0.5, 0, 1.63, 'red')
14  ss.addPlanet(m)
15
16  m = Planet('Jupiter', 100, 49000, 0.7, 0, 1, 'black')
17  ss.addPlanet(m)
18
19  numTimePeriods = 2000
20  for aMove in range(numTimePeriods):
21      ss.movePlanets()
22
23  ss.freeze()
```

Listing 10.15 Creating and moving a solar system with four planets

Description

As each planet is created, it is given an initial velocity, always in the *y*-component, and then added to the solar system. The real work of the animation is done by lines 19–21. The `numTimePeriods` constant sets the number of positions that will be calculated for each planet. Recall that each time period represents .001 of a second in this example.

Session 10.13 shows the code to run the animation. This assumes that all the classes are stored in the file `animateSS.py` and that the file also contains imports for the `math` and `turtle` modules. **Figure 10.9** displays the result. Each planet travels through an elliptical orbit, and the shape of that orbit is dictated by the planet's starting position, mass, and initial velocity. The time interval is sufficient to allow the inner two planets to make complete orbits, but the outer two planets would need more time to complete a single orbit. Note that this animation model takes into account only the interaction between a planet and the sun; it does not model the interactions of planets with each other. In reality, the accelerations must be computed by taking all of the other planets into account. We leave this modification as an exercise.

```
1 from animateSS import *
2 createSSandAnimate()
```

Session 10.13 Running the animation

Description



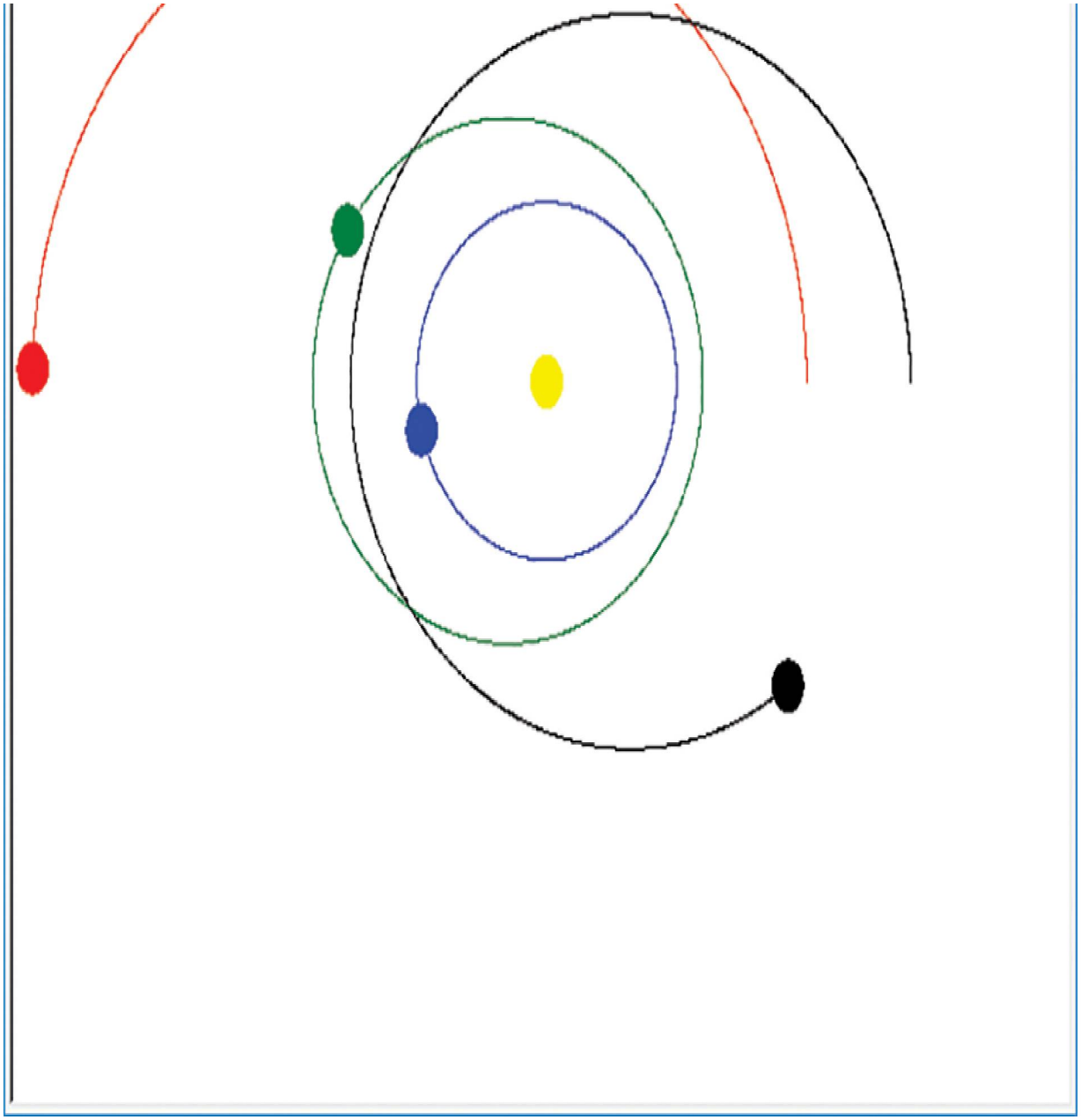


Figure 10.9 Running the solar system animation for 2,000 time units.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

Try It Out

10.38 Load the `Sun`, `Planet`, and `SolarSystem` classes, as well as the `createSSandAnimate` function. Run the solar system simulation.

10.39 Modify the `numTimePeriods` variable of `createSSandAnimate` to make the simulation run longer.

10.40 Modify the initial velocity of one planet, and observe the change in its orbit.

10.41 Modify the mass of the sun, and observe the change in the orbit of each planet. Make sure you try increasing and decreasing the sun's mass.

10.42 Create a new planet, and add it to the solar system. *Hint:* Make sure your planet is located inside the initial boundaries of the solar system, or change the initial boundaries.

10.43 The real value for Newton's gravitational constant is `G` = 6.673e-11. Using this value for `G` and a time step of 1800, look up the real values for the mass, distance, and velocity of the sun and one or more planets, and create a real model of the solar system.

10.44 Modify the `movePlanets` method so that planets interact with each other. *Hint:* The `accX` and `accY` variables will accumulate for the sun and all other planets. Try running the initial set of planets with this improved model. You may need to change their masses or initial velocities.

10.45 Add a moon to the earth or to any of the other planets.

10.7 Summary

In this chapter, we explored the implementation of our own data types by creating classes. The programmer can take advantage of this ability to create solutions using a data model that closely resembles the problem domain. Classes provide a description of objects (instances of classes) by defining data and methods. Objects are created by calling the constructor method.

Other methods, both accessors and mutators, allow the programmer to communicate with individual objects. We also introduced a number of special methods defined by Python, which in some cases are bound to operators. Finally, we used the `Turtle` to implement a graphical simulation of planetary movement.

Key Terms

accessor methods
class
constructor
data model
dunder methods
encapsulation
getter methods
information hiding
instance
instance data
instance variables
magic methods
methods
mutator methods
object
object state
object-oriented programming
setter methods
simulation
software model

Programming Exercises

10.1 Modify the `SolarSystem` class to support more than one sun.

10.2 Modify the `Planet` class so that the planet does not have the instance variables for `x` and `y` but simply uses the `x` and `y` location stored in the turtle.

10.3 Research *n*-body simulations. Using the ideas from this chapter, implement your own simple *n*-body simulation.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 11
Simulation
Computer Simulation Using Objects

11.1 OBJECTIVES

- To explore classes and objects further
- To design a large multiclass application
- To implement a graphical simulation using objects

11.2 Bears and Fish

People have been visiting Yellowstone National Park for many years hoping to see the Old Faithful geyser, the Yellowstone River waterfalls, and, of course, the bears.

For years, it was common to see bears along roads and in park campgrounds. Unfortunately, many tourists failed to realize that these great life-forms were actually wild and decidedly dangerous. Even with laws to prevent curious humans from feeding the bears, many of the bears had to be relocated. The hope was to reacquaint bears with their natural food sources and, at the same time, to reduce the number of injuries to humans.

One important food source for bears is fish. Bears depend on fish to provide a staple part of their diet, especially in the spring and summer when bears are leaving hibernation and beginning to care for young cubs. It turns out, however, that the relationship between bears and fish can be much more complicated.

Consider what happens if the number of bears decreases too quickly. Because there are now fewer bears, fish populations can increase. More fish, however, will place a strain on the plants that they feed on; these plants in turn provide oxygen for other water-dwelling life-forms.

Changing the population of one life-form can have drastic effects on others. In this chapter, we explore ways that computers can help us understand this potential impact.

11.3 What Is a Simulation?

A **computer simulation** is a computer program that is designed to model some specific aspects of a real situation or system. Computer simulations will often utilize random numbers as a way of introducing some realistic variability into the underlying model. The results can then be used to gain information about the way a real system behaves. The complexity of the computer simulation depends on the complexity of the underlying reality as well as the specific characteristics we want to represent.

One of the most common types of simulation models a real-world relationship, typically between two or more life-forms that must coexist and depend on one another in some fashion. These relationships, often referred to as **predator–prey relationships**, suggest that one life-form (the predator) hunts another (the prey) to survive. As the number of predators increases or decreases, it may cause the number of prey to increase or decrease. Some examples that have been widely studied are foxes and rabbits, big fish and little fish, ladybugs and aphids, and whales and plankton. In each of these cases, one life-form consumes another, and the existence of one depends on the other.

In this chapter, we will construct a computer simulation to graphically show a world that is inhabited by both predator and prey. In particular, we will model bears and fish. We will apply some rules that each must live by. Moreover, as time progresses, individuals will live, breed, die, and move about the world described by the model. We will be able to observe the impact of initial conditions, as well as the interaction of one life-form with another as the simulation plays out.

11.4 Rules of the Game

Our computer simulation models a world that contains two types of life-forms: bears and fish. We can think of this world as a two-dimensional grid with a fixed size for each dimension. Life-forms are able to live only at specific locations within the grid.

Each life-form is described by a set of rules that govern how it lives. Initially, a group of life-forms will be placed in the world at random locations. The simulation will progress by allowing one of the life-forms to live for one time unit. During this unit of time, all other life-forms are in suspended animation. Thus, at any given time, the life-form is in one of two states: (1) alive or (2) suspended. The life-form to be put into the alive state is chosen at random from the collection of all possible life-forms. Each time a life-form is in the alive state, it must reevaluate its surroundings because the rest of the world will likely have changed during the time units when it was suspended. In this way, the simulation takes on a sense of reality.

When a fish is alive, it is allowed to breed, move, and die. Once a fish has been in the alive state 12 times, it may attempt to breed—a process we will simplify greatly to make it easier to code. In our simulation, the fish does not have to spawn or lay eggs. It just randomly picks an adjacent location. If that location is empty, a new fish appears. If the location is occupied, the fish must wait until the next time it is in the alive state and try again.

Regardless of whether a fish breeds, it will next try to move. When a fish moves, it randomly picks an adjacent location. If that location is empty, the fish moves to this new location. If the location is occupied, the fish remains in its current location.

One additional environmental characteristic that will affect fish is overcrowding. If a fish discovers that two or more fish are living adjacent to its own space, then the fish will die. Thus, even in the complete absence of bears, fish will self-regulate their population size to some extent. In addition, in this version of the simulation, fish never need to eat.

When a bear is alive, it is allowed to breed, move, eat, and die. Breeding takes place in much the same fashion as described previously for fish. The

only difference is that bears need to be in the alive state eight times before they can start breeding. Bears move in exactly the same manner as fish.

Bears are not impacted by overcrowding, but they do need to eat. To eat, a bear must determine whether any fish are in an adjacent location. If so, then the bear randomly picks one of the fish and "eats" it. To consume the fish, the bear moves to the location currently occupied by the chosen fish. If there are no adjacent fish, the bear begins to starve. Any bear that has been in the alive state and starving 10 times in a row will die.

11.5 Design

We begin by creating a basic design for our simulation based on the previous description. The first step is to identify those parts of the problem that correspond to objects. One of the easiest ways to start the process of object identification is to consider the prominent nouns that appear in the description of the problem. In this case, words such as *bear*, *fish*, and *world* seem like good choices. The nouns that we identify now will likely become Python classes when we implement the design.

Next, we need to analyze each noun and come up with a list of things it should know and a list of actions it can perform. The list of things an object should know will become instance variables. The list of things an object can do will become methods. This process helps us decide how objects will interact with one another. You may not identify all the instance variables and methods at this stage, but you can always go back and add more later as you discover additional information about the problem.

GOOD PROGRAMMING PRACTICE:

As a first step in design, identify the objects. Then, for each object, identify the instance variables (what the object should know) and methods (what the object should be able to do).

The following lists provide a first attempt to identify the characteristic properties for the nouns previously identified:

1. World
    - A world should know:
        - The maximum *X* and *Y* dimensions.
        - A collection of all life-forms present.
        - A grid with the locations of each specific life-form.
    - A world should be able to:

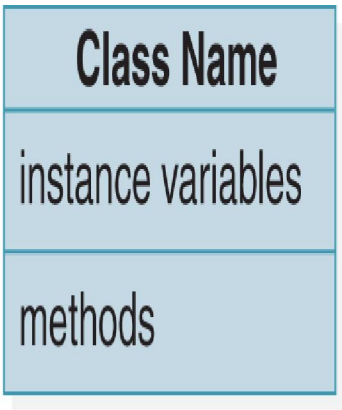
        - Return the dimensions of the world.
        - Add a new life-form at a specific location.
        - Delete a life-form wherever it is.
        - Move a life-form from one world location to another.
        - Check a world location to see if it is empty.
        - Return the life-form at a specific location.
        - Allow a life-form to live for one time unit.
        - Draw itself.

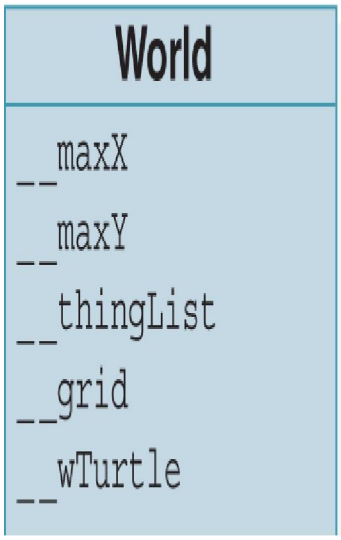
2. Bear
    - A bear should know:
        - The world to which it belongs.
        - Its specific world location given as (x, y).
        - How long it has gone without food.
        - How long it has gone without breeding.
    - A bear should be able to:
        - Return its location, both the x and y values.
        - Set its location, both the x and y values.
        - Set the world to which it belongs.
        - Appear if it is newly born.
        - Hide if it dies.
        - Move from one location to another.
        - Live for one time unit (includes breeding, eating, moving, and dying).

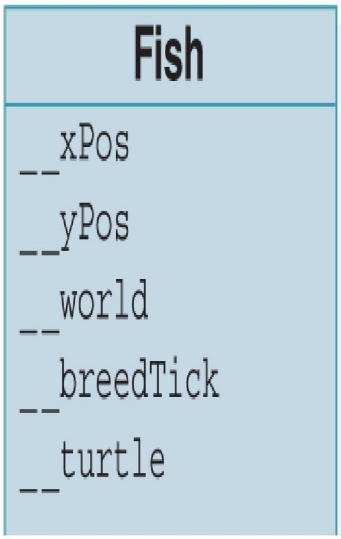
3. Fish
    - A fish should know:
        - The world to which it belongs.
        - Its specific location given as (x, y).
        - How long it has gone without breeding.
    - A fish should be able to:
        - Return its location, both the x and y values.
        - Set its location, both the x and y values.

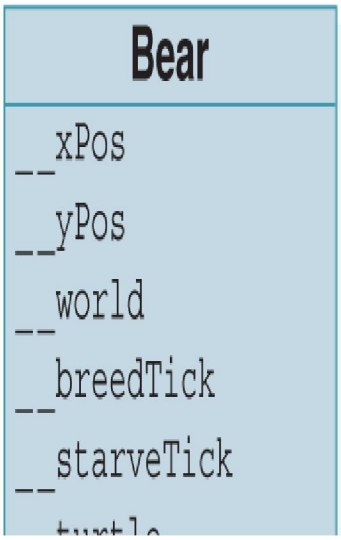
- Set the world that it belongs to.
- Appear if it is newly born.
- Hide if it dies.
- Move from one location to another.
- Live for one time unit (includes breeding, moving, and dying).

Figure 11.1 summarizes these results in a tabular form. Each table consists of the name of the class, a list of instance variables, and a list of methods. Names have been chosen that are consistent with the previous descriptions. Note that we have included an extra instance variable, a turtle, in each class. As we want this to be a graphical simulation, it will be necessary to include instances of the `Turtle` drawing tool in any class that will have some type of visual capability.









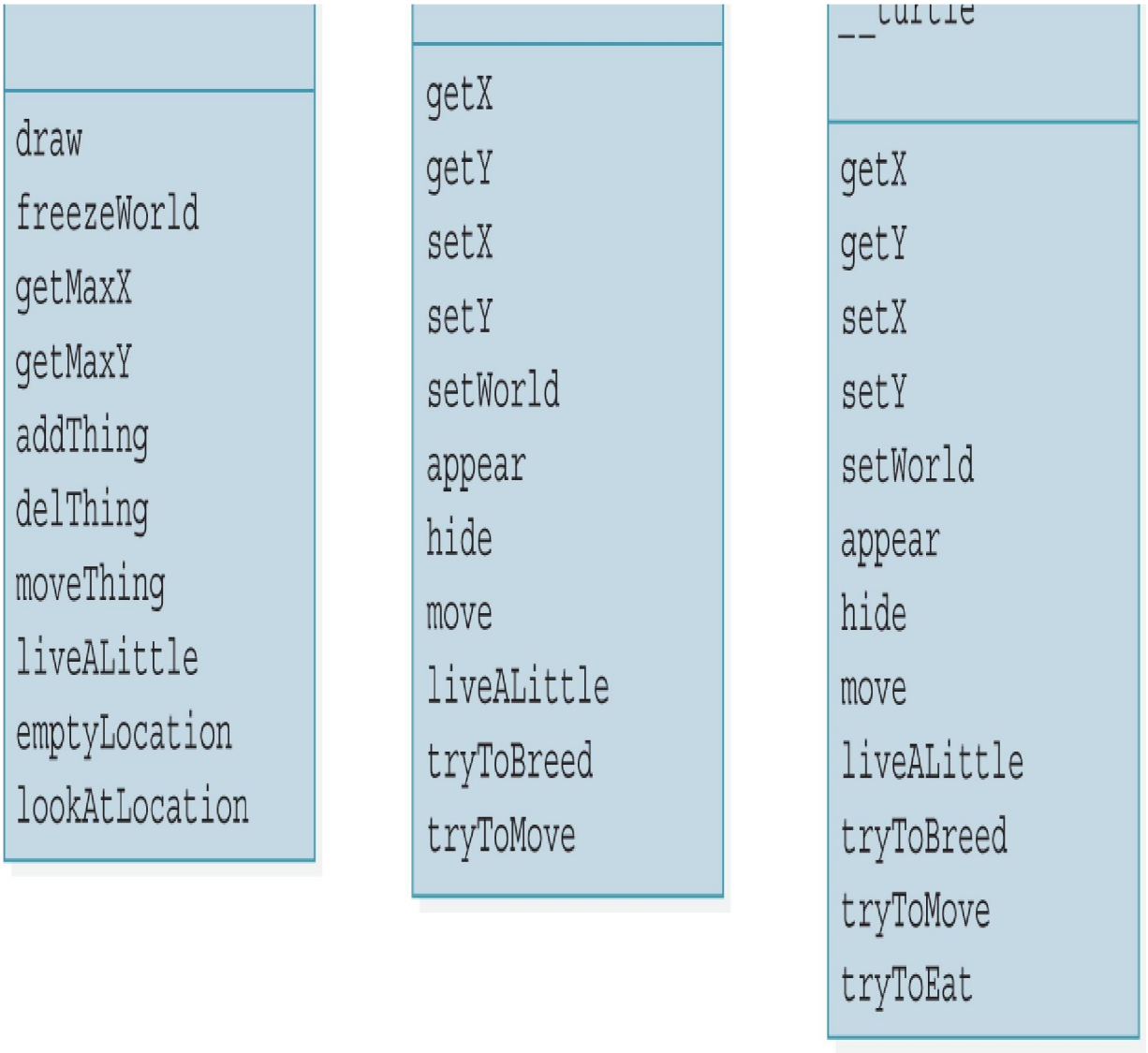


Figure 11.1 Listing instance variables and methods for each class.

Description

11.6 Implementation

We will create our simulation in Python by first implementing the instance variables and methods of the classes as described previously. Once we have the classes in place, we can complete the simulation by simply creating a world, adding life-forms to it, and then allowing them to live. The longer we let the simulation run, the more interesting the results may become.

11.6.1 The `World` Class

We will begin by implementing the `World` class. Recall that the `World` class will maintain dimensions as well as a list of the life-forms that are present. It will also have a grid to track the life-form positions. A `Turtle` will be used to set the initial coordinates for the world and to draw the grid.

Implementation of the grid requires some additional thought. The grid should maintain the exact two-dimensional position of each life-form. Each entry in the grid may contain a reference to a life-form. For those grid positions that do not contain a life-form, we will use the `None` value.

Figure 11.2 shows an example of a 12-by-6 grid with three life-forms present. Each position of the grid is addressed by a unique (x, y) pair. Bears are currently living at positions (2, 4) and (11, 5), and a fish is currently at position (5, 0).

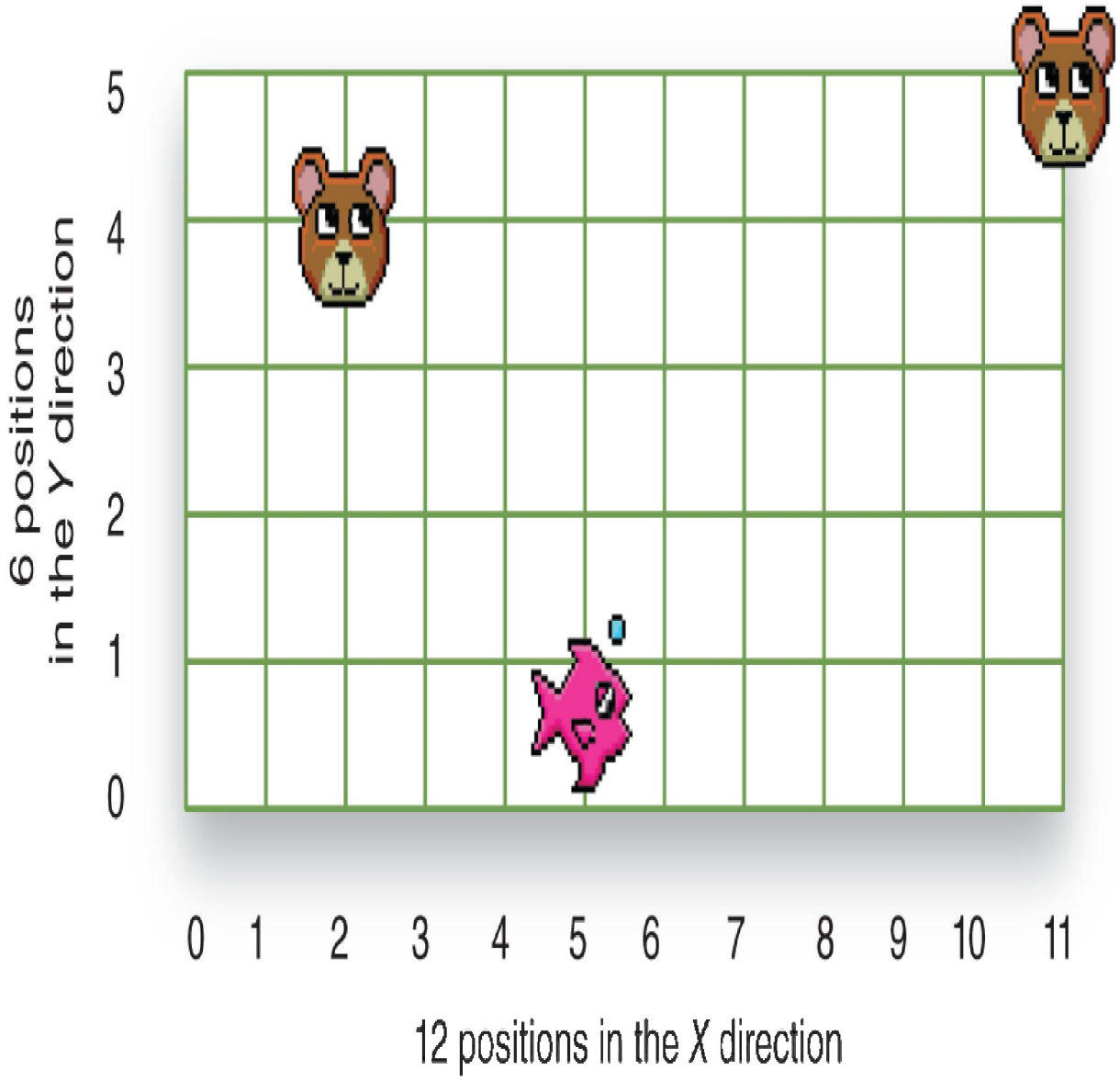


Figure 11.2 A grid with three life-forms.

Description

One of the easiest ways to implement a collection in Python is to use a list. However, since lists are one-dimensional, we need to be creative to represent the grid's two-dimensional structure. Our solution is to implement the rows and columns of the grid using a "list of lists" structure. To do so, we need to make a decision about whether the grid is actually a collection of rows or a collection of columns. Once we make this decision, the code that we write will simply need to be consistent with that decision.

Our choice will be to think of the grid as a collection of rows (list of rows). As mentioned in an earlier chapter, this arrangement is known as row-major order. Each row is, in turn, a list of items, one for each column. The entries are either a reference to a life-form or `None`. **Figure 11.3** shows a Python variable called `g` that is assigned to a list of six lists. This list of lists is designed to represent the information from Figure 11.2.

```
g = [   [ None,None,None,None,None,FISH,None,None,None,None,None,None ],

        [ None,None,None,None,None,None,None,None,None,None,None,None ],

        [ None,None,None,None,None,None,None,None,None,None,None,None ],

        [ None,None,None,None,None,None,None,None,None,None,None,None ],

        [ None,None,BEAR,None,None,None,None,None,None,None,None,None ],

        [ None,None,None,None,None,None,None,None,None,None,None,BEAR ]    ]
```

Figure 11.3 A list of lists representation for the grid example.

Description

Each item in the list `g` corresponds to a row in the grid. Row 0 can be accessed by `g[0]`, row 1 by `g[1]`, and so on. Within each row, there are 12 items. To access a specific item in a row simply requires another list index. For example, the fish is shown in position (5, 0). To access this position, we

would use `g[0]` to access row 0 and then use `[5]` to access the sixth item in the list. Thus we would use `g[0][5]` to access that item.

Generalizing this pattern, to access the item at any (*x*, *y*) location, we can give the list of lists grid location as `g[y][x]`. Because we decided that our rows would be "stored together," the first index will be the row or `y` value, and the second index will be the column or `x` value. In consequence, the two bears in the example can be accessed by `g[4][2]` and `g[5][11]`.

RECAP:
Lists of lists can be stored in row-major order or column-major order. In row-major order, each row is a list of items, one for each column. In column-major order, each column is a list of items, one for each row.

Listing 11.1 shows the constructor for the `World` class. In addition to `self`, it takes two arguments, the max *x* value and the max *y* value. The constructor creates six instance variables: `_ _maxX`, `_ _maxY`, `_ _thingList`, `_ _grid`, `_ _wTurtle`, and `_ _wScreen`. Remember that we precede each instance variable with two underscores to hide the name. This prevents code outside the class from directly accessing the instance variables and thereby achieves encapsulation. Initially, `_ _grid` is an empty list. Lines 9–13 show a nested iteration used to create the list of lists. Each row is first created by repeatedly appending `None` to an empty list. That entire list is then appended to the list of lists (`self._ _grid`).

```
1  import turtle
2  class World:
3      def __init__(self, mX, mY):
4          self._maxX = mX        #maximum x value
5          self._maxY = mY        #maximum y value
```

```
6           self._thingList = []   #list of living things
7           self._grid = []        #grid to hold living things
8
9           for aRow in range(self._maxY): #set all locations to empty
10              row = []
11              for aCol in range(self._maxX):
12                  row.append(None)
13              self._grid.append(row)
14
15          self._wTurtle = turtle.Turtle()
16          self._wScreen = turtle.Screen()
17          self._wScreen.setworldcoordinates(0, 0,
18                                  self._maxX - 1, self._maxY - 1)
19          self._wScreen.addshap("Bear.gif")
20          self._wScreen.addahape("Fish.gif")
21          self._wTurtle.hideturtle()
```

Listing 11.1 The `World` class constructor

Description

The remainder of the constructor (lines 15–21) modifies the coordinate system and adds two new shapes that will be used later as icons for fish and

bears. The turtle is hidden so that the default triangle shape no longer displays.

The `draw` method of the `World` class (**Listing 11.2**) will use `_ _wTurtle` to draw the grid system using the maximum *x* and *y* dimensions. It first draws the outer boundaries of the grid and then fills in the horizontal and vertical lines.

```
23      def draw(self):
24          self._wScreen.tracer(0)
25          self._wTurtle.forward(self._maxX - 1)
26          self._wTurtle.left(90)
27          self._wTurtle.forward(self._maxY - 1)
28          self._wTurtle.left(90)
29          self._wTurtle.forward(self._maxX - 1)
30          self._wTurtle.left(90)
31          self._wTurtle.forward(self._maxY - 1)
32          self._wTurtle.left(90)
33
34          for i in range(self._maxY - 1):
35              self._wTurtle.forward(self._maxX - 1)
36              self._wTurtle.backward(self._maxX -1)
37              self._wTurtle.left(90)
38              self._wTurtle.forward(1)
```

```
39              self._wTurtle.right(90)
40          self._wTurtle.forward(1)
41          self._wTurtle.right(90)
42          for i in range(self._maxX - 2):
43              self._wTurtle.forward(self._maxY - 1)
44              self._wTurtle.backward(self._maxY -1)
45              self._wTurtle.left(90)
46              self._wTurtle.forward(1)
47              self._wTurtle.right(90)
48          self._wScreen.tracer(1)
```

Listing 11.2 The `draw` method

Description

Listing 11.3 shows the other `World` methods. These methods will allow us to access or change some aspect of the simulation world. The simple accessor methods `getMaxX` and `getMaxY` return the maximum dimensions, `emptyLocation` returns `True` or `False` depending on whether a life-form is found at that location in the grid, and `lookAtLocation` returns the value at a grid location.

```
50      def addThing(self, aThing, x, y):
51          aThing.setX(x)
52          aThing.setY(y)
53          self._grid[y][x] = aThing  #add life-form to grid
```

```
54          aThing.setWorld(self)
55          self._thingList.append(aThing) #add to list of life-forms
56          aThing.appear()
57
58      def delThing(self, aThing):
59          aThing.hide()
60          self._grid[aThing.getY()][aThing.getX()] = None
61          self._thingList.remove(aThing)
62
63      def moveThing(self, oldX, oldY, newX, newY):
64          self._grid[newY][newX] = self._grid[oldY][oldX]
65          self._grid[oldY][oldX] = None
66
67      def getMaxX(self):
68          return self._maxX
69
70      def getMaxY(self):
71          return self._maxY
72
73      def liveALittle(self):
```

```
74          if self._thingList != []:  #If any life-forms
75              aThing = random.randrange(len(self._thingList))
76              randomThing = self._thingList[aThing]
77              randomThing.liveALittle()
78
79      def emptyLocation(self, x, y):
80          if self._grid[y][x] == None:
81              return True
82          else:
83              return False
84
85      def lookAtLocation(self, x, y):
86          return self._grid[y][x]
87
88      def freezeWorld(self):
89          self._wScreen.exitonclick()
```

Listing 11.3 The rest of the `World` class

Description

A new life-form is added to the world by `addThing`, in lines 50–56. This method needs the species and the (x, y) position where the life-form should be placed. Some of the methods called belong to the life-form itself and have

not yet been described; we will come back to them at a later time. For now, line 53 adds the life-form to the grid, and line 55 appends it to the list of life-forms maintained by the world.

Removing a life-form from the simulation world requires that it be taken off the grid and removed from the life-form list. These steps are performed by the `delThing` method. Line 60 sets the appropriate grid reference to `None`. Line 61 calls the list method `remove` to delete the fish or bear from the list of life-forms maintained by the world.

The `liveALittle` method, lines 73–77, does the majority of the simulation work. As long as any life-forms are in `thingList`, the world picks a random life-form from the list (lines 75–76) and calls the life-form's `liveALittle` method. Note that the world lets only one life-form live during every call to `liveALittle`.

The `freezeWorld` method is used at the end of the simulation. It calls the `exitonclick` method of `_ _wScreen` to allow the graphics window to remain drawn after the simulation has ended. Clicking on this window exits the simulation.

Try It Out

11.1 Implement the `World` using column-major order for the grid of locations. In other words, store the columns together in lists instead of the rows.

11.2 Add two instance variables, `_ _bearCount` and `_ _fishCount`, to the `World` class. Also add methods `getNumBears`, `getNumFish`, `incBears`, `decBears`, `incFish`, and `decFish` to increment and decrement the appropriate counters by one.

11.3 Add a method `showCounts` to show the count of bears and fish you added in Try It Out Exercise 11.2. You may want to use the `write` method of `_ _wTurtle` for this task.

11.6.2 The `Fish` Class

In this simulation, fish are the prey. They try to breed and move around, but they may end up in the vicinity of a bear, which will likely eat them for dinner. Fish need to know where they are and how long it has been since they have given birth. In addition, because this is a graphical simulation, each fish will have an instance of `Turtle` so that it can be displayed.

In **Listing 11.4**, which shows the constructor for the `Fish` class, `self._ _xPos` and `self._ _yPos` are initialized to (0, 0) but will be set later when the fish are placed in the world. Likewise, each fish will have a reference back to the world in which it lives. Line 10 sets this reference initially to `None`. This reference will also be set when a fish is added to the world.

```
1  class Fish:
2      def __init__(self):
3          self.__turtle = turtle.Turtle()
4          self.__turtle.up()
5          self.__turtle.hideturtle()
6          self.__turtle.shape("Fish.gif")
7
8          self.__xPos = 0
9          self.__yPos = 0
10         self.__world = None
11
12         self.__breedTick = 0
```

Listing 11.4 The constructor for the `Fish` class

Description

The next group of methods for the `Fish` class consists of the simple accessor and mutator methods used by other objects as they interact with `Fish` objects (**Listing 11.5**). They allow "getting" and "setting" of values as well as some basic `Turtle` functions. The `appear` and `move` methods need a bit of explanation. When a `Fish` is first created, it is added to the world at a specific location and then told to appear. The `appear` method takes care of moving the underlying `Turtle` object and then showing the icon in the simulation display.

```
14      def setX(self, newX):
15          self._xPos = newX
16
17      def setY(self, newY):
18          self._yPos = newY
19
20      def getX(self):
21          return self._xPos
22
23      def getY(self):
24          return self._yPos
25
26      def setWorld(self, aWorld):
27          self._world = aWorld
28
```

```
29      def appear(self):
30          self._turtle.goto(self._xPos, self._yPos)
31          self._turtle.showturtle()
32
33      def hide(self):
34          self._turtle.hideturtle()
35
36      def move(self, newX, newY):
37          self._world.moveThing(self._xPos, self._yPos, newX, newY)
38          self._xPos = newX
39          self._yPos = newY
40          self._turtle.goto(self._xPos, self._yPos)
```

Listing 11.5 The simple accessor and mutator methods of the `Fish` class

Description

Once the `Fish` object is present in the simulation, it can be moved to a new location using the `move` method. This method not only moves the underlying `Turtle` but also changes the *x* and *y* positions of the `Fish` object. The world must also move the object in its grid.

The most important method in the `Fish` class is `liveALittle`. This method is responsible for what happens to a `Fish` each time it is allowed to live for a time unit. Recall that fish can die if they are crowded by too many other fish. If that is not the case, then they will try to breed and then finally try to move.

The first thing the `liveALittle` method needs to do is count the number of `Fish` at adjacent locations to determine if overcrowding is occurring. This will require that we design a process to find all of the locations that are adjacent to a given location. For this task, we will use a list of offsets. Each offset is a tuple containing an amount to adjust the x coordinate and an amount to adjust the y coordinate. **Figure 11.4** shows the (x, y) values for any location and the eight locations around it. Note that, using the original (x, y), it is possible to compute the coordinates of all adjacent locations.

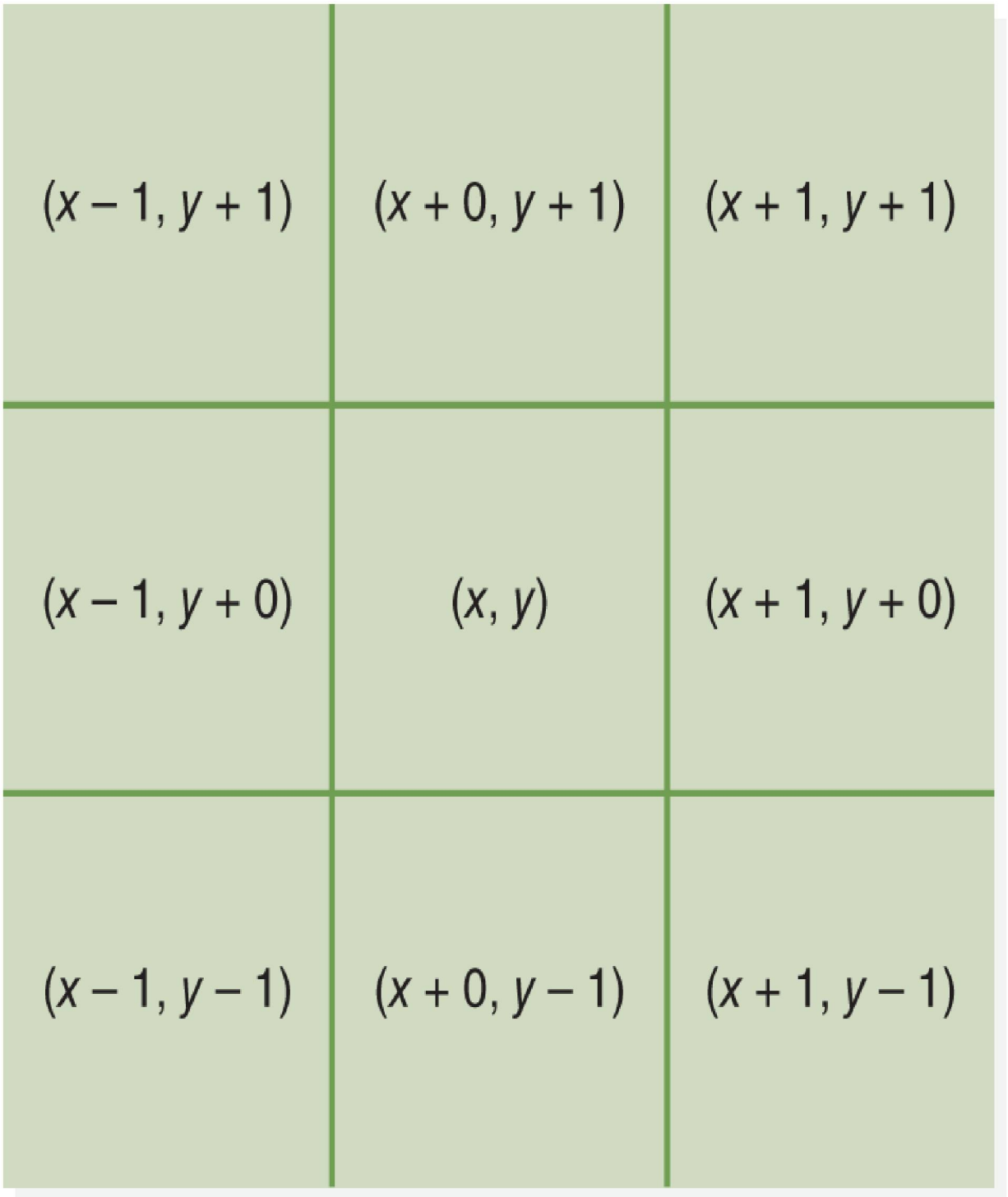


Figure 11.4 Coordinates of the eight adjacent locations around an (x, y) position.

Description

Figure 11.5 shows how we can reduce the coordinates to a set of offset tuples. In **Listing 11.6**, we create a list of these offset tuples. Now it is just a matter of iterating through the `offsetList` and adjusting the original (*x*, *y*) position by the appropriate tuple component to find the eight locations around the fish.

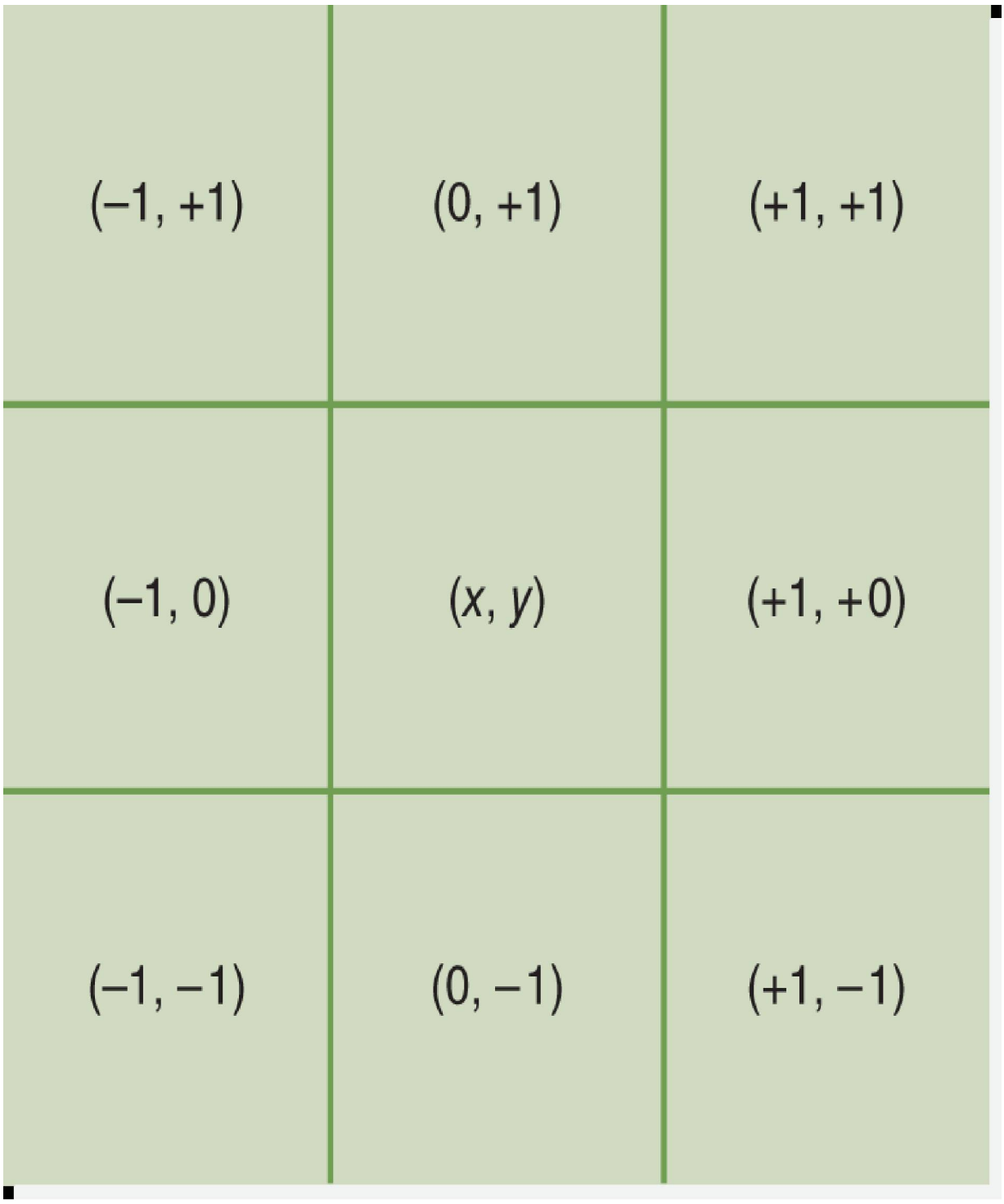


Figure 11.5 Offset tuples of the eight adjacent locations around an (x, y) position.

Description

```
43    def liveALittle(self):
```

```
44        offsetList = [(-1,1), (0,1), (1,1),
45                      (-1,0),        (1,0),
46                      (-1,-1),(0,-1),(1,-1)]
47        adjFish = 0  #count adjacent Fish
48        for offset in offsetList:
49            newX = self._xPos + offset[0]
50            newY = self._yPos + offset[1]
51            if 0 <= newX < self._world.getMaxX()  and \
52                  0 <= newY < self._world.getMaxY():
53                if (not self._world.emptyLocation(newX, newY)) and \
54                    isinstance(self._world.lookAtLocation(newX, newY), Fish):
55                    adjFish = adjFish + 1
56
57        if adjFish >= 2:   #if 2 or more adjacent Fish, die
58            self._world.delThing(self)
59        else:
60            self._breedTick = self._breedTick + 1
61            if self._breedTick >= 12:  #if alive 12 or more ticks, breed
62                self.tryToBreed()
63
```

```
64                self.tryToMove()          #try to move
```

Listing 11.6 The `liveALittle` method in the Fish class

Description

Listing 11.6 shows the `liveALittle` method for the `Fish`. Lines 44–46 create the list of offset tuples, and line 47 initializes a variable to count the number of neighboring fish (`adjFish`). Lines 49–55 are performed for each offset in `offsetList`. First, new x and y values are computed from the current (x, y) location and the current offset tuple. Lines 51–52 check whether this new location is a legal location, because the original (x, y) might have been on an edge of the grid.

If the location is legal, then we can check it for a life-form. We first check that the location is not empty; then we use the Python built-in `isinstance` function, shown in **Table 11.1**, to ask if an object is an instance of a particular class. In this case, we want to count only those objects that are instances of the `Fish` class. Line 55 increments the `adjFish` counter if this is the case.

TABLE 11.1 Python's Built-in `isinstance` Function

Function	Description	Examples

`isinstance(cbject, type)`	Returns `True` if the object is of the type specified; `False` if not. If `type` is a tuple, the function returns `True` if the object is one of the types in the tuple.	`isinstance(1, int)`
		`True`
		`isinstance(1, float)`
		`False`
		`isinstance(1, (int, flcat))`
		`True`

Once we have counted the adjacent `Fish`, line 57 checks whether there is overcrowding according to the rules described earlier. If so, `world` is told to delete the fish (`self`). Because `self` is a reference to the actual fish, `_world` can use this reference to locate and remove the object from `thingList`. This approach may look odd to you, given that we have always used `self` as a formal parameter rather than as an actual parameter. However, if you trace the call to `delThing` in Listing 11.3, you will see that `self` is assigned to the formal parameter `aThing`.

If the fish is not overcrowded, the next activity is to check whether the fish has been active for enough time units to try to breed. According to the rules, a fish must wait 12 time units before trying to breed. Regardless of whether the fish is successful in breeding, it then tries to move to a new location.

If it is determined that the `Fish` can try to breed, the `tryToBreed` method is called (**Listing 11.7**). According to the rules, this method must first pick a random adjacent location. To do this, we can use the same offset list technique that we employed earlier. In this case, instead of iterating through all of the offsets, we will simply pick one at random. To choose a random element of the list, we will first generate a random integer in the range of index values using the length of the `offsetList` as the upper bound. Once we compute this new (x, y) position, we must be sure that it is in the range of legal coordinates. If it is not, we must try again. The `while` loop (starting on lines 74–75) continues to choose random offset pairs until a legal result is obtained.

```
66    def tryToBreed(self):
67        offsetList = [(-1,1), (0,1), (1,1),
68                      (-1,0),        (1,0),
69                      (-1,-1),(0,-1),(1,-1)]
70        randomOffsetIndex = random.randrange(len(offsetList))
71        randomOffset = offsetList[randomOffsetIndex]
72        nextX = self._xPos + randomOffset[0]
73        nextY = self._yPos + randomOffset[1]
74        while not (0 <= nextX < self._world.getMaxX() and \
75                   0 <= nextY < self._world.getMaxY() ):
76            randomOffsetIndex = random.randrange(len(offsetList))
77            randomOffset = offsetList[randomOffsetIndex]
78            nextX = self._xPos + randomOffset[0]
79            nextY = self._yPos + randomOffset[1]
80
81        if self._world.emptyLocation(nextX, nextY):
82            childThing = Fish()
83            self._world.addThing(childThing, nextX, nextY)
84            self._breedTick = 0     #reset breedTick
```

Listing 11.7 The `tryTcBreed` method in the `Fish` class

Description

Once a new random location has been determined, the rules state that it must be empty for breeding to take place. Line 81 checks the location to see if the location is empty. If so, a new `Fish` is created and added to the world at that location. Recall that the `addThing` method sets the (x, y) and world values for the fish and makes the fish appear in the graphical display.

The `tryToMove` method (**Listing 11.8**) has much of the same functionality as `tryToBreed`. It first picks a random adjacent location and checks whether it is empty. If it is, the fish is allowed to move using its `move` method.

```
86    def tryToMove(self):
87        offsetList = [(-1,1), (0,1), (1,1),
88                      (-1,0),        (1,0),
89                      (-1,-1),(0,-1),(1,-1)]
90        randomOffsetIndex = random.randrange(len(offsetList))
91        randomOffset = offsetList[randomOffsetIndex]
92        nextX = self._xPos + randomOffset[0]
93        nextY = self._yPos + randomOffset[1]
94        while not (0 <= nextX < self._world.getMaxX() and \
95                   0 <= nextY < self._world.getMaxY() ):
96            randomOffsetIndex = random.randrange(len(offsetList))
97            randomOffset = offsetList[randomOffsetIndex]
98            nextX = self._xPos + randomOffset[0]
99            nextY = self._yPos + randomOffset[1]
100
101       if self._world.emptyLocation(nextX, nextY):
102           self.move(nextX, nextY)
```

Listing 11.8 The `tryTcMove` method in the `Fish` class

Description

Try It Out

11.4 Make the `offsetList` an instance variable of the `Fish` class.

11.5 Modify the `liveALittle` method to use instance variables for the number of fish that constitute overcrowding and the number of time units that must pass before breeding is allowed. Also, modify the `Fish` constructor to have parameters to initialize the new instance variables.

11.6 Add accessor and mutator methods for the instance variables you created in the previous exercise.

11.7 Modify the `tryToMove` method to allow the fish to try up to four random locations for its move.

11.6.3 The `Bear` Class

The `Bear` class is implemented in much the same way as the `Fish` class. In fact, many of the methods are identical, including the accessors and mutators for `__xPos` and `__yPos`, `setWorld`, `appear`, `hide`, and `move`.

Here, we will focus on those methods that are either different or new. The obvious place to start is with the constructor and the `liveALittle` method. As described earlier, bears will not be impacted by overcrowding. The first thing they will do is try to breed. The other important difference is that bears will eat fish if fish are available. However, any bear that does not get enough food will starve and die. In consequence, the constructor for the `Bear` class will need to include an instance variable that keeps track of the number of time units since the bear's last meal.

Listings 11.9 and **11.10** show these modifications. The obvious difference between the `Bear` constructor and the `Fish` constructor is the graphics icon used as the turtle shape and the `__starveTick` instance variable. The first three lines of the `liveALittle` method indicate that bears must wait for eight time units before attempting to breed (recall that fish waited for 12 time units). Implementation of the `tryToBreed` method for the `Bear` is identical to

the implementation in the `Fish` class except that a `Bear` will construct a new `Bear` as `childThing`.

```
1  class Bear:
2      def __init__(self):
3          self._turtle = turtle.Turtle()
4          self._turtle.up()
5          self._turtle.hideturtle()
6          self._turtle.shape("Bear.gif")
7
8          self._xPos = 0
9          self._yPos = 0
10         self._world = None
11
12         self._starveTick = 0
13         self._breedTick = 0
14
15     #same method as Fish
16     #setX, setY, getX, getY, setWorld, appear, hide, move,
17     #tryToBreed (create a Bear), tryToMove
```

Listing 11.9 The constructor method in the `Bear` class

Description

```
81  def liveALittle(self):
82      self._breedTick = self._breedTick + 1
83      if self._breedTick >= 8:  #if alive 8 or more ticks, breed
84          self.tryToBreed()
85
86      self.tryToEat()
87
88      if self._starveTick == 10:  #if not eaten for 10 ticks, die
89          self._world.delThing(self)
90      else:
91          self.tryToMove()
```

Listing 11.10 The `liveALittle` method in the `Bear` class

Description

The next step in the `liveALittle` method allows a bear to try to eat a fish (line 86). The `tryToEat` method is shown in **Listing 11.11**. Recall that bears will check in the adjacent locations to see if any fish are present. If so, the bear will pick one of the fish at random and eat it. To eat, the bear moves to the location currently occupied by the fish and the fish dies. If there are no fish to eat, the bear increments its `_starveTick`.

```
93  def tryToEat(self):
```

```
94      offsetList = [(-1,1), (0,1) ,(1,1),
95                    (-1,0),       (1,0),
96                    (-1,-1),(0,-1),(1,-1)]
97      adjPrey = []     #create list of adjacent prey
98      for offset in offsetList:
99          newX = self._xPos + offset[0]
100         newY = self._yPos + offset[1]
101         if 0 <= newX < self._world.getMaxX() and \
102            0 <= newY < self._world.getMaxY():
103             if (not self._world.emptyLocation(newX, newY)) and  \
104                     isinstance(self._world.lookAtLocation(newX, newY), Fish):
105                 adjPrey.append(self._world.lookAtLocation(newX, newY))
106
107     if len(adjPrey) > 0:  #if any Fish are adjacent, pick random Fish to eat
108         randomPrey = adjPrey[random.randrange(len(adjPrey))]
109         preyX = randomPrey.getX()
110         preyY = randomPrey.getY()
111
112         self._world.delThing(randomPrey)  #delete the Fish
113         self.move(preyX, preyY)           #move to the Fish's location
```

```
114             self._starveTick = 0
115         else:
116             self._starveTick = self._starveTick + 1
```

Listing 11.11 The `tryToEat` method in the `Bear` class

Description

As before, we will use the offset list technique to check the adjacent locations for `Fish`. However, instead of just counting the number of `Fish` present, we will create a list of adjacent fish. Line 97 creates an initially empty list, and line 105 adds each neighboring `Fish` to the list. The details for computing the adjacent (x, y) locations and checking the instance type are the same as we used to look for neighboring fish in the test for overcrowding.

Line 107 checks whether any `Fish` are adjacent to the bear. If not, then `_ _starveTick` is incremented (line 116). If fish are present, one is chosen at random by once again picking a random index value (line 108). Since the bear must move to the location of the fish to eat it, we must get the current (x, y) location of the randomly chosen fish by using `getX` and `getY`. Lines 112–113 tell the world to delete the fish and tell the bear to move to the location formerly occupied by the fish. Because the bear has now eaten, it is no longer starving and `_ _starveTick` can be reset to zero (line 114).

Bears move in exactly the same way as fish, so the `tryToMove` method is identical in both classes. Remember that they check adjacent locations and can move only to a location that is unoccupied. It is also important to note that in the `liveALittle` method for the `Bear`, the `tryToMove` method is called regardless of whether the bear eats. This means that it is possible to have the bear move twice during a time unit, once to eat and then again when it moves normally.

Try It Out

11.8 Make the `offsetList` an instance variable of the `Bear` class.

11.9 Modify the `liveALittle` method to use instance variables for the number of time units that must pass before a bear starves and the number of time units that must pass before breeding is allowed. Also modify the `Bear` constructor to have parameters to initialize the new instance variables.

11.10 Add accessor and mutator methods for the instance variables you created in the previous exercise.

11.11 Modify the `tryToMove` method to allow the bear to be more intelligent about the direction it moves. Perhaps the bear can look farther out to see if there are any fish in some direction.

11.6.4 Main Simulation

It is now time to write the function that will set up the entire simulation and start it. Recall that the simulation itself dictates how it progresses due to the random numbers that are generated. This main function will need to do four things:

1. Set initial constants for the number of bears and fish, the size of the world, and the length of the simulation.
2. Create an instance of the `World`.
3. Create the specified number of bears and fish, placing them at random locations in the world.
4. Let the world live for the specified number of time units.

Listing 11.12 shows the complete `mainSimulation` function. In this case, we initialize the world with 10 bears and 10 fish. Line 8 creates a `World`, called `myWorld`, with dimensions as defined. The grid for `myWorld` is then drawn (line 9).

```
1 def mainSimulation():
2     numberOfBears = 10
```

```
3     numberOfFish = 10
4     worldLifeTime = 2500
5     worldWidth = 50
6     worldHeight = 25
7
8     myWorld = World(worldWidth, worldHeight)  #create the world
9     myWorld.draw()
10
11    for i in range(numberOfFish):             #create fish
12        newFish = Fish()
13        x = random.randrange(myWorld.getMaxX())
14        y = random.randrange(myWorld.getMaxY())
15        while not myWorld.emptyLocation(x, y):
16            x = random.randrange(myWorld.getMaxX())
17            y = random.randrange(myWorld.getMaxY())
18        myWorld.addThing(newFish, x, y)
19
20    for i in range(numberOfBears):            #create bears
21        newBear = Bear()
22        x = random.randrange(myWorld.getMaxX())
```

```
23          y = random.randrange(myWorld.getMaxY())
24          while not myWorld.emptyLocation(x, y):
25              x = random.randrange(myWorld.getMaxX())
26              y = random.randrange(myWorld.getMaxY())
27          myWorld.addThing(newBear, x, y)
28
29      for i in range(worldLifeTime):              #run the simulation
30          myWorld.liveALittle()
31
32      myWorld.freezeWorld()
```

Listing 11.12 The `mainSimulation` function

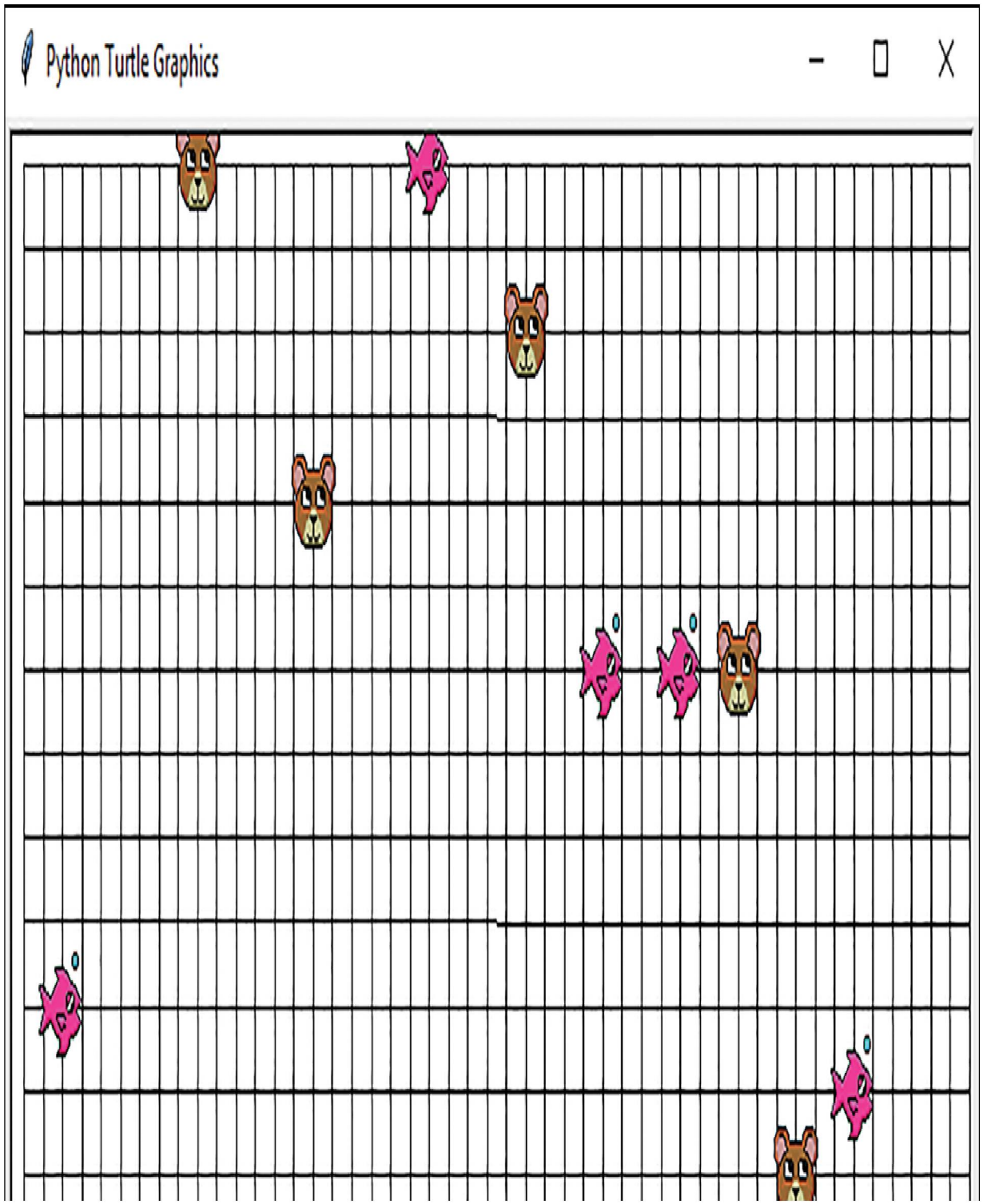
Description

Lines 11–18 create and place the `Fish` objects in the world. Note that once a fish has been created, the function must generate a random (x, y) location to hold it. Because we do not want more than one life-form living at a specific location, we must check whether the location is empty before using it. If it is not empty, we continue to generate random (x, y) locations (lines 16–17). For our purposes, we assume that there will always be enough space to find an empty location.

Bears are created and placed in the same way as fish (lines 20–27).

Most of the work is then accomplished by lines 29–30. For each time unit, the `World`'s `liveALittle` method is called. Recall that this means a random life-form is chosen to live for that time unit. The final line of the function puts the world into a wait mode so that we can observe the final state of the simulation. To exit the simulation, click anywhere in the window.

Figure 11.6 shows the simulation running with one possible way that the life-forms can be placed in the world. Of course, due to the random nature of the initial location selection, subsequent runs of the simulation will likely look different.



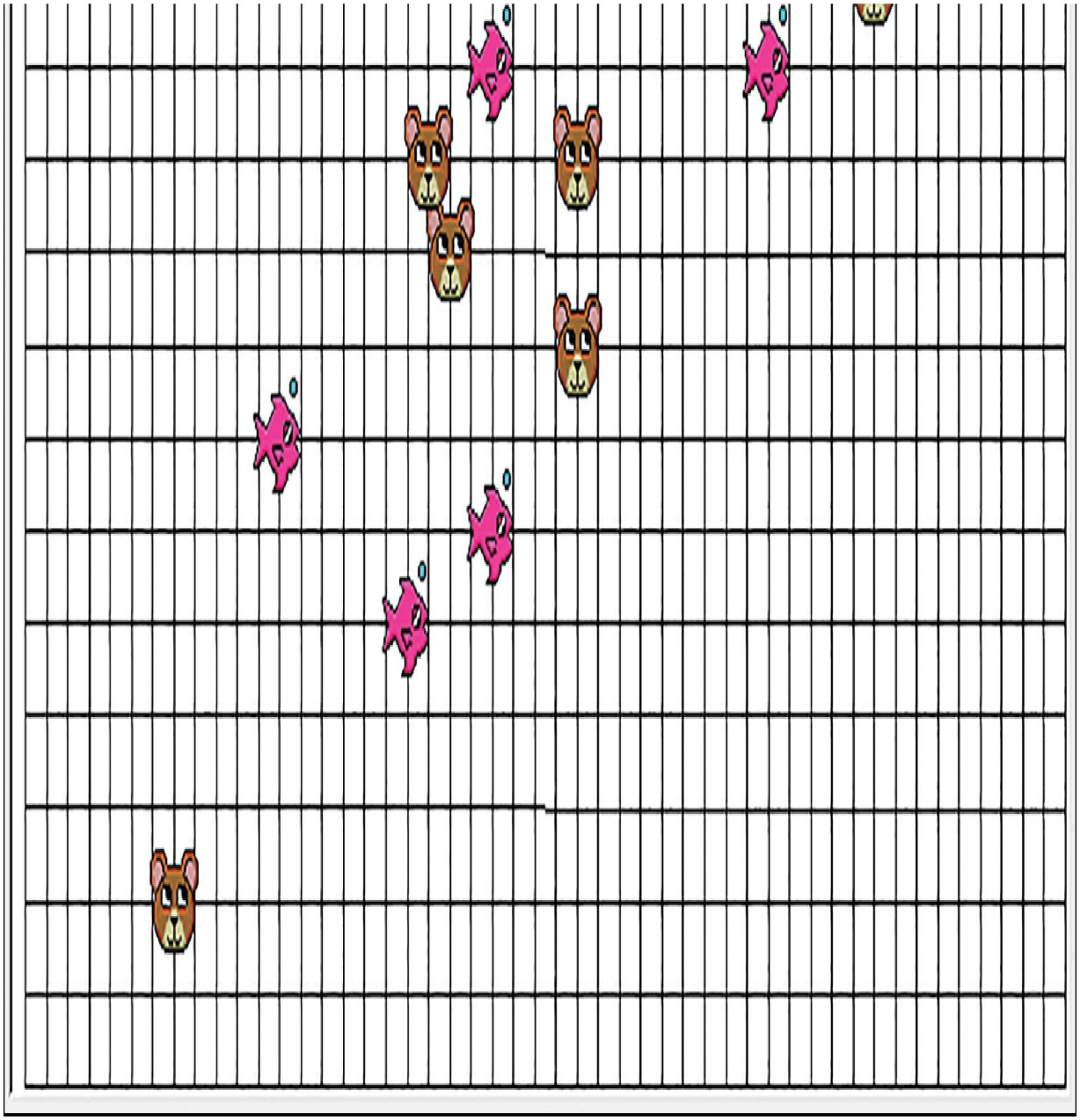


Figure 11.6 The initial simulation.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

Try It Out

11.12 Modify `addThing` and `delThing` to automatically update the fish and bear counters you added in an earlier exercise.

11.13 Run the simulation with different numbers of bears and fish. What do you observe about the number of bears and fish over time?

11.14 Modify `mainSimulation` to create bears and fish using different constants for the time to breed and time to starve. This will allow the life-forms to behave slightly differently.

11.15 Modify the simulation so that it will automatically stop if the number of bears or fish goes above or below some thresholds that you define.

11.16 Modify the `liveALittle` method of the `World` so that all bears and fish get a chance to live each time unit. To make it fair, you should randomly shuffle `thingList` each time unit.

11.17 Modify the `mainSimulation` function to create two lists. One list will keep track of the number of fish that are alive in each time unit, and the other will keep track of the number of bears that are alive. When the simulation is done, this data should be written to a file. The file should include three columns: one column for the time, one for the number of fish, and one for the number of bears.

11.18 Write a function to read the file produced by the previous exercise. This function should produce a graph that shows how the numbers of bears and fish change over time.

11.7 Growing Plants

In our initial description of the bears and fish model, we mentioned that fish can also be considered predators—that is, they can eat plants that live in the water. If not enough plants are available, the fish will starve and die. It is relatively easy to include another class to represent the plants by "repackaging" code that is already written.

We can begin by saying that plants will not be able to move and we will not have them eat anything. So, their only action is breeding. **Listing 11.13** shows the `liveALittle` method for the `Plant` class. Note that we are assuming that plants can attempt to breed after five time units.

```
56      def liveALittle(self):
57          self._breedTick = self._breedTick + 1
58          if self._breedTick >= 5:  #if alive 5 or more ticks, breed
59              self.tryToBreed()
```

Listing 11.13 The `liveALittle` method in the `Plant` class

Description

A change needed in the `World` class is to add the shape for the `Plant` object. Insert the line registering the plant image in **Listing 11.14** into the `World` constructor.

```
21          self.__wScreen.addshape("Bear.gif")
22          self.__wScreen.addshape("Fish.gif")
23          self.__wScreen.addshape("Plant.gif")
```

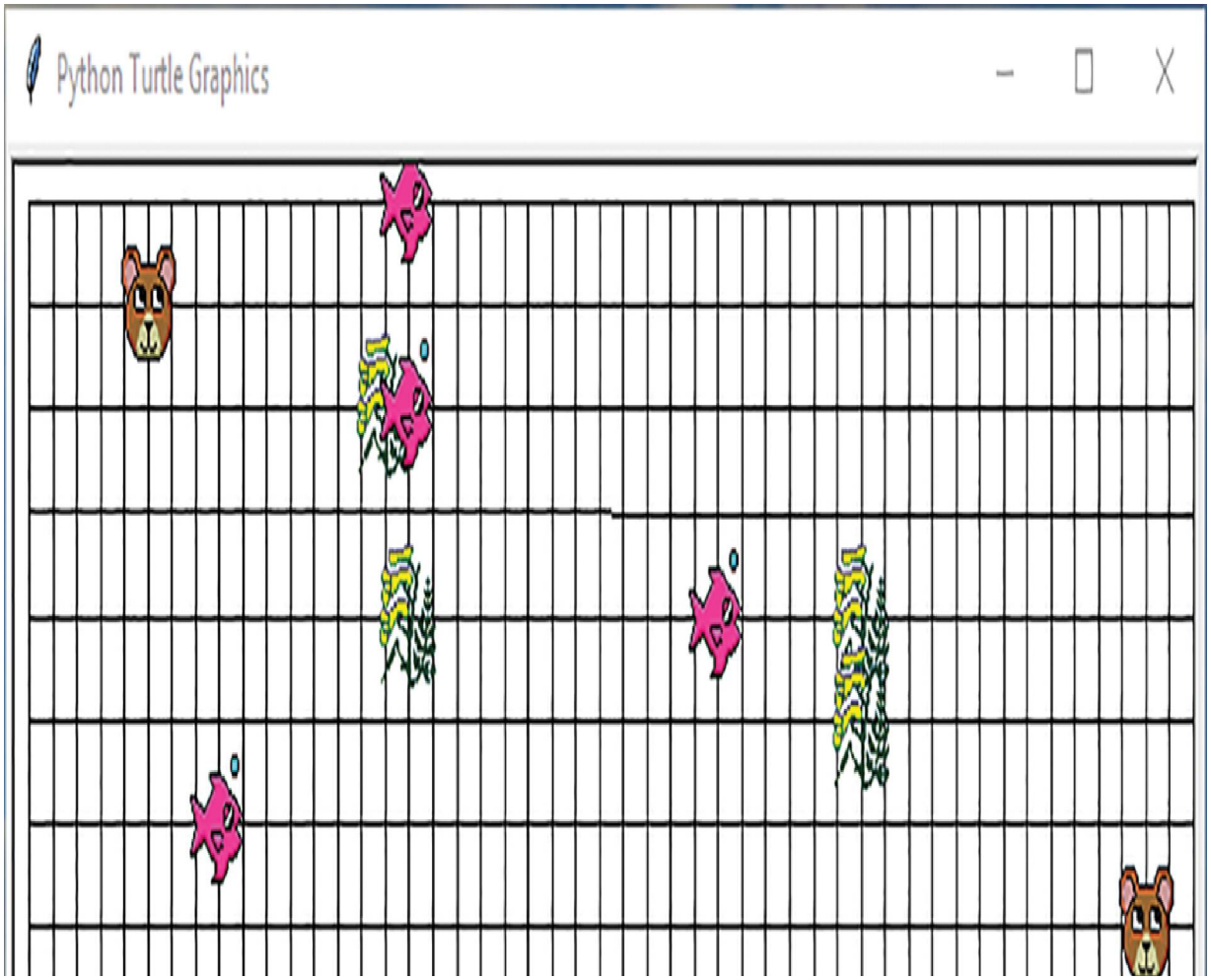
Listing 11.14 Registering the life-form images in the `World` class

Description

In the `Fish` class, we need to make three changes:

1. Include a `tryToEat` method. This method would be identical to the `tryToEat` method of the `Bear` class, except for the modification that fish eat plants.
2. Modify the `Fish`'s `liveALittle` method to call the `tryToEat` method.
3. Define a `__starveTick` instance variable in the `__init__` method.

Finally, we need to modify the main function (`mainSimulation()`) to add plants to the simulation at random locations. **Figure 11.7** shows a possible initial scenario that includes plants as well as the bears and fish.



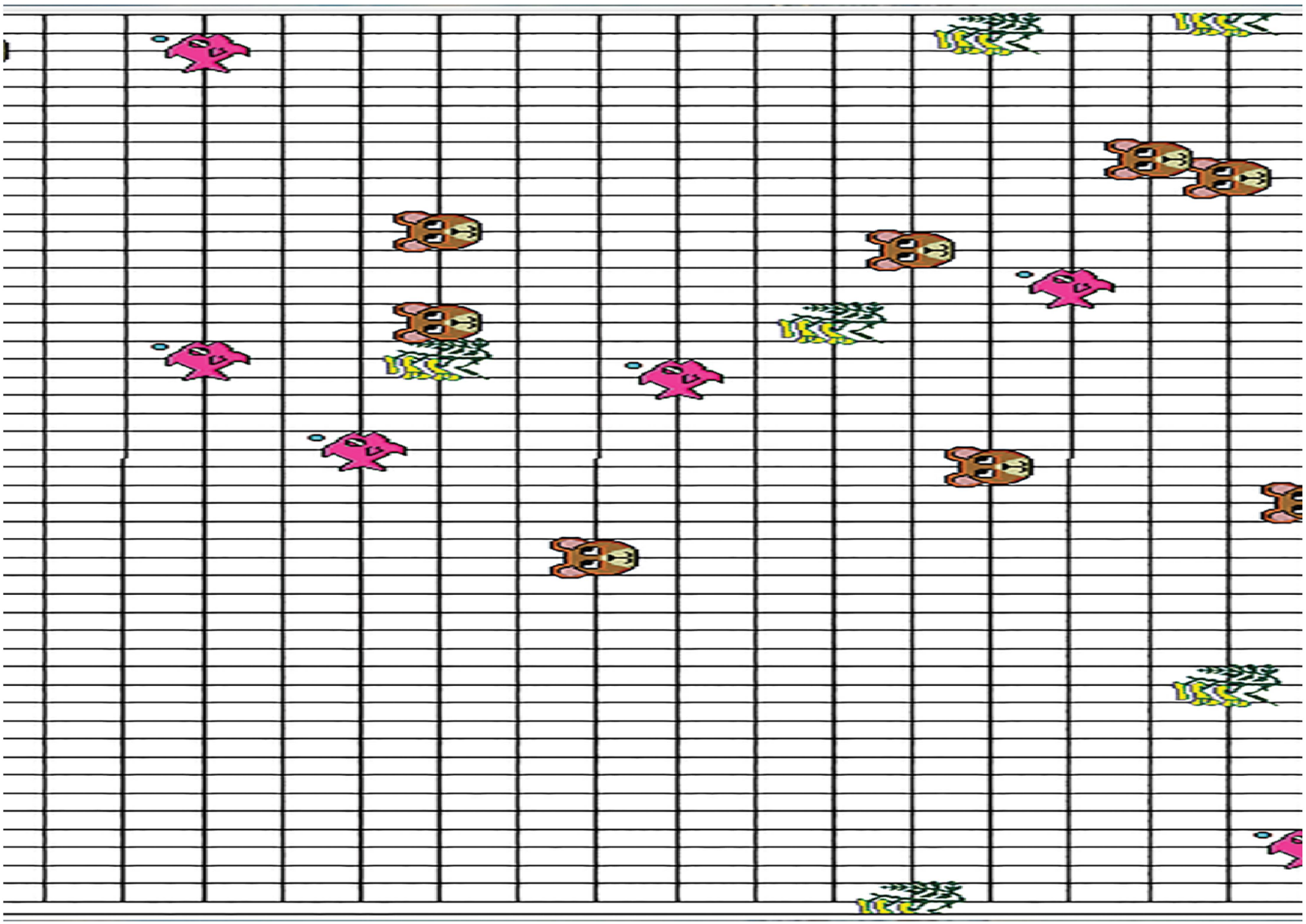




Figure 11.7 The initial simulation including plants.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

Try It Out

11.19 Create a `Plant` class that is similar to the `Fish` class using the modifications suggested in Section 11.7.

11.20 Modify the `liveALittle` method of the `Fish` class so that fish can try to eat plants. You will need to add a `_ _starveTick` counter for the fish.

11.21 Modify the main simulation to create some initial number of plants, and place them in the `World`.

11.22 Modify the `Plant` class to implement some kind of overcrowding mechanism.

11.8 A Note on Inheritance

As you considered the implementations of the `Fish`, `Bear`, and `Plant` classes, you probably noticed a number of places where the code was duplicated. Each class includes methods that are identical. For example, `getX`, `getY`, `setX`, `setY`, `appear`, and `hide` are all implemented the same way regardless of the class.

This observation leads us to think about the similarities among all life-forms in our simulation. It would make sense if we could somehow put all this duplication in one place and have all of the life-forms share it. This new implementation would allow us to efficiently add and modify functionality.

The solution that we are proposing is called **inheritance**—an object-oriented programming technique that is supported by many programming languages, including Python. We introduce inheritance in Chapter 12 and will ask you to reconsider the simulation as a programming exercise.

11.9 Summary

In this chapter, we designed and implemented a large, multiclass graphical simulation. First, we considered a description of the problem. We picked out the important subjects (nouns) and began to implement classes, one for each. Then, to design a class, we described those things that instances of a class would need to know about themselves (the instance variables) and those things that instances of a class would need to be able to do (the methods). After creating each class, we implemented a function that established many objects, and we let each object perform methods as defined by the initial rules of the simulation. By using random numbers, we were able to provide a degree of uncertainty so that no two runs of our simulation would look the same.

Key Terms

computer simulation
inheritance
predator–prey relationships

Programming Exercises

11.1 Modify the simulation so that a `World` consists of two kinds of locations, water and land. Fish and plants can live only in the water, and bears can go in the water only to eat fish.

11.2 Modify the simulation to give bears an energy instance variable. Each time a bear eats, its energy level increases. Bears should lose energy as they move and breed. If the energy level drops to zero, the bear dies.

11.3 Add a `Berry` class so that the bears have another food source. This class should behave much like the `Plant` class.

11.4 Choose another real-world predator–prey ecosystem and implement a simulation.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 12

A Hierarchy of Shapes

Inheritance and Object-Oriented Design

12.1 OBJECTIVES

- To introduce the concept of inheritance
- To create a working object-oriented graphics package
- To provide another example of object-oriented design

12.2 Introduction

Common computer graphics programs, such as Microsoft's Visio or The Omni Group's OmniGraffle, allow users to select the shapes they want in their drawing from a palette of available shapes. For simple drawing programs, this palette may include lines, circles, squares, triangles, and other common shapes. For more advanced drawing programs, such as architecture programs or computer-aided design (CAD) programs, the palette may include sophisticated shapes, such as cabinets, bathroom fixtures, or even specific parts for a car. These kinds of drawing programs are typically called **object-oriented drawing programs**.

Object-oriented drawing programs are easy to use. If your drawing calls for a square, or any other shape, you drag the shape from the palette of shapes and drop it into your drawing in the appropriate place. Once a shape is placed in the drawing, you can change its attributes, such as size, color, outline, or position.

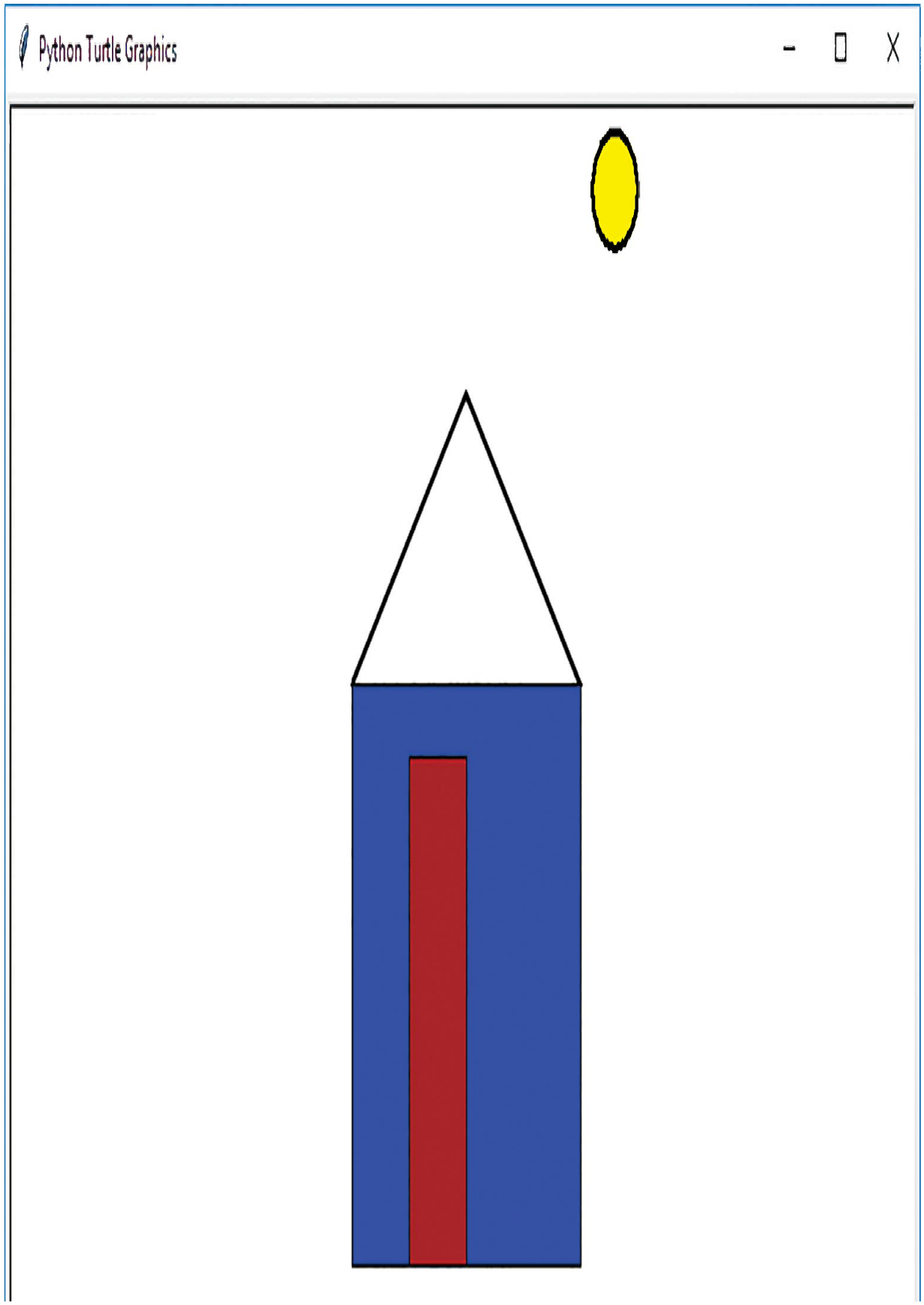
Our goal in this chapter is to design an object-oriented graphics module for Python that will allow you to create graphics drawings by creating simple graphical shapes and placing them on a drawing canvas. The shapes will be like the shapes for a drawing program, such as the ones described previously, except that we will not write the code to let a user drag the objects from a palette.

To be clear about where we are going in this chapter, consider the simple Python function `drawHouse`, shown in **Listing 12.1**. The code draws a blue-filled rectangle as the main part of a house, along with a brown door and a peaked roof. In addition, it draws a yellow sun. The picture drawn by the code in Listing 12.1 is shown in **Figure 12.1**.

```
1  def drawHouse():
2      myCanvas = Canvas(800, 600)
3      house = Rectangle((Point(-100, 100), Point(100, 100)))
4      house.setFill('blue')
5      door = Rectangle(Point(-50, -100), Point(0, 200))
6      door.setFill('brown')
7      roof1 = Line(Point(-100, 100), Point(0, 200))
8      roof2 = Line(Point(0, 200), Point(100, 100))
9      roof1.setWidth(3)
10     roof2.setWidth(3)
11     myCanvas.draw(house)
12     myCanvas.draw(door)
13     myCanvas.draw(roof1)
14     myCanvas.draw(roof2)
15     sun = Circle(Point(150, 250), 20)
16     sun.setFill('yellow')
17     myCanvas.draw(sun)
```

Listing 12.1 Using geometric objects to draw a house

Description


Python Turtle Graphics

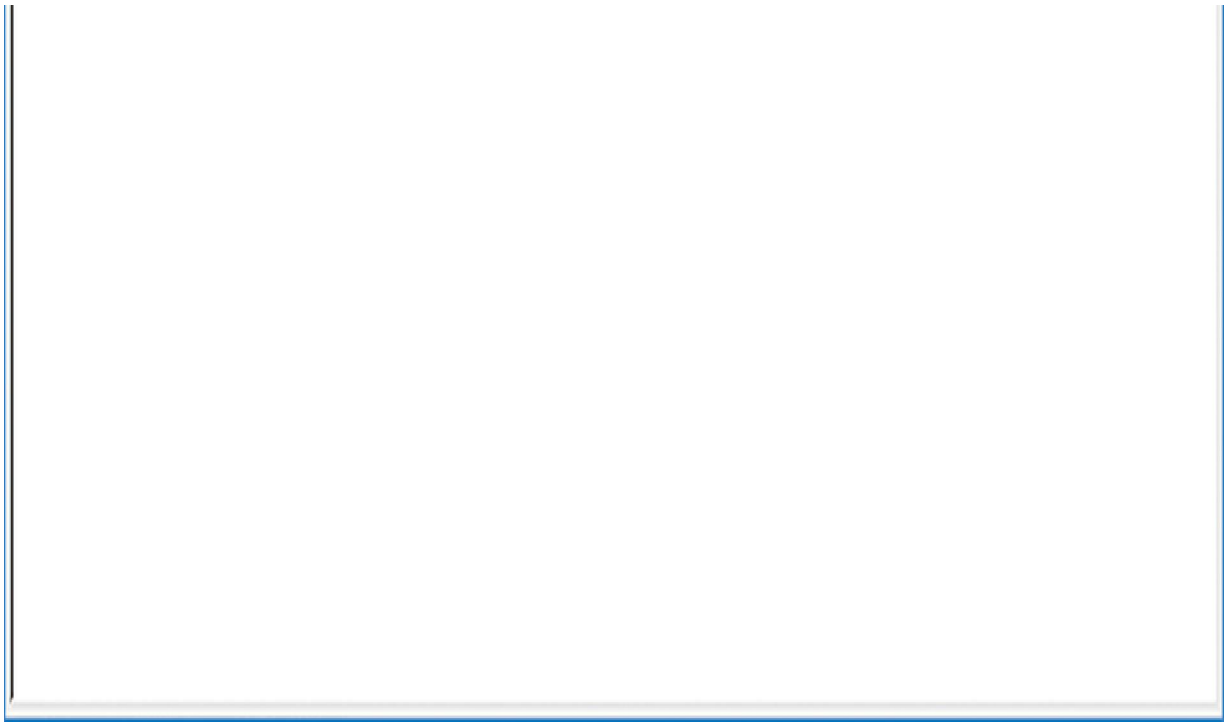


Figure 12.1 Using object-oriented graphics to draw a simple scene.

© 2001–2025 Python Software Foundation. All Rights Reserved.

In the rest of this chapter, we will design and implement a set of classes that enable us to write the function in Listing 12.1. Along the way we will learn about a powerful programming concept called **inheritance**.

12.3 First Design

When designing a project like our graphics module, it is a good idea to begin by making a list of the kinds of objects involved and the relationships among those objects. As we make the list of objects, we will try to identify two important kinds of relationships among the objects: IS-A and HAS-A.

The **IS-A relationship** describes two objects, one of which is a more specific instance of the other. For example, a square *is a* more specific instance of a rectangle, and a circle *is a* more specific instance of an ellipse. As we identify these IS-A relationships, we also look for functionality that each of the instances has in common. Classes with IS-A relationships are ideal for a design using inheritance.

A **HAS-A relationship** describes two objects, one of which uses the other. For example, if you think about a circle, each circle *has a* center point. A rectangle *has a* lower-left-corner point and an upper-right-corner point. Classes with HAS-A relationships are implemented using a design approach called **composition**. With composition, one class *has an* instance variable that is an object of the other class.

GOOD PROGRAMMING PRACTICE:

When designing a project, first identify the objects and look for IS-A and HAS-A relationships. An IS-A relationship is implemented using inheritance, while a HAS-A relationship is implemented using composition.

With these definitions in mind, let us now create a list of objects for our graphics module. An initial set of objects might look like the following:

- Square
- Circle
- Ellipse

- Rectangle
- Triangle
- Polygon
- Line
- Point

As you think about the words in the list, you will recognize that there are IS-A relationships between several of them. A Rectangle *is a* Polygon. A Square *is a* Rectangle. A Triangle *is a* Polygon. A Circle *is an* Ellipse. You may also recognize that all the words we have just mentioned are shapes. The word *shape* is not on our list of objects, but it may be a good idea to add Shape because it is an abstraction of many of the words we have listed. As you design an application, this will often be the case. As you organize the objects you have identified, you will very often uncover new, more abstract objects.

The next step is to arrange our words into a diagram that makes the IS-A relationships explicit. **Figure 12.2** illustrates the relationships between the objects we have discussed so far. This diagram uses **Unified Modeling Language (UML)**, which is a graphical tool for communicating relationships between classes. The arrows indicate inheritance. For example, a Polygon will inherit from a Shape, a Circle will inherit from an Ellipse, and so on. The most abstract or general object appears at the top of the **hierarchy**. At each lower level, the objects become more specific.

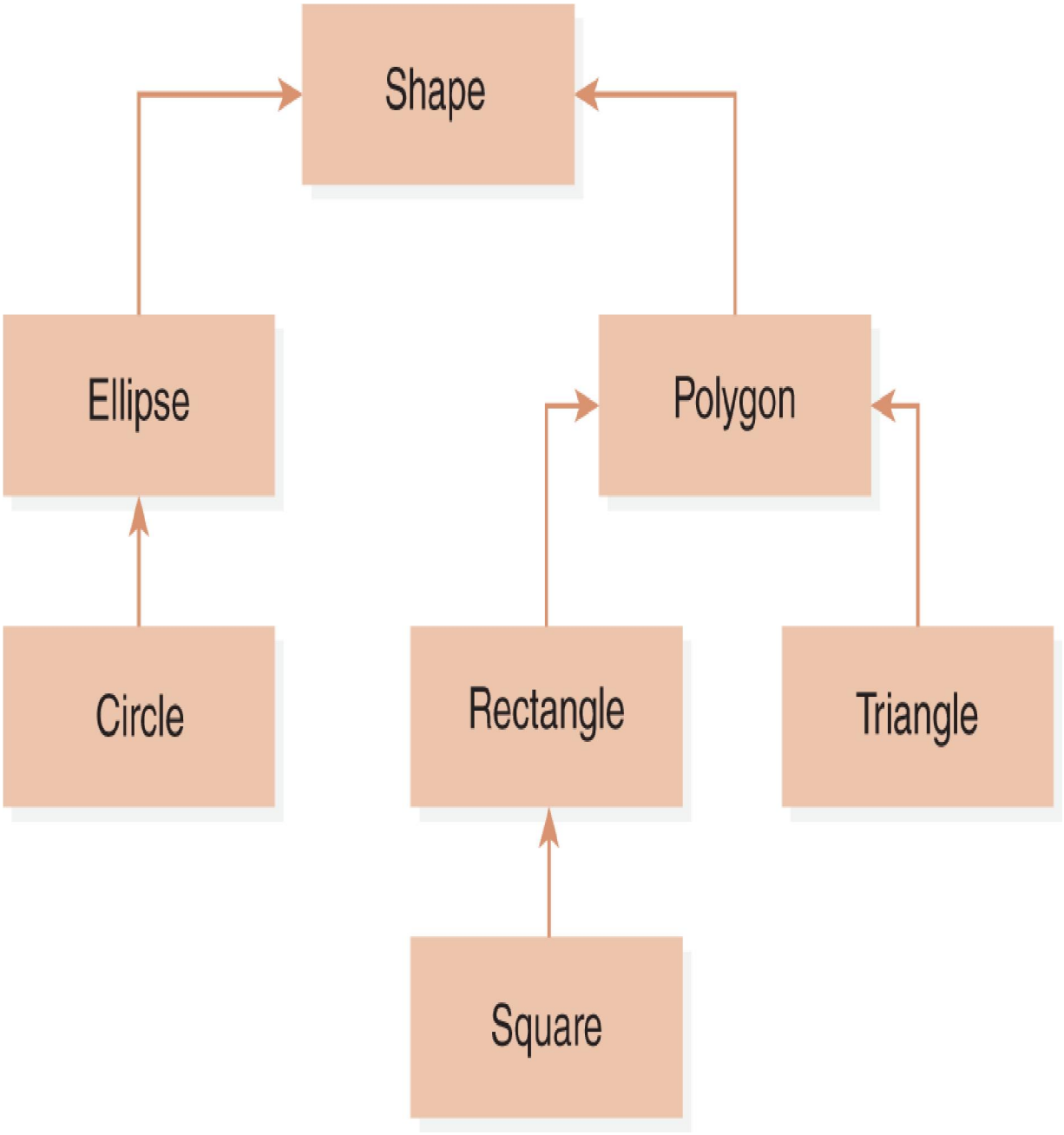


Figure 12.2 A simple object hierarchy.

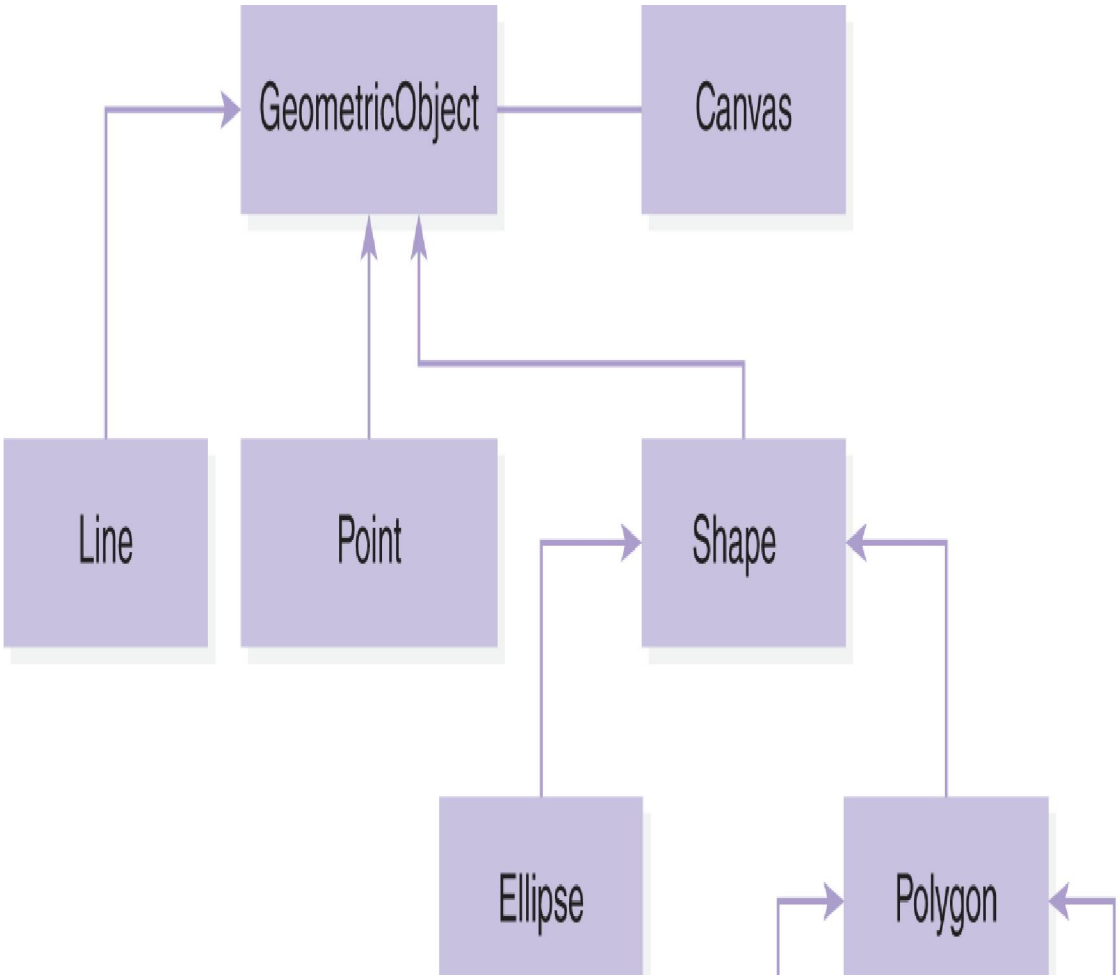
Description

When you look at Figure 12.2 and think about the words on our list, you may have some questions: What about Point and Line? Where do these objects belong? Should Lines and Points be included in the shapes hierarchy?

Is there another abstraction that we could use to relate Lines and Points to the others?

We might determine that although Lines and Points are not shapes, they are geometric objects. Likewise, shapes are geometric objects, so we might create another abstraction called GeometricObject.

Here's another question you might ask: What about the Canvas? The Canvas is not a GeometricObject or a Shape. Instead, the Canvas is the place where we will draw our lines, points, and other shapes. The relationship between Canvas and GeometricObject is a HAS-A relationship. A Canvas may contain (HAS-A) many geometric objects. This new hierarchy can be seen in **Figure 12.3**. This HAS-A relationship is indicated by a plain line drawn between the two classes. Similarly, most GeometricObjects will have one or more Points. To avoid muddying the diagram at this point, we do not indicate all these HAS-A relationships, but instead concentrate on displaying the IS-A relationships.



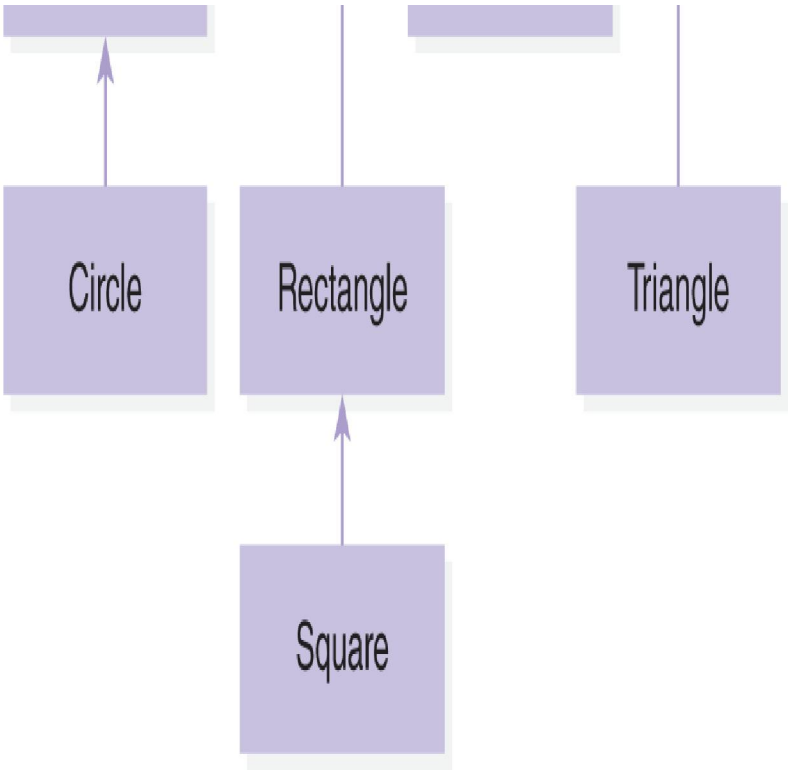


Figure 12.3 An expanded object hierarchy.

Description

With this hierarchy of objects complete, we can begin to think about implementing the objects as Python classes. In this chapter, we will use the hierarchical relationships to avoid the duplication of code.

In object-oriented programming, the IS-A links, indicated by arrows in Figure 12.2, define an **inheritance hierarchy**. Inheritance is the idea that a more general class is a **parent class** of a more specific class (a **child class**). When two classes are connected by an IS-A link, we call a class that is pointed to by another class in the inheritance hierarchy a **superclass**. The class where the IS-A link originates is called the **subclass**.

The superclass (parent) class may have methods that can be shared with any subclass (child). By defining shared methods, we can avoid duplicating code. In our example, the `Rectangle` class is a child of the `Polygon` class, which is a child of the `Shape` class.

In contrast, the HAS-A relationships are implemented using a design approach called composition. In composition, one class *has* an object of

another class as an instance variable.

RECAP:
The IS-A links define an inheritance hierarchy. A subclass (child class) inherits the methods and instance variables of the superclass (parent class). The HAS-A relationships are implemented using composition.

In the next stage of our design, we need to ask the following questions:

- What things should each object know?
- What things should each object be able to do?

The things we want each object to know will lead us to the set of instance variables. The things that each object can do will give us the set of methods we need to write.

We want our objects to remember several things about themselves:

- Fill color
- Position on the canvas
- Line width

If you look back at the `drawHouse` function in Listing 12.1, you will quickly come up with a list of things that our shapes should be able to do:

- Set or change the fill color
- Set or change the width of a line or outline
- Draw themselves

We can now talk about the advantages of using inheritance. Because all shapes have the same set of attributes, we can create the instance variables and write the methods to modify those instance variables in the `Shape` class. We can do this because whenever a method is defined in a superclass, the method can also be called by an instance of the subclass, or even any of the subclasses of the subclass. This can save us an enormous amount of work and

is one reason that spending time up front creating a good design is so important.

Good programming practice calls for developing a large program one step at a time. To help us with this task, we will write a simple test program that implements just a small part of the overall functionality we want to create. Once we get this simple program working, we can move on to something more complicated. Writing programs this way is much easier and more fun than writing all the code for a large program first and then trying to figure out where things went wrong.

GOOD PROGRAMMING PRACTICE:

Develop a program one step at a time. Test the functionality in each step before adding more code.

One of the easiest programs to write that moves us a long way toward our end goal is to draw a line on the screen. This program is shown in **Listing 12.2**. Although the program looks simple, it actually forces us to do some more design on the classes `Point`, `Line`, `GeometricObject`, and `Canvas`.

```
1 myCanvas = Canvas(800, 600)
2 myLine  Line(Point(-100, -100), Point(100, 100))
3 myCanvas.draw(myLine)
```

Listing 12.2 Drawing a line

Description

The `Canvas` class is responsible in some way for drawing a `GeometricObject`. How will `Canvas` do this? The answer: It will use a turtle. The turtle comes with a built-in window for drawing. With all of the turtle's functionality, implementing the `Canvas` class should be relatively

easy. But it does mean that one of the instance variables of the `Canvas` will be a `Turtle`, so it follows that another instance variable will be a turtle `Screen`. We will also need to write a `draw` method for the `Canvas` that can take any shape we give it and draw it using the turtle. Also, the `Canvas` will need to keep track of the `GeometricObject`s that have been drawn on the canvas. Again, we will preface our instance variable names with two underscores (_ _). This naming scheme hides each instance variable from code outside the class. Thus, we achieve encapsulation because only the code in the class can alter the value of an instance variable.

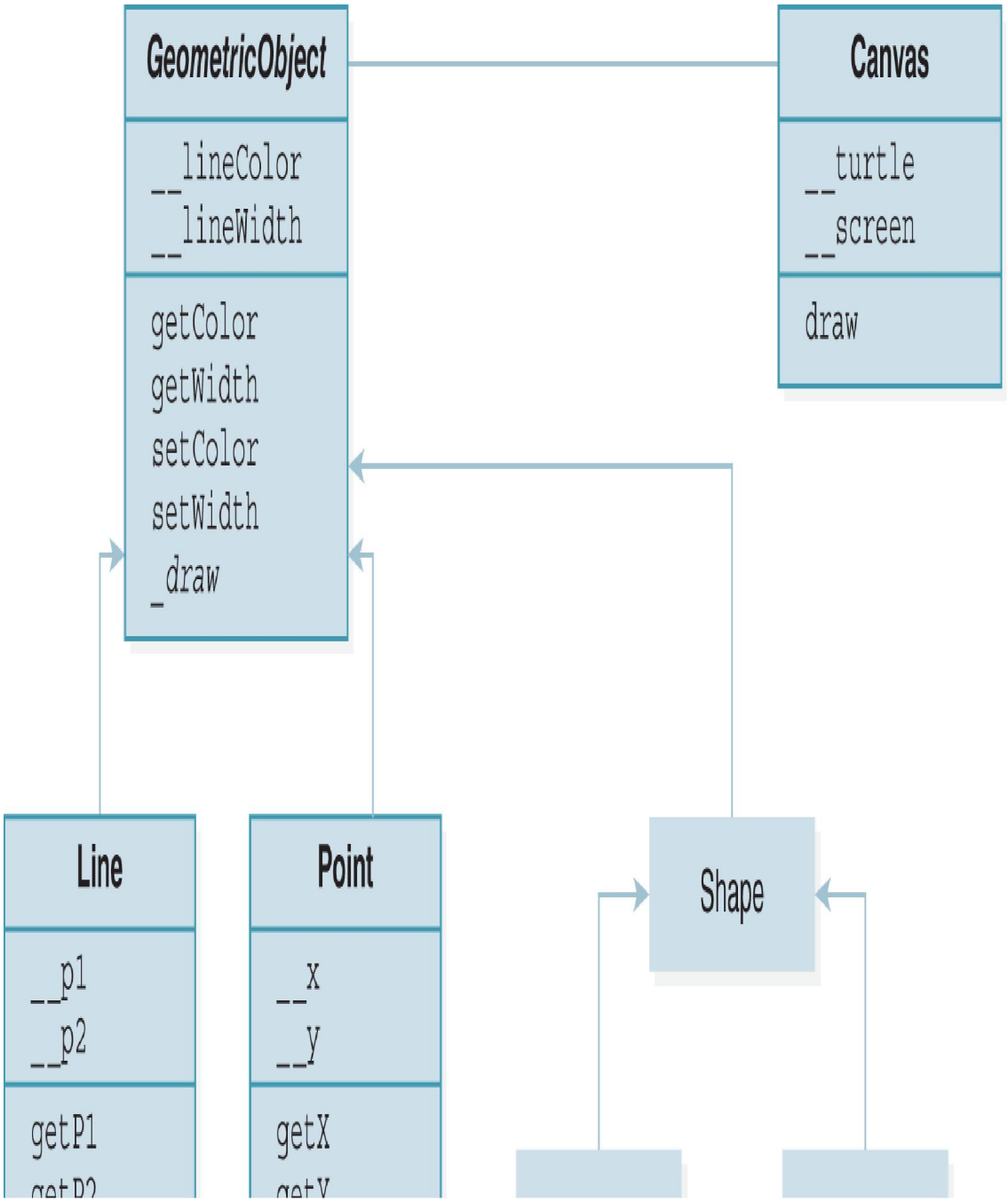
It follows, then, that although a subclass inherits the instance variables of the superclass, the subclass will not have direct access to those hidden instance variables. Thus, it is important for the superclass to provide accessors and mutators for its instance variables so that the subclass can access them.

The `GeometricObject` class will have the instance variables `_ _lineColor` and `_ _lineWidth`. We also know that the `GeometricObject` class will need methods for getting and setting the `_ _lineColor` and `_ _lineWidth` instance variables. Although we will not change the color or width of the lines in our simple test program, we will still implement the methods now.

The `Point` class, which will inherit from the `GeometricObject` class, is an abstraction for a geometric point in our two-dimensional coordinate system. The two instance variables of a `Point` are the *x* coordinate and the *y* coordinate. In addition, the `Point` class inherits the `_ _lineColor` and `_ _lineWidth` instance variables from `GeometricObject`. However, a `Point` needs to define only the following instance variables: `_ _x` and `_ _y`. In addition to the methods inherited from the `GeometricObject` class, the `Point` class provides three simple methods: `getX` returns the `x` coordinate, `getY` returns the `y` coordinate, and `getCoord` returns the `x` and `y` coordinates as a tuple. As you see, an advantage of inheritance is that the `Point` class needs only to write the methods to handle its specific instance variables. For this example, we have decided not to write mutator methods for setting the *x* and *y* values. Once a shape has been created and drawn, there is no need to change its location.

The `Line` class *has* two instance variables: a beginning `Point` and an ending `Point`. This is an example of composition: Like `Point`, `Line` inherits the instance variables `__lineColor` and `__lineWidth` and their associated methods from `GeometricObject`, so we do not explicitly implement them as part of the `Line` class.

Figure 12.4 shows our inheritance hierarchy with the details of instance variables and methods for each class added to the diagram.



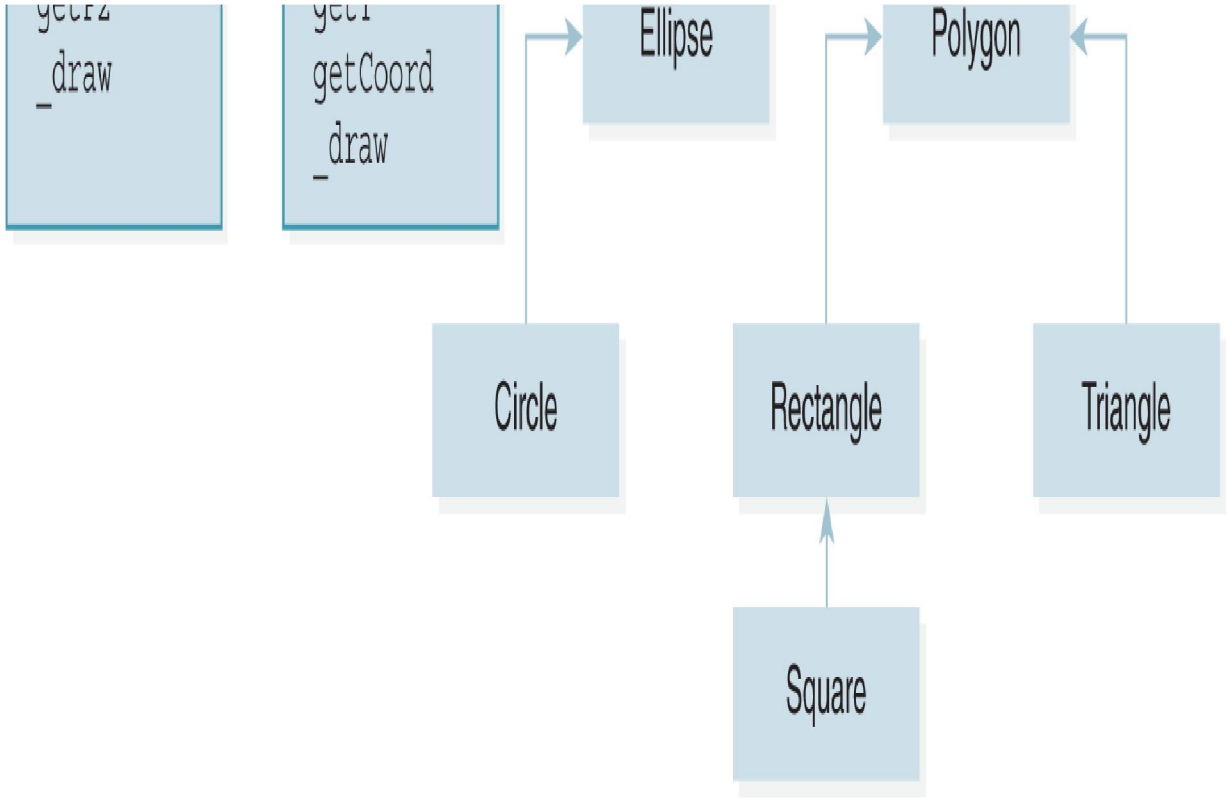


Figure 12.4 Instance variables and methods.

Description

12.4 Basic Implementation

Let's begin with a top-down implementation of the portion of the class hierarchy needed to implement the program in Listing 12.2. We will start with the `Canvas` class (shown in **Listing 12.3**), which serves the same purpose for our graphics module as it does for the turtle. It is a place for us to draw `GeometricObjects`. In fact, because we are using the turtle to do all of our behind-the-scenes drawing, we will make use of the same window and canvas as the turtle. This decision saves us a lot of work but also has some important ramifications for our program. For example, it means that we will use the turtle's coordinate system.

```
1  import turtle
2
3  class Canvas:
4      def __init__(self, w, h):
5          self._turtle = turtle.Turtle()
6          self._screen = turtle.Screen()
7
8          self._screen.setup(width = w, height = h)
9          self._turtle.hideturtle()
10
11     def draw(self, gObject):
12         self._turtle.up()        #prepare to move turtle without a trail
13         self._screen.tracer(0)   #turn off animation
14         gObject._draw(self._turtle)
15         self._screen.tracer(1)   #turn on animation
```

Listing 12.3 The `Canvas` class

Description

12.4.1 The `Canvas` Class

The most difficult task in writing our `Canvas` class is writing the constructor. As you have previously learned, the constructor method is always named `__init__`. The job of the constructor for `Canvas`, which takes the width and height of our drawing area as parameters, is to create an instance of a `Turtle` and store a reference to it in an instance variable called `__turtle`. We will also create an instance variable called `__screen` that will allow us to configure the canvas.

Once we have initialized the instance variables for the class, we set the width and height of the `Canvas` using the `Screen` method `setup`. Finally, because the people who will be using our graphics classes may not know anything about turtle graphics, we will hide the turtle using the `hideturtle` method. We do not want them to know that a `turtle` is actually doing the work.

The final method of our `Canvas` class is the `draw` method, which is simple to implement but exposes one of the most powerful and important parts of object-oriented programming. Notice that `draw` accepts a geometric object `gObject` as a parameter. After putting the turtle into a tail-up state and turning tracking off, the `draw` method just calls the `_draw` method (one underscore) on the `gObject`, passing our turtle as a parameter!

Let's be very clear, because these names may be confusing. The `draw` method belongs to the `Canvas`, whereas the `_draw` method (with one underscore) belongs to the `GeometricObject`. Although `_draw` may look a bit strange to you, it is a perfectly legal name for a function. We use `_draw` in this case to differentiate it from the `draw` method in the `Canvas` class and to prevent programmers from inadvertently calling the `draw` method directly on a `GeometricObject`.

If you think about it, you may realize that the formal parameter of the `draw` method can be any `GeometricObject`. As a consequence, the `Canvas` does not know whether it should draw a `Point`, a `Line`, or any other `Shape`. We could solve this problem by using the following code:

```
if isinstance(gObject, Point):
    #code to draw a Point
elif isinstance(gObject, Line):
    #code to draw a Line
elif isinstance(gOBject, Circle):
    #code to draw a Circle
...
else:
    print('unknown geometric object')
```

Description

While using `isinstance` to check the type of `gObject` would certainly work, it is cumbersome to use. Also, each time we create a new shape, we would need to add code to check for that new shape. Fortunately, object-oriented programming provides us with **polymorphism**—a much more powerful way to handle exactly this kind of problem and an approach that allows us to write a `_draw` method for each of our specific geometric objects. Python is smart enough to call the right method, depending on the type of object referenced by the `gObject` parameter. After we have implemented the `_draw` methods, we will discuss the underlying mechanism that Python uses to implement polymorphism.

12.4.2 The `GeometricObject` Class

Let's continue our implementation by writing the `GeometricObject` class. `GeometricObject` and `Canvas` are peers at the top level of our inheritance hierarchy. Recall that the `Canvas` *has* one or more `GeometricObjects`.

Although the `GeometricObject` class will contain the instance variables and methods that are common to all its subclasses, we do not intend for any objects to be created as a `GeometricObject`. Instead, we create this class because it is an abstraction of its child classes: `Point`, `Line`, `Rectangle`, and others. To that end, we define the `GeometricObject` class as an **abstract class**. An abstract class provides us with one convenient place to define instance variables and methods used by all of its child classes.

To help in defining abstract classes, Python provides a module called `abc` (Abstract Base Class). **Listing 12.4** shows the `GeometricObject` class, in which we first import the `abc` module. Then we specify that the `GeometricObject` class inherits from the `ABC` class by putting `ABC` in parentheses following the class name (line 2). By defining the class as `abstract`, we make it impossible to create a `GeometricObject` instance.

```
1  from abc import *
2  class GeometricObject(ABC):          #inherit from Abstract Base Class
3      def __init__(self):
4          self._lineColor = 'black'  #default drawing color
5          self._lineWidth = 1        #default line size
6
7      def getColor(self):
8          return self._lineColor
9
10     def getWidth(self):
11         return self._lineWidth
12
```

```
13      def setColor(self, color):
14          self._lineColor = color
15
16      def setWidth(self, width):
17          self._lineWidth = width
18
19      @abstractmethod                 #indicate that _draw is abstract
20      def _draw(self, someturtle):
21          pass                        #do nothing
```

Listing 12.4 The `GeometricObject` class

Description

The `GeometricObject` class contains two instance variables: `_lineColor` and `_lineWidth`. Because these variables are initialized by the `GeometricObject` constructor, they do not need to be initialized by any of the child classes.

An important aspect of inheritance is that each class handles only the instance variables it defines. Recall that the `Canvas` class expects that a `GeometricObject` will have a method named `_draw` that takes a turtle object as a parameter. But a `GeometricObject` has nothing to draw. Instead, we want each subclass of `GeometricObject` to provide its own `_draw` method. We can enforce this rule by defining the `_draw` method in the `GeometricObject` class as an **abstract method**. We do this by using a Python syntax element called a **decorator**. The decorator we use is `@abstractmethod` (line 19), which identifies this method as being abstract. Decorators must immediately precede the method and must be on a line by

themselves. Then we define the `_draw` method as a do-nothing method containing only the `pass` statement. As described in **Table 12.1**, the `pass` statement serves as a placeholder when a statement is needed, but no code is appropriate. With the `_draw` method defined as abstract, any subclass of `GeometricObject` must either provide a nonabstract implementation of the `_draw` method or that class will also be abstract. A class that is not abstract and from which objects can be created is called a **concrete class**.

TABLE 12.1 The `pass` Statement

Statement	Description
`pass`	A statement that does nothing when executed. Useful when a statement is needed, but no code is appropriate.

12.4.3 The `Point` Class

The `Point` class is important because it is used by all other `GeometricObject`s for positioning. For example, the `Line` class *has* two `Point` objects that anchor the ends of the `Line`. The `Rectangle` class *has* two `Point` objects that position the opposing corners of the `Rectangle`, and so on. The code for the `Point` class is given in **Listing 12.5**. The `Point` class inherits from the `GeometricObject` class; thus, the `Point` class is a subclass of `GeometricObject`. The `Point` class designates `GeometricObject` as the superclass by putting `GeometricObject` in parentheses after the class name (line 1). The syntax we use to designate one class as the superclass of another is to place the superclass name in parentheses after the subclass name.

```
class Point(GeometricObject):
    def __init__(self, x, y):
        super().__init__()
        self.__x = x
        self.__y = y

    def getCoord(self):
        return (self.__x, self.__y)

    def getX(self):
        return self.__x

    def getY(self):
        return self.__y

    def _draw(self, turtle):
        turtle.goto(self.__x, self.__y)
        turtle.dot(self.getWidth(), self.getColor())
```

Listing 12.5 The `Point` class

Description

Another way that we can speak about the parent–child relationship between classes is to say that `Point` extends `GeometricObject`. Specifically, `Point` begins with the instance variables and methods provided by `GeometricObject` and then adds new instance variables and methods of its own.

Two key ideas are important in the `Point` class. First, line 3

```
super().__init__()
```

makes sure that the `__init__` method in the superclass (`GeometricObject`) is called so that the instance variables inherited from `GeometricObject` are given their proper initial values. The `super` function returns a special super-object that knows how to properly call the `__init__` method for the superclass.

RECAP:

To designate the inheritance source, a subclass includes the superclass name in parentheses in the class definition. The constructor of the subclass should call the constructor of the superclass to initialize the instance variables of the superclass.

Second, the `Point` class defines a `_draw` method that implements the drawing of a `Point`, using the turtle that is passed as a parameter. This is important because the `GeometricObject` superclass has defined the `_draw` method as abstract, requiring all subclasses to define a concrete method named `_draw`. If the `Point` class did not define a `_draw` method, then the `Point` class would be abstract and Python would generate a type error on any attempt to create a `Point` object.

In the `_draw` method, the `goto` method moves the turtle to the correct place on the canvas. Its tail is up because the `Canvas` object lifts the tail before

calling the `_draw` method. The parameters passed to `goto` are the instance variables `self._ _x` and `self._ _y`. Once the turtle is in position, a dot is drawn using the instance variables for the line width and color defined in the superclass.

Notice that the `Point` class does not define `_ _lineWidth` and `_ _lineColor`. Instead, the `Point` class automatically inherits these instance variables from the `GeometricObject` class. However, because the line width and color are defined beginning with double underscores in the `GeometricObject` class, the `Point` class must call the accessor methods in the superclass. Thus, line 18 is written as

```
turtle.dot(self.getWidth(), self.getColor())
```

Description

HEADS UP:
A subclass does not define the instance variables and methods of the superclass. The subclass needs only to define the instance variables and methods specific to the subclass.

12.4.4 The `Line` Class

The `Line` class is similar to the `Point` class, except that `Line` has two instance variables that are `Points`. Also, the `_draw` method for `Line` draws a line rather than a simple point. Notice that we make two calls to the turtle to set the current line color and the line width. This is an important step because we cannot make any assumptions about what color or line width the turtle is set to use. The code for the `Line` class is given in **Listing 12.6**.

```
1  class Line(GeometricObject):
2      def __init__(self, p1, p2):
3          super().__init__()
4          self.__p1 = p1
5          self.__p2 = p2
6
7      def getP1(self):
8          return self.__p1
9
10     def getP2(self):
11         return self.__p2
12
13     def _draw(self, turtle):
14         turtle.color(self.getColor())
15         turtle.width(self.getWidth())
16         turtle.goto(self.__p1.getCoord())
17         turtle.down()
18         turtle.goto(self.__p2.getCoord())
```

Listing 12.6 The `Line` class

Description

12.4.5 Testing Our Implementation

With our first four classes written, let's try our first test program and see what we can learn about our implementation. Assume that all the classes we have defined so far are stored in `draw.py` and that `turtle` is imported at the top of that file. **Session 12.1** creates a `Canvas` and a `Line`, and it draws the line. If everything works, you should see a window with a black diagonal line in it.

```
1 from draw import *
2 myCanvas = Canvas(800,600)
3 myLine = Line(Point(-100, -100), Point(100, 100))
4 myCanvas.draw(myLine)
```

Session 12.1 First test of graphics classes

Description

Let's continue to explore this example and see what we can learn about the objects we have created. **Session 12.2** on the next page illustrates that a `Line` and a `Point` are both `GeometricObjects`. We also illustrate that the `Line` and `Point` classes have inherited the `getColor` and `getWidth` methods from the `GeometricObject` class. In addition, the `Point` class has defined the `getX` and `getY` methods. First, we verify that `myLine` has the default color and width. Next, we get one of the `Points` used to create the line and check its width and color, along with its `x` and `y` coordinates.

```
In [1]: 1 from draw import *
        2 myLine = Line(Point(-100, -100), Point(100, 100))
```

In [2]:
```
1 isinstance(myLine, Line)
```

Out[2]: True

In [3]:
```
1 isinstance(myLine, GeometricObject) #Line IS-A GeometricObject
```

Out[3]: True

In [4]:
```
1 myLine.getWidth()    #call inherited methods
```

Out[4]: 1

In [5]:
```
1 myLine.getColor()
```

Out[5]: 'black'

In [6]:
```
1 p = myLine.getP1()   #get Point object from Line
```

In [10]:
```
1 isinstance(p, Point)
```

Out[10]: True

In [11]:
```
1 isinstance(p, GeometricObject) #Point IS-A GeometricObject
```

Out[11]: True

In [12]:
```
1 p.getWidth()        #call inherited methods
```

Out[12]: 1

In [13]:
```
1 p.getColor()
```

Out[13]: 'black'

In [14]:
```
1 p.getX()            #call Point methods
```

Out[14]: -100

```
In [15]: 1 p.getY()

Out[15]: -100

In [16]: 1 p.getCoord()

Out[16]: (-100, -100)
```

Session 12.2 Investigating inheritance

Description

Try It Out

12.1 Modify the test program to change the line color and width.

12.2 Rewrite the `Point` class, but rather than storing the coordinate information as separate numbers, store the coordinates as a tuple. When you are done, use your `Point` class in place of the `Point` class we have written and make sure the test program still works.

12.3 Modify the `Line` class so that a line can be either solid or dashed.

12.4 Write a program that plots a sine wave using instances of the `Point` class. *Note*: Look into the `sin` and `radians` functions of the `Math` module.

12.5 Create an instance of a `GeometricObject` and then try to draw it. Explain why you get an error message.

12.5 Understanding Inheritance

To better understand how inheritance works, we can look behind the scenes at how Python keeps track of instance variables and methods. Each class and instance of a class has some special instance variables that we can use to see how Python finds the right method to call or the right instance variable.

First, the `__bases__` instance variable tells us the name of the superclass of any class. For example, in **Session 12.3**, you can see that the superclass of `Line` is `GeometricObject`. The parent of `GeometricObject` is `ABC` (Abstract Base Class). The parent of `ABC` is `object`, and `object` has no parent. By default, Python uses the special class `object` at the top of the inheritance hierarchy. This chain of child-to-parent links is very important for Python when a method is called that was defined in a parent class.

```
In [1]:  1 from draw import *
         2 myLine = Line(Point(-100, -100), Point(100, 100))
```

```
In [2]:  1 Line.__bases__
```

```
Out[2]: (draw.GeometricObject,)
```

```
In [3]:  1 GeometricObject.__bases__
```

```
Out[3]: (abc.ABC,)
```

```
In [4]: 1 ABC.__bases__
```

```
Out[4]: (object,)
```

```
In [5]: 1 object.__bases__
```

```
Out[5]: ()
```

```
In [6]: 1 Line.__dict__.keys()
```

```
Out[6]: dict_keys(['__module__', '__init__', 'getP1', 'getP2', '_draw', '__doc__',
        '__abstractmethods__', '_abc_impl'])
```

```
In [7]: 1 GeometricObject.__dict__.keys()
```

```
Out[7]: dict_keys(['__module__', '__init__', 'getColor', 'getWidth', 'setColor',
         'setWidth', '_draw', '__dict__', '__weakref__', '__doc__',
        '__abstractmethods__', '_abc_impl'])
```

```
In [8]: 1 myLine._class_

Out[8]: draw.Line

In [9]: 1 myLine._dict_.keys()

Out[9]: dict_keys(['_GeometricObject_lineColor', '_GeometricObject_lineWidth',
        '_Line_p1', '_Line_p2'])
```

Session 12.3 Investigating the inheritance hierarchy

Description

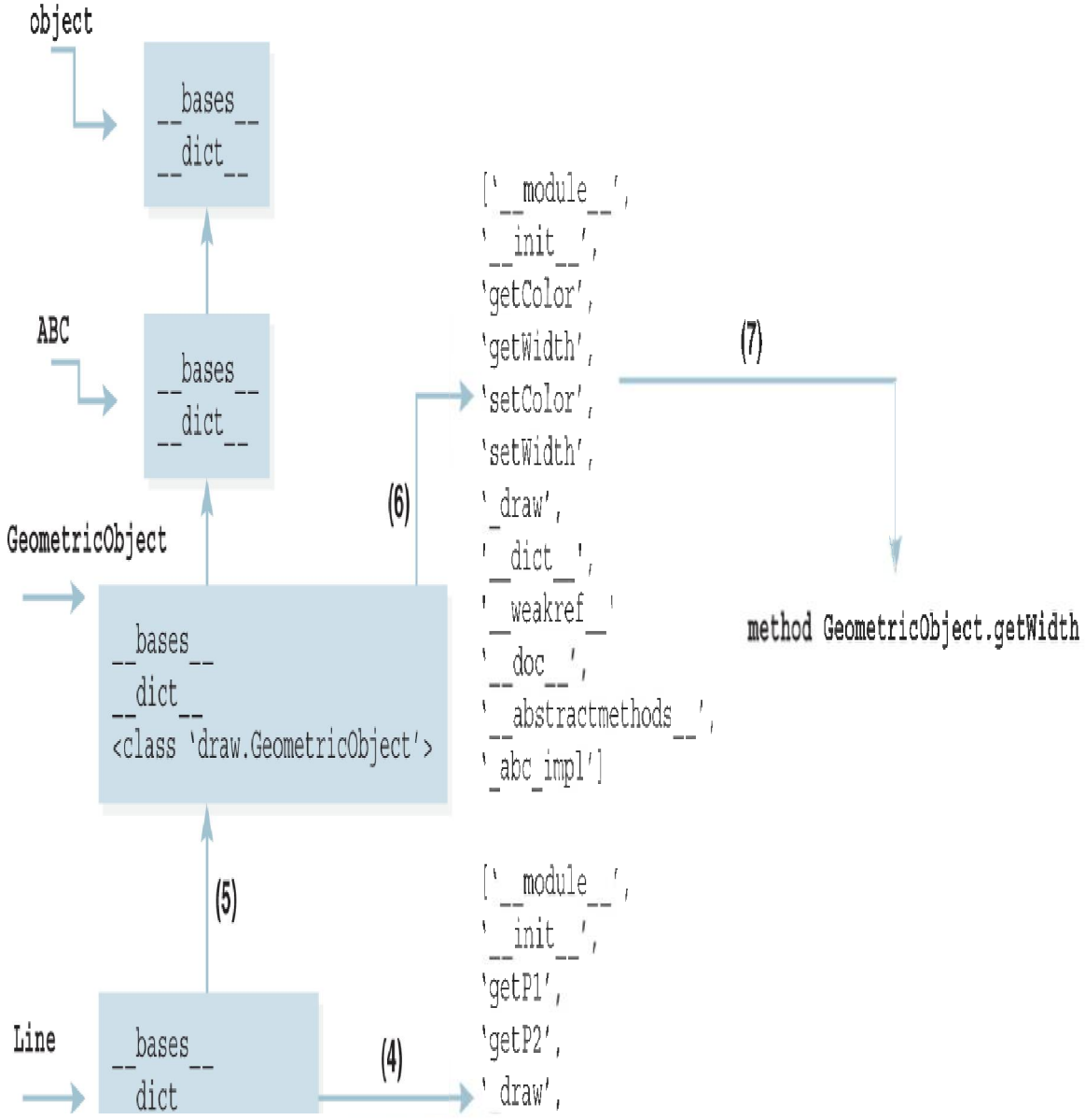
The name `__bases__` comes from the term *base class*, which is an alternative way of thinking about the superclass–subclass relationship. Some programmers say that the superclass is the **base class** and the subclass **extends** the base class. In our hierarchy, `GeometricObject` is the base class of the `Line` class. The critical idea is that `Line` has all the functionality of `GeometricObject`, plus new functionality that extends the base class.

Another instance variable that Python maintains for each class is the `__dict__` variable. This instance variable is a dictionary that keeps track of the methods defined for the class. For example, in Session 12.3 you can see that the dictionary of methods for the `Line` class has keys that correspond to the name of every method we defined.

Instances of Python objects also have some special instance variables. Every object has an instance variable called `__class__`, which contains a reference to the object's class. In Session 12.3, you can see that `myLine.__class__` refers to the `Line` class. In addition, the `__dict__` variable

for instances of classes contains a dictionary of user-defined instance variables. For example, Session 12.3 shows that the `myLine` instance of the `Line` class also has a `__dict__` instance variable that contains keys for `__lineColor` and `__lineWidth` that are inherited from the `GeometricObject` class, as well as `__p1` and `__p2` defined in the `Line` class.

Putting all these special variables together, we can see in more detail how inheritance works in Python. **Figure 12.5** shows a reference diagram that illustrates the chain of events in the Python interpreter for the expression `myLine.getWidth()`.



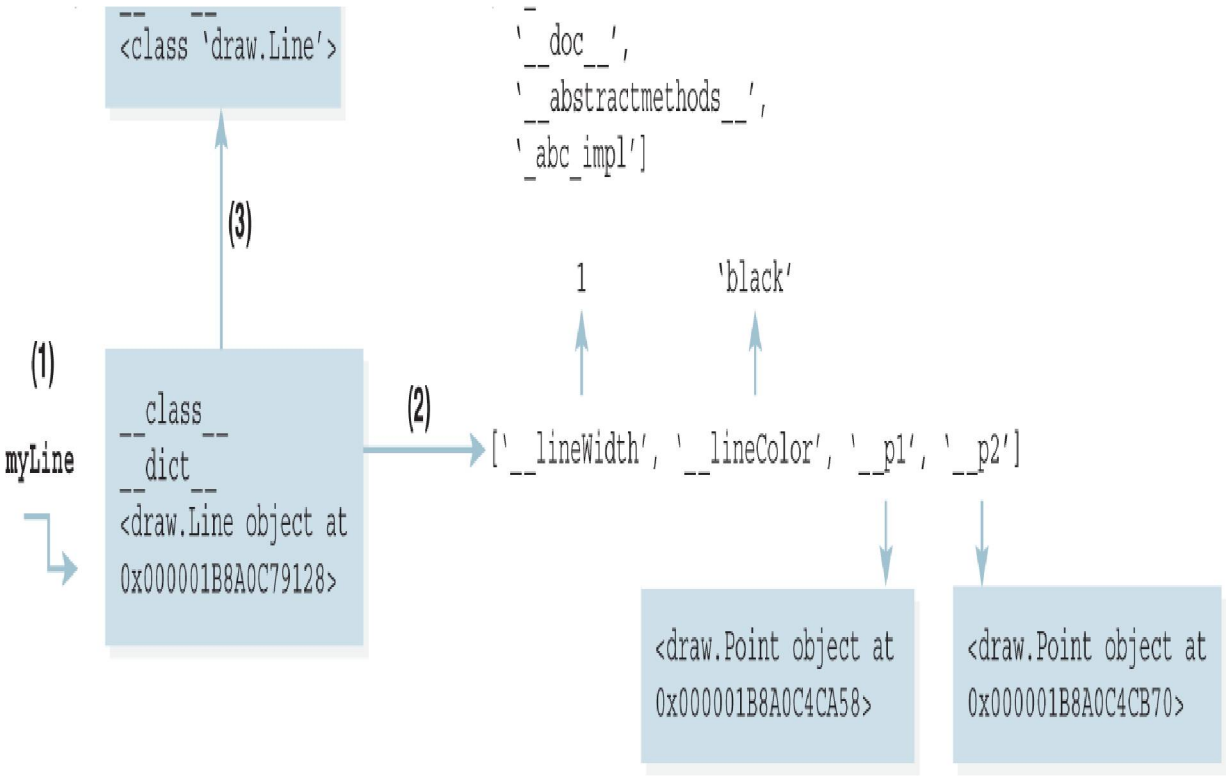


Figure 12.5 A reference diagram for following an inheritance chain. The numbered arrows indicate the order of the search operation.

Description

1. When Python evaluates the expression `myLine.getWidth()`, the first step is to dereference the name `myLine`—that is, to find the object in memory. Python next tries to dereference the name `getWidth` using the following chain of lookups.
2. If `__dict__` in `myLine` contains `getWidth` as a key, stop searching and use it.
3. The method is not in `myLine`, so follow the `__class__` link to the `Line` class.
4. If `__dict__` in the `Line` class contains `getWidth` as a key, stop searching and use it.
5. The method is not in `Line`, so follow the `__bases__` link to the superclass (`GeometricObject`).
6. If `__dict__` contains `getWidth` as a key, stop searching and use it.

7. In this case, we do find the `getWidth` method in the `GeometricObject` class.

Steps 5 and 6 can be repeated until all of the classes listed in `__bases__` have been tried. If all base classes are exhausted and `getWidth` is not found, then generate an error.

Once the name `getWidth` is dereferenced, the next step is to apply the function call operators `()`. Because we are applying the call operators to a method of a class, we take the value of `myLine` and pass it as the first parameter to the `getWidth` function. In the `getWidth` function, `myLine` is called `self`. **Figure 12.6** illustrates the binding of `myLine` as an actual parameter to the formal parameter `self`. The main thing to notice is that `myLine` and `self` both reference the same instance of a `Line`.

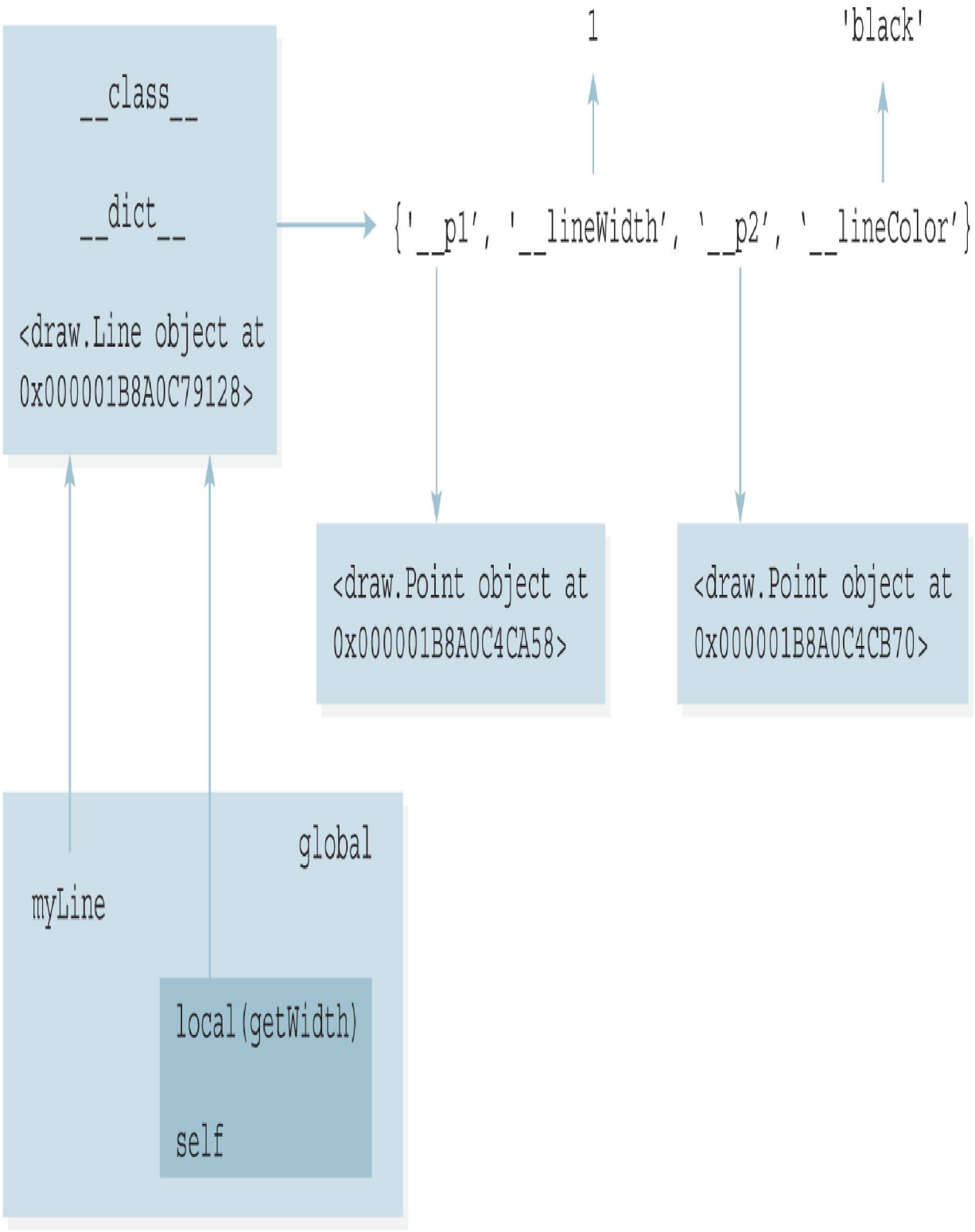


Figure 12.6 The `self` parameter with an inherited method.

Description

Try It Out

12.6 Using the reference diagram in Figure 12.5, trace the series of reference links that Python uses to evaluate the expression `self._lineWidth`.

12.7 Using the reference diagram in Figure 12.5, trace the series of reference links that Python uses to evaluate the expression `myLine._draw()`.

12.8 Draw a reference diagram illustrating the sequence of lookups required for the statement `myLine.getP1().getX()`.

12.9 In the Python shell, follow the sequence of lookups to find the location in memory of `myLine._p1`.

12.6 Limitations

Now that you have a better understanding of how inheritance works, let's try a slightly more difficult test program. **Session 12.4** contains a simple function called `test2` to draw two lines that cross. Once the lines are drawn, the color of the first line is changed to red and the width of the second line is changed to 4. What do you think the final picture will look like?

In [1]:
```
1 from draw import *
```

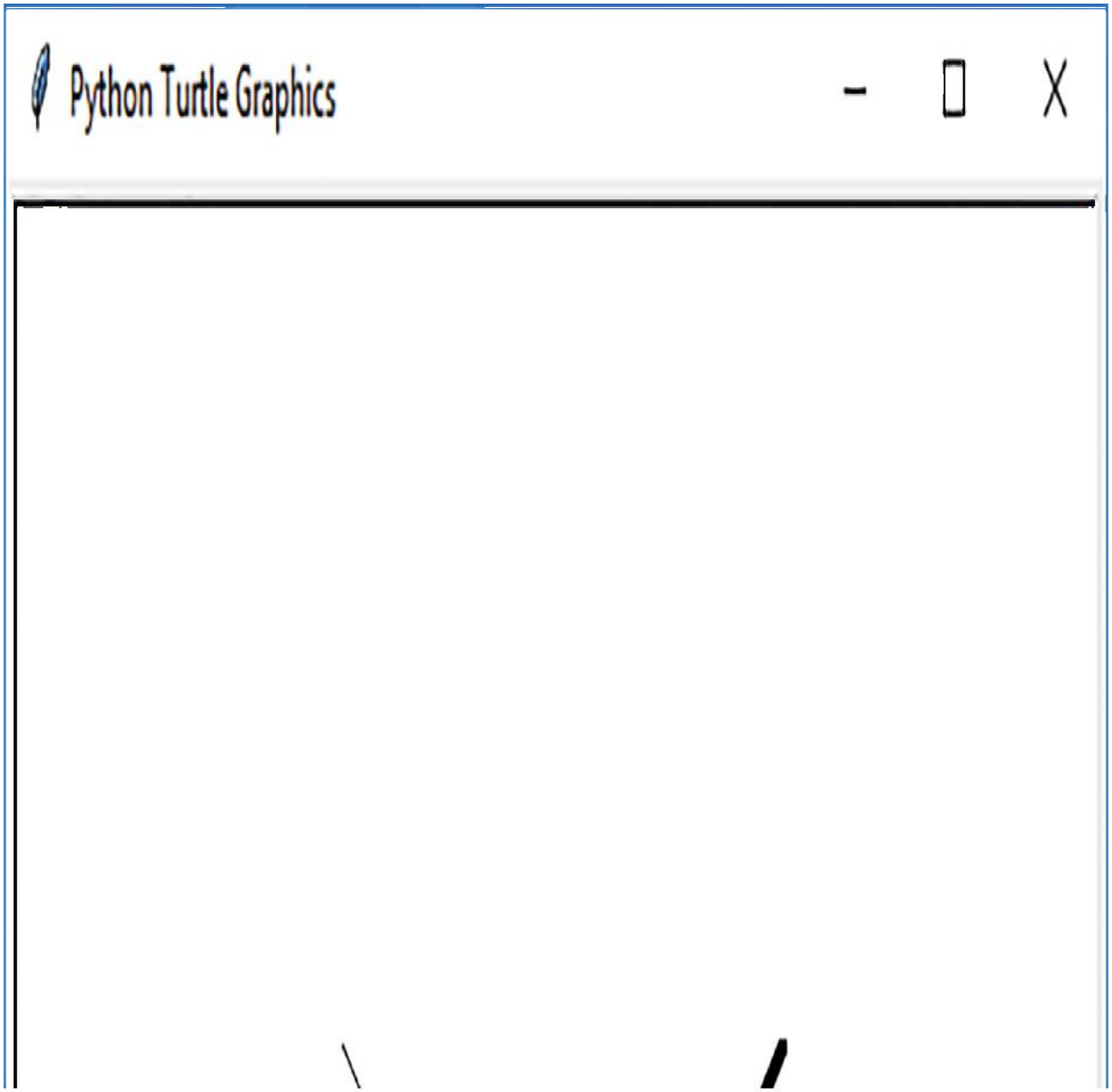
In [2]:
```
1  def test2():
2      myCanvas = Canvas(500, 500)
3      line1 = Line(Point(-100, -100), Point(100, 100))
4      line2 = Line(Point(-100, 100), Point(100, -100))
5      line1.setWidth(4)
6      myCanvas.draw(line1)
7      myCanvas.draw(line2)
8
9      line1.setColor('red')
10     line2.setWidth(4)
11
```

In [3]:
```
1 test2()
```

Session 12.4 Drawing two lines that cross

Description

If you think that the result should be two crossing black lines, one of which is thicker than the other, you are correct, as **Figure 12.7** shows. Why didn't the call to `setColor` actually change the color of `line1`? And why didn't `setWidth` change the width of `line2`? The answer is that calling `setColor` or `setWidth` modifies only the `__lineColor` or `__lineWidth` instance variables; it does not change the way the line appears on the screen. To change the appearance of the line on the screen, we need to redraw it. Although the obvious fix for this small program is to move the `setColor` and `setWidth` calls before the call to `draw`, that is not an effective general solution.



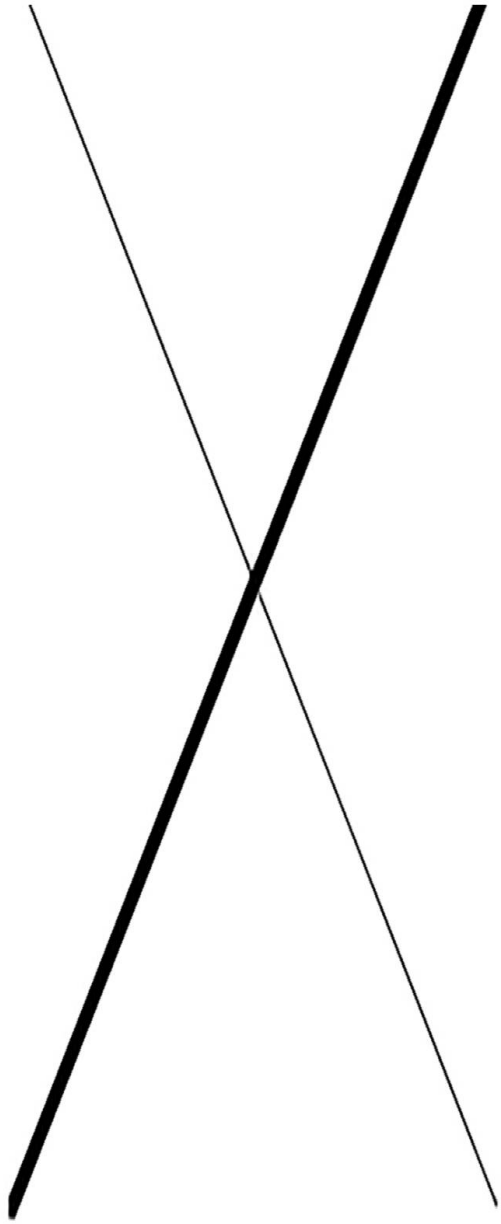




Figure 12.7 Testing the shape hierarchy.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Another possible solution is that every time we change the color or thickness of the line, we tell the canvas to draw the line again. This solution can produce unintended consequences, however. For example, we might add one more line to the program in Session 12.4 to call `myCanvas.draw(myLine)`. Then the line will be red, but it would be in front of the other line. If we want the black line to be in front of the red line, we need to redraw the black line as well. For a complicated drawing, this can turn into a complicated set of dependencies.

The behavior that we really want can be specified as follows:

1. If the object is not drawn yet, change the value of the instance variable.
2. If the object is already visible, change the attribute and redraw all the objects on the screen in the same order as they were originally drawn.

One possible solution to this problem would be to add a call to `_draw` in the `setColor` and `setWidth` methods. However, that creates the same problem with the order of drawing just mentioned. Another problem also arises with this approach: To draw the line, we need a turtle. Recall that the turtle is passed as a parameter to the `_draw` method. However, when we call the `setWidth` method, we do not (and should not!) have a reference to a turtle object. A final problem is that requirement 1 stipulates that if the line has not yet been drawn, the only thing that `setColor` should do is change the value of the color instance variable.

12.7 An Improved Implementation

The solution to the problem outlined previously is valuable to understand because it will help you recognize the principles behind many professional graphics systems, including the TKinter graphics system used by Python. It also helps you understand what is happening in the graphics systems used in the Java programming language. Another benefit of the solution is that once you have the mechanism in place to solve the current problem, you will be able to implement a `move` method for all graphics objects very easily.

The solution to the problem begins with two additional instance variables for the `GeometricObject` class and one additional instance variable and one more method for the `Canvas` class. For the `GeometricObject` class, one variable will keep track of whether the object has been drawn. We might call this instance variable `_ _visible`. The second instance variable, `_ _myCanvas`, will keep track of the `Canvas` on which the object is drawn. The additional instance variable for the `Canvas` object, `_ _visibleObjects`, will allow the `Canvas` to remember an ordered list of the objects it has drawn.

To better understand the importance of these three instance variables, let's think about what will happen when we make a call like the following: `theCanvas.draw(myLine)`.

1. Set the value of the `_ _visible` instance variable of the line to `True`.
2. Set the value of the `_ _myCanvas` instance variable of the line to point to `theCanvas`. At the moment this may seem circular to you, but it should become clear when we talk about what happens when you call `setColor`.
3. Call the line's `_draw` method.
4. Add the line object to the `_ _visibleObjects` instance variable of the canvas.

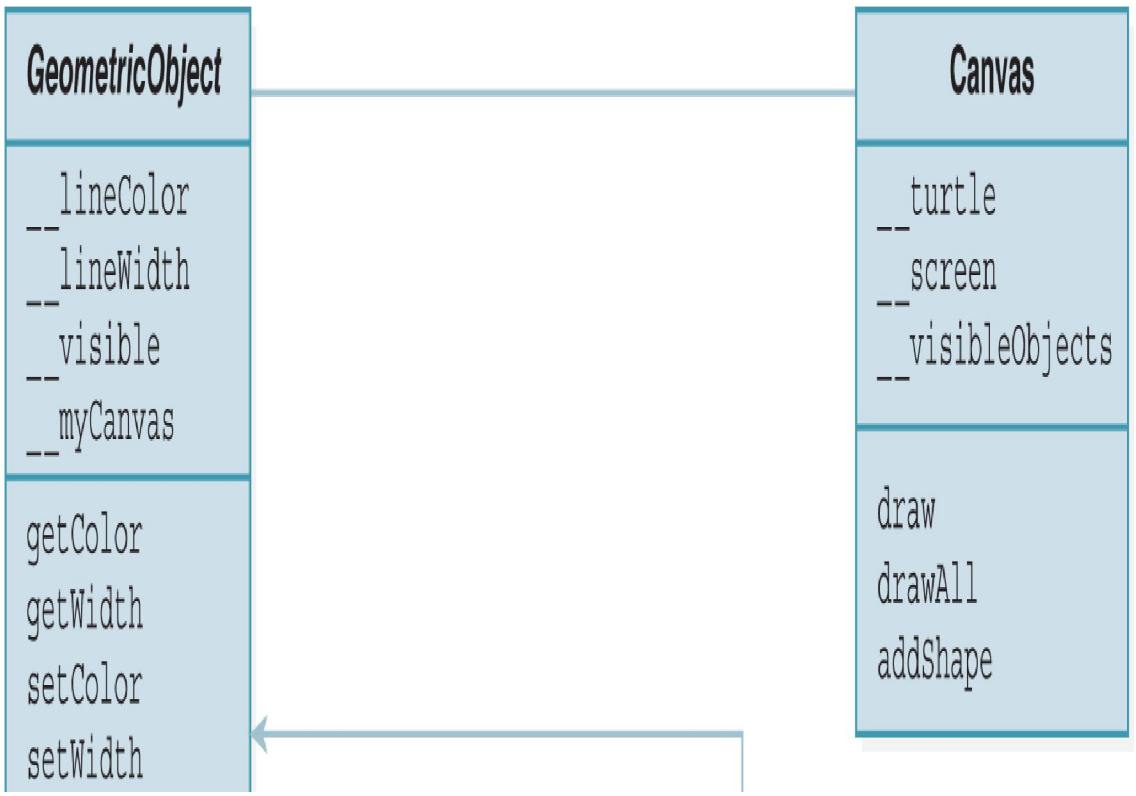
At the end of this sequence, the line will be drawn on the canvas. More importantly, the line will now "remember" which canvas it was drawn on,

and the canvas will know all the objects that have been drawn on it.

Next, let's consider the sequence of events that occurs when we make a call to change the color or width of the line such as `myLine.setColor('red')`.

1. Modify the `__lineColor` instance variable of the line to have the new value of `'red'`.
2. If the `__visible` instance variable is `True`, then use the `__myCanvas` instance variable to call the method `drawAll` on the canvas.

In the `Canvas` class, the new `drawAll` method clears the canvas and redraws all objects on the canvas in the same order as they were drawn originally. Although we may not need to redraw all the objects on the canvas, it would be more work to figure out which objects need to be redrawn than it would be to just redraw everything. The new design for our classes is shown in **Figure 12.8**. This figure includes all the new instance variables and methods we have discussed. In UML, abstract classes and abstract methods are indicated by putting the names in italics. Thus, in Figure 12.8, `GeometricObject` and the `_draw` method in that class are italicized.



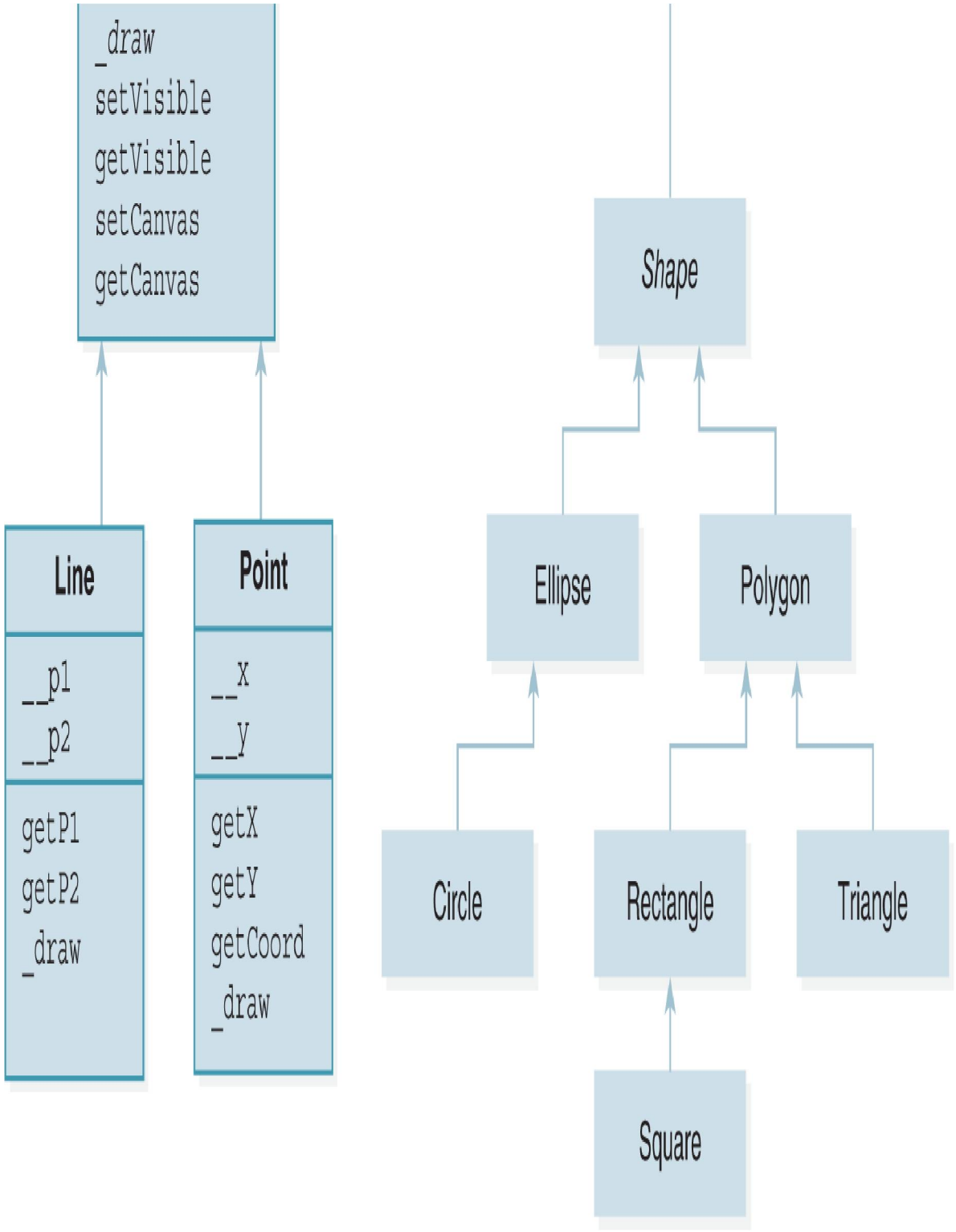


Figure 12.8 New design to allow proper color and width changes.

Description

Even though we have made some major improvements to the functionality of the `Point` and `Line` classes, we have actually changed only the `Canvas` and `GeometricObject` classes. **Listing 12.7** presents the new version of the `Canvas` class. Relative to our earlier implementation in Listing 12.3, we have added two new methods: `drawAll` and `addShape`. In addition, we have made small modifications to `draw` and `_ _init_ _`.

```
1  class Canvas:
2      def _init_(self, w, h):
3          self._visibleObjects = []   #list of shapes to draw
4          self._turtle = turtle.Turtle()
5          self._screen = turtle.Screen()
6          self._screen.setup(width = w, height = h)
7          self._turtle.hideturtle()
8
9      def drawAll(self):
10         self._turtle.reset()
11         self._turtle.up()
12         self._screen.tracer(0)
13         for shape in self._visibleObjects: #draw all shapes in order
14             shape._draw(self._turtle)
15         self._screen.tracer(1)
16         self.  turtle.hideturtle()
```

```
17
18      def addShape(self, shape):
19          self._visibleObjects.append(shape)
20
21      def draw(self, gObject):
22          gObject.setCanvas(self)
23          gObject.setVisible(True)
24          self._turtle.up()
25          self._screen.tracer(0)
26          gObject._draw(self._turtle)
27          self._screen.tracer(1)
28          self.addShape(gObject)
```

Listing 12.7 A new and improved `Canvas` class

Description

The modifications to `__init__` and `draw` affect the initialization and updating of the `__visibleObjects` instance variable. When a canvas is first created, the list of objects drawn on the canvas is initialized to an empty list. Whenever `draw` is called, the `gObject` passed as a parameter is appended to the end of the list. Consequently, all objects that are visible on the canvas are stored in the `__visibleObjects` list, and the order in which they were drawn is preserved.

The `addShape` method is simply a convenient way to add an object to the `__visibleObjects` list. The reason we do not call `_`

`_visibleObjects.append` directly is that by consistently using the `addShape` method, we limit any dependencies on how we represent the objects drawn on the canvas to one method. Imagine that sometime in the future you decide to use a dictionary, rather than a list, to keep track of the objects on the canvas. If you had used `_ _visibleObjects.append` in several places, you would need to change each of those places to accommodate the new representation. However, because we have been clever and used the `addShape` method, we would need to change the way we add an object to the canvas in only one place.

The biggest addition to the `Canvas` class is the `drawAll` method. This method iterates over all the `GeometricObject` objects on the `_ _visibleObjects` list and has each object perform its `_draw` method. In this way, we can recreate whatever picture is visible on the canvas by calling a single method. This loop demonstrates polymorphism in object-oriented programming in a very powerful way. Notice that we do not need to keep a separate list of lines, points, and other shapes; instead, we need only one list of objects. Because each object inherits from `GeometricObject` and provides a `_draw` method, Python finds the correct `_draw` method for each object.

Also notice that the first line of the method calls `self._ _turtle.reset()`. The `reset` method of the turtle creates a blank canvas by erasing anything that the turtle has previously drawn and puts the turtle in the center of the canvas with all default attributes.

Listing 12.8 shows the new `GeometricObject` class, which has fairly small changes. We have added two new setter methods: `setVisible` and `setCanvas`. In addition, the `setColor` and `setWidth` methods have been modified to call the `drawAll` method whenever the object being changed is visible.

```
1 from abc import *
2 class GeometricObject(ABC):
3     def __init__(self):
4         self.lineColor = 'black'
```

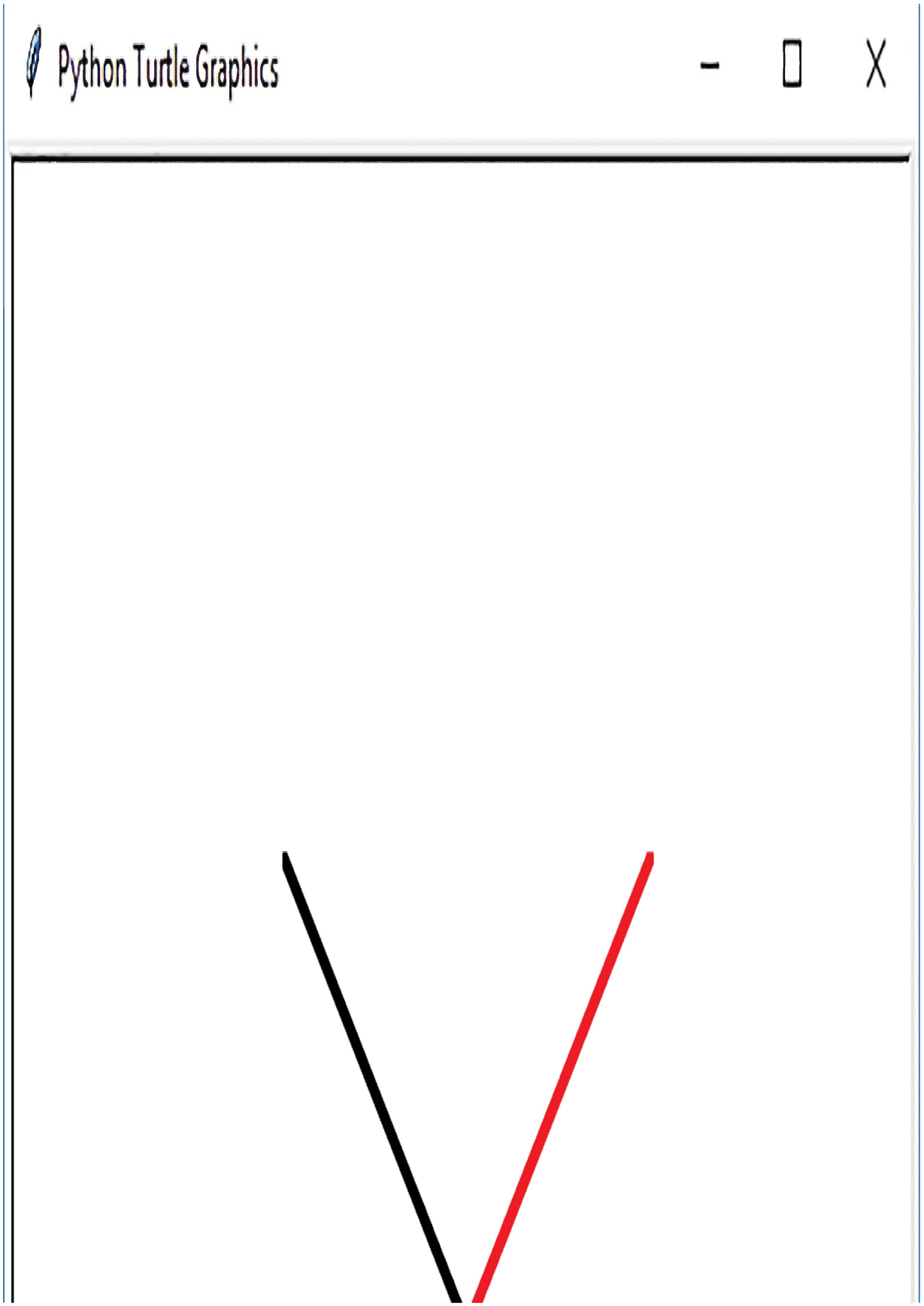
```
4           self._lineColor = 'black'
5           self._lineWidth = 1
6           self._visible = False
7           self._myCanvas = None
8
9       def setColor(self, color):  #modified to redraw visible shapes
10          self._lineColor = color
11          if self._visible:
12              self._myCanvas.drawAll()
13
14      def setWidth(self, width):  #modified to redraw visible shapes
15          self._lineWidth = width
16          if self._visible:
17              self._myCanvas.drawAll()
18
19      def getColor(self):
20          return self._lineColor
21
22      def getWidth(self):
23          return self._lineWidth
24
```

```
25      @abstractmethod
26      def _draw(self):
27          pass
28
29      def setVisible(self, vFlag):
30          self._visible = vFlag
31
32      def getVisible(self):
33          return self._visible
34
35      def setCanvas(self, theCanvas):
36          self._myCanvas = theCanvas
37
38      def getCanvas(self):
39          return self._myCanvas
```

Listing 12.8 A new and improved `GeometricObject` class

Description

Now, if we run the `test2` code in Session 12.4, **Figure 12.9** shows that the changes made to our lines after being drawn do take effect: `line1` is red and `line2` has a width of 4.


Python Turtle Graphics

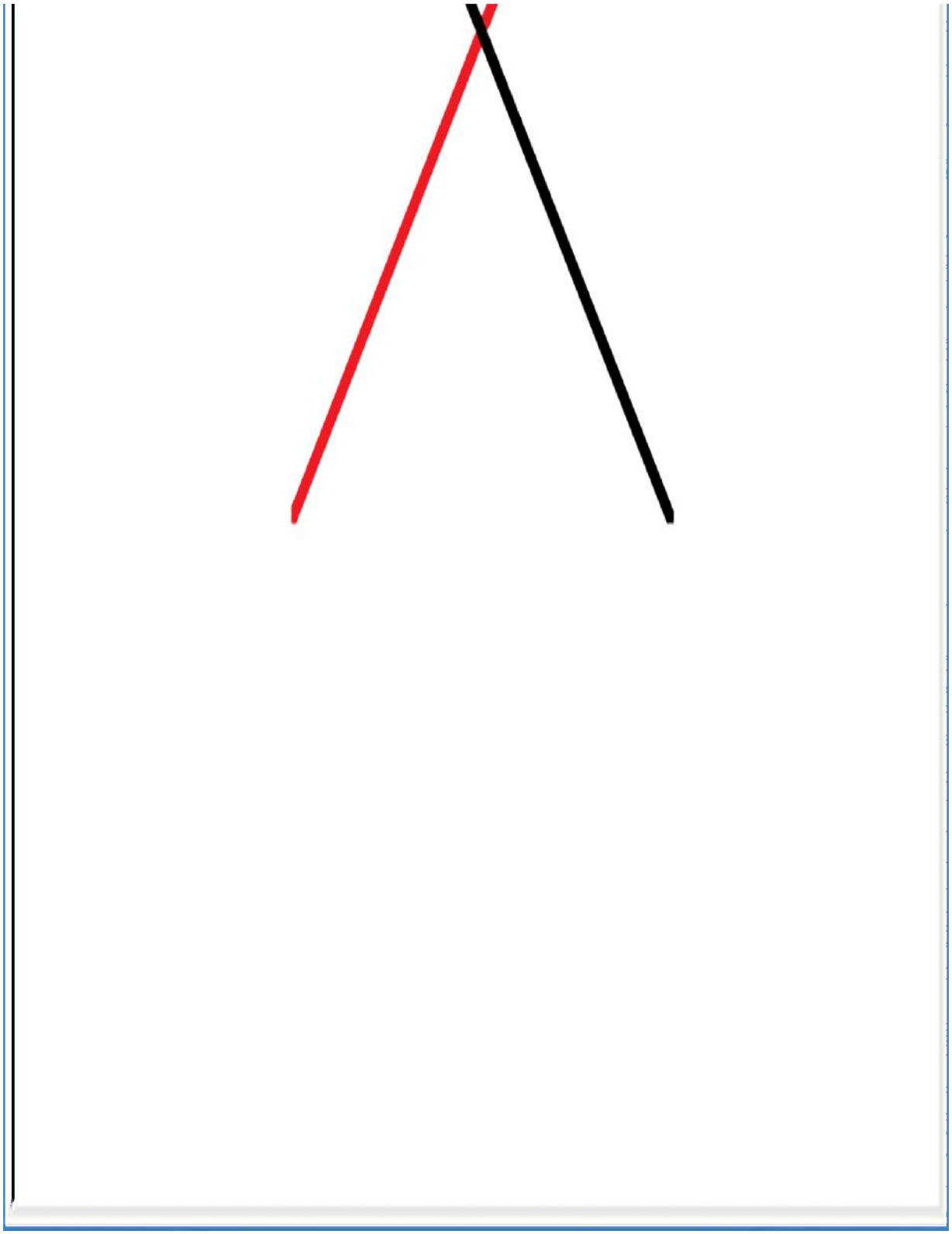


Figure 12.9 Improved code allows color and width changes.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Try It Out

12.10 Write an "undraw" method to remove a `GeometricObject` from the canvas.

12.11 Add the ability to move objects.

12.12 Because each object now has a reference to the canvas on which it is drawn, it is no longer necessary to pass a reference to a turtle as a parameter to `_draw`. Modify the classes to remove this parameter.

12.8 Implementing Polygons

We have neglected a large part of our initial inheritance hierarchy in favor of focusing on a few key implementation details. Now that we have made it possible to properly modify the color, line width, and visibility of `Points` and `Lines`, we can turn our attention to adding new graphical objects. You will see that inheritance makes it easy to add new objects without writing a lot of code.

Let's begin by working down the inheritance hierarchy, starting with `Shape`. `Shape` is an abstract class that will hold an instance variable that will let us know whether an instance of `Shape` should appear as an outline or whether the shape should be filled in with a color. **Listing 12.9** shows that the `Shape` class also inherits from the `GeometricObject` class. Like the `setColor` and `setWidth` methods in the `GeometricObject`, the `setFill` method triggers a call to redraw all visible objects. Note that because the `Shape` class inherits from the abstract `GeometricObject` class and does not implement the `_draw` method, the `Shape` class is also abstract. In consequence, an attempt to create an object of the `Shape` class will generate a type error.

```
class Shape(GeometricObject):
    def __init__(self):
        super().__init__()
        self.__fillColor = None

    def setFillCover(self, aColor):
        self.__fillColor = aColor
        if self.getVisible():
            self.getCanvas().drawAll()

    def getFillColor(self):
        return self.__fillColor
```

Listing 12.9 The `Shape` class

Description

The next class to consider is `Polygon`, which can be either abstract or concrete. That is, we can imagine creating an instance of a `Polygon` as representing some irregular multisided closed shape. In fact, we can just stop with `Polygon` and implement squares, triangles, octagons, and similar shapes as polygons. This is one of those design decisions where advantages and disadvantages should be considered. If we choose to make `Polygon` abstract, then we need to add explicit subclasses for every possible polygon we might want to draw. If we choose to allow `Polygon` to be concrete, then we can implement some subclasses to make it easy to construct the most common

polygons. We will choose the latter approach and make `Polygon` a concrete class—that is, we will implement a `_draw` method.

For each shape, the most important method we must develop is `_draw`. If we think carefully, we can implement a single `_draw` method to draw any polygon. But how can we represent the shape of a polygon with instance variables? The answer is that we can represent any polygon using a list of points. A triangle has three points corresponding to its three corners, a square has four points corresponding to its four corners, and so on. In addition to knowing the positions of the corners of our polygon, it is important to specify them in some order so that we can go from corner to corner and get an outline of the shape. If the corners are randomly ordered, we may just draw a bunch of random lines.

There are two logical choices for the order in which to specify the corners: clockwise and counterclockwise. Most graphics systems use a counterclockwise ordering for the corners of a polygon.

A counterclockwise ordering means that we can draw any polygon by drawing lines between points that are adjacent in our ordered list of points. **Figure 12.10** shows an example of a parallelogram with four points. These four points could be stored in a list such as `[p0, p1, p2, p3]`. We can then draw lines from `p0` to `p1`, from `p1` to `p2`, and from `p2` to `p3` because they are next to one another in the list. However, this gives us only three of the four sides: We must also draw a line from `p3` to `p0` to complete the shape.

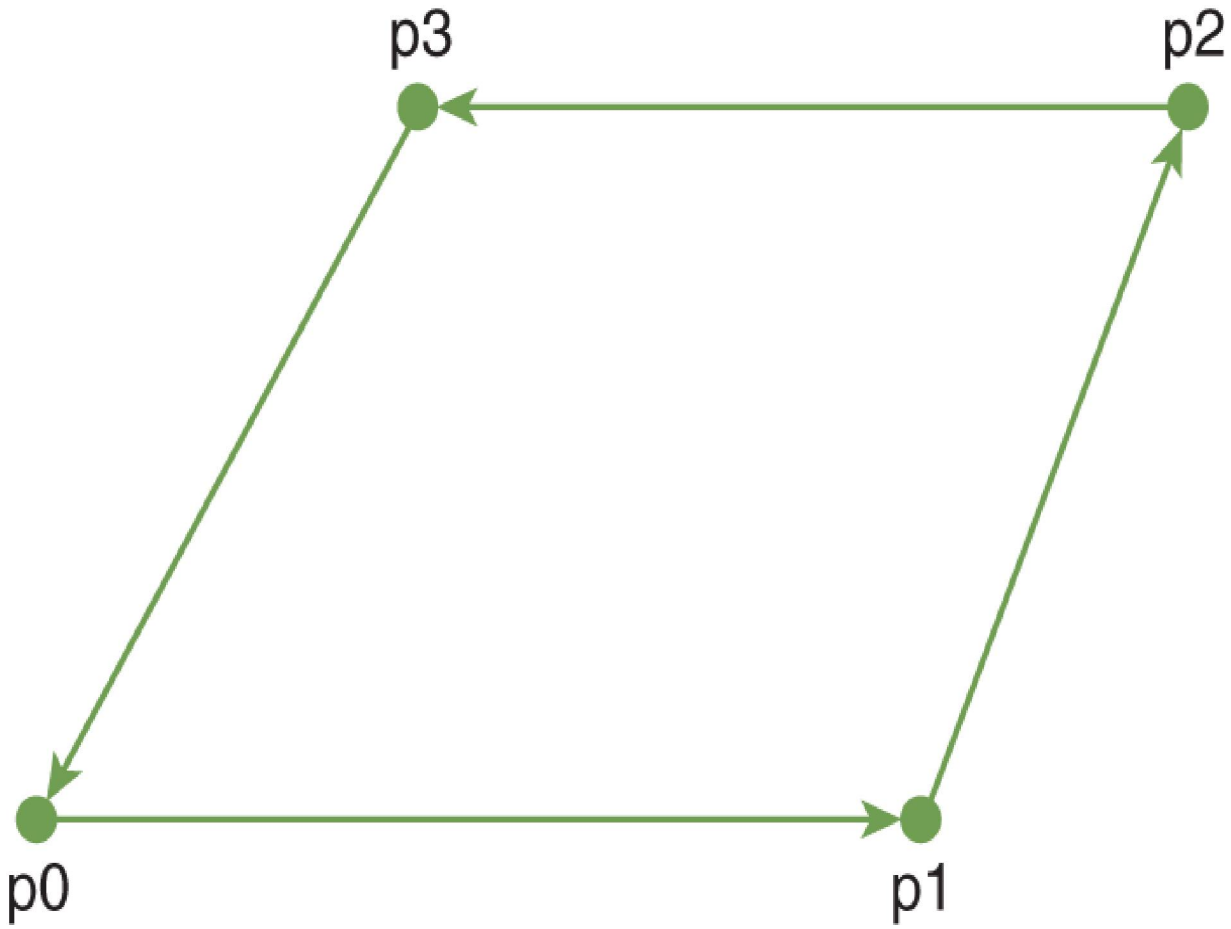


Figure 12.10 A correctly drawn polygon with corners specified in counterclockwise order.

Description

If we had specified the corners of our polygon in the order `[p0, p1, p3, p2]`, we would get a completely different shape that is not a polygon at all.

To implement the `_draw` method for any polygon, you can simply iterate over the list of corners and have the turtle draw a line from point to point. When you run out of points in the list, you must have the turtle draw one final line back to the first point on the list. If a polygon has a `__fillColor` other than `None`, then you will need to call the `begin_fill` and `end_fill` methods of the turtle before drawing the first line and after drawing the last line.

Now that you know how to represent and draw any polygon, what about `Rectangle`, `Triangle`, and `Square`? The key for these three classes is how we write our constructor. For an arbitrary polygon, the constructor must take a list of points. For a specific polygon, we can make it a bit more user-

friendly. For example, the constructor for `Triangle` can take three points, and the constructors for `Square` and `Rectangle` can each take just two points.

The points needed to specify a rectangle are the lower-left corner and the upper-right corner. If you know the lower-left and upper-right corners, you can figure out the position of the other two corners quite easily, as shown in **Figure 12.11** (where the lower-left coordinates are prefaced with LL, and the upper-right coordinates are prefaced with UR). Listing 12.1 provides examples of the constructors for a `Rectangle`. The constructor for a `Square` is identical to the constructor for a `Rectangle` except that it should check that the lengths of all sides are the same.

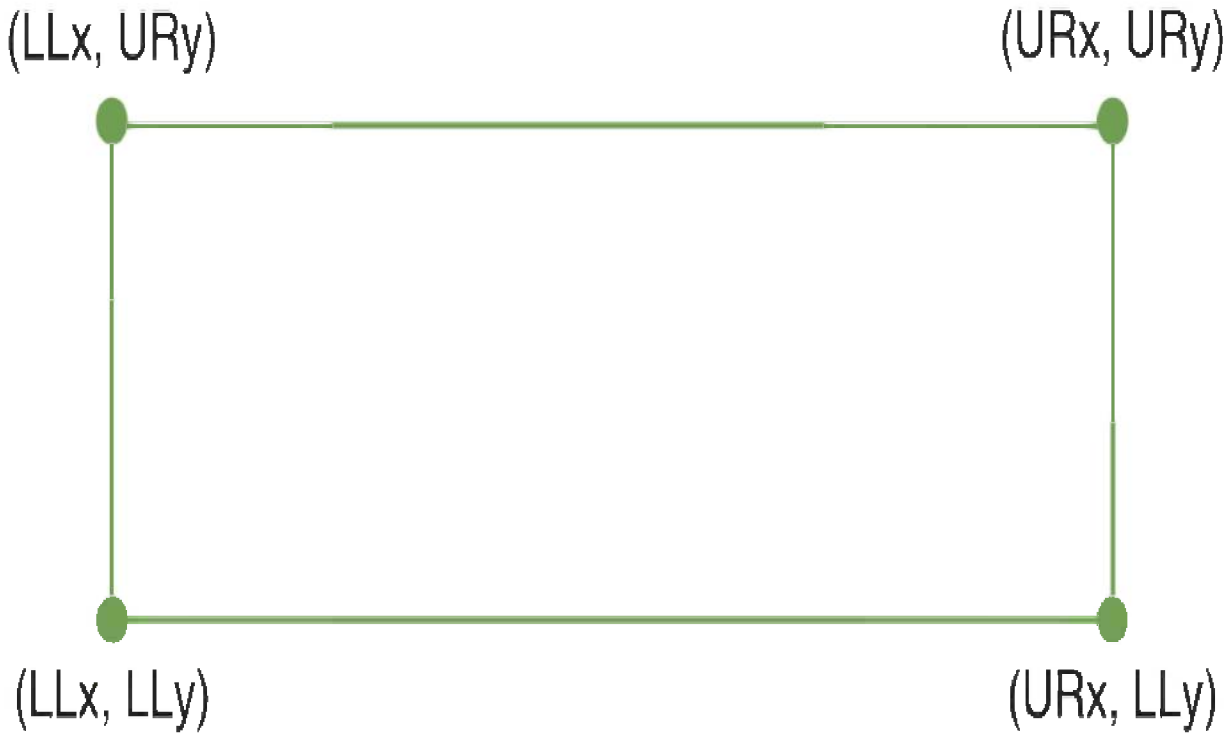
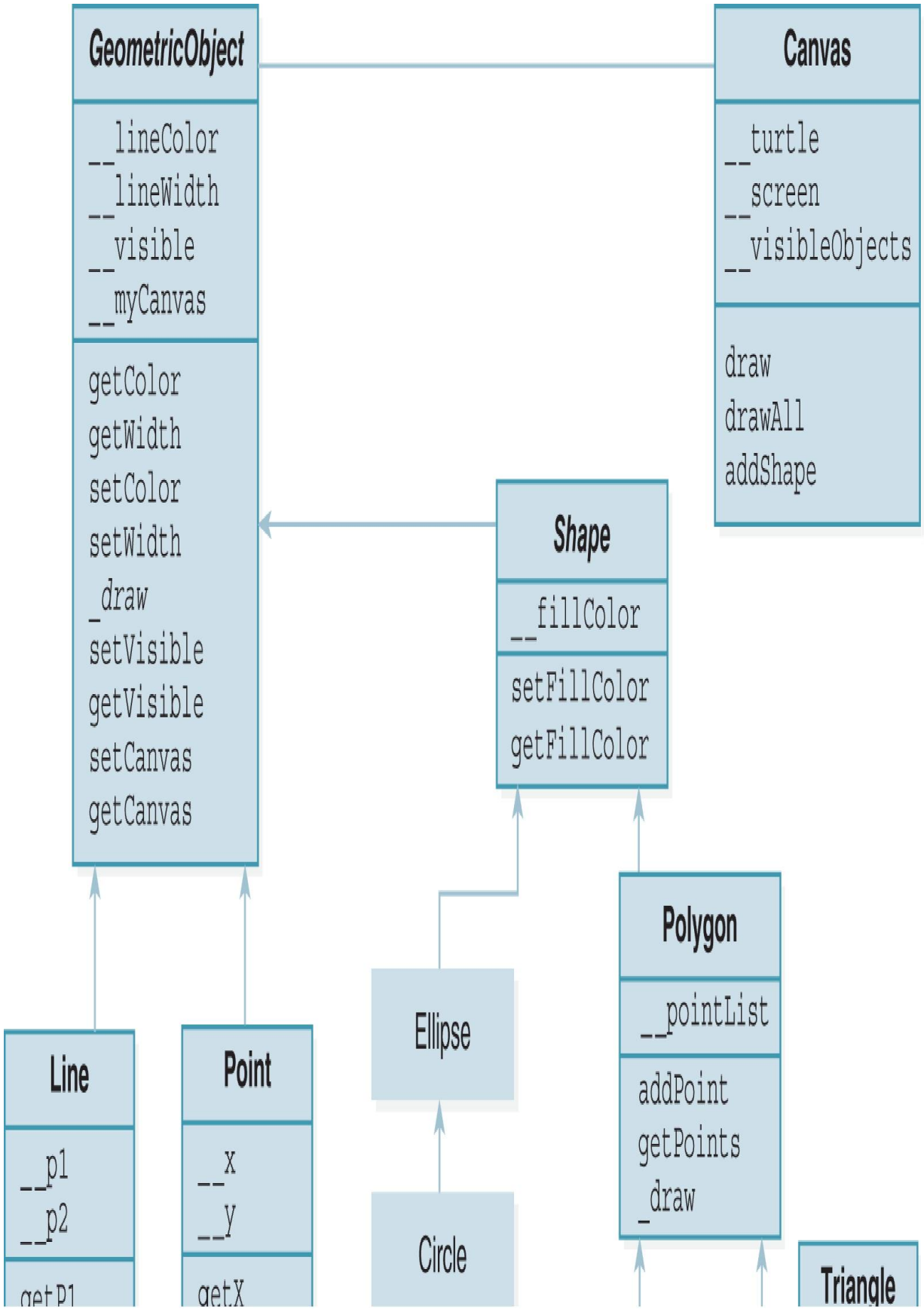


Figure 12.11 Using the information for the lower-left and upper-right corners to determine other corners of a rectangle.

Description

Figure 12.12 depicts the new version of our class hierarchy. We will not show you the Python implementation for `Polygon`; instead, implementing the rest of the hierarchy is left as an exercise for you.


GeometricObject
__lineColor
__lineWidth
__visible
__myCanvas
getColor
getWidth
setColor
setWidth
_draw
setVisible
getVisible
setCanvas
getCanvas
Canvas
__turtle
__screen
__visibleObjects
draw
drawAll
addShape
Shape
__fillColor
setFillColor
getFillColor
Polygon
__pointList
addPoint
getPoints
_draw
Ellipse
Circle
Line
__p1
__p2
getP1
Point
__x
__y
getX
Triangle

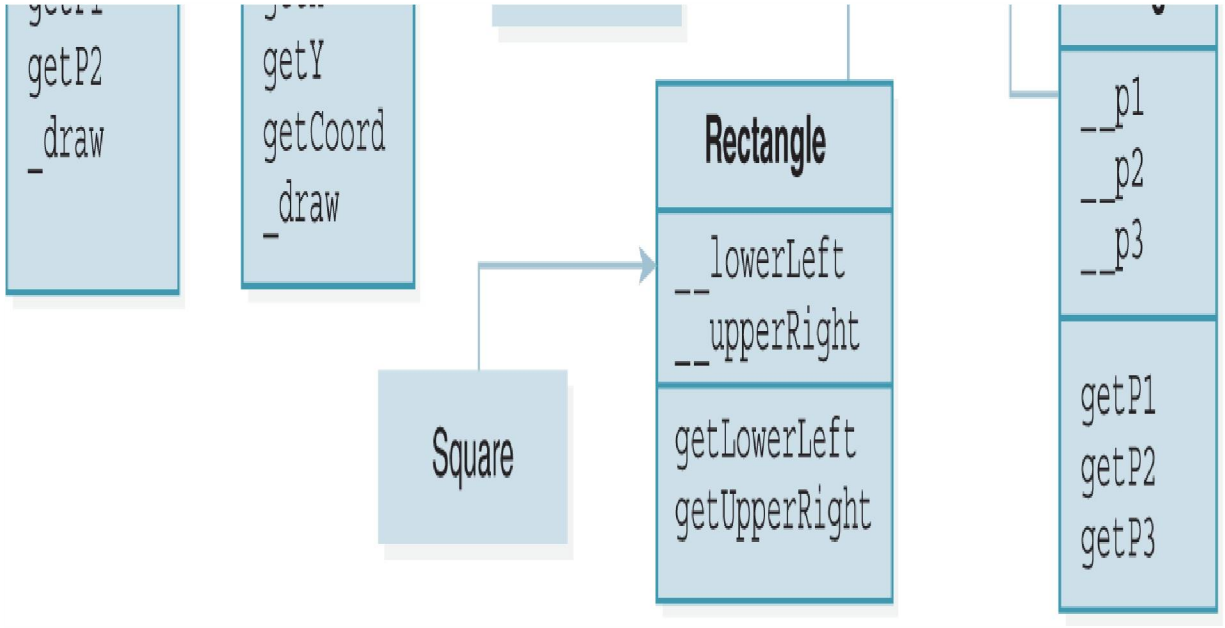


Figure 12.12 Inheritance hierarchy, including details for polygons.

Description

Try It Out

12.13 Implement the design for `Polygon` and its children.

12.14 Implement additional `Polygon` convenience classes such as `Square`, `Triangle`, `Rectangle`, and `Octagon`. (*Hint*: These subclasses can just set up the points list for the specific shape.)

12.9 Summary

In this chapter we learned about inheritance, the ability to design and implement classes that take advantage of natural relationships existing between them. These relationships provide a way for us to organize our classes. Superclasses contain general details (instance variables and methods) that pertain to an object. Subclasses contain details that allow objects to be more specific. This parent–child relationship, called an IS-A relationship, allows child objects to have all the functionality of parent objects while adding specific additional details. Another relationship called HAS-A exists when an object of one class is an instance variable of another class. This relationship is also called composition.

Key Terms

abstract class
abstract method
base class
child class
composition
concrete class
decorator
extends
HAS-A relationship
hierarchy
inheritance
inheritance hierarchy
IS-A relationship
object-oriented drawing programs
parent class
polymorphism
subclass
superclass
Unified Modeling Language (UML)

Programming Exercises

12.1 Design and implement the Ellipse part of the shape hierarchy.

12.2 Design and implement a `Text` class for the `GeometricObject` hierarchy.

12.3 Test your entire set of classes using Listing 12.1.

12.4 Using our graphics module, implement your own scene.

12.5 Reimplement the bear and fish simulation from Chapter 11 using inheritance. *Hint:* You will want to create an abstract class called `LifeForm` that captures the commonalities of bears and fish.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

CHAPTER 13

Interactive Games

Multithreading, Event Handlers, and Static Variables and Static Methods

13.1 OBJECTIVES

- To write an event-driven program
- To understand and write callback functions
- To understand multithreading
- To see another pattern for using inheritance
- To learn about static variables and static methods

13.2 Introduction

So far, the programs we have written have a beginning, a middle, and an end. We know the order in which things will happen because we have implemented the steps in a linear fashion. That is not the way most programs work. Think of the programs you use on a daily basis—from your web browser and word processor to video games. These programs do not have a beginning, a middle, and an end. Instead, they sit there and wait for you to click your mouse, touch your screen, or press a key. When you take some action, the program responds with its own action. The order of the actions is determined by you, not the program.

13.2.1 Event-Driven Programming

Programs that wait for an event to occur and then respond to that event by taking an action are called **event-driven programs**. In this chapter, you will learn how to write event-driven programs using the `turtle` module. When writing event-driven programs, one of the most important concepts is that of a **callback function**, which is just like any other function except that it is designed to be called in response to an event.

The main structure of an event-driven program is an **event loop**, which performs the following tasks:

- Check the event queue for the next event.
- Call the callback function to handle the event.

Figure 13.1 illustrates the idea behind the event loop. Notice the similarity between this event loop diagram and the read–eval–print loop that Python uses. One difference between the two types of loops is that in event-driven programming we use a **queue**—a data structure that works something like a to-do list to keep track of events that need to be handled. A queue allows you to remove the first thing on your to-do list and add things at the end. In this way, a queue enforces a first-come, first-served strategy for handling events.

You will learn much more about queues and their usefulness in a data structures class.

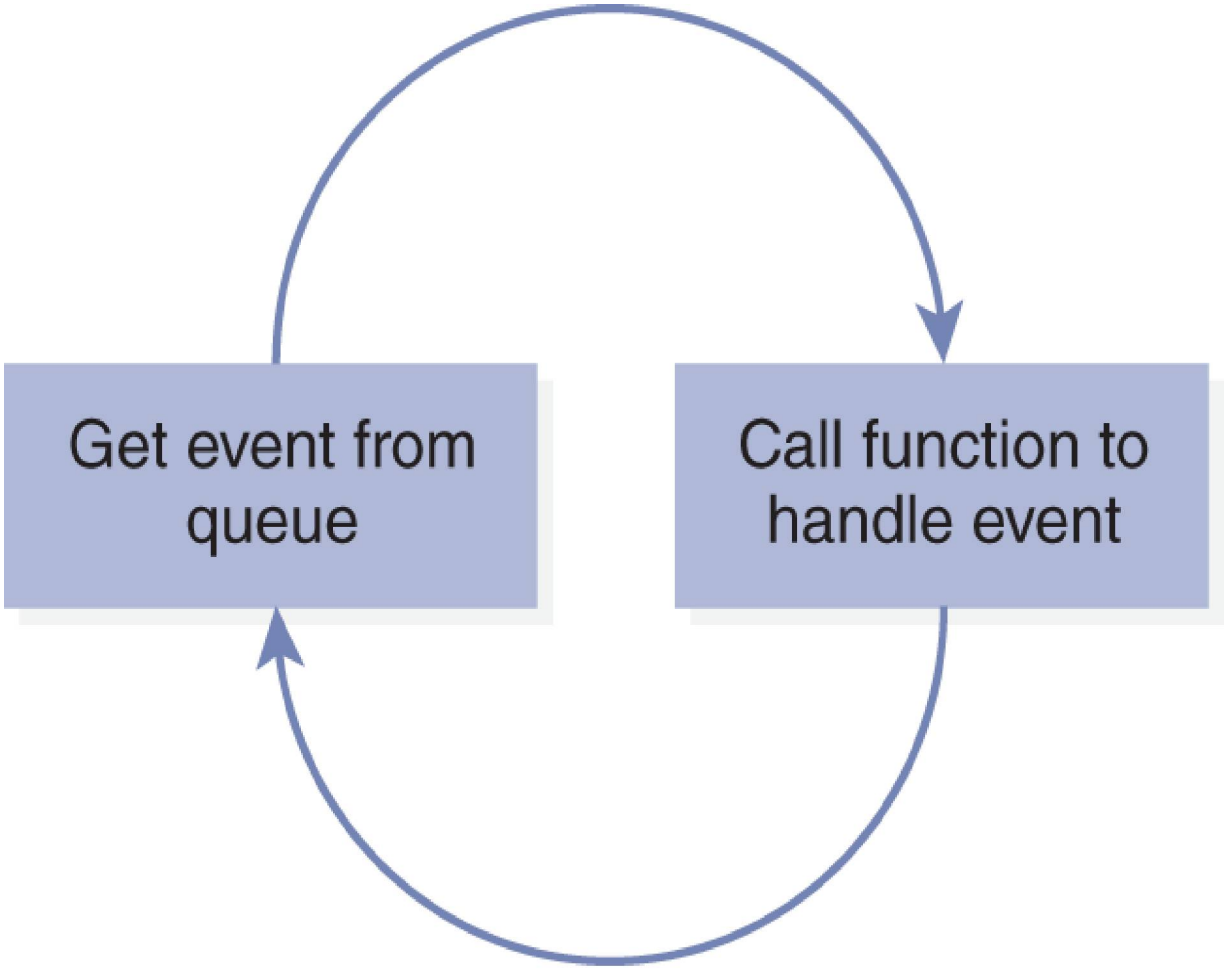


Figure 13.1 The event loop.

Description

13.2.2 Simulating an Event Loop

The event loop in Figure 13.1 looks pretty simple, but one important piece of the puzzle is not shown in the diagram. How do we, as programmers, control which function is called for a particular event? Of course, if we were writing the event loop, we could do it easily with some `if` statements. But we are not writing the event loop, and we do not even have access to the source code for the event loop—so there must be some other way.

To solve this control problem, the event-processing system provides us with a way to **register** a function that we want to call when a specific event occurs. For example, we might tell the event-processing system to call function A when the user clicks the mouse and to call function B when the user presses a key on the keyboard.

To illustrate how this all works, let's write a simple simulation of an event-processing system. Our event processor will have the following functions:

- Add an "event" to the queue.
- Register a function to be called in response to an event.
- Run the simulation.

Listing 13.1 implements a simple class to illustrate an event-processing loop. Although the code is short, some powerful things are happening. In the `__init__` method, we set up two important instance variables: `__queue`, which will hold the list of events that need to be processed, and `__eventKeeper`, which is a dictionary that maps events to their callback functions.

```python
class EventHandler:
    def __init__(self):
        self._queue = []          #holds events
        self._eventKeeper = {}    #dict of event types and callbacks

    def addEvent(self, eventName):
        self._queue.append(eventName)

    #specify function to call when event occurs
    def registerCallback(self, event, func):
        self._eventKeeper[event] = func

    def run(self):
        while (True):    #endless loop
            if len(self._queue) > 0:
                nextEvent = self._queue.pop(0) #get next event
                self._eventKeeper[nextEvent]() #run callback function
            else:
                print('queue is empty')
```

Listing 13.1 A simple event-processing simulation

Description

The `registerCallback` method (lines 9–11) allows us to specify an event type and a function to call when the event occurs. The method then adds the dictionary entry to the `_ _eventKeeper` instance variable. Suppose that we have two events: `'mouse'` and `'key'`. We can store a reference to the callback function for the mouse event in `_ _eventKeeper` using the string `'mouse'` as the key. Similarly, we can store a reference to the callback function for the key event in `_ _eventKeeper` using the string `'key'`.

The `addEvent` method allows us to add events to the event queue. A new event is appended to the queue, enforcing the first-come, first-served ordering of event processing.

RECAP:
The event queue handles events in a first-come, first-served manner. New events are added to the end of the queue, and events to be handled are removed from the front of the queue.

The `run` method does the event handling. If the queue is not empty, we retrieve the first event in the queue. Recall that the list method `pop` will remove and return the `i`th item in the list. So, we use `pop(0)` to retrieve the first item in the event queue (line 16).

To call the correct callback function for the next event, we can then call the function (line 17) using the rather odd-looking expression `self_ _eventKeeper[nextEvent]()`. If we break that expression apart, it is not so mysterious. The square brackets tell Python to look up the key specified by `nextEvent` in the `_ _eventKeeper` dictionary. The dictionary lookup returns a reference to the function stored there. Finally, the left and right parentheses let Python know that it should call the referenced object as a function.

Let's give our simulation a try. You can follow along in **Session 13.1**. First, we will define two callback functions. We will call the functions

`myMouse` and `myKey` to handle mouse and key events, respectively. Although these two functions are simple, keep in mind that they could do anything we want them to do. Second, we will register our callback functions with the event handler. Third, we will add some events using the `addEvent` method.

In [1]:
```
1 from eventhandler import *
```

In [2]:
```
1 def myMouse():
2     print('Oh no, the mouse was clicked!')
3
```

In [3]:
```
1 def myKey():
2     print('A key has been pressed.')
3
```

In [4]:
```
1 def runEventHandler():
2     eh = EventHandler()
3     eh.registerCallback('key', myKey)
4     eh.registerCallback('mouse', myMouse)
5     eh.addEvent('mouse')
6     eh.addEvent('key')
```

```
6     eh.addEvent('key')
7     eh.addEvent('mouse')
8     eh.run()
9
```

```
In [*]: 1  runEventHandler()
```

```
Oh no, the mouse was clicked!
A key has been pressed.
Oh no, the mouse was clicked!
queue is empty
queue is empty
queue is empty
...
```

Session 13.1 Testing the event-processing simulation

Description

With the callback functions registered and some events added to the queue, we tell the event handler to run. Note that the `run` method does something that we have never tried to do on purpose before: It enters an infinite loop. In each pass through the loop, we check the length of the event queue to see if it contains any entries. If there are entries in the queue, then the first one is removed and the appropriate callback function is called. If the queue is empty, then the message "`queue is empty`" is printed.

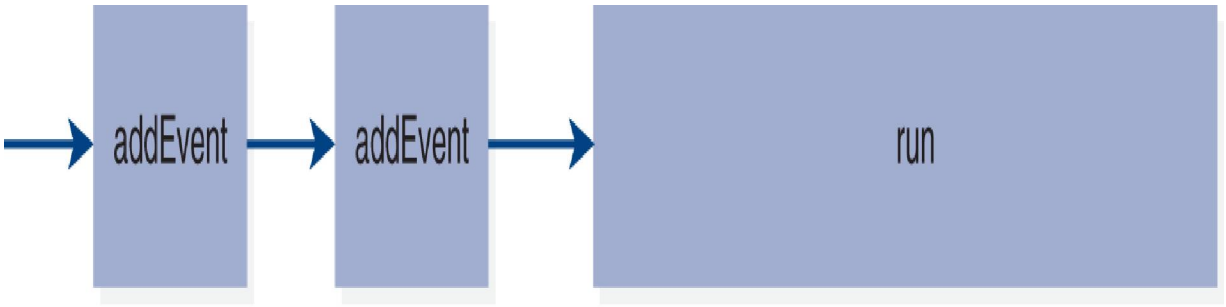
HEADS UP:
Remember to register the callback function for any event you want to handle.

When we run this simulation, after the first three messages, "`queue is empty`" will be printed forever or until you stop the program from running (try pressing CTRL-C).

13.2.3 A Multithreaded Event Loop

As demonstrated, the problem with the code in Listing 13.1 is that once you call the `run` method, the program goes into an infinite loop. The only events that will be processed are events that you placed on the event queue before you called the `run` method. In the real world, this is not the case: The event loop runs simultaneously with other code that is capable of putting events on the event queue. A **multithreaded program** can have multiple things happening at the same time. All modern desktop applications are multithreaded applications, including the `turtle` module.

Figure 13.2 illustrates the difference between **single-threaded** and multithreaded execution. In the top sequence, there is a single **thread** of execution, which means that once the `run` method is started, no more events can be added to the queue. In the bottom sequence, which illustrates **multithreading**, the `start` call launches a new thread and calls the `run` method inside the new thread. The new thread runs in parallel with the old thread. With two threads of execution, events can be added in one thread while the `run` method is executing in the other.



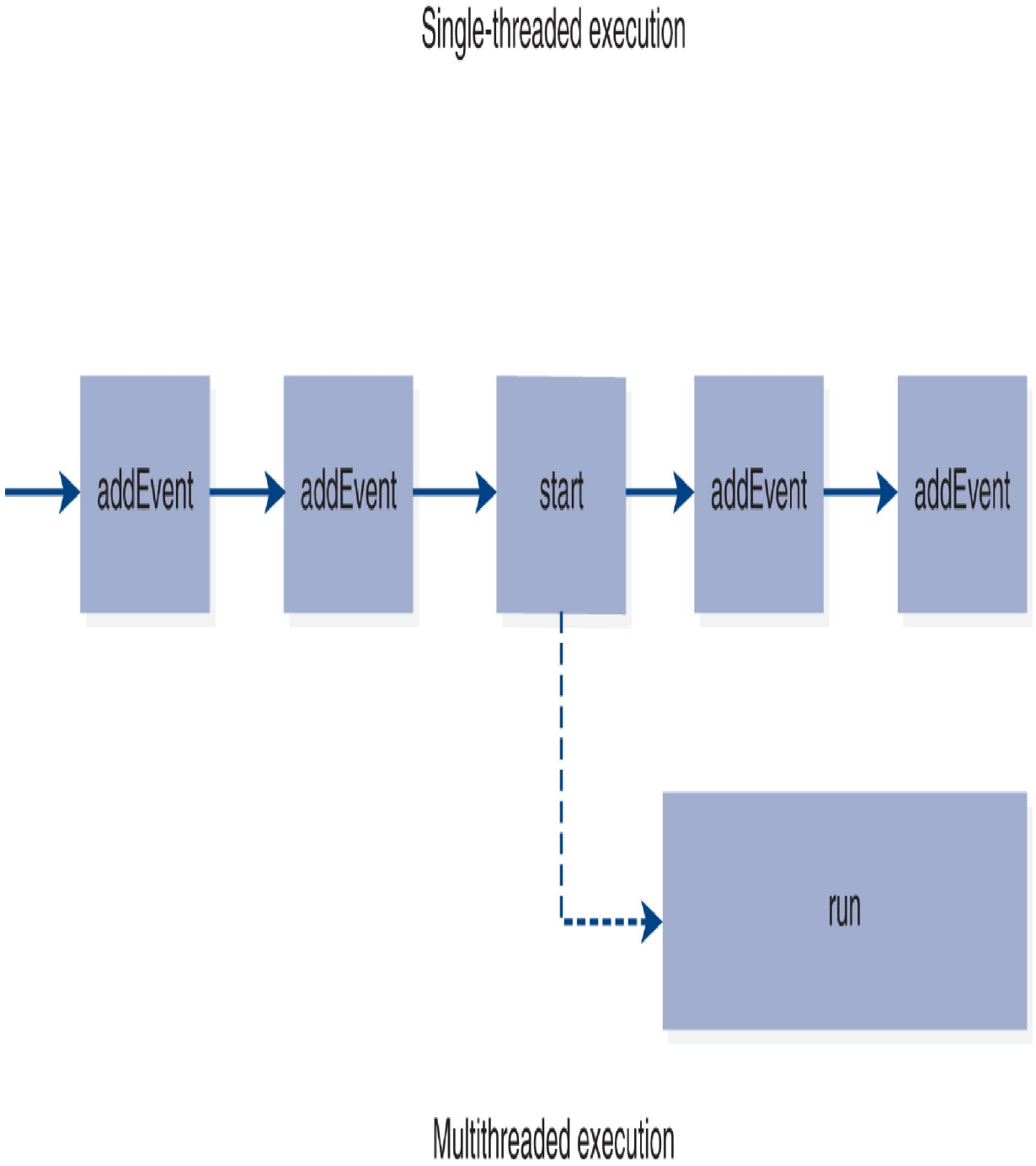


Figure 13.2 Comparing single-threaded and multithreaded execution.

Description

Inheritance makes it possible to easily convert a single-threaded `EventHandler` class to a multithreaded class. We can make our class

multithreaded by having it inherit from the class `Thread`. This is an extremely powerful pattern that we will revisit when we implement our video game. For now, we can just use the code shown in **Listing 13.2**. Notice that we have made only a few small changes between Listings 13.1 and 13.2. The first change is that we now import the `Thread` class from the `threading` module. The second change is that we import the `time module`. The third is in the class declaration: Instead of inheriting from `object` (the default), we now inherit from `Thread`.

```
1  from threading import Thread
2  import time
3
4  class EventHandler(Thread):   #inherits from Thread class
5      def __init__(self):
6          super().__init__()    #call Thread constructor
7          self._queue = []
8          self._eventKeeper = {}
9
10     def addEvent(self, eventName):
11         self._queue.append(eventName)
12
13     def registerCallback(self, event, func):
14         self._eventKeeper[event] = func
15
```

```
16      def run(self):
17          while (True):
18              if len(self._queue) > 0:
19                  nextEvent = self._queue.pop(0)
20                  callBack = self._eventKeeper[nextEvent]
21                  callBack()      #call registered function for event
22              else:
23                  time.sleep(1)  #pause for 1 second
```

Listing 13.2 A multithreaded event handler

Description

When a class inherits from the `Thread` class, it has two options for specifying which code to run when a new thread is created. The first option is to provide a `run` method that contains the code. The second option is to pass a function to the constructor of the `Thread` class. We already have defined the event handler code in a method called `run`, so we are using the first option here.

By inheriting from `Thread`, we gain access to the important `start` method, which creates a new thread and calls the `run` method of the class inside that new thread. The `run` and `start` methods are explained in **Table 13.1**.

TABLE 13.1 The `run` and `start` Methods in the `Thread` Class

Method	Description
`run()`	Contains the code to do the work of the thread. The subclass provides its own version of this method.

Method	Description
`start()`	Creates the thread and calls the thread's `run` method. This method can be called only once per thread.

Once our `run` method starts, it enters an infinite loop just as before. In this version, however, if the event queue is empty, the method `time.sleep` is called. The `time.sleep` method causes the thread to pause for the number of seconds passed as a parameter. The `sleep` method is shown in **Table 13.2**. To call this method, first import the `time` module.

TABLE 13.2 The `sleep` Method in the `time` Class

Method	Description
`sleep(n)`	Pauses for `n` seconds.

We can test the multithreaded version of our event loop as shown in **Session 13.2**. As you can see, once the `start` method is called, the three original events are handled as before. However, with the `run` method executing in its own thread, we can now type additional Python statements to be executed by the original thread. Also, because two threads are running simultaneously, you might see that the output of the main Python read–eval–print loop overlaps the output from the event handler.

```
In [1]: 1 from eventhandlermultithread import *
```

```
In [2]: 1 eh = EventHandler()
```

```
In [3]: 1 eh.addEvent('mouse')
```

In [4]:
```
1 eh.addEvent('key')
```

In [5]:
```
1 eh.addEvent('mouse')
```

In [6]:
```
1 eh.registerCallback('key', myKey)
```

In [7]:
```
1 eh.registerCallback('mouse', myMouse)
```

In [8]:
```
1 eh.start()
```

```
Oh no, the mouse was clicked!
A key has been pressed.
Oh no, the mouse was clicked!
```

In [9]:
```
1 eh.addEvent('mouse')
2 eh.addEvent('key')
```

```
3 eh.addEvent('mouse')
```

```
Oh no, the mouse was clicked!
A key has been pressed.
Oh no, the mouse was clicked!
```

Session 13.2 A multithreaded event loop example

Description

Try It Out

13.1 Write your own callback functions for the `'mouse'` and `'key'` events. Register these events with the `EventHandler` and rerun the simulation.

13.2 Add a third event type called `'timer'`. Supply a simple callback function for the timer.

13.3 Draw a reference diagram to illustrate how a dictionary of callback functions works.

13.4 Devise a way to modify the simulation so that you can have a key event and pass the name of the key as a parameter to the callback function.

13.5 Devise a way to modify the simulation so that you can have a mouse event and pass simulated window coordinates of the mouse to the callback function.

13.6 Modify the simulation to support the timer event. When you add a timer event, you should supply the number of seconds until the timer expires. When the timer goes off, it should call the timer callback function.

13.3 Event-Driven Programming with the Turtle

The `turtle` module provides three methods to register callbacks for key, mouse, and timer events:

- `onkey`
- `onclick`
- `ontimer`

In addition, the `listen` method is used to give the turtle canvas focus so that the event handler can respond to key presses. Each method has conventions for the parameters it accepts. **Table 13.3** explains each of these methods to register callbacks for events, as well as the `listen` method.

TABLE 13.3 `Turtle` Methods to Register Callbacks

Method	Description
`onkey(function, key)`	Binds a function to a key press so that `function` is called when a specified key is pressed. The `function` must take no parameters. `key` is a string representing a key on the keyboard. The keyboard must have focus (see `listen`). Passing `None` as the `function` parameter cancels the callback.
`onclick(function, btn=1, add=None)`	Binds a function to a mouse click, so that `function` is called whenever the mouse button is pressed. Note that `function` must accept two parameters that will be the (*x*, *y*) position of the mouse when the button is clicked. The `btn` parameter specifies the mouse button; 1 is the left button. If `add` is `True`, a new callback will be added; otherwise, the function will replace the previous callback. Passing `None` as the `function` parameter cancels the callback.
`ontimer(function, time=0)`	Binds a function to a timer event. The `function` will be called once after `time` milliseconds. The `function` must take no parameters.
`listen(x=None, y=None)`	Gives the current turtle's canvas focus so it can listen for key events. Key events will not be recognized until this function is called. This `listen` method is defined to take two dummy parameters so that `listen` can be bound to a mouse-click event.

Our video game will use all three of these callback types. In this section, we will investigate the use of these callbacks in more detail.

13.3.1 A Simple Drawing Using Key Presses

Let's begin our investigation of event-driven turtle programming by creating a simple drawing program. In this program, the user can draw just by using the arrow keys. Each arrow key will have the following effect:

- `Up` (↑): The turtle goes forward 5 pixels.
- `Left` (←): The turtle turns counterclockwise by 10 degrees.
- `Right` (→): The turtle turns clockwise by 10 degrees.
- `Down` (↓): The turtle backs up 5 pixels.

In addition, pressing the "q" key will cause the program to quit. Note that the arrow keys go by the special names `Up`, `Left`, `Right`, `Down`, respective to the direction of the arrows on the keys.

The `turtle` module allows us to add a callback function for any key by using the `onkey` method of a `Screen` object. This method takes two parameters: (1) a reference to a function and (2) the name of a key. For example, `myScreen.onkey(myKey,'a')` sets up a callback so that when the user presses the "a" key, the function `myKey` is called. This approach allows us to set up a different callback function for each key on the keyboard.

One important question remains: What about parameters? The first solution that might occur to you is to have the callback function for the left arrow call the turtle method `left`. But the `left` method requires a parameter. Perhaps you might think that the solution is to set up the callback by using `myScreen.onkey(myTurtle.left(5),'Left')`—but this would result in an error. Why? If you think about how Python evaluates the statement, you will realize that call-by-assignment parameter passing evaluates the parameters to a function call. Python will actually try to evaluate `myTurtle.left(5)`, which will return the value `None`. The `onkey` method, however, expects to receive a reference to a function.

In the `turtle` module, the callback for keys cannot accept any parameters. In consequence, we will have to implement our callbacks with zero

parameters and have the callbacks call the turtle methods with the appropriate parameters. With this point in mind, we will write five callback functions for our program, one for each of the arrow keys and one for the "q" key. In the case of the arrow keys, the callback functions will be short, because all we need to do is call the appropriate turtle function. The callback functions are shown on lines 20–33 of **Listing 13.3**.

```
1  import turtle
2
3  class Etch:
4      def __init__(self):
5          self._myT = turtle.Turtle()
6          self._myScreen = turtle.Screen()
7          self._myT.color('blue')
8          self._myT.pensize(2)
9          self._myT.speed(0)
10         self._distance = 5                  #distance to move
11         self._turn = 10                     #degrees to turn
12         self._myScreen.onkey(self.fwd, "Up")   #register callbacks
13         self._myScreen.onkey(self.bkwd, "Down")
14         self._myScreen.onkey(self.left, "Left")
15         self._myScreen.onkey(self.right, "Right")
16         self._myScreen.onkey(self.quit, "q")
```

```
17          self._myScreen.listen()             #listen for key presses
18          self.main()                         #start the Turtle mainloop
19
20      def fwd(self):
21          self._myT.forward(self._distance)
22
23      def bkwd(self):
24          self._myT.backward(self._distance)
25
26      def left(self):
27          self._myT.left(self._turn)
28
29      def right(self):
30          self._myT.right(self._turn)
31
32      def quit(self):
33          self._myScreen.bye()
34
35      def main(self):
36          turtle.mainloop()
```

Listing 13.3 A simple drawing program

Description

The most interesting part for us is the `__init__` method. In that method, we create a turtle, initialize its attributes, create a screen, and set up the callback functions. Line 17 of `__init__` calls the screen method `listen`, which tells the screen to start listening for keyboard events.

The final method is `main`, which just calls the `turtle.mainloop` function. By now, you can probably guess that `mainloop` starts the event loop for the turtle so that the turtle can process events. This method call needs to be the last statement in a turtle graphics program. **Table 13.4** describes the `mainloop` method.

TABLE 13.4 The `mainloop` Method

Method	Description
`mainloop()`	Starts the turtle event handler so that the turtle graphics program can respond to events. This method call must be the last statement in a turtle graphics program.

To run this little drawing program, you load the `Etch` class as defined in Listing 13.3 and then execute the following command:

```
draw = Etch()
```

Once the program is running, you can create a drawing by pressing the arrow keys to move the turtle around the window or press "q" to end the program.

You might have noticed that the program in Listing 13.3 uses composition —a style of object-oriented programming in which the `Etch` class uses a turtle to do the bulk of the work and, therefore, "HAS-A" reference to a turtle as an instance variable. This approach is a common and useful way to think about programming. However, we could also apply the concept of inheritance when thinking about this problem. The inheritance view of this

problem would say that an `Etch` "IS-A" special kind of `Turtle` and, therefore, our `Etch` class would inherit from `Turtle`.

Listing 13.4 shows how to implement our drawing program using inheritance. The main difference between this program and the previous one is that we do not explicitly create a reference to a turtle inside the `Etch` class, because now an `Etch` "IS-A" `Turtle`. So, the real work of creating the window and the `Turtle` part of our `Etch` object happens on lines 5–6. Notice that this also simplifies the rest of the code because we can now just call `self.forward` rather than `self._ _myT.forward`.

```
1  from turtle import Turtle, mainloop
2
3  class Etch(Turtle):
4      def __init__(self):
5          super().__init__()
6          self._screen = self.getscreen()
7          self.color('blue')
8          self.pensize(2)
9          self.speed(0)
10         self._distance = 5
11         self._turn = 10
12         self._screen.onkey(self.fwd, "Up")
13         self._screen.onkey(self.bkwd, "Down")
14         self._screen.onkey(self.left10, "Left")
```

```
15          self.__screen.onkey(self.right10, "Right")
16          self.__screen.onkey(self.quit, "q")
17          self.__screen.listen()
18          self.main()
19
20      def fwd(self):
21          self.forward(self.__distance)
22
23      def bkwd(self):
24          self.backward(self.__distance)
25
26      def left10(self):
27          self.left(self.__turn)
28
29      def right10(self):
30          self.right(self.__turn)
31
32      def quit(self):
33          self.__screen.bye()
34
```

```
35      def main(self):
36          mainloop()
37
38  if __name__ == '__main__':
39      etch = Etch()
```

Listing 13.4 The drawing program implemented with inheritance

Description

Neither implementation of `Etch` is implicitly "right" or "wrong." That is, the programs in Listings 13.3 and 13.4 are alternative ways of accomplishing the same goal. Each has both advantages and disadvantages.

Try It Out

13.7 Add a new key event to the `Etch` program to allow the turtle to pick up or put down its tail.

13.8 Add a key event and reset function to the `Etch` program that clears the screen and returns the turtle to the home position.

13.9 You could make the `Etch` program in Listing 13.4 even cleaner by using polymorphism and a default parameter. Rather than implementing a `fwd` method that calls `self.forward`, you can replace the definition of `forward` with a default parameter for the distance. The implementation of the `forward` method would use the `super` function to call the turtle's implementation of `forward` passing the default parameter. Rewrite the `Etch` implementation using `forward`, `backward`, `left`, and `right` with default parameters.

13.3.2 Placing Turtles Using Mouse Clicks

We will continue our investigation of event-driven programming by looking at how our programs can respond to mouse-click events. To set up a callback for a mouse click, we call the `Turtle Screen`'s `onclick` method and pass it the function to call whenever the mouse is clicked. The main difference between the keyboard callbacks and the mouse callbacks is that the function we write to handle mouse callbacks must accept two parameters. When the callback method executes, the parameters will contain the `x` and `y` coordinates of the cursor location when the mouse was clicked.

As a first example, let's write an application to place turtles in the window. Wherever the user clicks the mouse, a new turtle will appear. The number of turtles that the user is allowed to create is passed as a parameter during start-up time. Once the user has created the maximum number of turtles, no more turtles should be created.

To solve this problem, we create a class called `TurtlePlace`, which has three methods (see **Listing 13.5**). The `_ _init_ _` method creates the initial window and—unfortunately—an initial turtle. We hide the initial turtle. The `_ _init_ _` method also calls `onclick` to register `placeTurtle` as the method to be called whenever the mouse is clicked. In addition, `_ _init_ _` sets up a turtle counter so we can keep track of how many turtles we have created at any time. The constructor calls the `drawField method`, which draws a rectangle for the turtles to live in. Finally, `_ _init_ _` calls `mainloop` to begin processing events.

```
1 from turtle import Turtle, mainloop
2 import random
3
4 class TurtlePlace:
5     def __init__(self, maxTurtles, hWall = 200, vWall = 200):
6         self._bigT = Turtle()
```

```
7           self._bigTscreen = self._bigT.getscreen()
8           self._bigT.color('blue')
9           self._bigTscreen.onclick(self.placeTurtle)
10          self._bigT.hideturtle()
11          self._numTurtles = 0
12          self._maxTurtles = maxTurtles
13          self._hWall = hWall
14          self._vWall = vWall
15          self.drawField()
16          mainloop()
17
18      def placeTurtle(self, x, y):
19          newT = Turtle()
20          newT.hideturtle()
21          newTscreen = newT.getscreen()
22          newTscreen.tracer(0)
23          newT.up()
24          newT.goto(x, y)
25          newT.shape('turtle')
26          newT.showturtle()
```

```
27          newT.setheading(random.randint(1, 359))
28          newTscreen.tracer(1)
29          self._numTurtles = self._numTurtles + 1
30          if self._numTurtles >= self._maxTurtles:
31              self._bigTscreen.onclick(None)   #cancel event handler
32
33      def drawField(self):
34          self._bigTscreen.tracer(0)
35          self._bigT.up()
36          self._bigT.goto(-(self._hWall), -(self._vWall))
37          self._bigT.down()
38          for i in range(4):
39              self._bigT.forward(2 * self._hWall)
40              self._bigT.left(90)
41          self._bigTscreen.tracer(1)
```

Listing 13.5 A turtle placement program

Description

The `placeTurtle` method creates a new turtle at position `(x, y)` in the window. It updates the number of turtles created, gives the turtle a random heading, and gives the turtle a fancy shape. Finally, `placeTurtle` checks whether `maxTurtles` have been created. If the limit on turtles is reached,

then `placeTurtle` calls `onclick` and passes `None` as the function to call. Passing `None` to the `onclick` function effectively cancels the callback mechanism so further mouse clicks do not call `placeTurtle`.

To test the `TurtlePlace` implementation, create an instance of `TurtlePlace`, passing the maximum number of turtles as a parameter. Because `mainloop` is called on the last line of the `__init__` method, `__init__` will never return and the event loop is started.

Try It Out

13.10 Modify the `placeTurtle` method so that turtles cannot be placed outside the boundary box.

13.11 Modify the `TurtlePlace` class so that after all the turtles have been created, subsequent mouse clicks cause all the turtles to turn and move 10 units toward the point where the mouse was clicked.

13.12 Add an `onclick` event to `Etch` that causes the turtle to move to wherever the mouse was clicked.

13.13 Modify the drawing package from Chapter 12 to allow a user to place geometric objects on a canvas.

13.14 Create a program to draw a regression line through a set of points. The user should be able to click in the window to create the set of points. When the user clicks in a certain part of the window, the program should calculate and draw a best-fit line through the points. You will find the equations needed for calculating the regression line in Chapter 4.

13.3.3 Bouncing Turtles

The final turtle event handler we examine is `ontimer`. We will use a timer to animate the turtles that are placed in the window using the `TurtlePlace` program. The `ontimer` method takes two parameters: (1) the callback

function and (2) the number of milliseconds to wait before calling the function. Timer callback functions have no parameters.

For this part of our program, we would like to move the turtles that we placed in the last section. We can animate a turtle by calling the `forward` method every few milliseconds. Furthermore, we want the turtles to bounce off the walls of an imaginary box and off each other as they move around.

We can approach this problem in several ways. The first solution, which we leave for you to implement later, is to define a list instance variable, which keeps track of all turtles that have been created. Every so many milliseconds, a timer callback function can iterate over the turtles on the list and call the `forward` method on each turtle. This approach uses composition.

An alternative solution relies on inheritance. Suppose that rather than making a plain `Turtle`, we create a special kind of turtle called an `AnimatedTurtle`. Once an `AnimatedTurtle` is created, it automatically begins to wander around the box, bouncing off the walls and the other turtles.

Because the animated turtle class is not meant to be controlled like a regular turtle, we will not add any methods that we expect the user to call. To this end, we name our two methods beginning with double underscores to hide those methods from code outside the `AnimatedTurtle` class. As we have noted before, names that begin with two underscores are hidden from any functions outside the class, a process that Python calls **mangling**.

The important point to note in the `AnimatedTurtle` class is on line 12 of **Listing 13.6**, which calls the `ontimer` method to set up a callback for the `__moveOneStep` method, causing the method to execute after 100 milliseconds.

```
1 from turtle import Turtle, mainloop
2 import random
3
4 class AnimatedTurtle(Turtle):
5     def __init__(self, hWall, vWall):
```

```
6         super().__init__()
7         self.__scr = self.getscreen()
8         self.__xMin = -vWall
9         self.__xMax = vWall
10        self.__yMin = -hWall
11        self.__yMax = hWall
12        self.__scr.ontimer(self.__moveOneStep, 100)
13
14    def __moveOneStep(self):
15        self.__computeNewHeading()
16        self.forward(5)
17        self.__scr.ontimer(self.__moveOneStep, 100)
18
19    def __computeNewHeading(self):
20        xPos, yPos = self.position()
21        oldHead = self.heading()
22        newHead = oldHead
23
24        if xPos > self.__xMax or xPos < self.__xMin:
25            newHead = 180 - oldHead
```

```
26          if yPos > self.__yMax or yPos < self.__yMin:
27              newHead = 360 - oldHead
28          if newHead != oldHead:
29              self.setheading(newHead)
```

Listing 13.6 A class to implement an animated turtle

Description

The `_ _moveOneStep` method calls the `_ _computeNewHeading` method to check whether the turtle has run into one of the boundary walls. If so, `_ _computeNewHeading` turns the turtle 180 degrees. If not, `_ _moveOneStep` method moves the turtle forward by 5 units. Note that a timer callback is good for only one interval; it does not repeat. Therefore, the last thing we do in the `_ _moveOneStep` method is reset the timer callback to execute again after another 100 milliseconds.

HEADS UP:
The `ontimer` callback will be called only once when the interval expires; it does not repeat. The callback code usually re-registers the callback for another interval.

To integrate the `TurtlePlace` class with the `AnimatedTurtle` class, we need to modify the `placeTurtle` method so that it creates an `AnimatedTurtle` rather than a plain `Turtle`. The `AnimatedTurtle` also requires that we pass its boundaries as parameters. The new `placeTurtle` method is shown in **Listing 13.7**, where line 2 is different from the same method in Listing 13.5. We also color the animated turtles green in line 9.

```
1   def placeTurtle(self, x, y):
2       newT = AnimatedTurtle(self._hWall, self._vWall)
3       newT.hideturtle()
4       newTscreen = newT.getscreen()
5       newTscreen.tracer(0)
6       newT.up()
7       newT.goto(x, y)
8       newT.shape('turtle')
9       newT.color('green')
10      newT.showturtle()
11      newT.setheading(random.randint(1, 359))
12      newTscreen.tracer(1)
13      self._numTurtles = self._numTurtles + 1
14      if self._numTurtles >= self._maxTurtles:
15          self._bigTscreen.onclick(None) #remove event handler
```

Listing 13.7 A modified `placeTurtle` method

Description

At this point, we must load both the modified `TurtlePlace` and `AnimatedTurtle` turtle classes. Once the classes are loaded, we can start the whole program by typing

```
TurtlePlace(5, 200, 200)
```

This statement creates a new instance of the `TurtlePlace` class that allows us to create five new turtles in a box with dimensions of 400 by 400 units.

The turtles bounce off the walls, but at this point they go right through each other if their paths cross. Clearly, this is not a realistic simulation of turtle behavior. The next step to make our program more realistic is to add the ability for turtles to bounce off each other.

There is no magic to this part of the solution. To find out if a turtle has "hit" another turtle, we will need to check the distance between the moving turtle and all the other turtles. You might be tempted to think that some very elegant solution to this problem exists. In fact, for this application, a simple brute-force check of all other turtles will work quite well.

The question is how to keep track of all the turtles. Of course, we will want to use a list, but where will we keep that list? Who is responsible for adding turtles to the list, and how can we ensure that each `AnimatedTurtle` has access to the list of all other turtles? We could use a `turtleList` instance variable in the `TurtlePlace` class, but then every animated turtle would need to have a reference to an instance of `TurtlePlace`.

The answer is to use a **static variable**—a variable that is shared by all instances of a class and is available to all the methods in the class. **Listing 13.8** creates a static variable on line 6. As you can see, static variables are defined inside the class, but outside any method definition. Because a static variable is shared by all instances, we reference a static variable using the class name rather than `self`.

```
1 from turtle import Turtle, mainloop
2 import random
3
4 class AnimatedTurtle(Turtle):
5
```

```
    _allTurtles = []

    def _init_(self, hWall, vWall):
        super()._init_()
        self._scr = self.getscreen()
        self._xMin = -vWall + 10
        self._xMax = vWall - 10
        self._yMin = -hWall + 10
        self._yMax = hWall - 10
        self._scr.ontimer(self._moveOneStep, 100)
        AnimatedTurtle._allTurtles.append(self)

    def _moveOneStep(self):
        self._computeNewHeading()
        self.forward(5)
        self._checkCollisions()
        self._scr.ontimer(self._moveOneStep, 100)

    def _computeNewHeading(self):
        xPos, yPos = self.position()
```

```
        oldHead = self.heading()
        newHead = oldHead

        if xPos > self._xMax or xPos < self._xMin:
            newHead = 180 - oldHead
        if  yPos > self._yMax or yPos < self._yMin:
            newHead = 360 - oldHead
        if newHead != oldHead:
            self.setheading(newHead)

    def _checkCollisions(self):
        for otherT in AnimatedTurtle._allTurtles:
            if self != otherT:
                if self.distance(otherT) < 20:    #if close, swap headings
                    tempHeading = self.heading()
                    self.setheading(otherT.heading())
                    otherT.setheading(tempHeading)
                    while self.distance(otherT) < 20: #move away
                        self.forward(1)
                        otherT.forward(1)
```

Listing 13.8 An `AnimatedTurtle` class with turtle collision detection

Description

RECAP:
A static variable is:

- Shared by all instances of a class
- Defined inside a class definition, but outside any method
- Referenced using the class name rather than `self`

With the static variable `__allTurtles` defined, the `__init__` method can add new turtles to this list as they are created using the statement

```
AnimatedTurtle.__allTurtles.append(self)
```

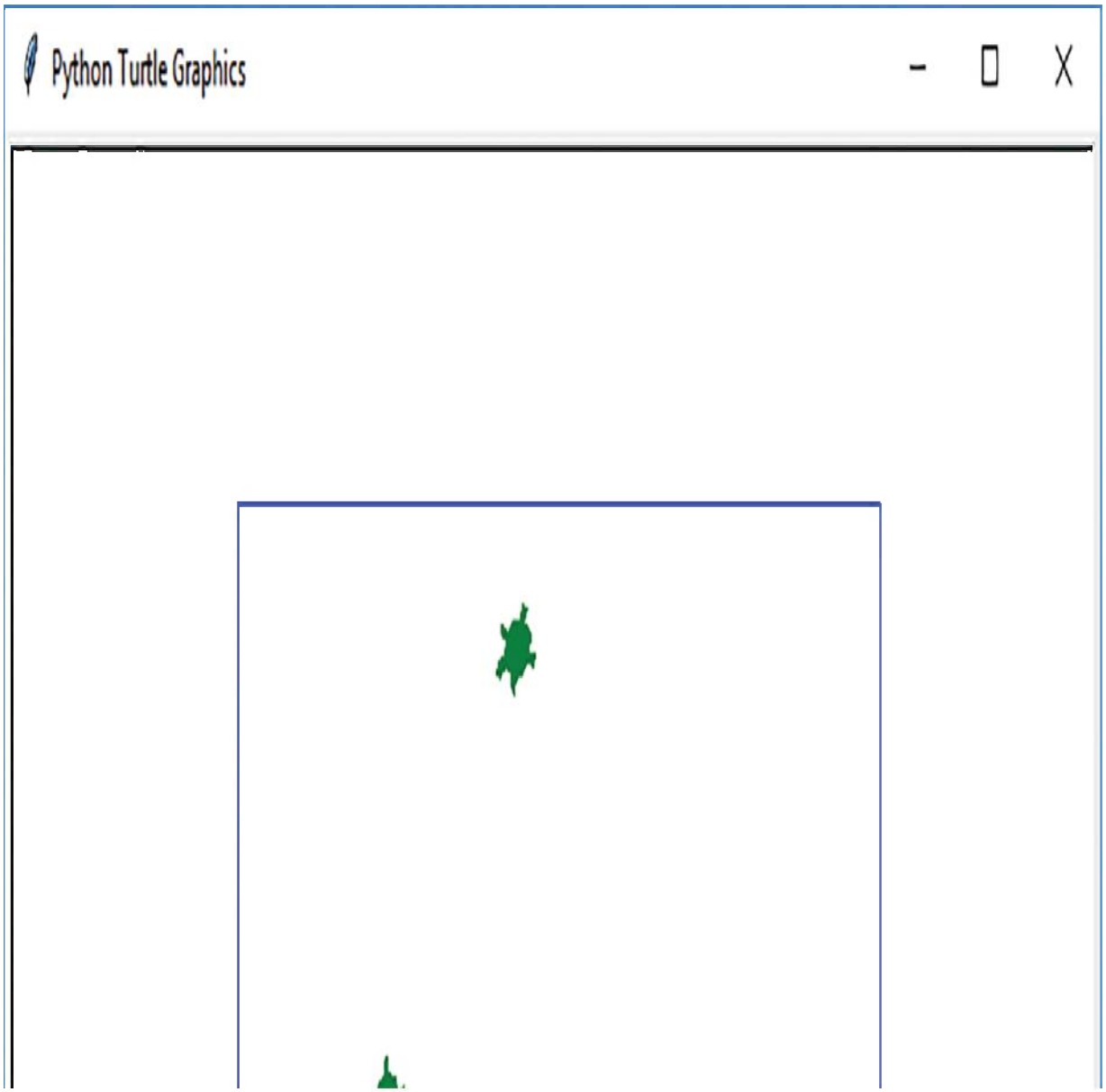
Similarly, the `__checkCollisions` method can now access the list of all turtles by referencing the variable `AnimatedTurtle.__allTurtles`. Remember that when we reference a static variable, we use the class name rather than `self`. The beauty of the static variable approach is that it keeps the accounting of all the animated turtles inside the `AnimatedTurtle` class.

To calculate the approximate distance between two turtles, we will use the `distance` method provided by the `turtle` module; this method calculates the distance between the center points of two turtles. To make our simulation look more realistic, we need to think about the distance between the edges of the circles that circumscribe the two turtles. In other words, if the edges of the circles touch, we should consider that the two turtles have collided.

With this in mind, take a look at the `__checkCollisions` method in Listing 13.8. This method uses two loops. One loop iterates over all the turtles on the `AnimatedTurtle.__allTurtles` list. The statement `if self != otherT` ensures that we do not compute the distance between the

same turtle using two different references. If the turtles are too close, then they bounce off each other by exchanging their headings. This behavior assumes that turtles collide in a perfectly elastic collision, which may not be the case in nature but is easy to program. The loop—a `while` loop—moves both turtles forward along their new headings to ensure that they are no longer inside the collision distance. If we omit the `while` loop, our program may produce some funny-looking simulations in which the turtles appear to be stuck together for a short time.

Figure 13.3 shows five turtles moving around the screen in various directions.



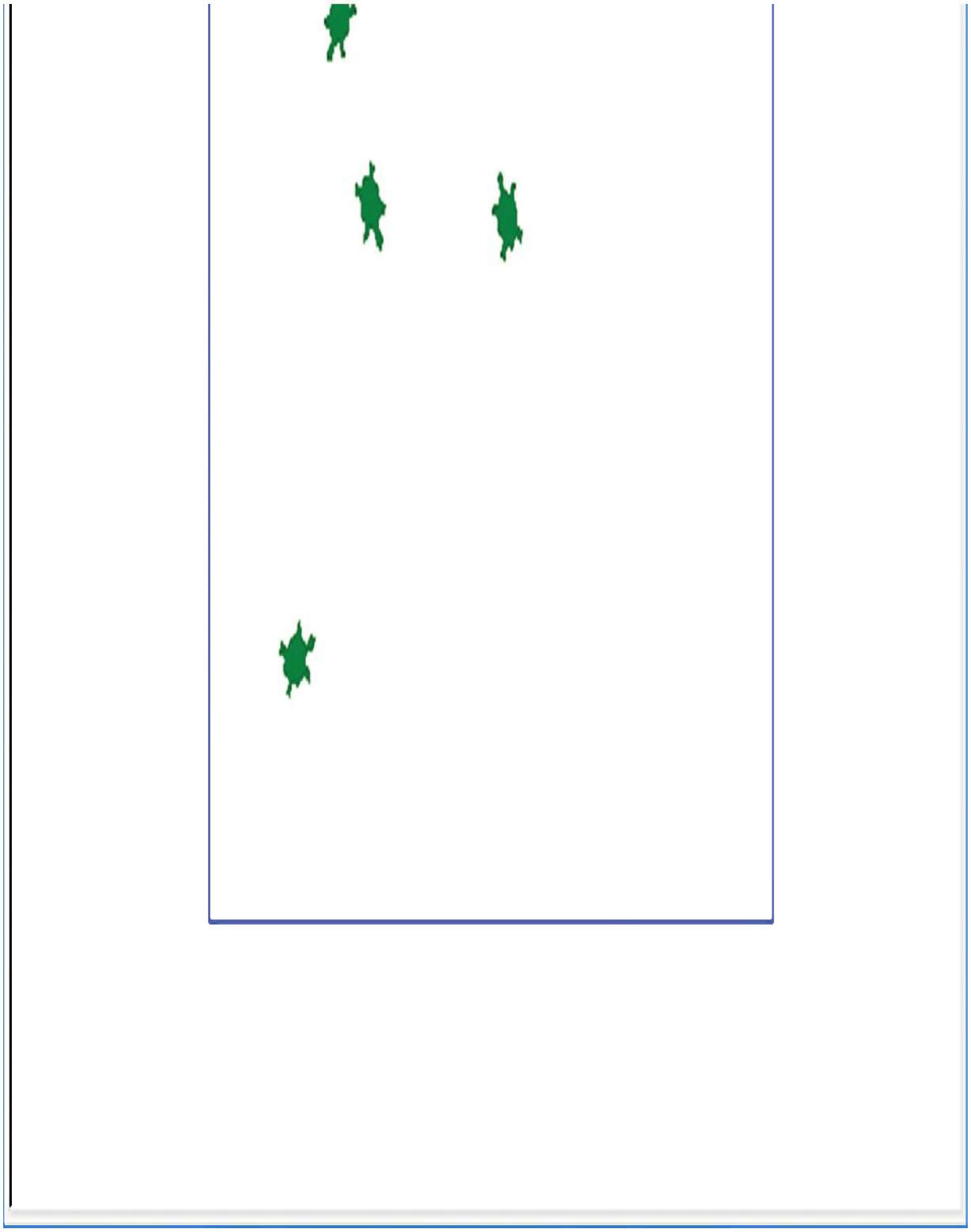


Figure 13.3 Animated turtles with collision detection.

© 2001–2025 Python Software Foundation; All Rights Reserved.

Try It Out

13.15 Modify the `__init__` method to give each new turtle a random color.

13.16 Add start and stop methods to the `AnimatedTurtle` class.

13.17 Add a speed method that allows you to alter the speed of a turtle by changing the number of milliseconds passed to the `ontimer` method.

13.18 Add a "q" key event to the bouncing turtles program. When the user presses the "q" key, the program should exit.

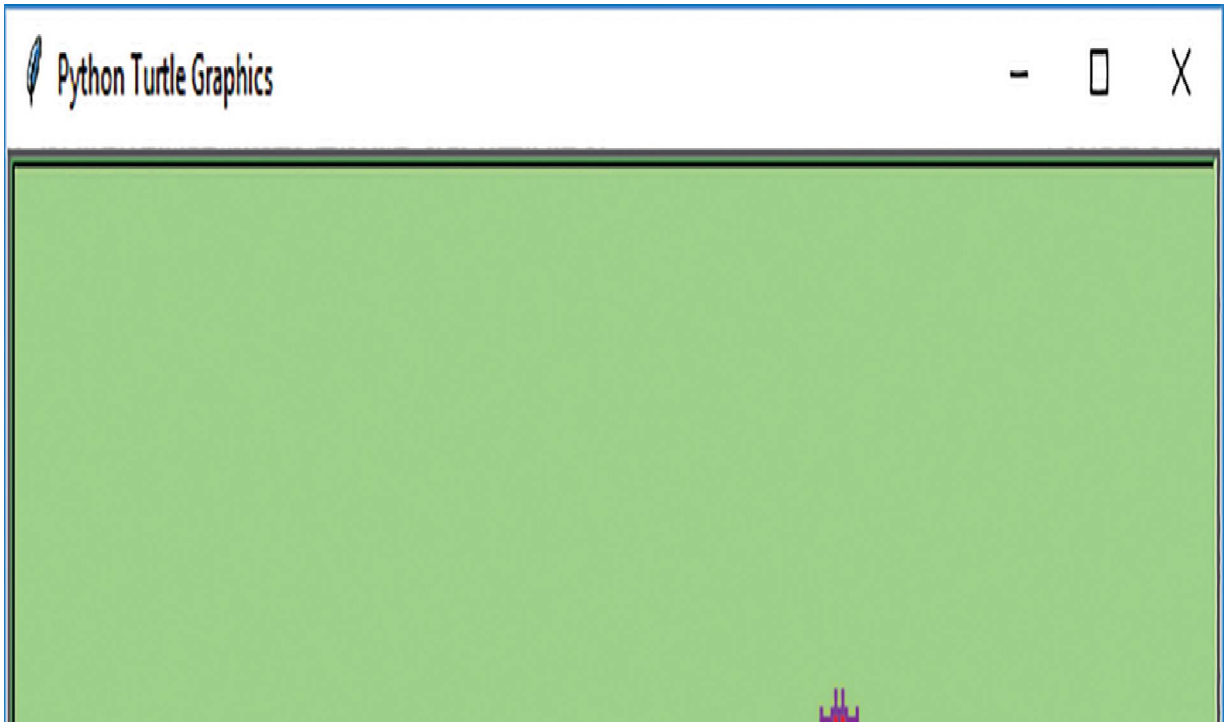
13.19 Add a mouse event to the bouncing turtles program so that if the user clicks on a turtle that is moving, the turtle stops. Conversely, if the user clicks on a turtle that is stopped, the turtle starts.

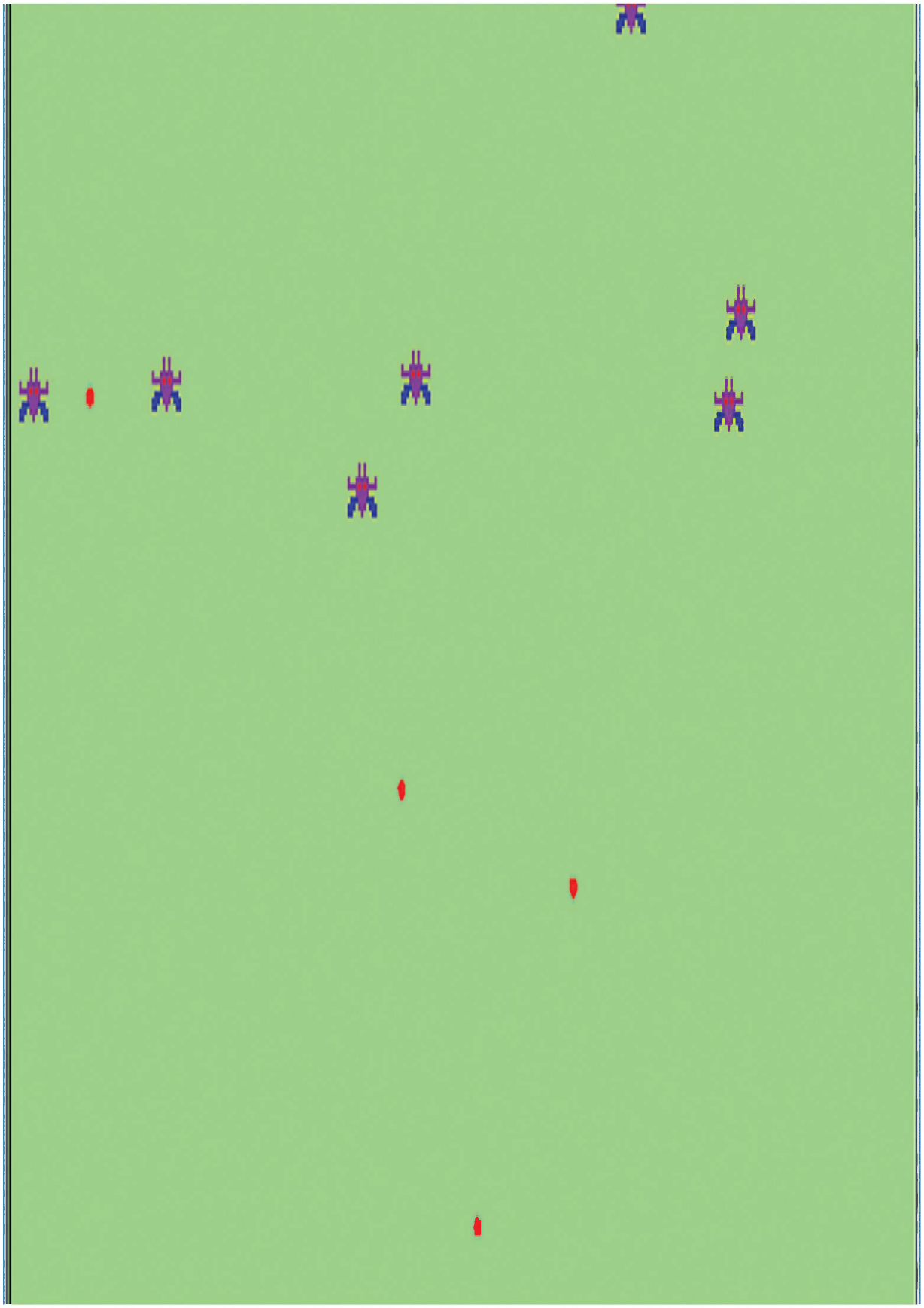
13.20 Rewrite the bouncing turtles program using composition rather than inheritance. After you are done, compare the two solutions. What are the strengths and weaknesses of each approach?

13.4 Creating Your Own Video Game

By now, you should have a good understanding of event-driven programming using the `turtle` module. We will finish this chapter by pulling everything together to create a simple video game—our updated version of the old Space Invaders game, in which aliens appear in the sky and you must shoot them with a laser cannon before they reach the ground. In our game, however, drones will appear in the sky rather than aliens.

Figure 13.4 shows a screenshot of the game in action. In the figure, notice the three main objects that we will implement to develop our video game. At the bottom center of the window is the laser cannon, which is the usual turtle triangle. The drones are the enemies, and the small circles are the bombs that have been shot from the laser cannon. These three things will become our main classes: `LaserCannon`, `Drone`, and `Bomb`. Each of these classes inherits from the `Turtle` class, so we can control each element of our game as a `Turtle`. The different appearance of each object comes from using the turtle `shape` method that allows us to use image files and even shapes you draw yourself to represent a turtle.





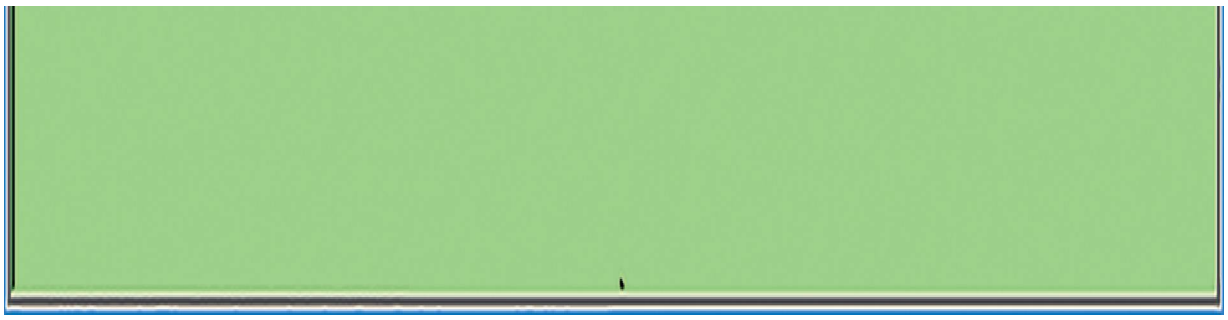


Figure 13.4 Drone invaders.

© 2001–2025 Python Software Foundation. All Rights Reserved.

Description

The next step is to identify the capabilities of each of our classes. The `LaserCannon` must be able to do two things:

- Aim at `Drone`s.
- Fire `Bomb`s.

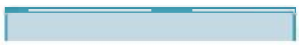
A `Bomb` must be able to do three things:

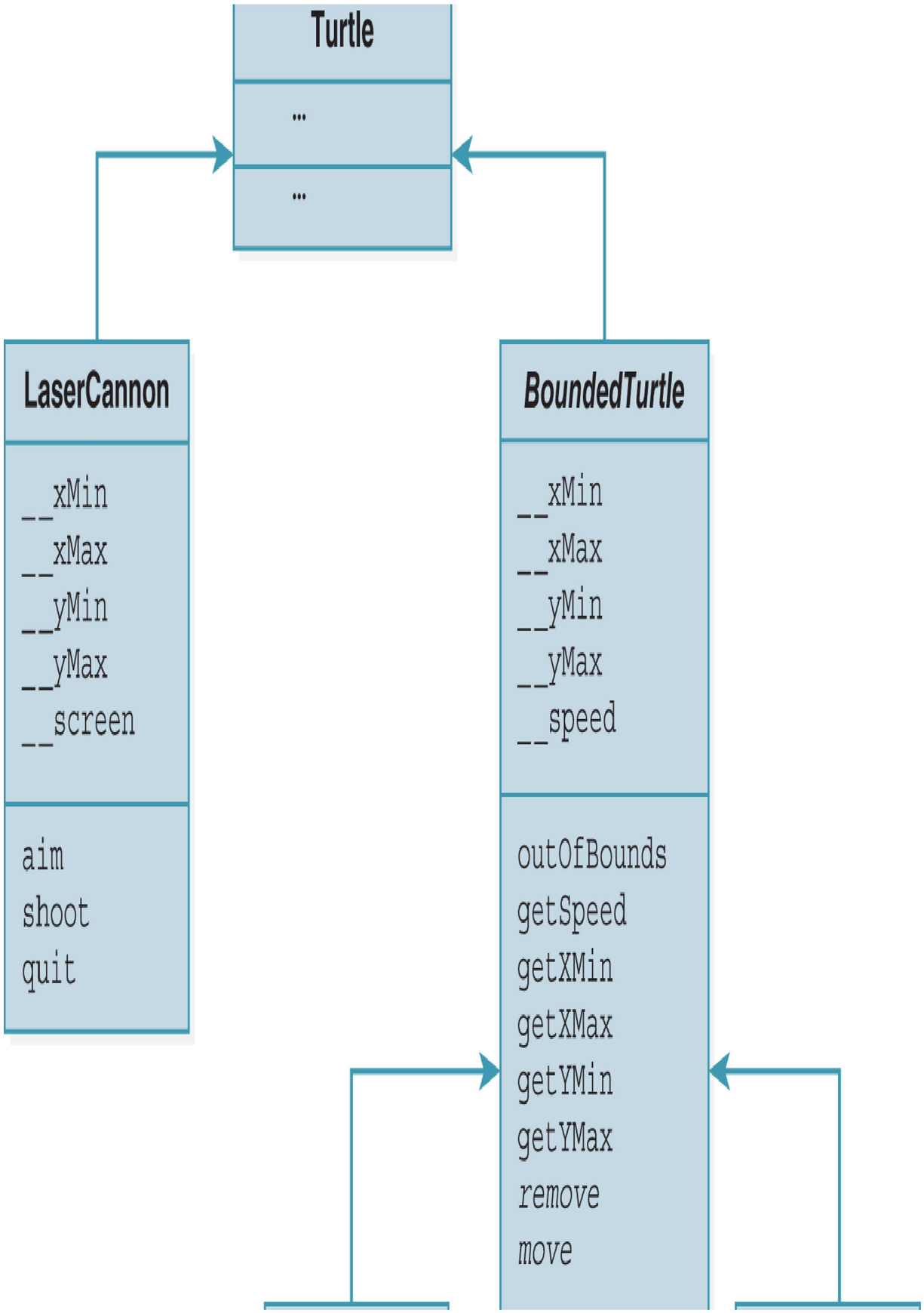
- Move in the direction it was fired.
- Detect when it hits a `Drone`.
- Detect when it disappears out of the window.

A `Drone` must be able to do two things:

- Move from the top of the screen to the bottom.
- Remove itself from the screen after it has been hit by a bomb or when it disappears from the window.

This high-level design is captured in **Figure 13.5**. As is standard UML practice, the abstract `BoundedTurtle` and its two abstract methods, `move` and `remove`, are italicized. Also as is standard practice with UML, in the `Drone` class, the static variable `droneList` and the static method `getDrones` are underlined.




Turtle
...
...
LaserCannon
__xMin
__xMax
__yMin
__yMax
__screen
aim
shoot
quit
BoundedTurtle
__xMin
__xMax
__yMin
__yMax
__speed
outOfBounds
getSpeed
getXMin
getXMax
getYMin
getYMax
remove
move

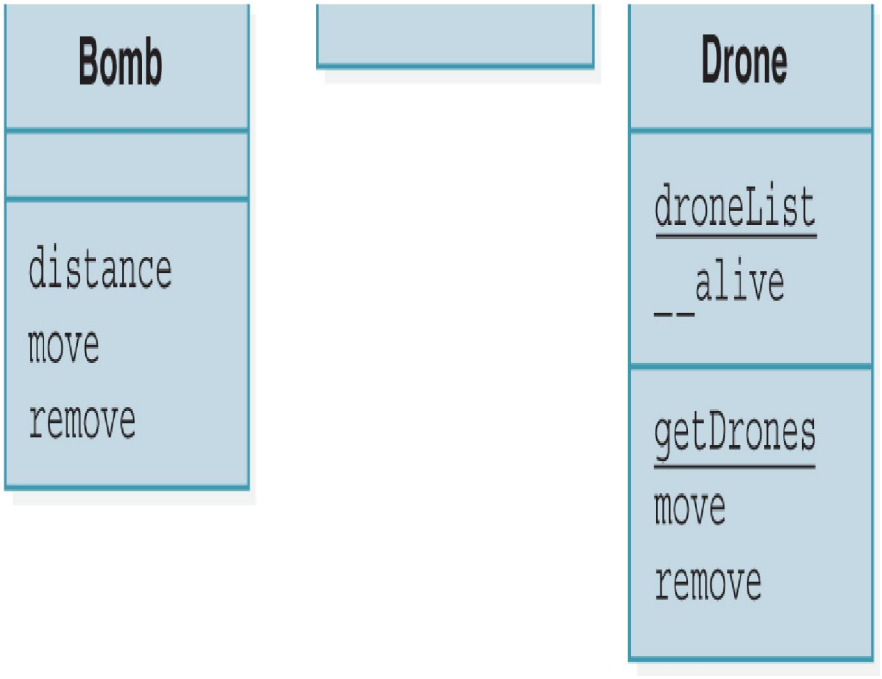


Figure 13.5 Drone invaders design.

Description

13.4.1 The `LaserCannon` Class

Let's look at the implementation of the `LaserCannon` class in more detail (see **Listing 13.9**). This class should look somewhat familiar to you after studying the bouncing turtles example from the previous section. The `LaserCannon` will serve as our initial turtle, so it will create the window for all the other turtles that come later. In the `_ _init_ _` method, we create a turtle and set up three callback functions: `aim`, `shoot`, and `quit`. A mouse click calls the `aim` method, which turns the laser cannon toward the position of the cursor where the mouse was clicked. Pressing the "s" key calls `shoot`. The `shoot` method is very short: It just creates a new `Bomb`. As you will see later in Listing 13.12, the constructor for `Bomb` takes care of the rest of the work.

```
1 class LaserCannon(Turtle):
2     def __init__(self, xMin, xMax, yMin, yMax):
```

```
        super().__init__()
        self._xMin = xMin
        self._xMax = xMax
        self._yMin = yMin
        self._yMax = yMax
        self._screen = self.getscreen()
        self._screen.onclick(self.aim)
        self._screen.onkey(self.shoot, 's')
        self._screen.onkey(self.quit, 'q')

    def aim(self, x, y):
        heading = self.towards(x, y)
        self.setheading(heading)

    def shoot(self):
        Bomb(self.heading(), 5, self._xMin, self._xMax,
                                self._yMin, self._yMax)

    def quit(self):
        self._screen.bye()
```

Listing 13.9 The `LaserCannon` class

Description

13.4.2 The `BoundedTurtle` Class

The `BoundedTurtle` class is an abstract class that will be the superclass of both the `Drone` and `Bomb` classes. The difference between a regular `Turtle` and a `BoundedTurtle` is that the `BoundedTurtle` knows when it is outside the window. When the turtle is outside the window, we want it to disappear.

Listing 13.10 shows the code for the `BoundedTurtle` class. In the `outOfBounds` method, the statement `xPos, yPos = self.position()` is equivalent to two statements that get the `x` and `y` coordinates of the turtle separately. The turtle `position` method returns a tuple, but the assignment statement unpacks the tuple and assigns the corresponding elements to the variables on the left-hand side. The `outOfBounds` method returns `True` or `False` depending on whether the turtle has gone outside its boundaries.

```
1 from abc import *
2 class BoundedTurtle(Turtle):
3     def __init__(self, speed, xMin, xMax, yMin, yMax):
4         super().__init__()
5         self.__xMin = xMin
6         self.__xMax = xMax
7         self.__yMin = yMin
8         self.__yMax = yMax
9         self.__speed = speed
```

```
10
11      def outOfBounds(self):
12          xPos, yPos = self.position()
13          out = False
14          if xPos < self._xMin or xPos > self._xMax:
15              out = True
16          if yPos < self._yMin or yPos > self._yMax:
17              out = True
18          return out
19
20      def getSpeed(self):
21          return self._speed
22
23      def getXMin(self):
24          return self._xMin
25
26      def getXMax(self):
27          return self._xMax
28
29      def getYMin(self):
```

```
30          return self._yMin
31
32      def getYMax(self):
33          return self._yMax
34
35      @abstractmethod
36      def remove(self):
37          pass
38
39      @abstractmethod
40      def move(self):
41          pass
```

Listing 13.10 Bounded turtles can check whether they are out of bounds

Description

The `BoundedTurtle` class defines two abstract methods: `move` and `remove`. The `Bomb` and `Drone` classes will need to define these methods, and Python will use polymorphism to determine which class's version of the method to call, depending on the class of the object used to call the methods.

13.4.3 The `Drone` Class

The next class we will implement is `Drone`. Our drones need to be able to move down the screen, and they need to remove themselves from the game when they are hit by a `Bomb` or when they reach the bottom of the screen.

As in the bouncing turtle simulation, we will keep track of all drones so that when a bomb moves, we can test whether it has hit any of the drones. A static variable, `droneList`, keeps track of all the drones that are alive.

To keep things simple, we want only one method to add or remove drones from the `droneList`. We will perform this task in the `Drone` constructor. When a new drone is created, we append it to the list. When a drone is hit by a bomb, we set the `_ _alive` instance variable for that drone to `False`.

Because the `Bomb` class needs to know which drones it can blow up, it must get a list of all the "live" drones. To do this, the `Drone` class will provide a **static method** called `getDrones` to return a list of live `Drone`s. Like static attributes, static methods are independent of objects. Static methods are useful when you have a method that belongs inside a class but does not apply to any instance of the class.

Listing 13.11 shows the code for the `Drone` class. Notice that no `self` parameter appears in the formal parameter list of the static method `getDrones`. Also notice that we use a new decorator, `@staticmethod`, to create a static method. The `@staticmethod` decorator tells the Python interpreter that there is no need to pass an instance variable as an implicit parameter to the function `getDrones`. Instead, we call the method using the class name, as in `Drone.getDrones()`, without needing an instance of the `Drone` class.

```
1 class Drone(BoundedTurtle):
2
3     droneList = []     #static variable
4
5     @staticmethod
6     def getDrones():
7         return [x for x in Drone.droneList if x._alive]
```

```
8
9     def __init__(self, speed, xMin, xMax, yMin, yMax):
10        super().__init__(speed, xMin, xMax, yMin, yMax)
11        self.getscreen().tracer(0)
12        self.up()
13        if 'Drone.gif' not in self.getscreen().getshapes():
14            self.getscreen().addshape('Drone.gif')
15        self.shape('Drone.gif')
16        self.goto(random.randint(xMin - 1, xMax - 1), yMax - 20)
17        self.setheading(random.randint(250, 290))
18        self.getscreen().tracer(1)
19        Drone.droneList = Drone.getDrones()
20        Drone.droneList.append(self)
21        self.__alive = True
22        self.getscreen().ontimer(self.move, 200)
23
24    def move(self):
25        self.forward(self.getSpeed())
26        if self.outOfBounds():
27            self.remove()
```

```
28          else:
29              self.getscreen().ontimer(self.move, 200)
30
31      def remove(self):
32          self._alive = False
33          self.hideturtle()
```

Listing 13.11 The `Drone` class

Description

RECAP:
A static method:

- Cannot access instance variables
- Does not take a `self` parameter
- Is called using the class name
- Is indicated using the `@staticmethod` decorator

`Drone.getDrones` returns all the live drones by constructing a new list created from the instances of `Drone` where `_ _alive` is `True`. The other Python construct that is part of the `getDrones` method is the list comprehension on line 7. List comprehension provides a concise way to create a list without using a `for` loop with an append inside it. **Session 13.3** illustrates some uses of list comprehensions. The first example shows it is legal to apply Python operators to the variable you are using to create the list.

```
In [1]: 1 [x**2 for x in range(5)]        #list of squares from 1 to 4

Out[1]: [0, 1, 4, 9, 16]
```

```
In [2]: 1 movies = ['alien', 'starwars', 'bourne']
```

```
In [3]: 1 [x.capitalize() for x in movies]  #capitalize each movie

Out[3]: ['Alien', 'Starwars', 'Bourne']
```

```
In [4]: 1 [x[-1] for x in movies if len(x) > 6] #last letter if longer than 6 chars

Out[4]: ['s']
```

```
In [5]: 1 [x[-1] for x in movies if len(x) <= 6] #last letter if 6 or fewer chars

Out[5]: ['n', 'e']
```

Session 13.3 Using list comprehensions

Description

The `getDrones` method on lines 6–7 of Listing 13.11 uses a list comprehension and is equivalent to the `getDrones` method shown here:

```
1  def getDrones():
2      myList = []
3      for x in Drone.droneList:
4          if x.__alive:
5              myList.append(x)
6      return myList
```

Description

Notice two important points about the `__init__` method. First, it is the only method that modifies the `droneList`. This ensures that we avoid the situation in which one callback method could be adding a drone to the list while another is trying to remove a drone from the list. This could happen if a bomb hit a drone at the same time that another drone was being created.

Second, notice that the `Drone`'s `remove` method sets the value of `__alive` to `False` and hides the turtle, but does not remove the drone from the `droneList`. In the constructor, we use a list comprehension to filter out the dead drones every time a new drone is created. This performance improvement keeps the list nice and short.

Also notice the conditional statement on lines 13–14. This statement checks whether the `'Drone.gif'` file has been loaded. If it has been loaded once, it is not necessary to load it again, because the `addshape` method just loads a list of shapes that works for all the turtles in the same window.

The `move` method checks whether the drone is out of bounds. If the drone stays in bounds, the timer is reset so that the drone will move again in 200 milliseconds. If the drone is out of bounds, the drone is removed, and the timer is not reset.

The `remove` method marks the drone as dead and hides it so it disappears. By setting `_ _alive = False`, the drone stays on the list (until another drone is created), but will not be included in any calculations involving bombs that are shot from the laser cannon.

13.4.4 The `Bomb` Class

Our final class, `Bomb`, is no more complicated than the bouncing turtle. A bomb is created with an initial heading corresponding to the direction in which the cannon is pointing. Once the bomb is created, it continues moving in that direction. If it runs into a drone, the bomb explodes, the drone is marked as dead, and the bomb hides itself and ceases to move. **Listing 13.12** shows the `Bomb` class. In the `distance` method, we calculate the distance between the bomb and each `Drone` using the `math.dist` method (lines 24–27).

```
1 class Bomb(BoundedTurtle):
2     def __init__(self, initHeading, speed, xMin, xMax, yMin, yMax):
3         super().__init__(speed, xMin, xMax, yMin, yMax)
4         self.resizemode('user')
5         self.color('red', 'red')
6         self.shape('circle')
7         self.turtlesize(.25)
8         self.setheading(initHeading)
9         self.up()
```

```
        self.getscreen().ontimer(self.move, 100)

    def move(self):
        exploded = False
        self.forward(self.getSpeed())
        for a in Drone.getDrones():
            if self.distance(a) < 5:
                a.remove()
                exploded = True
        if self.outOfBounds() or exploded:
            self.remove()
        else:
            self.getscreen().ontimer(self.move, 100)

    def distance(self, other):
        p1 = self.position()
        p2 = other.position()
        return math.dist(p1, p2)

    def remove(self):
```

```
30          self.hideturtle()
```

Listing 13.12 The `Bomb` class

Description

13.4.5 Putting All the Pieces Together

To put together all the pieces of the game and start things running, we will create one final class, `DroneInvaders`, which will be the main application. **Listing 13.13** provides the code for the `DroneInvaders` class. The main task of this class is to create a `LaserCannon` object and to set up a timer to add drones to the playing field at regular intervals. These tasks are implemented in the `play` method.

```
1  class DroneInvaders:
2      def __init__(self, xMin, xMax, yMin, yMax):
3          self._xMin = xMin
4          self._xMax = xMax
5          self._yMin = yMin
6          self._yMax = yMax
7
8      def play(self):
9          self._mainWin = LaserCannon(self._xMin, self._xMax,
10                                     self._yMin, self._yMax).getscreen()
11         self._mainWin.bgcolor('light green')
```

```
12          self._mainWin.setworldcoordinates(self._xMin, self._yMin,
13                                            self._xMax, self._yMax)
14          self._mainWin.ontimer(self.addDrone, 1000)
15          self._mainWin.listen()
16          mainloop()
17
18      def addDrone(self):
19          if len(Drone.getDrones()) < 7:
20              Drone(1, self._xMin, self._xMax,
21                       self._yMin, self._yMax)
22          self._mainWin.ontimer(self.addDrone, 1000)
```

Listing 13.13 Putting it all together

Description

To play the game, we execute the two commands shown in **Session 13.4**. We should add these commands to the bottom of the `.py` file with all of the class definitions so we can run everything with a single command. At the top of the `.py` file, we should include `import` statements for `turtle`, `random`, `math`, and `abc`.

```
In [1]:  1 game = DroneInvaders(-200, 200, 0, 400)
         2 game.play()
```

Description

Try It Out

13.21 Modify the drone invaders game to keep score. The user should earn 10 points for each drone hit and should lose 10 points for each drone that falls to the ground.

13.22 Modify the game to increase the difficulty level as the score gets higher. That is, more drones should appear on the screen, and they should move faster.

13.23 Modify the game so that the user can press the arrow keys instead of using a mouse click to aim the laser cannon.

13.24 Modify the game so that if a drone lands on the laser cannon, the game is over.

13.25 Modify the game so that the laser cannon always shoots straight up, but the user can slide it back and forth across the bottom of the window.

13.26 Modify the game so that the user can click a button that resets the score and all invaders.

13.5 Summary

In this chapter, we extended our discussion of object-oriented programming by introducing event-driven programming. We used multithreaded event-driven programming to implement a simple video game. We designed our video game by creating classes to represent the components of the game. We tied functions to mouse clicks and key presses so that actions would occur when those input events took place. We used timers to place the video game in motion. As part of the implementation, we used inheritance in building turtle objects used in the game. In addition, we used a static variable that could be shared by each object of the class. We also created a static method that did not belong to any object of the class, but could access static data defined in the class.

Key Terms

callback function
event-driven programs
event loop
mangling
multithreaded program
multithreading
queue
register
single-threaded
static method
static variable
thread

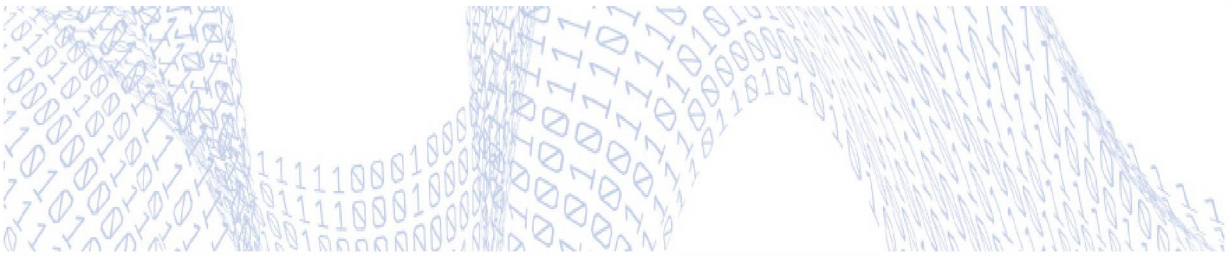
Programming Exercises

13.1 Write your own video game. Here are some simple games to think about:

- One-player Pong
- One-player Brickout
- Frogger
- Racetrack
- Asteroids
- Two-player Pong
- Tank battle

Your game should keep score and utilize each of the callbacks discussed in this chapter.

Design Credits: Calculator Icon made by Smashicons from www.flaticon.com



© Yuri Hoyda/Shutterstock.

Appendix A
Installing the Required Software

All of the Python software used in this text is free and open source. The following sections provide pointers about where to find the software, as well as some links to instructions for installing it. As software is always a moving target and the versions of Python and some of the modules used in this text are bound to change, we will not try to give you specific instructions for installing Python on every operating system and platform.

A.1 Installing Python

The latest version of Python is always available from www.python.org. Click the **Download** link and follow the instructions for your operating system to install it.

Mac OS X

If you have a Mac and are using an older version of the Mac OS, an earlier version of Python (2.x) is already installed on your machine. However, many of the examples in this text will not run with this older version. We strongly recommend that you install Python version 3.10 or later. Do not uninstall the earlier version, however, because some Apple and third-party software use the earlier version.

A.2 Installing the Jupyter Notebook IDE

The IDLE development system is included with the Python download. You are welcome to use this system for entering and running Python code.

In this text, however, we have used Jupyter Notebook to generate the listings and sessions. We find that Jupyter Notebook has a better user interface, making it easier to use. To install the Jupyter Notebook library, type the following command in the command line[1]:

```
pip install jupyter
```

Once installed, to run the Jupyter Notebook, execute the following from the command line:

```
jupyter notebook
```

Jupyter Notebook runs in your web browser. When you run the above command, a browser window will open and you should see a home page like the following:

Files | Running | Clusters

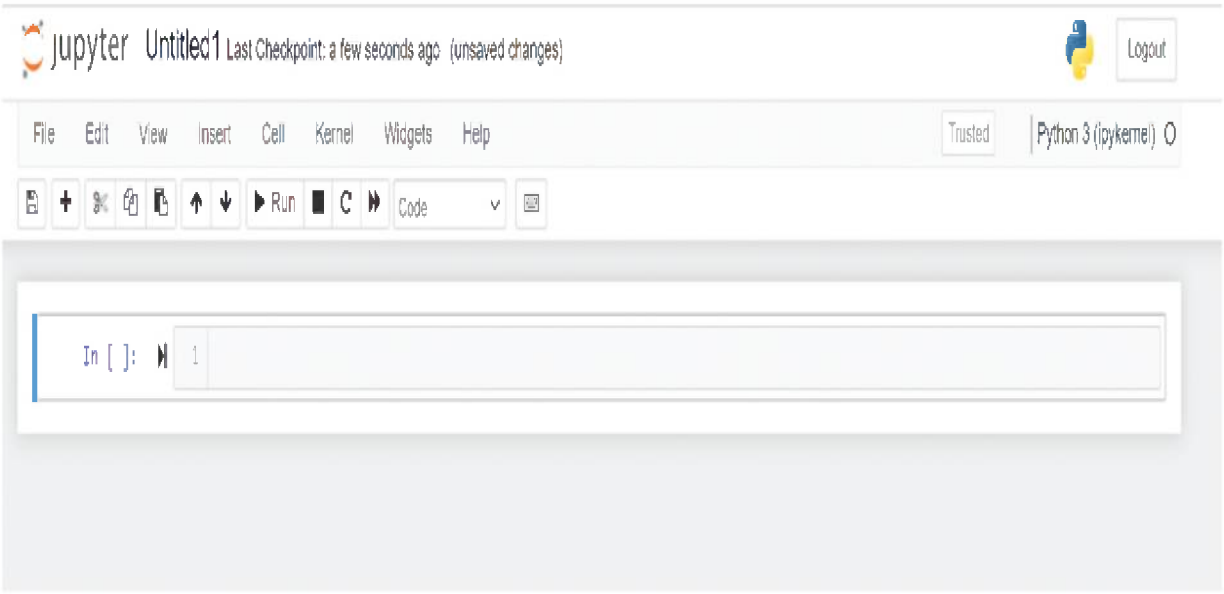
Select items to perform actions on them.

Upload | New ▾ | ↻

0 ▾ /	Name ↓	Last Modified	File size
3D Objects		3 minutes ago	
AppData		a year ago	
Contacts		2 years ago	
Desktop		4 days ago	
Documents		a month ago	
Downloads		14 days ago	
Dropbox		6 months ago	
Favorites		2 years ago	
Links		2 years ago	
Music		2 years ago	
OneDrive		4 hours ago	
Pictures		a year ago	
Saved Games		a minute ago	
Searches		2 years ago	
Videos		2 years ago	



To create a new Python file, navigate to the appropriate folder, click `new` menu item on the top right, and select `Python 3` as the type of file. You should see a new window like this:



© Jupyter.

Type your Python commands in the box and press `Run` to run the command. To run all commands, click `>>`.

[1] If you get a "pip not found" error in your terminal window, enter `pip install` to install the package installer for Python. Then resume installing Jupyter Notebook.

A.3 Installing the `Matplotlib` and `pandas` Libraries

If you have installed Jupyter Notebook, two external modules, `Matplotlib` and `pandas`, are automatically installed as well. If you are using another development system, you might need to install these libraries.

To install, type the following commands in the command line:

```
pip install matplotlib
pip install pandas
```

A.4 Installing the Python Image Library and `cImage`

The image library used in Chapter 6 relies on two Python modules. The first module is `cImage.py`, which is provided with the source code for Chapter 6. You can either use the `cImage.py` file in the Chapter 6 folder or move it to the `site-packages` folder. In Windows, the `site-packages` folder is located in the `Lib` folder under Python's installation folder. In this way, Python will find `cImage.py` from any working folder. The `cImage` module will work with gif and ppm files with no additional modules required.

The second module, called the Python Image Library, should also be installed. `Pillow` allows image processing of JPGs and several other image formats. To install `Pillow`, on the command line, type

```
pip install pillow
```



© Yuri Hoyda/Shutterstock.

Appendix B
Python Quick Reference

This quick reference provides an overview of the primary Python constructs used in this text. It is not a detailed reference but rather a compendium of tables and syntax. These are described in more detail within the chapters. In addition, more detailed documentation can be found at https://docs.python.org/.

B.1 Python Reserved Words

The words shown in **Table B.1** are reserved by Python. You should not use any of these words as identifiers in your program. If you do, you will get a `SyntaxError`.

TABLE B.1 Python's Reserved Words

and	class	except	if	nonlocal	True
as	continue	False	import	not	try
assert	def	finally	in	or	while
async	del	for	is	pass	with
await	elif	from	lambda	raise	yield
break	else	global	None	return	

Description

B.2 Numeric Data Types

Python has three numeric data types:

- Integers
- Floating-point numbers
- Complex numbers

Table B.2 shows the most common operators for numeric data types.

TABLE B.2 Arithmetic Operators in Python

Operation	Operator	Explanation
Addition	`+`	Calculates the sum of two values.
Subtraction	`-`	Calculates the difference between two values.
Multiplication	`*`	Calculates the product of two values.
Floating-point division	`/`	Calculates the quotient of two values.
Integer division	`//`	Calculates the integer quotient.
Modulo	`%`	Finds the remainder after performing a division.
Exponentiation	`**`	Raises a number to a power. Example: `(x ** y)` is equivalent to x^y.

B.3 Built-in Functions

Python has a number of functions that can be called without an object reference. **Table B.3** shows some common built-in functions.

TABLE B.3 Built-in Functions

`chr(num)`	Returns the character represented by the Unicode value `num`.
`dir()`	Returns the list of names defined in the current local scope.
`float(numOrString)`	Converts the number or string to a floating-point number.
`help(str)`	Returns documentation on `str`.
`input(prompt)`	Outputs the prompt, then returns what the user types as a string.
`isinstance(object, class)`	Returns `True` if `object` is an instance of `class`; `False` if not.
`int(numOrString)`	Converts the number or string to an integer.
`len(sequence)`	Returns the number of elements in `sequence`.
`list(sequence)`	Creates a list from the elements in `sequence`.
`max(sequence)`	Returns the maximum item in the `sequence`.
`min(sequence)`	Returns the minimum item in the `sequence`.
`next(iterator)`	Returns the next item from the iterator.
`open(filename, mode)`	Opens the file named `'filename'`. Mode can be `'r'(read)`, `'w'(write)`, or `'a'(append)`. Returns a `file` reference.
`ord(ch)`	Returns the Unicode value of a single-character string.
`print(value1, …, separator=None, end=endstring, file=None, flush=False)`	Outputs each value separated by a space by default, ending with a line separator by default.
`round(num, nDigits)`	Returns `num` rounded to `nDigits` after the decimal point; if `nDigits` is not given, returns the nearest integer.
`str(num)`	Converts a number to a string.
`sum(sequence)`	Returns the sum of the items in a numeric sequence.
`super()`	Calls the superclass version of a method.

B.4 Sequence Operators

Python has three sequential data types: strings, lists, and tuples. **Table B.4** summarizes the operators that apply to all sequences.

TABLE B.4 Operations on Any Sequence in Python

Operation	Operator	Explanation
Indexing	`[n]`	Accesses the nth element of a sequence.
Concatenation	`+`	Combines sequences.
Repetition	`*`	Concatenates a repeated number of times.
Membership	`in`	Returns `True` if an item is in a sequence; `False` otherwise.
Membership	`not in`	Returns `True` if an item is not in a sequence; `False` otherwise.
Slicing	`[ : ]`	Extracts a part of a sequence.

B.4.1 String

Strings are immutable sequences of characters that are indexed by integers, starting at zero. To create string literals, you can enclose them in single, double, or triple quotes as follows:

```
a = "Don't say Hello."
b = 'She said, "Hello World!"'
c = """He said, "You don't say!", didn't he"""
```

Description

Table B.5 summarizes some useful string methods.

TABLE B.5 Summary of String Methods

Method	Use	Explanation
center	`aString.center(w)`	Returns the string `aString` surrounded by spaces to make `aString` `w` characters long.
count	`aString.count(item)`	Returns the number of occurrences of `item` in `aString`.
ljust	`aString.ljust(w)`	Returns `aString` left-justified in a field of width `w`.
rjust	`aString.rjust(w)`	Returns `aString` right-justified in a field of width `w`.
upper	`aString.upper()`	Returns `aString` in all uppercase.
lower	`aString.lower()`	Returns `aString` in all lowercase.
index	`aString.index(item)`	Returns the index of the first occurrence of `item` in `aString`, or an error if not found.
rindex	`aString.rindex(item)`	Returns the index of the last occurrence of `item` in `aString`, or an error if not found.
find	`aString.find(item)`	Returns the index of the first occurrence of `item` in `aString`, or –1 if not found.
rfind	`aString.rfind(item)`	Returns the index of the last occurrence of `item` in `aString`, or –1 if not found.
replace	`aString.replace(old, new)`	Replaces all occurrences of `old` substring with `new` substring in `aString`.
split	`aString.split(sChar)`	Returns a list of substrings, split at `sChar` or at whitespace if `sChar` is omitted.
contains	`aString.contains(item)`	Returns `True` if `item` is found in `aString`; `False` otherwise.
startswith	`aString.startswith(item)`	Returns `True` if `aString` starts with `item`; `False` otherwise.
join	`separator.join(list)`	Returns a string composed of the elements in the list, separated by the string used to call the method.

B.4.2 Lists

Lists are mutable sequences of references to any object. They are indexed by integers, starting with zero. You can create a list as follows:

```
In [1]:  1 emptyList = []
         2 myList = [1, 2, 3.0, 'hello', 5, 6]
```

```
In [2]:  1 newList = list("hello")
         2 newList

Out[2]: ['h', 'e', 'l', 'l', 'o']
```

Description

Table B.6 summarizes some useful list methods.

TABLE B.6 Methods Provided by Lists in Python

Method	Use	Explanation
append	`aList.append(item)`	Adds a new item to the end of a list.
insert	`aList.insert(i, item)`	Inserts an item at the `i`th position in a list.
pop	`aList.pop()`	Removes and returns the last item in a list.
pop	`aList.pop(i)`	Removes and returns the `i`th item in a list.
sort	`aList.sort(key=None, reverse=False)`	Modifies a list to be sorted from low to high. Optional `key` specifies a function to extract a field as the sort value. If `reverse` is `True`, the list is sorted from high to low.
reverse	`aList.reverse()`	Modifies a list to be in reverse order.
index	`aList.index(item)`	Returns the index of the first occurrence of `item`, or an error if not found.
count	`aList.count(item)`	Returns the number of occurrences of `item`.
remove	`aList.remove(item)`	Removes the first occurrence of `item`.
copy	`copyList = aList.copy()`	Puts a copy of the items in `aList` into the list `copyList`.
clear	`aList.clear()`	Removes all items from `aList`.

You can convert sequences to lists using the `list` function, described in Table B.3.

You can also create sequences using the following `range` sequences.

- `range(stop)` Creates a sequence of numbers starting at 0 and going up to `stop-1`.
- `range(start, stop)` Creates a sequence of numbers starting at `start` and going up to `stop-1`.
- `range(start, stop, step)` Creates a sequence of numbers starting at `start` and going up to `stop-1` counting by `step`.

```
In [2]:  1 list(range(10))
```

```
Out[2]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [3]:  1 list(range(1, 11))
```

```
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [4]:  1 list(range(10, 0, -1))
```

```
Out[4]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Description

B.4.3 Tuples

Tuples are immutable sequences of references to any object. You can create a tuple by enclosing the sequence in parentheses as follows:

In [1]:
```
1 t = (1, 2, 3)
2 t
```

Out[1]: (1, 2, 3)

In [2]:
```
1 u = ('a', 'b')
2 u
```

Out[2]: ('a', 'b')

In [3]:
```
1 y = tuple([1, 2, 3])
2 y
```

Out[3]: (1, 2, 3)

In [4]:
```
1 z = tuple('abc')
2 z
```

Out[4]: ('a', 'b', 'c')

Description

Tuples support the same operators as lists, such as indexing and slicing. The main difference is that because tuples are immutable, you cannot use an operator in a way that would modify the tuple. You may not use any of the list methods with a tuple.

The `zip` built-in function can be used to create a list of tuples from two or more sequences. The tuples consist of the `i`th element from each sequence. For example:

```
In [1]: 1 list(zip('hello', range(len('hello'))))

Out[1]: [('h', 0), ('e', 1), ('l', 2), ('l', 3), ('o', 4)]
```

```
In [2]: 1 list(zip([1, 2, 3], [4, 5, 6]))

Out[2]: [(1, 4), (2, 5), (3, 6)]
```

```
In [3]: 1 list(zip('abc', 'def', 'ghi'))

Out[3]: [('a', 'd', 'g'), ('b', 'e', 'h'), ('c', 'f', 'i')]
```

Description

B.5 Dictionaries

A dictionary is a collection of pairs of `key:objects`. The objects, or values, are referenced by their keys. Python maintains the elements in the order in which they were added to the dictionary. You can create a Python dictionary as follows:

```
In [3]:  1 emptyDict = {}
         2 emptyDict
```

```
Out[3]: {}
```

```
In [4]:  1 myD = {'a':123, 'b':789, 'c':246}
         2 myD
```

```
Out[4]: {'a': 123, 'b': 789, 'c': 246}
```

```
In [5]:  1 myD['d'] = 359
         2 myD
```

```
Out[5]: {'a': 123, 'b': 789, 'c': 246, 'd': 359}
```

Description

Table B.7 summarizes the Python dictionary methods.

TABLE B.7 Methods, Functions, and Operators Provided by Dictionaries in Python

Name	Use	Explanation
`keys`	`aDict.keys()`	Returns a `dict_keys` object of keys in the dictionary.
`values`	`aDict.values()`	Returns a `dict_values` object of values in the dictionary.
`items`	`aDict.items()`	Returns a `dict_items` object of key–value tuples.
`get`	`aDict.get(k)`	Returns the value associated with key `k`; `None` otherwise.
`get`	`aDict.get(k, alt)`	Returns the value associated with key `k`; `alt` otherwise.
`setdefault(key, default)`	`value = nDict.setdefault(key,defaultValue)`	If the key is not in the dictionary, insert the key with a value of `defaultValue` and return `defaultValue`. If the key is in the dictionary, return the value for the key.
`in`	`key in aDict`	Returns `True` if `key` is in the dictionary; `False` otherwise.
`not in`	`key not in aDict`	Returns `True` if `key` is not in the dictionary; `False` otherwise.
`index`	`aDict[key]`	Returns the value associated with `key`.
`clear()`	`aDict.clear()`	Deletes all elements in the dictionary.
`del`	`del aDict[key]`	Removes the `key` entry from the dictionary.
| (merge)	`aDict | bDict`	Merges the two dictionaries. Unique entries in `bDict` are appended to `aDict`. Entries with the same key in both dictionaries are assigned the value in `bDict`.

B.6 Files

Files are sequences of characters that are stored on a disk. It is also useful to think of files as sequences of lines. **Table B.8** summarizes `File` methods.

TABLE B.8 Methods Provided by a `File` Object

Name	Use	Explanation
`open (filename, mode)`	`with open(filename,'r') as fileVar`	Opens a file called `filename` and uses it for reading. This will return a reference `(fileVar)` to a `file` object. Other modes are `'w'` for writing, and `'a'` for appending to a file. Read `('r')` is the default. With this use of `open`, the file is closed automatically when processing finishes.
`close`	`fileVar.close()`	File use is complete.
`write`	`nChars = fileVar.write(aString)`	Adds `aString` to the end of the file. `filevar` must refer to a file that has been opened for writing. Returns the number of characters written.
`read(n)`	`aString = fileVar.read(n)`	Reads and returns a string of `n` characters, or the entire file as a single string if `n` is not provided.
`readline(n)`	`aString = fileVar.readline(n)`	Returns the next line of the file with all text up to and including the newline character. If `n` is provided as a parameter, then only `n` characters will be returned if the line is longer than `n`. Returns an empty string when the end of the file is reached.
`readlines(n)`	`stringList = fileVar.readlines(n)`	Returns a list of `n` strings, each representing a single line of the file, including the terminating newline character. If `n` is not provided, then all lines of the file are returned.

The following program opens a file and reads it line by line, printing each line of the file.

```
with open('myFile.dat', 'r') as f:
    line = f.readline()
    while line:
        print(line)
        line = f.readline()
#f is closed automatically
```

Description

It is also possible to read and print a file using a simple `for` loop:

```
with open('myFile.dat') as f:
    for line in f:
        print(line)
```

Description

B.7 Formatting Output

Python allows you to format strings using the string `format` method. For example:

```
In [1]:  1 print("{0} + {1} = {2}".format(5, 6, 11))
```

```
5 + 6 = 11
```

```
In [2]:  1 city = "Orlando"
         2 rainfall = 9.5
         3 print("{0:>12s} {1:5.1f} inches\n".format(city, rainfall))
```

```
     Orlando   9.5 inches
```

Description

The `format` method substitutes replacement fields for placeholders, indicated by curly braces `{}` in the string, using the formats specified by the placeholder. By default, the replacement fields are substituted in the same order as the placeholders. It is recommended, however, that each placeholder be identified by using its sequence number in curly braces `{}`, as shown by `{0}`, `{1}`, and `{2}` in the preceding code.

The format for a placeholder is

```
{Position:AlignmentWidthPrecisionType}
```

Table B.9 shows the various format characters that can be used to specify the type of the value.

TABLE B.9 Common Type Specifications

Type of Replacement Field	Type Conversion Character	Output Format
String	`s`	String. This is the default format.
Integer	`c`	Character. Converts an integer to its Unicode equivalent.
	`d`	Decimal integer. This is the default.
	`n`	Number. Same as decimal integer.
Floating-point	`e` or `E`	Scientific notation with a default precision of 6, as `m.dddddde+/-xx` or `m.ddddddE+/-xx`.
	`f` or `F`	Fixed-point with default precision of 6, as `m.dddddd`.
	`g`	General format. Uses scientific or fixed-point depending on the magnitude of the number.
	`n`	Same as `g`.
	`%`	Percentage. Multiplies the number by 100, then displays as `f` format and appends a percent sign.

Table B.10 shows the various format modifiers that can be used to specify alignment, width, and precision for the value.

TABLE B.10 Common Format Modifiers

Type	Character	Output Format
Alignment	`<`	Left aligns the value in its space. This is the default for strings.
	`>`	Right aligns the value in its space. This is the default for numbers.
	`^`	Centers the value within its space.
Width	`w`	The value will use the space of `w` characters. The default is to use the minimum space required to display the value.
Precision	`.n`	Displays `n` numbers after the decimal point.

B.8 Iteration

B.8.1 Simple Iteration over Collections of Data

The `for` statement allows you to easily iterate over any collection of data using the following form:

```
for item in sequence:
    statement1
    statement2
    ...
```

Description

The next three examples show how to iterate over strings, lists, and tuples.

In this example, the variable `i` will be bound to a new integer created by the `range` function each time through the loop:

```
for i in range(n):
    print(i)
```

Description

In this example, the loop variable `i` will be bound to the next element in the list each time through the loop:

```
for i in ['a', 1, 'b', 2, 'c', 3]:
    print(i)
```

Description

In this example, the loop variable `ch` is bound to the next character of the string each time through the loop:

```
for ch in "hello world":
    print(ch)
```

Description

It is also possible to iterate over a dictionary. The following examples show you two common patterns. In this case, the loop variable `i` is bound to each key contained in the dictionary `myDictionary` and prints the value associated with each key.

```
for i in myDictionary:
    print(myDictionary[i])
```

Description

It is also possible to iterate over keys and values simultaneously using the following code:

```
for key, value in myDictionary.items():
    print(key, value)
```

Description

In this example, the key–value pairs are pulled out of the dictionary ahead of time using the `items` method.

B.8.2 Iteration with `while`

The `while` loop is an indefinite loop. It will continue to perform the statements in the body of the loop until the loop condition becomes `False`. Typically, the last statement in the loop performs an action that updates the loop condition. If the condition is never `False`, then the loop is an infinite loop and will continue forever.

```
while <condition>:
    statement1
    statement2
    ...
    loopUpdate statement
```

Description

B.8.3 List Comprehensions

List comprehensions allow you to create lists by embedding a `for` loop in a list creation expression. It is more convenient than writing a `for` loop and using the `append` method. Here is the general format for a list comprehension:

```
[<expression> for <item1> in <sequence1>
              for <item2> in <sequence2>
              ...
              if <condition>]
```

Description

The `if` statement is optional and only one `for` loop is required. Here are some examples of simple list comprehensions:

In [1]:
```
x = [1, 2, 3]
y = ['a', 'b', 'c']
```

In [2]:
```
[a * a for a in x]    #square each element in x
```

Out[2]: [1, 4, 9]

In [3]:
```
[a for a in x if a % 2 != 0]  #odd elements in x
```

Out[3]: [1, 3]

In [4]:
```
[(a, b) for a in x for b in y]  #tuples with corresponding
                                #elements from x and y
```

Out[4]:
```
[(1, 'a'),
 (1, 'b'),
 (1, 'c'),
 (2, 'a'),
 (2, 'b'),
 (2, 'c'),
```

```
(3, 'a'),
(3, 'b'),
(3, 'c')]
```

Description

B.9 Boolean Expressions

B.9.1 Relational Operators

A simple Boolean expression can be constructed using any of the relational and equality operators shown in **Table B.11**.

TABLE B.11 Relational and Equality Operators and Their Meaning

Operator	Meaning
`<`	Less than
`<=`	Less than or equal to
`>`	Greater than
`>=`	Greater than or equal to
`==`	Equal
`!=`	Not equal

B.9.2 Compound Boolean Expressions

Multiple Boolean expressions can be joined using the logical operators `and`, `or`, and `not` as shown in **Table B.12**. In this table, *x* and *y* are `Boolean expressions`.

TABLE B.12 Logical Operator Behavior

`x and y`	`True` if both `x` and `y` are `True`; otherwise `False`.
`x or y`	`True` if either `x` or `y` is `True`; otherwise `False`.
`not x`	`True` if `x` is `False`, otherwise `False`.

B.10 Selection

B.10.1 Binary Selection

```
if ‹condition›:
    ‹statements›  #execute if condition is True
else:
    ‹statements›  #execute if condition is False
```

Description

B.10.2 Unary Selection

```
if ‹condition›:
    ‹statements›  #execute if condition is True
```

Description

B.10.3 Nested Selection with `elif`

```
if ‹condition1›:
    ‹statements›      #condition1 is True
elif ‹condition2›:
    ‹statements›      #condition1 is False; condition2 is True
...
elif ‹conditionN›:
    ‹statements›      #conditions 1 - N-1 are False; conditionN is True
else:
    ‹statements›      #conditions 1 - N are False
```

Description

B.10.4 Selection with `match`

```
match ‹subject›:
    case value1:
        #statements to execute if subject == value1
    case value2:
        #statements to execute if subject == value2
    ...
    case _: #optional
        #statements to execute if subject does not match any case values
```

Description

B.11 Python Modules

B.11.1 Math

Table B.13 summarizes the main functions from the `math` module used in this text. To use these functions, first `import math`.

TABLE B.13 Some Math Functions and Constants

Name	Use	Explanation
`dist`	`math.dist(p1, p2)`	Returns the Euclidean distance between two points, `p1` and `p2`, specified as a tuple.
`sqrt`	`math.sqrt(x)`	Returns the square root of `x`.
`log10`	`math.log10(x)`	Returns the base 10 logarithm of `x`.
`log2`	`math.log2(x)`	Returns the base 2 logarithm of `x`.
`pi`	`math.pi`	Evaluates to 3.141592653589793.
`e`	`math.e`	Evaluates to 2.718281828459045.

B.11.2 Random Numbers

Table B.14 summarizes the main functions from the `random` module used in this text. To use these functions, first `import random`.

TABLE B.14 Random Number Generation

Method Name	Use	Explanation
`random`	`random.random()`	Returns a random number between 0.0 and 1.0.
`randint`	`random.randint(start,end)`	Returns a random number in the range start–end, including the endpoints.
`randrange`	`random.randrange(n)`	Returns a random number in the range 0 to n − 1. Uses the same syntax as the `range` function for specifying ranges of numbers.

B.11.3 Statistics

Table B.15 summarizes some useful functions of the `statistics` module. To use these functions, first `import statistics`.

TABLE B.15 Some Useful Functions of the `statistics` Module

Method	Explanation
`mean(data)`	Returns the average of the data values.
`median(data)`	Returns the middle value of ordered numeric data. If the number of data items is even, returns the average of the two middle values.
`multimode(data)`	Returns a list containing the values that appear most often.
`mode(data)`	Returns the value that appears most often. If more than one value appears with the same highest frequency, returns the first mode in the list.
`stdev(data)`	Returns the standard deviation.
`stdev(data, mean)`	Returns the standard deviation, given the mean.

B.11.4 Reading CSV Files

Table B.16 summarizes the `reader` function of the `csv` module for reading `CSV` files and the `next` function, which can be used with any iterator. To use these functions, first `import csv`.

TABLE B.16 Useful Functions for Reading a `csv` File

Method	Explanation
`reader(file)`	Returns an iterator object for moving through the rows of a `csv` file. Each row is returned as a list of its fields.
`next(iterator)`	Returns the next row from the iterator.

B.11.5 Opening a URL

Table B.17 summarizes the `urlopen` function of the `urllib.request` module for opening a website address. To use this function, first `import urllib.request`.

TABLE B.17 The `urlopen` Function in the `urllib.request` Module

Method	Explanation
`urlopen(url)`	Opens a web address and returns a file handle. Raises a `URLError` if unsuccessful.

B.11.6 Reading JSON Data

Table B.18 summarizes the `loads` function of the `json` module for converting JSON data to Python data formats. To use this function, first `import json`.

TABLE B.18 The `loads` Function in the `json` Module

Method	Explanation
`loads(jsonData)`	Converts JSON objects to Python objects. JSON objects are translated to Python dictionaries; JSON arrays are translated to Python lists; and JSON data types are translated to Python data types.

B.11.7 Abstract Classes and Methods

Table B.19 shows the abstract base class, `ABC`, and **Table B.20** shows the decorator used to specify that a method in an abstract class is itself abstract. To use this base class, first `import abc`.

TABLE B.19 The `ABC` Class in the `abc` Module

Class	Explanation
`ABC`	Abstract base class for convenience in defining an abstract class. The abstract class specifies `ABC` as its superclass.

TABLE B.20 The `@abstractmethod` Decorator in the `abc` Module

Decorator	Explanation
`@abstractmethod`	Decorator that indicates a method is an abstract method.

B.11.8 Multithreading

The `threading` module provides support for defining a multithreaded application. To use multithreading, first `import Thread`. A multithreaded application needs to perform these three actions:

1. The class should inherit from the `Thread` class. Ex: `class MyApp(Thread)`
2. The class should define a `run` method that contains the code for the thread to execute.
3. The class should call the `start` method. This will create a `Thread` and call the `run` method.

Table B.21 shows the `run` and `start` methods.

TABLE B.21 The `run` and `start` Methods in the `Thread` Class

Method	Description
`run()`	Contains the code to do the work of the thread. The subclass provides its own version of this method.
`start()`	Creates the thread and calls the thread's `run` method. This method can be called only once per thread.

B.12 Regular Expression Patterns

The following is a list of some simple regular expression patterns:

Expression	Meaning
`.`	Matches any single character.
`[abc]`	Matches `a`, `b`, or `c`.
`[^abc]`	Matches any characters other than `a`, `b`, or `c`.
`[abc]+`	Matches one or more occurrences of the characters `abc`—for example, match `b` or `aba` or `ccba`.
`[abc]+[de]*`	Matches zero or more occurrences of the characters `abcde`—for example, match `b` or `aba` or `bd` or `abae`.
(regex)	Creates a capture group.
`$`	Ends with.

Table B.22 summarizes the main regular expression functions in the `re` module. To use these functions, first `import re`.

TABLE B.22 Regular Expression Module Functions and Methods

Method Name	Use	Explanation
`match(pattern, string)`	`re.match('[abc]XY.', 'aXYv')`	Matches any string that starts with `a`, `b`, or `c`, followed by `XY`, followed by any single character. Returns a `match` object on success or `None`.
`sub(pattern, replacement, string)`	`re.sub('[tv]', 'X','vxyzbgtt')`	Returns `'XxyzbgXX'`. Similar to `replace`, except that regular expression matching is used.
`findall(pattern, string)`	`{re.findall('[bc]+', 'abcdefedcba')`	Returns `['bc','cb']`. Returns a list of all substrings matching the regular expression.
`groups()`	`matchObj.groups()`	Returns a list of all capture groups matched. `matchObj` is created by a call to `match`.
`group(i)`	`matchObj.group(2)`	Returns a single capture group at index 2. `matchObj` is created by a call to `match`.

B.13 Syntax for Defining a Function

```
def functionName(param1, param2, ...):
    statement1
    statement2
    ...
```

Description

B.14 Syntax for Defining a Class

```
def ClassName:
    def __init__(self, ...): #constructor
        ...
    def method1(self, ...):
        ...
    def method2(self, ...):
        ...
    @staticmethod              #decorator
    def staticmethod(...):     #defines a static
        ...                    #method
```

Description

B.14.1 Syntax for Defining Subclasses

```
class ClassName(SuperClass):
    def __init__(self, ...):   #constructor
        super().__init__()     #call superclass constructor
        ...
    def method1(self, ...):
        ...
    def method2(self, ...):
        ...
```

Description

B.15 Deleting Objects

The `del` statement is used to delete objects, such as variables, functions, and items in mutable sequences.

Deleting a function:

In [1]:
```
def bar():
    return 2 * 5

```

In [2]:
```
bar()
```

Out[2]: 10

In [3]:
```
del bar   #delete the bar function
```

In [4]:
```
bar()    #attempt to call deleted function
```

NameError Traceback (most recent call last)

Input In [4], in <cell line: 1>()

----> 1 bar()

NameError: name 'bar' is not defined

Description

Deleting an object:

```
In [1]:  1 import turtle
```

```
In [2]:  1 t = turtle.Turtle()
         2 t.forward(50)
```

```
In [3]:  1 del t                    #delete the turtle
```

```
In [4]:  1 t.forward(75)            #attempt to use deleted turtle
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [4], in <cell line: 1>()
----> 1 t.forward(75)

NameError: name 't' is not defined
```

Description

Deleting a list item:

```
In [1]:  1 myList = ['a', 'b', 'c', 'd']
         2 myList

Out[1]: ['a', 'b', 'c', 'd']

In [2]:  1 del myList[2]     #delete the item at index 2

In [3]:  1 myList

Out[3]: ['a', 'b', 'd']
```

Description

Deleting a dictionary item:

In [1]:
```
1 myDict = {'a':123, 'b':456, 'c':789}
2 myDict
```

Out[1]:
```
{'a': 123, 'b': 456, 'c': 789}
```

In [2]:
```
1 del myDict['b']      #delete the item with key b
```

In [3]:
```
1 myDict
```

Out[3]:
```
{'a': 123, 'c': 789}
```

Description

B.16 `matplotlib`

Table B.23 shows some common functions used for visualization. To use these functions, first `import matplotlib`. The parameters listed for each function are only a subset of the possible parameters that can be set to customize a visualization. See matplotlib.org for information on other options.

TABLE B.23 `matplotlib` Functions

Name	Use	Explanation
`bar`	`bar(x = xValues, height = hValues, color = colorNames)`	Creates a bar chart given *x* values and their corresponding heights. The `color` of the bars can be set using a string representation of that color (e.g., `"red"`, `"green"`, `"blue"`). If not set, the default color is blue.
`scatter`	`scatter(x = xValues, y = yValues, color = colorNames)`	Creates a scatterplot of points for each (`x`, `y`) coordinate. The `color` of the points can be set using a string representation of that color, as described above.
`axvline`	`axvline(x, color = colorName, ls = lineStyle)`	Plots a vertical line at the provided *x* value. The `color` of the line can be set using a string representation of that color, as described above. The line style (`ls`) can be set using a string representation of the style (e.g., `"solid"`, `"dashed"`). If the line style is not set, a solid line will be generated.
`axhline`	`axhline(y, color = colorName, ls = lineStyle)`	Plots a horizontal line at the provided *y* value. The `color` of the line and the line style (`ls`) can be set using string representations, as described above.
`xlabel`	`xlabel(str)`	Sets the title of the *x*-axis to the provided string (`str`).
`ylabel`	`ylabel(str)`	Sets the title of the *y*-axis to the provided string (`str`).
`show`	`show()`	Displays the visualization.

B.17 pandas

Table B.24 shows a few pandas library functions for reading files.

TABLE B.24 pandas Functions for Reading Files

Function Name	Explanation
`read_csv(filename, delimiter)`	Reads and parses the CSV file name, using the delimiter specified (defaults to comma-separated if nothing is specified). Returns a DataFrame object.
`read_excel(filename, sheetName)`	Reads a Microsoft® Excel file and parses the worksheet specified by sheetName. Returns a DataFrame object.

Table B.25 shows a few useful pandas properties and methods for DataFrames.

TABLE B.25 Some pandas DataFrame Properties and Methods

Name	Explanation
`columns`	Property that returns a list of column names for this DataFrame.
`index`	Property that returns a list of row labels.
`iloc[]`	Property that allows for selection by integer position.
`shape()`	Method that returns the number of rows and columns in the DataFrame as a tuple.
`head(n)`	Method that gets the first `n` rows of the DataFrame.
`tail(n)`	Method that gets the last `n` rows of the DataFrame.

Table B.26 shows a few useful pandas series methods.

TABLE B.26 Useful pandas Series Methods

Method	Explanation
`sum()`	Returns the sum of all numeric data in the series.
`mean()`	Returns the mean of all numeric data in the series.
`mode()`	Returns the mode of all numeric data in the series.
`median()`	Returns the median of all numeric data in the series.
`std()`	Returns the standard deviation of all numeric data in the series.

B.18 Common Error Messages

Learning how to read and understand error messages can save a lot of time and trouble once you know what to look for.

One of the most common errors is the syntax error, which indicates that you have written a line of code that does not follow the rules of Python's grammar. Here is an example of a syntax error that is due to a missing colon (`:`) at the end of the function definition line `def foo(x, y)`. It is also very common to get a syntax error like this when you are using an `if` statement.

```
In [1]:  1 def foo(x, y)
         2
```

```
  Input In [1]
    def foo(x, y)
                  ^
SyntaxError: expected ':'
```

Description

Another common error is to use a variable that has not yet been assigned a value. The following example shows an attempt to use the variable `spam` when `spam` has no value. Note that this error message includes the word `Traceback`, which can help you pinpoint the line and the function where an error occurs. The error message says that a `NameError` occurred. This message tells you that `'spam'` is undefined. Because the only way to define a variable in Python is to assign a value to it, this message should help you

realize that you need to assign a value to `spam` before using it in an expression.

```
In [1]: 1 4 + spam * 3
```

```
NameError                                 Traceback (most recent call last)
Input In [1], in <cell line: 1>()
----> 1 4 + spam * 3

NameError: name 'spam' is not defined
```

Description

Let's look at another example of a traceback using a simple Python program (**Listing B.1**). Notice that `spam` is not defined in the function `foo`.

```
1  def foo():
2      a = 1 + 2
3      b = a + spam
4      return b
5
6  def main():
7      print('hello')
8      x = foo()
9      print(x)
10
11 main()
```

Listing B.1 A simple Python traceback

Description

When we run this program, we get the following output:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <cell line: 11>()
      8     x = foo()
      9     print(x)
---> 11 main()

Input In [1], in main()
      6 def main():
      7     print('hello')
----> 8     x = foo()
      9     print(x)

Input In [1], in foo()
      1 def foo():
      2     a = 1 + 2
----> 3     b = a + spam
      4     return b

NameError: name 'spam' is not defined
```

Description

The first line of the traceback points to line 11 of the listing, where `main()` is called. The next line of the file points to line 8, where `foo()` is called. The last line points to line 3 in the function `foo`, where we find the expression `b = a + spam`. Once again Python tells us that `spam` is not defined, and it gives us exactly the line that contains the error.

When working with a mixture of strings and numeric data types, you might try to "add" two incompatible things together. The next example shows an expression that tries to add a string and an integer. Python does not know whether it should convert `'2'` to an integer and add two integers, or convert `2` to a string and concatenate two strings. Thus, it will give you this error message:

```
In [1]: 1 '2' + 2
```

```
TypeError                                 Traceback (most recent call last)
Input In [1], in <cell line: 1>()
----> 1 '2' + 2

TypeError: can only concatenate str (not "int") to str
```

Description

Here is another situation where Python cannot figure out what to add:

```
In [1]:  1 def bar():
         2     return 2 * 5
```

```
In [2]:  1 a = 3 + bar
```

```
TypeError                                 Traceback (most recent call last)
Input In [2], in <cell line: 1>()
----> 1 a = 3 + bar

TypeError: unsupported operand type(s) for +: 'int' and 'function'
```

Description

In this case, the statement `a = 3 + bar` is missing the parentheses after the function named `bar`. Even though `bar` does not require any parameters, Python does not know that it should call the function without the "call operators" `()`.

Here is an example of using the call operators on an object that is not a function:

```
In [1]:  1 b = 1
         2 c = 2 + b()
```

```
TypeError                                 Traceback (most recent call last)
Input In [1], in <cell line: 2>()
      1 b = 1
----> 2 c = 2 + b()

TypeError: 'int' object is not callable
```

Description

The next error is fairly easy to recognize, but it is a common error:

```
In [1]:  1 10 * (1/0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [1], in <cell line: 1>()
----> 1 10 * (1/0)

ZeroDivisionError: division by zero
```

Description

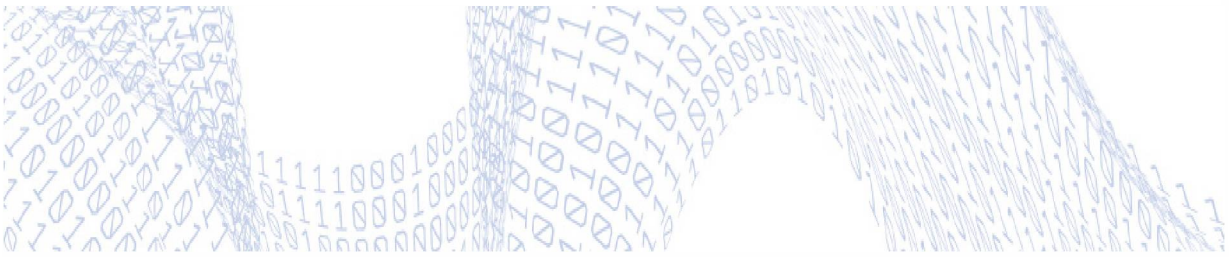
Finally, here is an example of the error message you get when you attempt to iterate over something that is not a sequence. In this case, `d` is an integer rather than a string, list, tuple, dictionary, or file.

In [1]:
```
1 d = 1
2 for i in d:
3     print(i)
```

```
TypeError                                 Traceback (most recent call last)
Input In [1], in <cell line: 2>()
      1 d = 1
----> 2 for i in d:
      3     print(i)

TypeError: 'int' object is not iterable
```

Description



© Yuri Hoyda/Shutterstock

Appendix C
`turtle` Reference

The definitive guide to the `turtle` module can be found at

https://docs.python.org/3/library/turtle.xhtml

In this reference, we include `turtle` methods we used in this book.

C.1 Basic Move and Draw

`forward(distance)` Move forward along the current heading by `distance` units.

`backward(distance)` Move backward along the current heading by `distance` units.

`right(angle)` Turn the turtle right by `angle` degrees.

`left(angle)` Turn the turtle left by `angle` degrees.

`goto(pos, y=None)` Move the turtle to an absolute position. If the tail is down, then a line will be drawn. The turtle's orientation does not change. The position may be specified by a pair of numbers, or a tuple representing the `x`, `y` coordinates.

`setx(x)` Move the turtle to absolute (`x`, `y`) position where `x` is specified as a parameter and `y` does not change.

`sety(y)` Move the turtle to absolute (`x`, `y`) position where `y` is specified as a parameter and `x` does not change.

`setheading(direction)` Set the turtle facing the given direction. Some common directions in degrees include 0 = right, 90 = up, 180 = left, 270 = down.

`home()` Move the turtle to its home position (0, 0), and set the heading to its original orientation.

`circle(radius, angles=None, sides=None)` Draw a circle or part of a circle given by `angles`, and change the turtle's orientation by `angles`. The center is to the turtle's left (considering its heading) by `radius` units. The `sides` parameter specifies the number of sides for the polygon used to draw the circle. If omitted, `sides` is automatically determined.

`dot(size=None, *color)` Make a dot at the current position of `size`; `color`, if given, is a color string or a color tuple.

```
dot(30)
dot(40, 'blue')
```

`undo()` Undo the turtle's last action.

`write(text, move=False, align='left', font=('Arial', 8, 'normal'))` Write some text to the screen at the turtle's current position. If `move` is `False`, the turtle's pen will not move. `align` can be `'left'`, `'right'`, or `'center'`. The `font` specification allows you to choose a font and size for the message.

C.2 Turtle State

`position()` Return the turtle's current location as an (`x`, `y`) tuple.

`towards(pos, y=None)` Return the angle between the turtle's current heading and the location passed as a parameter.

```
towards(x, y)                #location as x and y
towards((x, y))              #location as a tuple
towards(otherTurtle)         #another turtle
```

Description

`heading()` Return the current heading of the turtle.

`distance(pos, y=None)` Return the distance from the turtle's current location and the location passed as a parameter.

```
distance(x, y)               #to a location as x and y
distance((x, y))             #to a location as a tuple
distance(otherTurtle)        #to another turtle
```

Description

C.3 Drawing State

`down()` Put down the turtle's tail.

`up()` Raise up the turtle's tail.

`width(width=None)` Change the width of the line to draw. If no parameter is passed, the method will return the width of the line.

C.4 Filling

`begin_fill()` Call `begin_fill` before drawing the shape you want filled in.

`end_fill()` Call `end_fill` after you draw the shape you want filled in. For example, you can use the following code to create a filled square:

```
turtle.begin_fill()
for i in range(4):
    turtle.forward(100)
    turtle.right(90)
turtle.end_fill
```

Description

C.5 More Drawing Control

`reset()` Delete all the turtle's drawing from the window. Re-center the turtle, and reset its variables to their default values.

`clear()` Delete all the turtle's drawing, but leave the turtle at the current position.

C.6 Controlling the Shape and Appearance

The turtle is not limited to the triangular shape you see by default, but instead can appear as a turtle or any other shape that you can draw with the turtle. In addition, you can use a GIF file to provide the turtle with a shape. Once you have added a shape to your program, any turtle can use that shape by name.

`addshape(name, shape=None)` Add a new shape to the `TurtleScreen`'s shape list. A shape must be registered before being used. This method is called using a screen object or the `turtle` module name. The arguments are interpreted as follows:

- If `name` is the name of a GIF file and `shape` is `None`, the image is registered.
- If `name` is a string and `shape` is a tuple of pairs of coordinates, then the tuple of pairs is used to create a polygon to use as the turtle shape.
- If `name` is a string and `shape` is a shape object, then the shape object is registered.

`resizemode(rmode=None)` Set resize mode to one of the following values: `auto`, `user`, or `noresize`. `auto` adapts the appearance of the turtle according to the value of `pensize`. `user` adapts the appearance of the turtle according to the values of `stretchfactor` and outline width. Both of these can be set by `turtlesize`. The `noresize` mode forces the turtle to stay the same size. If no parameter is passed, then the method returns the current mode.

`turtlesize(stretch_wid=None, stretch_len=None, outline=None)` If the `resizemode` is `user`, then the turtle will be displayed with its width and length stretched according to the parameters. The `outline` parameter determines the width of the shape's outline.

`getshapes()` Return a list of all shapes in the `TurtleScreen`'s shape list. The default list of shapes is `['arrow', 'blank', 'circle', 'classic', 'square', 'triangle', 'turtle']`.

`begin_poly()` Start recording the vertices of a polygon.

`end_poly()` Stop recording the vertices of a polygon.

`get_poly()` Return the last recorded polygon. This polygon is an object that can be passed to `addshape`.

`shape(name=None)` Set the turtle shape or return the turtle shape if `name` is `None`. You can make your own shapes as shown in the following example:

```
turtle.begin_poly()
for i in range(4):
    turtle.forward(10)
    turtle.right(90)
turtle.end_poly()
squareShape = turtle.get_poly()
turtle.addshape('squarepants', squareShape)
turtle.shape('squarepants')
```

Description

`hideturtle()` Make the turtle invisible.

`showturtle()` Make the turtle visible.

C.7 Measurement Settings

`degrees(fullcircle=360.0)` Change the angle measurement units to degrees. For some applications, it may be convenient for a full circle to have a different number of degrees. For example, you might want to make a circle have 100 degrees to make it easy to map percentages to portions of a circle.

`radians()` Change all angle measurement units to radians.

C.8 Drawing Speed

`speed(speed=None)` If no parameter is given, return the current speed of the turtle. Otherwise, set the speed to an integer value between 0 and 10, where 0 means no animation, 1 is the slowest, and speeds increase as the number increases up to 10.

`tracer(n=None, delay=None)` Only every `n`th update is performed when animating the turtle. The second parameter specifies a delay value in milliseconds.

`delay(delay=None)` Set or return the drawing delay in milliseconds.

`update()` Update the screen with all drawing. This is useful when `tracer` is set to 0.

C.9 Color

`bgcolor(*args)` Set or return the background color of the `TurtleScreen`.

The color can be set several ways:

```
bgcolor('colorname')          #most reasonable color names are recognized
bgcolor(red, blue, green)     #red, blue, and green values between 0.0 and 1.0
bgcolor((red, green, blue))   #tuple with values between 0.0 and 1.0
```

Description

`bgpic(picname=None)` Set the background of the window to the image in the filename. `picname` can be a string, a filename of a GIF, `nopic`, or `None`. If `nopic`, the current background image is deleted. If `None`, return the current background image.

`pencolor(*args)` Set the color of the pen, using the same options as `bgcolor`.

```
pencolor('colorname')          #most reasonable color names are recognized
pencolor(red, blue, green)     #red, blue, and green values between 0.0 and 1.0
pencolor((red, green, blue))   #tuple with values between 0.0 and 1.0
```

Description

See `color` for more explanation.

`fillcolor(*args)` Set the current `fillcolor` using the same options as `bgcolor`.

`color(*args)` When called without parameters, return the current pen and fill colors. Otherwise, you can use this method to set the `pencolor` and `fillcolor` in one call. Legal ways to call this method include

```
color(penRed, penGreen, penBlue, fillRed, fillGreen, fillBlue)
color((penRed, penGreen, penBlue), (fillRed, fillGreen, fillBlue))
color('pencolorname', 'fillcolorname')
```

Description

The first group of numbers sets the pen color, and the second group of numbers sets the fill color.

`colormode(cmode=None)` Set the specification of red-green-blue colors to floating-point values between 0 and 1 or to integer values between 0 and 255. Calling `colormode(255)` changes to integer mode. The default is 1.0.

```
colormode(255)                  #range is int 0 - 255
color(200, 149, 58)
colormode(1.0)                  #range is float 0.0 to 1.0
color(.75, .69, .02)
```

Description

C.10 Events

`onclick(function, button=1, add=None)` Register a function to a mouse click so that `function` is called whenever the user presses the button. Note that `function` must accept two parameters that will give the position of the mouse when the button is clicked. `button` identifies the mouse button, with 1 being the left button. If `add` is `True`, a new callback will be added; otherwise, the function will replace the previous callback. If `None` is passed as the function parameter, the callback is canceled.

`onkey(function, key)` Register a function to a key press. `function` must be defined with no parameters. `key` is a string representing a key on the keyboard. If `None` is passed as the function parameter, the callback is canceled.

`ontimer(function, time=0)` Register a function to a timer event. The `function` will be called once after `time` milliseconds. The `function` must be defined with no parameters.

`listen(x=None, y=None)` Give focus to the current turtle's canvas so it can listen for key events. Key events will not be recognized until this function is called. This `listen` method is defined to take two dummy parameters so that `listen` can be bound to a mouse click event.

`mainloop()` Start the event loop. Must be the last statement in a turtle graphics program.

C.11 Miscellaneous

`turtles()` Return a list of all the turtles on the screen.

`done()` `static` method to stop turtle activity. ex: `turtle.done()`

`clearscreen()` Delete all drawing and all turtles from the screen. The screen is set to a white background, there are no event bindings, and tracing is on.

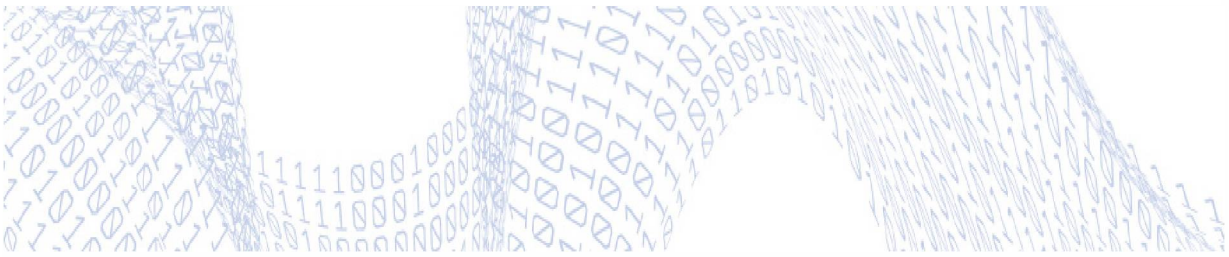
`resetscreen()` Reset all turtles on the screen to their initial positions.

`screensize(canvwidth=None, canvheight=None, bg=None)` Resize the drawing canvas to `canvwidth` and `canvheight`. Does not resize the window; just adds scroll bars for viewing areas not in original drawing area. If no arguments are given, returns the current width and height.

`bye()` Close the turtle graphics window.

`exitonclick()` Wait for a mouse click inside the window, then close and exit.

`setworldcoordinates(llx, lly, urx, ury)` Modify the world coordinates of the drawing canvas. This will automatically scale the canvas between `llx` (lower-left `x`) and `urx` (upper-right `x`) in the `x` direction and `lly` (lower-left `y`) and `ury` (upper-right `y`) in the `y` direction.



© Yuri Hoyda/Shutterstock.

Appendix D
Answers to Selected Try It Out Exercises

Chapter 1

1.1

```
8 + 9 + 10
# 27
```

1.3

```
365 * 24 * 60 * 60
# 31536000
```

1.6

```
(17*(17 + 1)) // 2  # assuming that you have 18 classmates
# 153
```

Description

1.9

```
2 **120
# 1329227995784915872903807060280344576
```

1.10

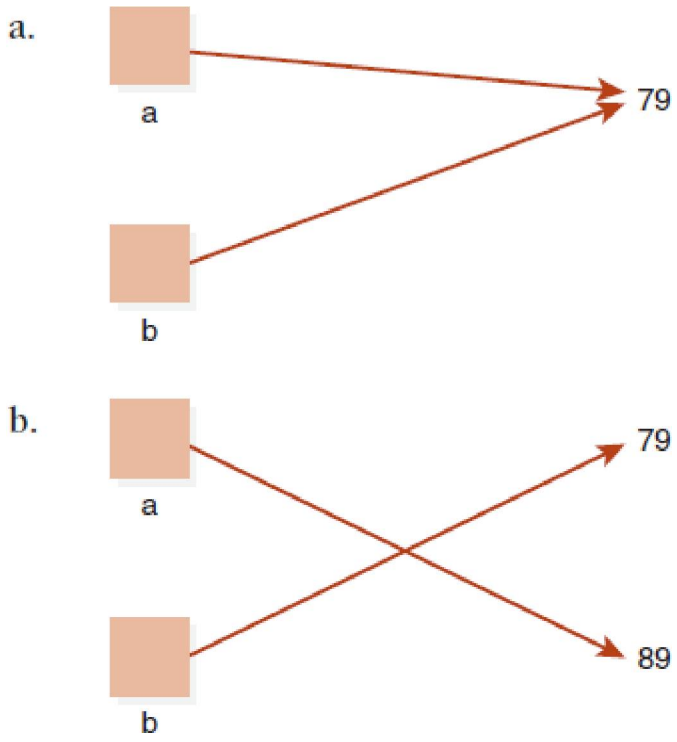
```
(18 + 19 + 19 + 20 + 25 ) / 5
# 20.2
```

Description

1.13

```
2900000 * 5.878e12
# 1.70462e+19
```

1.15



Description

1.18

a is 20; *b* is 15

1.21

```
ole = turtle.Turtle()

ole.right(45)

ole.forward(50)
```

Description

1.27

```
def drawRectangle(myTurtle, width, height):
    myTurtle.forward(width)
    myTurtle.left(90)
    myTurtle.forward(height)
    myTurtle.left(90)
    myTurtle.forward(width)
    myTurtle.left(90)
    myTurtle.forward(height)
    myTurtle.left(90)
```

Description

1.32

```
range( -10, 11, 1 )
```

Description

1.37

```
def drawSpiral(turtle1, turtle2, maxSide):
    for sideLength in range(1, maxSide+1, 5):
        turtle1.forward(sideLength)
        turtle2.forward(sideLength)
        turtle1.right(140)
        turtle2.right(140)
```

Description

1.39

```
def drawSquare(myTurtle, sideLength):
    myTurtle.forward(sideLength)
    myTurtle.left(90)
    myTurtle.forward(sideLength)
    myTurtle.left(90)
    myTurtle.forward(sideLength)
    myTurtle.left(90)
    myTurtle.forward(sideLength)
    myTurtle.left(90)

def drawNestedSquares(myTurtle):
    for i in range(200,(200 - 5 * 10),-5):
        drawSquare(myTurtle, i)
```

Description

1.43

```
def plotCircle(myTurtle, radius):
    circumference = 2 * 3.1415 * radius
    sideLength = circumference / 360
    myTurtle.up()
    myTurtle.left(90)
    myTurtle.forward(radius)
    myTurtle.right(90)
    myTurtle.down()
    drawPolygon(myTurtle, sideLength, 360)
    myTurtle.up()
    myTurtle.right(90)
    myTurtle.forward(radius)
    myTurtle.left(90)
    myTurtle.down()
```

Description

Chapter 2

2.1

```
False or True
# True
```

2.3

```
not 7 > 3
# False
```

2.6

```
not (7 > 3 and 8 < 2)
# True
```

2.9

```
if result == 100:
    answer = 1
else:
    answer = 2
```

Description

2.11

```
if score >= 90:
    gradePoint = 4
elif score >= 80:
    gradePoint = 3
elif score >= 70:
    gradePoint = 2
elif score >= 60:
    gradePoint = 1
else
    gradePoint = 0
```

Description

2.13

```
def costOfOranges(pounds):
    costFruit = pounds * 0.32
    shipping = 7.50
    if pounds > 100:
        shipping = shipping - 1.50
    totalCost = costFruit + shipping
    return totalCost
```

Description

2.15

```
def paycheck(rate, hours):
    if hours <= 40:
        totalPay = rate * hours
    else
        totalPay = rate * 40 + (1.5 * rate * (hours - 40))
    return totalPay
```

Description

2.17

```
def mychatbot(question):
    question = question.lower()

    if question.startswith("who") or "name" in question:
        print("The tooth fairy")
    elif question.startswith("what"):
        print("To be kind to one another")
    elif question.startswith("how") and "feeling" in question:
        print("I don't feel anything. I'm a computer!")
    elif question.startswith("where") and "world" in question:
        print("The south pole")
    elif question.startswith("why"):
        print("You'll understand when you're older.")
    elif question.startswith("when"):
        print("This afternoon")
    else:
        print("You stumped me on that!")
```

Description

2.20

```
def askmychatbot():
    question = input("what is your question?")
    question = question.lower()

    if question.startswith("who") or "name" in question:
        print("The tooth fairy")
    elif question.startswith("what"):
        print("To be kind to one another")
    elif question.startswith("how") and "feeling" in question:
        print("I don't feel anything. I'm a computer!")
    elif question.startswith("where"):
        print("At the dentist")
    elif question.startswith("why"):
        print("You'll understand when you're older.")
    elif question.startswith("when"):
        print("This afternoon")
    else:
        print("You stumped me on that!")
```

Description

2.22

```
def validateHashtag(inputText):
    if inputText.startswith("#") and " " not in inputText:
        isHashtag = True
    else:
        isHashtag = False
    return isHashtag
```

Description

2.23

```
def sumOfEvenNumbers():
    totalSum = 0
    count = 0
    num = 0 # mathematically, 0 is considered an even number
    while count < 100:
        if num % 2 == 0:
            totalSum = totalSum + num
            count = count + 1
        num = num + 1
    return totalSum
```

Description

2.26

```
def avgOfOddNumbers():
    totalSum = 0
    count = 0
    num = 1
    while count < 100:
        if num % 2 != 0:
            totalSum = totalSum + num
            count = count + 1
        num = num + 1
    return totalSum/count
```

Description

2.27

```
round(math.log(50))
round(math.log(100))
round(math.log(300))
```

Description

2.28

```
def guessANumber(start, end):
```

```
def guessANumber(start, end):

    print("Pick a number between " + str(start) + " and " + str(end) + ".")
    print("I'll try to guess it in as few guesses as possible.")

    guess = (end + start) // 2
    numAttempts = 1
    response = input("Is " + str(guess) + " too high, too low, or correct? ")
    response = response.lower()

    while response != 'correct':
        if response == 'too high':
            end = guess - 1
        elif response == 'too low':
            start = guess + 1
        else:                           # invalid response
            response = input("Please enter 'too high', 'too low', or 'correct' ")
            response = response.lower()
            continue                    # skip the remainder of the loop
```

```
        continue                    # skip the remainder of the loop

    guess = (end + start) // 2
    numAttempts = numAttempts + 1
    response = input("Is " + str(guess) + " too high, too low, or correct? ")
    response = response.lower()


print("Your number is " + str(guess))
print("I guessed it in " + str(numAttempts) + " trys.")
```

Description

2.30

```
import random
def flipCoin():
    side = random.randint(0,1)
    if side == 0:
        print("Heads")
    else:
        print("Tails")
```

Description

2.32

```
def nim2(pile):
    if pile < 5:
        print("The pile must contain at least 5 stones.")
    else:
        print("The pile has " + str(pile) + " stones.")

        winner = ""
        while pile != 0:
            c1Move = randomMove(pile)
            pile = pile - c1Move
            print("Random Randy takes " + str(c1Move) + " stone(s).")
            print("Pile is now " + str(pile))
            if pile == 0:
                winner = "Random Randy"
            else:
                c2Move = nimbleMove(pile)
```

```
            pile = pile - c2Move
            print("Nimble Nancy takes " + str(c2Move) + " stone(s).")
            print("Pile is now " + str(pile))
            if pile == 0:
                winner = "Nimble Nancy"
print("The winner is " + winner + "!")
```

Description

Chapter 3

3.2 `fullName[:x]` where `x` is the index of the space after your first name

3.5

```
len(firstName)
```

3.7

```
for i in range(len(name) + 1):
    print(name[:i])
```

Description

3.10

```
river = "mississippi"
river.find("p")
river.index("p")
```

Description

3.13

```
def toInt(ch):
    if ord(ch) > ord('9') or ord(ch) < ord('0'):
        print( 'error:', ch, 'is not a digit')
        return -1
    else:
        return ord(ch) - ord('0')
```

Description

3.16

```
def letterGrade(score):
    if score >= 90:
        letter = 'A'
    elif score >= 80:
        letter = 'B'
    elif score >= 70:
        letter = 'C'
    elif score >= 60:
        letter = 'D'
    else:
        letter = 'F'
    return letter
```

Description

3.18 This solution is more general in that it will remove any characters in `removeString`.

Thus, to strip spaces, call this method with " " as the `removeString` parameter.

```
def removeMatches(myString, removeString):
    newStr = ""
    for ch in myString:
        if ch not in removeString:
            newStr = newStr + ch
    return newStr
```

Description

3.22

```
def substitutionEncryptNoSpaces(plainText, key):
    alphabet ="abcdefghijklmnopqrstuvwxyz"
    plainText = plainText.lower()
    plainText = plainText.replace(" ", "")
    cipherText = ""
    for ch in plainText:
        idx = alphabet.find(ch)
        cipherText = cipherText + key[idx]
    return cipherText
```

Description

3.25

```
def substitutionEncrypt(plainText, password):
    key = genKeyFromPass(password)
    alphabet = "abcdefghijklmnopqrstuvwxyz "
    plainText = plainText.lower()
    cipherText = ""
    for ch in plainText:
        idx = letterToIndex(ch)
        cipherText = cipherText + key[idx]
    return cipherText
```

Description

3.27

```
def rotEncrypt(message, rotNum):
    alphabet = "abcdefghijklmnopqrstuvwxyz "
    result = ""
    for ch in message:
        idx = alphabet.find(ch)
        idx = (idx + rotNum) % len(alphabet)
        result = result + alphabet[idx]
    return result

def rotDecrypt(cyberText, rotNum):
    return rotEncrypt(cyberText, -rotNum)
```

Description

3.28

```
def undoVig(keyLetter, ctLetter):
    keyIndex = letterToIndex(keyLetter)
    ctIndex = letterToIndex(ctLetter)
    newIdx = (ctIndex - keyIndex) % 26
    return indexToLetter(newIdx)
```

Description

Chapter 4

4.2

```
myList = [7, 9, "a", "cat", False]

myList.append(3.14)                    # (a)
myList.append(7)
myList

myList.insert(3, "dog")                # (b)
myList

myList.index("cat")                    # (c)

myList.count(7)                        # (d)

myList.remove(7)                       # (e)
myList

myList.pop(myList.index("dog"))        # (f)
myList
```

Description

4.5

```
def wordCount(sentence):
    wordList = sentence.split()
    return len(wordList)
```

Description

4.7

```
import random
def shuffle(aList):
    copyList = aList[:]
    newList = []
    for i in range(len(copyList)):
        item = copyList.pop(random.randrange(len(copyList)))
        newList.append(item)
    return newList
```

Description

4.10

```
myList = [ [] ]*3
myList
```

Description

The statement creates a list containing three copies of an empty list.

4.13

```
myList = []
for i in range(3):
    myList.append([])

myList

myList[1].append(2)
myList
```

Description

4.15

```
def getMin(aList):
    minSoFar = aList[0]
    for item in aList[1:]:
        if item < minSoFar:
            minSoFar = item
    return minSoFar
```

Description

4.18

```
def mean(aList):
    sumOfItems = 0
    for item in aList:
        sumOfItems = sumOfItems + item
    mean = sumOfItems / len(aList)
    return mean
```

Description

4.24

```
def makeDictionary(list1, list2):
    newDict = {}
    for i in range(len(list1)):
        newDict[list1[i]] = list2[i]
    return newDict
scoreDict = makeDictionary(names, scores)
```

Description

4.25

```
scoreDict.get('barb')
```

4.27

```
scoresList = list(scoreDict.values())
scoresList.sort()
```

Description

4.31

```
scoreDict = {'joe': 10, 'barb': 13, 'sue': 18, 'sally': 13, 'john': 19}
keys = scoreDict.keys()
keys = list(keys)
keys
# ['joe', 'barb', 'sue', 'sally', 'john']
keys.sort()
keys
# ['barb', 'joe', 'john', 'sally', 'sue']
for aName in keys:
    print(aName, scoreDict[aName])
```

Description

4.33

```
def mode(aList):
    maxValue = max(aList)
    countList = [0] * (maxValue + 1)

    for item in aList:
        countList[item] = countList[item] + 1

    maxCount = max(countList)
    modeList = []
    for i in range(len(countList)):
        if countList[i] == maxCount:
            modeList.append(i)
    return modeList
```

Description

4.35

```
def averages(scoreList):
    dataDict = {}
    for item in scoreList:
        name = item[0]
        score = item[1]
        if name in dataDict:
            dataDict[name].append(score)
        else:
            dataDict[name] = [score]
    for aKey in dataDict:
        print(aKey, sum(dataDict[aKey])/len(dataDict[aKey]))
```

Description

4.37

```
def frequencyTable(aList):
    countDict = {}

    for item in aList:
        if item in countDict:
            countDict[item] = countDict[item] + 1
        else:
            countDict[item] = 1

    items = countDict.items()
    items = list(items)
    items.sort()

    print("ITEM", "FREQUENCY")
    for item in items:
        print(item[0], "      ", item[1])
```

Description

```
import matplotlib.pyplot as plt
import statistics

def frequencyChart(aList):

    countDict = {}

    for item in aList:
        if item in countDict:
            countDict[item] = countDict[item] + 1
        else:
            countDict[item] = 1

    keys = list(countDict.keys())      #the x-axis
    values = list(countDict.values()) #the y-axis
    mean = statistics.mean(values)
    plt.bar(x = keys, height = values)
    plt.axvline(x = mean, color = 'red')
```

```
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```

Description

Chapter 5

5.1

```
with open("rainfall.txt", "r") as rainFile:
    with open("rainfallFmt.txt", "w") as outFile:
        for aLine in rainFile:
            values = aLine.split()
            city = values[0]
            inches = float(values[1])
            nChars = outFile.write(
             "{0:>25s} {1:5.1f}\n".format(city, inches))
```

Description

5.3

```
with open("rainfall.txt") as inFile:
    line1 = inFile.readline()
    line2 = inFile.readline()
    restOfFile = inFile.readlines()


restOfFile
# ['Algona 30.69\n', 'Allison 33.64\n', 'Alton 27.43\n', 'AmesW 34.07\n', 'AmesSE
33.95\n', 'Anamosa 35.33\n', 'Ankeny 33.38\n', 'Atlantic 34.77\n', 'Audubon
33.41\n', 'Beaconsfield 35.27\n', 'Bedford 36.35\n', 'BellePlaine 35.81\n',
'Bellevue 34.35\n', 'Blockton 36.28\n', 'Bloomfield 38.02\n', 'Boone 36.30\n',
'Brighton 33.59\n', 'Britt 31.54\n', 'Buckeye 33.66\n', 'BurlingtonKBUR 37.94\n',
'Burlington 36.94\n', 'Carroll 33.33\n', 'Cascade 33.48\n']
```

Description

Note: The `readlines` function returns lines 3 through the end of the file.

5.6

```
with open("rainfall.txt", "r") as inFile:
    lineCount = 0
    wordCount = 0
    charCount = 0
    for aLine in inFile:
        lineCount = lineCount + 1
        wordList = aLine.split()
        wordCount = wordCount + len(wordList)
        charCount = charCount + len(aLine[0:-1])
    print("Lines Words Characters")
    print("{0:>5} {1:>5} {2:>10}".format(lineCount, wordCount, charCount))
```

Description

5.9

```
def makeMagListByDate():
    magDict = {}
    with open("earthquakes.csv", "r") as inFile:
        csvReader = csv.reader(inFile)
        titles = next(csvReader)   #skip titles line
        for line in csvReader:
            mag = line[4]
            dateTime = line[0]
            dateTimeValues = dateTime.split("T")
            date = dateTimeValues[0]
            if date in magDict:
                magDict[date].append(mag)
            else:
                magDict[date] = [mag]
        dataList = []
        for aDate in magDict:
            dataList.append([aDate] + magDict[aDate])
        return dataList
```

Description

5.12

```
import pandas as pd
df = pd.read_csv("earthquakes.csv")
filteredDf = df[(df["latitude"] < 0) & (df["depth"] < 10)]
print(filteredDf)
```

Description

5.14

```
import pandas as pd
df = pd.read_csv("earthquakes.csv")
print("mean", df["depth"].mean())
print("median", df["depth"].median())
print("std", df["depth"].std())
```

Description

5.16

```
def getProperty(data, key):
    valueList = []
    earthquakes = data.get("features")
    for i in range(len(earthquakes)):
        features = earthquakes[i]
        props = features.get("properties")
        value = props.get(key)
        valueList.append(value)
    return valueList
```

Description

5.18

```
feltShortList = []
earthquakes = eData.get("features")
for i in range(len(earthquakes)):
    features = earthquakes[i]
    props = features.get("properties")
    felt = props.get("felt")
    if (felt is not None):
        feltShortList.append(felt)
```

Description

Chapter 6

6.2

```
def drawRectangle():
    rectX = 100
    rectY = 100
    rectWidth = 100
    rectLength = 125
    whitePixel = Pixel(255, 255, 255)
    im = EmptyImage(300, 300)
    #draw top and bottom lines
    for i in range(rectX, rectX + rectWidth):
        im.setPixel(i, rectY, whitePixel)
        im.setPixel(i, rectY + rectLength, whitePixel)
    #draw left and right lines
    for i in range(rectY, rectY + rectLength):
        im.setPixel(rectX, i, whitePixel)
        im.setPixel(rectX + rectWidth, i, whitePixel)
```

```
    return im


imWin = ImageWin("Drawing a Rectangle", 300, 300)
rectWin = drawRectangle()
rectWin.draw(imWin)
```

Description

6.6 Each color intensity is stored in a single byte (8 bits). The maximum value that can be stored in 8 bits is 255. Each color intensity ranges from 0 to 255, which is 256 values per intensity.

6.7

```
def clearRed(oldPixel):
    newPixel = Pixel(0, oldPixel.getGreen(), oldPixel.getBlue())
    return newPixel
```

Description

6.11

```
def blackWhite(oldPixel):
    red = oldPixel.getRed()
    green = oldPixel.getGreen()
    blue = oldPixel.getBlue()
    gray = (red + green + blue) // 3
    if gray <= 128:
        newPixel = Pixel(0, 0, 0)
    else:
        newPixel = Pixel(255, 255, 255)
    return newPixel
```

Description

6.17

```
def sepia(oldPixel):
    red = oldPixel.getRed()
    green = oldPixel.getGreen()
    blue = oldPixel.getBlue()
    newRed = int(red*0.393 + green*0.769 + blue*0.189)
    newGreen = int(red*0.349 + green*0.686 + blue*0.168)
    newBlue = int(red*0.272 + green*0.534 + blue*0.131)
    if newRed > 255:
        newRed = 255
    if newGreen > 255:
        newGreen = 255
    if newBlue > 255:
        newBlue = 255
    newPixel = Pixel(newRed, newGreen, newBlue)
    return newPixel
```

Description

6.22

```
def fourTimes(oldImage):
    oldW = oldImage.getWidth()
    oldH = oldImage.getHeight()
    newIm = EmptyImage(oldW * 4, oldH * 4)
    for row in range(newIm.getHeight()):
        for col in range(newIm.getWidth()):
            origCol = col // 4
            origRow = row // 4
            oldPixel = oldImage.getPixel(origCol, origRow)
            newIm.setPixel(col, row, oldPixel)
    return newIm
```

Description

6.24

```
def halfSize(oldImage):
    newWidth = oldImage.getWidth() // 2
    newHeight = oldImage.getHeight() // 2
    newIm = EmptyImage(newWidth, newHeight)
    for row in range(newHeight):
```

```
        for col in range(newWidth):
            #get average of neighbors
            n1Pixel = oldImage.getPixel(2 * col, 2 * row)
            n2Pixel = oldImage.getPixel(2 * col + 1, 2 * row)
            n3Pixel = oldImage.getPixel(2 * col, 2 * row + 1)
            n4Pixel = oldImage.getPixel(2 * col + 1, 2 * row + 1)
            red = (n1Pixel.getRed() + n2Pixel.getRed() \
                   + n3Pixel.getRed() + n4Pixel.getRed()) // 4
            green = (n1Pixel.getGreen() + n2Pixel.getGreen() \
                   + n3Pixel.getGreen() + n4Pixel.getGreen()) // 4
            blue = (n1Pixel.getBlue() + n2Pixel.getBlue() \
                   + n3Pixel.getBlue() + n4Pixel.getBlue()) // 4
            newPixel = Pixel(red, green, blue)
            newIm.setPixel(col, row, newPixel)
    return newIm
```

Description

6.26

```
def reduceNoise(oldImage):
```

```
    import statistics
    height = oldImage.getHeight()
    width = oldImage.getWidth()
    newIm = EmptyImage(width, height)
    for row in range(height-1):
        for col in range(width-1):
            n1 = oldImage.getPixel(col, row)
            n2 = oldImage.getPixel(col, row + 1)
            n3 = oldImage.getPixel(col + 1, row)
            n4 = oldImage.getPixel(col + 1, row + 1)
            red = [n1.getRed(), n2.getRed(), n3.getRed(), \
                   n4.getRed()]
            green = [n1.getGreen(), n2.getGreen(), \
                   n3.getGreen(), n4.getGreen()]
            blue = [n1.getBlue(), n2.getBlue(), n3.getBlue(), \
                   n4.getBlue()]
            medRed = int(statistics.median(red))
            medGreen = int(statistics.median(green))
```

```
            medBlue = int(statistics.median(blue))
            newPixel = Pixel(medRed, medGreen, medBlue)
            newIm.setPixel(col, row, newPixel)
    # bring over end pixels unchanged
    for row in range(height - 1):
        oldPixel = oldImage.getPixel(width - 1, row)
        newIm.setPixel(width-1, row, oldPixel)
    for col in range(width - 1):
        oldPixel = oldImage.getPixel(col, height - 1)
        newIm.setPixel(col, height-1, oldPixel)

    return newIm
```

Description

6.28

```
def completeFlip(oldImage):
    oldW = oldImage.getWidth()
    oldH = oldImage.getHeight()

    newIm = EmptyImage(oldW,oldH)

    for row in range(oldH):
        for col in range(oldW):
            oldPixel = oldImage.getPixel(oldW - 1 - col,
                                         oldH - 1 - row)
            newIm.setPixel(col, row, oldPixel)

    return newIm
```

Description

6.29

```
def verticalMirror(oldImage):
    oldW = oldImage.getWidth()
    oldH = oldImage.getHeight()
    newIm = EmptyImage(oldW, oldH)
    if oldW % 2 == 0:
        halfway = oldW // 2
    else:
        halfway = oldW // 2 + 1
    maxP = oldW -1
    for row in range(oldH):
        for col in range(halfway):
            oldPixel = oldImage.getPixel(maxP - col, row)
            newIm.setPixel(maxP - col, row, oldPixel)
            newIm.setPixel(col, row, oldPixel)
    return newIm
```

Description

6.32

```
def rotateImage90(oldImage):
    oldW = oldImage.getWidth()
    oldH = oldImage.getHeight()
    newIm = EmptyImage(oldH, oldW)
    rightCol = oldH - 1
    for row in range(oldH):
        for col in range(oldW):
            oldPixel = oldImage.getPixel(col, row)
            newIm.setPixel(rightCol, col, oldPixel)
        rightCol = rightCol - 1
    return newIm
```

Description

6.35 Smaller numbers yield more edges; larger numbers yield fewer edges. The actual results depend on the original image.

Chapter 7

7.1

```
with open("cs150exams.txt") as datafile:
    scores = []
    line = datafile.readline()
    while line != "":
        score = line.replace("\n","")
        scores.append(score)
        line = datafile.readline()
print(scores)
```

Description

7.2

```
def scoreAverage():
    sum = 0
    howMany = 0
    score = input("Please enter a score ('stop' to exit) ")
    while score != 'stop':
        sum = sum + int(score)
        howMany = howMany + 1
        score = input("Please enter a score ('stop' to exit) ")
    average = sum / howMany
    return average
```

Description

7.6

```
def createClusters(k, centroids, dataList, repeats):
    prevClusters = []
    for i in range(k):
        prevClusters.append([])
    aPass = 1
    doAgain = True
```

```
while doAgain:
    print("****PASS", aPass, "****")
    clusters = []
    for i in range(k):
        clusters.append([])

    for dataIndex in range(len(dataList)): # calculate distance to centroid
        distances = []
        for clusterIndex in range(k):
            dtoC = euclidD(dataList[dataIndex], centroids[clusterIndex])
            distances.append(dtoC)

        minDist = min(distances)                    # minimum distance
        closestCluster = distances.index(minDist)   # index of min distance

        clusters[closestCluster].append(dataIndex)  # add to cluster

    # check previous cluster against new cluster
    if clusters != prevClusters:
        prevClusters = clusters
```

```
        prevClusters = clusters
    else:
        doAgain = False

    dimensions = len(dataList[0])   # recompute the clusters
    for clusterIndex in range(k):
        sums = [0] * dimensions     # init sum for each dimension
        for dataIndex in clusters[clusterIndex]:
            dataPoints = dataList[dataIndex]
            for ind in range(len(dataPoints)): # calculate sums
                sums[ind] = sums[ind] + dataPoints[ind]
        for ind in range(len(sums)):  # calculate average
            clusterLen = len(clusters[clusterIndex])
            if clusterLen != 0:       # do not divide by 0
                sums[ind] = sums[ind] / clusterLen

        centroids[clusterIndex] = sums # assign avg to centroids

    for c in range(k):    # output the clusters
        print("CLUSTER #" + str(c+1))
```

```
        print("CLUSTER #" + str(c + 1))
        for dataIndex in clusters[c]:
            print(dataList[dataIndex], end = " ")
        print()

    aPass = aPass + 1

return clusters
```

Description

Chapter 8

8.1

```
def createWordList(filename):
    myList = []
    with open(filename, "r") as myFile:
        for aLine in myFile:
            myList.append(aLine[:-1])
    return myList
```

Description

8.2

```
import time
def timeFunctions():
    start = time.time()
    createWordDict("wordlist.txt")
    end = time.time()
    print("Dict:", end - start)
    start = time.time()
    createWordList("wordlist.txt")
    end = time.time()
    print("List:", end - start)
```

Description

Note that it takes longer to create the dictionary.

8.4

```
wList = ['the', 'quick', 'brown', 'fox']
' '.join(wList)
# 'the quick brown fox'
':'.join(wList)
# 'the:quick:brown:fox'
','.join(wList)
# 'the,quick,brown,fox'
'--'.join(wList)
# 'the--quick--brown--fox'
```

Description

8.9

```
def railBreak(cipherText):
    wordDict = createWordDict("wordlist.txt")
    cipherLen = len(cipherText)
    maxGood = 0
    for i in range(1, len(cipherText) // 2): # part (a)
        if cipherLen % i == 0: # part (b)
            print("trying rails:", i)
            words = railDecrypt(cipherText, i)
            goodCount = 0
            for w in words:
                if w in wordDict:
                    goodCount = goodCount + 1
            if goodCount > maxGood:
                maxGood = goodCount
                bestGuess = " ".join(words)
    return bestGuess
```

Description

8.14

```
def wordPop(text, wordLength):
    wordList = []
    words = text.split()
    for w in words:
        if len(w) == wordLength and w not in wordList:
            wordList.append(w)
    wordList.sort()
    return wordlist
```

Description

8.15

```
def sortKey(x):
    return x[1]


def endPop(myString):
    wordList = myString.split()
    popDict = {}
    for word in wordList:
        popDict[word[-1]] = popDict.get(word[-1], 0) + 1
    popList = list(popDict.items())
    popList.sort(key=sortKey, reverse=True)
    return popList[0]
```

Description

8.18

```
re.match('.*ing', myWord)
```

Description

8.22

```
re.match('[a-z][aeiou][aeiou][a-z]$', myWord)
```

Description

8.24

```
def getNameSuffix(filename):
    g = re.match('(.*)\.(.*)',filename)
    if g:
        return g.group(1), g.group(2)
    else:
        return None
```

Description

Chapter 9

9.2

```
def drawCenteredSquare(aTurtle, x, y, side):
    aTurtle.up()
    aTurtle.goto(x, y)
    aTurtle.dot()
    aTurtle.left(90)
    aTurtle.forward(side / 2)
    aTurtle.left(90)
    aTurtle.forward(side / 2)
    aTurtle.right(180)
    aTurtle.down()
    for i in range(4):
        aTurtle.forward(side)
        aTurtle.right(90)

def concentricBox(aTurtle, x, y, side):
```

```
    if side >= 1:
        drawCenteredSquare(aTurtle, x, y, side)
        concentricBox(aTurtle, x, y, side - 10)

import turtle
t = turtle.Turtle()
concentricBox(t, 0, 0, 200)
```

Description

9.4

```
def sumList(aList):
    if aList == []:
        return 0
    else:
        return aList[0] + sumList(aList[1:])
```

Description

9.7

```
def reverseString(aString):
    if aString == '':
        return ''
    else:
        return reverseString(aString[1:]) + aString[0]
```

Description

9.10

```
def treeLeft(t, trunkLength):
    if trunkLength < 5:
        return
    else:
        t.forward(trunkLength)
        t.left(30)
        treeLeft(t, trunkLength - 15)
        t.right(60)
        treeLeft(t, trunkLength - 15)
        t.left(30)
        t.backward(trunkLength)
```

Description

9.12

```
import random
def tree(t, trunkLength):
    if trunkLength < 5:
        return
    else:
        t.forward(trunkLength)
        t.left(30)
        tree(t, trunkLength - random.randint(5, 25))
        t.right(60)
        tree(t, trunkLength - random.randint(5, 25))
        t.left(30)
        t.backward(trunkLength)
```

Description

9.15

```
def drawTriangle(t, p1, p2, p3, aColor):
    t.color(aColor)
    t.up()
    t.goto(p1)
    t.down()
    t.goto(p2)
    t.goto(p3)
    t.goto(p1)

def sierpinski(myTurtle, p1, p2, p3, depth, aColor):
    if depth > 0:
        sierpinski(myTurtle,p1,midPoint(p1,p2),midpoint(p1,p3),
                                    depth-1, 'blue')
        sierpinski(myTurtle,p2,midPoint(p2,p3),midpoint(p2,p1),
                                    depth-1, 'green')
        sierpinski(myTurtle,p3,midPoint(p3,p1),midpoint(p3,p2),
                                    depth-1, 'red')
    else:
        drawTriangle(myTurtle, p1, p2, p3, aColor)
```

To execute these functions:

```
import turtle
t = turtle.Turtle()
sierpinski(t, (0,0), (200,200), (200,0), 4, 'black')
t.hideturtle()
```

Description

9.23

```
def recKoch(aT, distance, level):
    if level == 0:
        aT.forward(distance)
    else:
        recKoch(aT, distance/3, level - 1)
```

```
        aT.left(60)
        recKoch(aT, distance/3, level - 1)
        aT.right(120)
        recKoch(aT, distance/3, level - 1)
        aT.left(60)
        recKoch(aT, distance/3, level - 1)

import turtle
t = turtle.Turtle()
t.up()
t.goto(-100, 100)
t.down()

recKoch(t, 200, 4)
t.right(120)
recKoch(t, 200, 4)
t.right(120)
recKoch(t, 200, 4)
```

```
t.hideturtle()
```

Description

9.25

```
AB
ABA
ABAAB
ABAABABA
ABAABABAABAAB
```

Description

9.28

```
import turtle
t = turtle.Turtle()
axiom = 'F++F++F'
myRules = {'F': 'F-F++F-F'}
instructions = applyProduction(axiom, myRules, 3)
t.up()
t.goto(-100, 100)
t.down()
drawLS(t, instructions, 60, 10)
t.hideturtle()
```

Description

9.30 Insert this code into the `match` command:

```
case: 'G':
    aTurtle.up()
    aTurtle.forward(distance)
    aTurtle.down()
```

Description

Chapter 10

10.1

```
class Planet:
    def __init__(self, iName, iRad, iM, iDist, iMoons):
        self.__name = iName
        self.__radius = iRad
        self.__mass = iM
        self.__distance = iDist
        self.__numMoons = iMoons
```

Description

10.4 and 10.5

```
class Sentence:
    def __init__(self, initialString):
        self.__theString = initialString

    def getSentence(self):
        return self.__theString

    def getWords(self):
        return self.__theString.split()

    def getLength(self):
        return len(self.__theString)

    def getNumWords(self):
        return len(self.__theString.split())
```

Description

10.8

```
def setMoons(self, nummoons):
    self.__numMoons = nummoons;
```

Description

10.11

```
def addPunc(self, punc):
    self.__theString = self.__theString + punc
```

Description

10.14

```
def __getitem__(self, index):
    return self.__theString.split()[index]
```

Description

10.17

```
def __add__(self, otherSentence):
    newString = self.__theString + ' '  \
                + otherSentence.getSentence()
    return Sentence(newString)
```

Description

10.18 and 10.19

```
class Die:
    def __init__(self, inumsides):
        self._numSides = inumsides
        self._currRoll = self.roll()
    def roll(self):
        import random
        self._currRoll = random.randint(1, self._numSides)
        return self._currRoll
    def getCurrRoll(self):
        return self._currRoll
    def __str__(self):
        return str(self._currRoll)
    def __eq__(self, otherDie):
        return self.getCurrRoll() == otherDie.getCurrRoll()
    def __ne__(self, otherDie):
        return self.getCurrRoll() != otherDie.getCurrRoll()
    def __lt__(self, otherDie):
```

```
        return self.getCurrRoll() < otherDie.getCurrRoll()
    def __le__(self, otherDie):
        return self.getCurrRoll() <= otherDie.getCurrRoll()
    def __gt__(self, otherDie):
        return self.getCurrRoll() > otherDie.getCurrRoll()
    def __ge__(self, otherDie):
        return self.getCurrRoll() >= otherDie.getCurrRoll()
```

Description

Example:

```
d1 = Die(6)
d2 = Die(6)
d1.roll()
print(d1)
# 6
d2.roll()
print(d2)
# 3
d1 == d2
```

```
# False
d1 != d2
# True
d1 < d2
# False
d1 <= d2
# False
d1 > d2
# True
d1 >= d2
# True
```

Description

10.23 and 10.24

```
class BankAccount:
    def __init__(self, iid):
        self.__idNum = iid
        self.__balance = 0
    def deposit(self, amount):
        self.__balance = self.__balance + amount
    def withdraw(self, amount):
        self.__balance = self.__balance - amount
    def getBalance(self):
        return self.__balance
    def transferTo(self, otherAcct, amount):
        self.withdraw(amount)
        otherAcct.deposit(amount)
```

Description

10.28

```
class Sun:
    def __init__(self, iname, irad, im, itemp):
        self.__name = iname
```

```
        self.__radius = irad
        self.__mass = im
        self.__temp = itemp
    def getMass(self):
        return self.__mass
    def getRadius(self):
        return self.__radius
    def getTemperature(self):
        return self.__temp
    def getVolume(self):
        import math
        v = 4/3 * math.pi * self.__radius**3
        return v
    def getSurfaceArea(self):
        import math
        sa = 4 * math.pi * self.__radius**2
        return sa
    def getDensity(self):
```

```
        d = self.__mass / self.getVolume()
        return d
    def __str__(self):
        return self.__name
    def setName(self, newName):
        self.__name = newName
    def setRadius(self, newRad):
        self.__radius = newRad
```

Description

10.31

```
def getTotalMass(self):
    totalMass = self.theSun.getMass()
    for aPlanet in self.planets:
        totalMass = totalMass + aPlanet.getMass()
    return totalMass
```

Description

10.33

```
def getNearest(self):
    nearest = self.planets[0]
    for aPlanet in self.planets:
        if aPlanet.getDistance() < nearest.getDistance():
            nearest = aPlanet
    return nearest


def getFarthest(self):
    farthest = self.planets[0]
    for aPlanet in self.planets:
        if aPlanet.getDistance() > nearest.getDistance():
            farthest = aPlanet
    return farthest
```

Description

To call these methods with a `SolarSystem` named `ss`:

```
print(ss.getNearest())
print(ss.getFarthest())
```

Chapter 11

11.1

```
for aCol in range(self.__maxX):
    col = []
    for aRow in range(self.__maxY):
        col.append(None)
    self.__grid.append(col)
```

Description

11.7

```
def tryToMove(self):
    offsetList = [(-1,1), (0,1), (1,1),
                  (-1,0),        (1,0),
                  (-1,-1),(0,-1),(1,-1)]
    tryCount = 0
    goodMove = False
    while tryCount < 4 and not goodMove:
        randomOffsetIndex = random.randrange(len(offsetList))
```

```
randomOffset = offsetList[randomOffsetIndex]
nextX = self._xPos + randomOffset[0]
nextY = self._yPos + randomOffset[1]
while not (0 <= nextX < self._world.getMaxX() and \
           0 <= nextY < self._world.getMaxY() ):
        randomOffsetIndex = \
                random.randrange(len(offsetList))
        randomOffset = offsetList[randomOffsetIndex]
        nextX = self._xPos + randomOffset[0]
        nextY = self._yPos + randomOffset[1]

        if self._world.emptyLocation(nextX, nextY):
            self.move(nextX, nextY)
            goodMove = True
        else:
            tryCount = tryCount + 1
```

Description

11.9

```
class Bear:
    def __init__(self):
        self.__turtle = turtle.Turtle()
        self.__turtle.up()
        self.__turtle.hideturtle()
        self.__turtle.shape("Bear.gif")

        self.__xPos = 0
        self.__yPos = 0
        self.__world = None

        self.__starveTick = 0
        self.__breedTick = 0
        self.__starveThreshold = 10
        self.__breedThreshold = 8

    # other methods …
```

```
def liveALittle(self):
    self.__breedTick = self.__breedTick + 1
    if self.__breedTick >= breedThreshold:
        self.tryToBreed()

    self.tryToEat()

    if self.__starveTick == starveThreshold:
        self.__world.delThing(self)
    else:
        self.tryToMove()
```

Description

Chapter 12

12.2

```
class Point(GeometricObject):
    def __init__(self, x, y):
        super().__init__()
        self.pos = (x, y)
    def getCoord(self):
        return self.pos
    def getX(self):
        return self.pos[0]
    def getY(self):
        return self.pos[1]
    def _draw(self, turtle):
        turtle.goto(self.pos[0], self.pos[1])
        turtle.dot(self.lineWidth, self.lineColor)
```

Description

12.4

```
from draw import *
myCanvas = Canvas(800, 600)
x = 0
for i in range(360):
    y = 100 * math.sin(math.radians(i))
    myPoint = Point(x, y)
    myPoint.setWidth(3)
    myPoint.setColor('blue')
    myCanvas.draw(myPoint)
    x = x + 1
```

Description

12.7 Using Figure 12.5, the search follows the arrows numbered 1, 2, 3, and 4.

12.10 Add this code to the `Canvas` class:

```
def undraw(self, gObject):
    self.__visibleObjects.remove(gObject)
    gObject.setVisible(False)
    self.drawAll()
```

Description

12.13

```
class Polygon(Shape):
    def __init__(self, plist):
        super().__init__()
        self._corners = plist

    def _draw(self, aturtle):
        aTurtle.color(self.getColor())
        aTurtle.width(self.getWidth())
        aTurtle.goto(self._corners[0].getCoord())
        aTurtle.down()
        if self.getFillColor() != None:
            aTurtle.fillcolor(self.getFillColor())
            aTurtle.begin_fill()
        for cIndex in range(1, len(self._corners)):
            aTurtle.goto(self._corners[cIndex].getCoord())
        aTurtle.goto(self._corners[0].getCoord())
```

```
        if self.getFillColor() != None:
            aTurtle.end_fill()


    def addPoint(self, aPoint):
        self._corners.append(aPoint)


    def getPoints(self):
        return self._corners
```

Description

12.14 Each subclass of `Polygon` should simply create a list of `Point`s and send it to the `Polygon` constructor. For example, here is a `Rectangle` class:

```
class Rectangle(Polygon):
    def __init__(self, lowerLeft, upperRight):
        cornerList = []
        cornerList.append(lowerLeft)
        cornerList.append(Point(lowerLeft.getX(), upperRight.getY()))
        cornerList.append(upperRight)
        cornerList.append(Point(upperRight.getX(), lowerLeft.getY()))
        super().__init__(cornerList)

    def getLowerLeft(self):
        return lowerLeft

    def getUpperRight(self):
        return upperRight
```

Description

Chapter 13

13.4
(13.5 is similar)

```
from threading import Thread
import time
class EventHandler(Thread):
    def _init_(self):
        super()._init_()
        self._queue = []
        self._eventKeeper = {}

    def addEvent(self, eventName, eventData):
        self._queue.append((eventName, eventData)) #function and param

    def registerCallback(self, event, func):
        self._eventKeeper[event] = func

    def run(self):
```

```
    def run(self):
        while (True):
            if len(self._queue) > 0:
                nextEvent, eventData = self._queue.pop(0)
                callBack = self._eventKeeper[nextEvent]
                callBack(eventData)     #pass param to function
            else:
                time.sleep(1)  #pause for 1 second


def myKey(key):
    print('The {0} key was pressed'.format(key))


eh = EventHandler()
eh.addEvent('key', 'j')
eh.registerCallback('key', myKey)
eh.start()
# The j key was pressed
```

Description

13.7

```
class Etch(Turtle):
    def __init__(self):
        super().__init__()
        self.__screen = self.getscreen()
        self.color('blue')
        self.pensize(2)
        self.speed(0)
        self.__distance = 5
        self.__turn = 10
        self.__screen.onkey(self.fwd, "Up")
        self.__screen.onkey(self.bkwd, "Down")
        self.__screen.onkey(self.left5, "Left")
        self.__screen.onkey(self.right5, "Right")
        self.__screen.onkey(self.quit, "q")
        self.__screen.onkey(self.tailup, "u")
        self.__screen.onkey(self.taildown, "d")
        self.__screen.listen()
```

```
        self.main()

    def fwd(self):
        self.forward(self.__distance)

    def bkwd(self):
        self.backward(self.__distance)

    def left5(self):
        self.left(self.__turn)

    def right5(self):
        self.right(self.__turn)

    def quit(self):
        self.screen.bye()

    def tailup(self):
```

```
        self.up()

    def taildown(self):
        self.down()

    def main(self):
        mainloop()
```

Description

13.11 In the `placeTurtle` method, change the last two lines to

```
if self.__numTurtles >= self.__maxTurtles:
    self.__bigTscreen.onclick(self.moveTo)
```

Description

and then add this method to the class:

```
def moveTo(self, x, y):
    for t in self.__turtleList:
        t.setheading(t.towards(x, y))
        t.forward(10)
```

13.19 Make all these changes to the `AnimatedTurtle` class:

1. In the `__init__` method:
   Add a `__moving` instance variable initialized to `True`:

   ```
   self.__moving = True
   ```

   Add a callback for `onclick`:

   ```
   self.onclick(self.__stopGo)
   ```

2. Add this method:

   ```
   def __stopGo(self, x, y):
       for t in AnimatedTurtle.__allTurtles:
           if t.distance(x, y) < 10:
               if t.__moving:
                   t.__moving = False
               else:
                   t.__moving = True
   ```

3. Change the `moveOneStep` method to:

   ```
   def __moveOneStep(self):
       if self.__moving:
           self.__computeNewHeading()
           self.forward(5)
           self.__checkCollisions()
       self.__scr.ontimer(self.__moveOneStep, 100)
   ```

Description

13.21

a. Add this ScoreKeeper class:

```
class ScoreKeeper(Turtle):
        gameScore = 0    # static variable

        @staticmethod
        def alterScore(amount):
            ScoreKeeper.gameScore = ScoreKeeper.gameScore + amount

        def __init__(self, scoreX, scoreY):
            super().__init__()
            self.hideturtle()
            self.up()
            self.goto(scoreX, scoreY)
            self.down()
            self.__oldScore = -1  # force display first time
            self.writeScore()

        def writeScore(self):
            if ScoreKeeper.gameScore != self.__oldScore:
                # update if changed
                self.clear()
                self.write('Score: ' + str(ScoreKeeper.gameScore),
                           move = False)
             self.__oldScore = ScoreKeeper.gameScore
```

Description

b. Change the `DroneInvaders` class:

```
class DroneInvaders:
    def __init__(self, xMin, xMax, yMin, yMax):
        self.__xMin = xMin
        self.__xMax = xMax
        self.__yMin = yMin
        self.__yMax = yMax

    def play(self):
        self.mainWin = LaserCannon(self.__xMin, self.__xMax,
                                   self.__yMin, self.__yMax).getscreen()
        self.mainWin.bgcolor('light green')
        self.mainWin.setworldcoordinates(self.__xMin, self.__yMin,
                                         self.__xMax, self.__yMax)
        #start ScoreKeeper
        self.__scoreKeeper = ScoreKeeper(self.__xMin + 50,
                                         self.__yMax - 50)

        self.mainWin.ontimer(self.addDrone, 200)
        self.mainWin.listen()
        mainloop()


def addDrone(self):
    if len(Drone.getDrones()) < 7:
        Drone(1, self.__xMin, self.__xMax,
                 self.__yMin, self.__yMax)
    self.__scoreKeeper.writeScore()    #write score every 200 ms
    self.mainWin.ontimer(self.addDrone, 200)
```

Description

c. Change the `move` method in the `Bomb` class to add 10 when a `Drone` is hit:

```
def move(self):
    exploded = False
    self.forward(self.getSpeed())
    for a in Drone.getDrones():
        if self.distance(a) < 5:
            a.remove()
            ScoreKeeper.alterScore(10)
            exploded = True
    if self.outOfBounds() or exploded:
        self.remove()
    else:
       self.getscreen().ontimer(self.move, 100)
```

Description

d. Change the `move` method in the `Drone` class to subtract 10 from the score when the `Drone` goes out of bounds.

```
def move(self):
    self.forward(self.getSpeed())
    if self.outOfBounds():
        self.remove()
        ScoreKeeper.alterScore(-10)
    else:
        self.getscreen().ontimer(self.move, 200)
```

Description



© Yuri Hoyda/Shutterstock.

Glossary

A

abstract class A convenience class for defining instance variables and methods used by all of the child classes; cannot be used to construct objects.

abstract method A method in an abstract class that must be implemented in all concrete child classes.

abstraction A concept or idea not associated with any specific instance.

accessor methods Allow code outside the class to access the current values of the instance variables. Also referred to as “getter methods” because the word “get” often appears in the name.

accumulator pattern A commonly occurring pattern where a total is computed by repeatedly computing a running sum.

accumulator variable The variable that is continuously updated with the current value of the running sum.

actual parameters References to objects passed to the function that represent the “actual” data that the function will receive.

algorithms Sets of step-by-step instructions that can be performed by a computer (creating a computational process).

American Standard Code for Information Interchange (ASCII) The mapping of 128 characters to numbers defined in the 1960s.

assignment statement A statement containing an object name on the left-hand side, an expression on the right-hand side, and the assignment operator (=). The value of the expression is assigned to the object name.

attributes Special values for Python objects.

axiom The base case for a grammar.

B

backslash continuation character Character used to indicate that a statement is unfinished and continues on the next line.

bar chart A graphical representation using vertical or horizontal bars to represent values in a collection.

base case The nonrecursive case that implements a simple solution.

base class Another way of thinking of the superclass.

binary search A type of search in which possible numbers are cut in half with each guess.

block A group of Python statements that are indented at the same level.

Boolean expression An expression that compares values of objects.

boundary values Edge values that should cause different results when testing to ensure that code works properly.

brute force A method for solving a cipher in which all possible values are tried.

`builtins` **namespace** A namespace created when Python is started that includes all the system-defined names of functions and data types regularly used in Python.

C

call by assignment In Python, the mechanism for passing all parameters. References to the actual parameters are assigned to the formal parameters.

call tree A way to graph a function that shows how the recursive calls are ordered.

callback function Function designed to be called in response to an event.

calltips Reminders displaying the parameters that a method accepts.

camelCase A naming convention in which the name begins with a lowercase letter and the first letter of internal words is capitalized.

capture group Patterns specified within parentheses to help determine which characters in the target word matched letters in different parts of our pattern.

case block Statements to be executed when a specified value is equal to the subject in a `match` statement.

central tendency A measure that estimates where the “typical value” of a collection will be found.

centroid The mean of a collection of data points.

characters The letters of the alphabet and other symbols commonly used in writing.

child class A class more specific than a parent class.

ciphertext Messages that have been encoded.

class Used in object-oriented programming to describe methods and data for objects.

close a file Release the resources, such as memory, allocated to the file.

cluster analysis A data mining technique that attempts to divide the data into meaningful groups called clusters.

clusters Groups that represent data values that show some kind of similarity to one another while exhibiting a dissimilar relationship to data values outside.

collections Ways to organize and store data values.

color intensity The amount of each color component in a pixel. The value ranges from 0 to 255.

column-major order Mapping method that stores all the values in the first column, one after another, followed by the second column, the third column, and so on.

comment Descriptive documentation placed in code that will not impact the execution of the program. In Python, comments begin with the # character.

complex number Primitive numeric type in Python; consists of a real part and an imaginary part.

composition Design approach that implements classes with HAS-A relationships; one class has an instance variable that is an object of the other class.

compound Boolean expressions Expressions composed of simple Boolean expressions connected by `and`, `or`, or `not`.

computer science The study of algorithms.

computer simulation A computer program that is designed to model some specific aspects of a real situation or system.

concrete class A class that is not abstract and from which objects can be created.

condition A test that resolves to `True` or `False`.

constructor The method that creates an instance (object) of a class.

convolution A process in which pixels in the original image are mathematically combined with masks called kernels. The result is then used to decide whether that pixel represents an edge.

correlation A measure of the strength and direction of the relationship between two variables.

correlation coefficient A measure of the tendency of two variables to increase or decrease at the same time.

counter A variable used in the accumulator pattern that starts with an initial value, usually 0 or 1, and increases incrementally each time a particular event occurs.

cryptanalysis Field of code breaking.

cryptography The art of encoding and decoding text for secure communications.

CSV (comma-separated values) A data set file format where the first line identifies the data names and succeeding lines comprise the data delimited by commas.

D

data An assortment of items, many times numeric, that have been observed, measured, or collected by some means.

data mining The application of automated techniques that attempt to discover underlying patterns.

data model A conceptual representation of the data in a system.

data science The use of computational algorithms and scientific processes to gain understanding of or find patterns in large data sets.

DataFrame `Pandas`'s two-dimensional tabular form for storing data, similar to a spreadsheet.

decomposition Process of breaking down a problem into smaller parts.

decorator A Python syntax element that immediately precedes the method and appears on its own line.

decryption The process of converting ciphertext into plaintext.

define To give a name to an object.

delimiter A special character that indicates the end of a value.

dereferencing A process in which Python looks up a name and returns the object named.

dictionary A collection of pairs of items where each pair consists of a key and a related value.

dictionary view object The keys, values, or items that can be used to iterate through the dictionary.

digital image A collection of pixels organized in a two-dimensional grid.

digital image processing Using algorithms to edit and manipulate digital images.

dispersion A measure of data value spread.

dot operator Operator that indicates Python should dereference an object name in order to call a method.

dunder methods Another name for magic methods because they start and end with double underscores.

E

edge detection An image processing technique that tries to find places in the image that have dramatic changes in color intensity values.

empty list A list with zero data objects, represented by [].

encapsulation Design principle of classes in which only the class methods should be able to directly change the value of an object's instance variables.

encryption The process of converting plaintext into ciphertext.

equality operators Operators used to compare if two values are equal.

Euclidean distance The simple distance between two location values.

event loop Main structure of an event-driven program, which checks the event queue for the next event and calls the callback function to handle the event.

event-driven programs Programs that wait for an event to occur and then respond to that event by taking an action.

expression A combination of operators and operands.

extends Refers to a child class adding instance variables or methods to the parent class.

F

file pointer A marker that denotes the current read position in the file.

filtering Operation that allows for the creation of a shorter list by keeping only values from a longer list that meet certain criteria.

flip axis A line that indicates where a flip is to occur.

flipping Modifying an image by reversing pixels along a flip axis.

floating-point numbers Python's approximations of real numbers.

`for` **loop** A statement for repeating a block of code multiple times.

formal parameters Parameters named in the function header.

formatted strings String made from a template in which words or spaces that will remain constant are combined with placeholders for variables to be inserted into the string.

fractal A geometric figure in which the parts are similar at different scales.

frequency analysis Method of breaking a code by comparing the frequency of letters in the ciphertext to the frequency of letters in the English language.

frequency distribution A representation of the number of times each value occurs in a data collection.

function body The indented block of code in a function.

functions Prewritten Python code that defines statements to be executed as a unit.

G

generalization A technique that enables you to go from some specific examples to a solution that can be implemented as a program.

getter methods Allow code outside the class to access the current values of the instance variables. Also referred to as accessor methods.

grammar Computer science concept that can be used for the specification of programming languages and other purposes; consists of a set of symbols and one or more production rules.

grayscale A color manipulation in which every pixel will be converted to a shade of gray, ranging from very dark (black) to very light (white).

H

HAS-A relationship Describes two objects, one of which uses the other; also called composition.

heterogeneous collections Collections composed of mixtures of various data types.

hierarchy A group of related superclasses and subclasses.

histogram A graphical representation of the frequency of occurrences of values.

homogeneous collections Collections in which each item is the same data type.

I

immutable collections Collections of data in which individual items cannot be changed.

index operator Used to access characters in specific positions in a string.

infinite loop A loop that never ends.

information hiding Practice in which the internal representation of data is hidden from the user of the class.

inheritance The idea that a more general class is a parent class of a more specific class (a child class).

inheritance hierarchy A group of superclasses and subclasses defined by IS-A links.

initialization Assigning a variable a value at the beginning of a process.

in-place sort Modifying a list that is being sorted without creating a new list.

instances Objects of a class.

instance data Data that each object of a class has.

instance variables Data items defined in a class. Each object will have its own values.

integers Whole numbers.

invocation The two-step process to pass data when the function is called.

IS-A relationship Describes two objects, one of which is a more specific instance of the other.

iterate To visit each object in a collection one at a time.

iteration One execution of the statements in a loop body.

iterator An object that can return one item at a time from a group.

J

JSON (JavaScript Object Notation) A standardized data transfer format with self-identifying tags.

K

kernel A filter or mask used to weight the intensities of the surrounding pixels.

key A scrambled version of the alphabet used with a substitution cipher.

keyword parameter Parameter that consists of the keyword, the assignment operator, and a value.

keywords Names corresponding to Python capabilities, which are reserved by Python for its own use.

K-means algorithm A cluster algorithm.

Koch snowflake curve Fractal algorithm developed by Helge von Koch in 1906.

L

libraries Prewritten Python code that can be incorporated into programs.

list An ordered, sequential collection of zero or more Python data objects, written as comma-delimited values enclosed in square brackets.

list comprehension A Python feature that allows for easy creation of one list based on processing or applying selection criteria to another list.

list of lists One or more lists defined inside of a list.

local namespace A namespace created when a function is invoked that corresponds to the function and its data.

local variables Names created within the function that are part of the local namespace.

logical operators Operators `and`, `or`, and `not` that compare Boolean expressions.

loop body Contains the executable statements inside a loop.

loop variable The variable that follows the keyword `for` and takes on a different value for each iteration.

L-system A formal mathematical theory designed to model the growth of biological systems.

M

machine learning An area of data science where a computer program is fed lots of data showing both correct and incorrect results, and the program uses sophisticated algorithms to "learn" to distinguish the correct results from the incorrect results.

magic methods Special methods that perform the work of an operator.

`main` **namespace** A namespace created when Python is started that is initially empty.

mangling A mechanism Python uses with names that begin with double underscores to hide those names from code outside the class.

mean A measure of central tendency in which the sum of values in a data collection is divided by the number of items; also known as the average.

median A measure of central tendency found by locating the middle item or items in a collection that is in numeric order.

megapixels Measure of image resolution; 1 million pixels.

metadata Descriptive information about the data in a file.

methods What an object can do; the functions defined within a class.

mode The most frequently occurring value in a data set; a measure of central tendency.

modules Prewritten Python code that implements an abstraction designed to make programming easier.

multithreaded program A program that can have parallel code sequences happening at the same time.

multithreading Launching a new thread and calling a method inside the new thread, which runs in parallel with the old thread.

mutable collections Capable of being modified.

mutator methods Modify an object in some way but do not usually return a value to the caller. Also called setters because they set the values of the instance variable.

N

namespaces Collections of all names that are available to be accessed at a particular time during the execution of a Python program.

negative image A color-reversed image.

nested iteration Placing an iteration inside of another iteration.

nested selection The practice of placing one `if` statement inside another `if` statement.

newline Special character represented as `\n`; terminates a line of text in a file.

null object An object with no value, represented by the keyword `None`.

O

object An instance of a class containing data. Methods of the class can manipulate the object data.

object-oriented drawing programs Drawing programs that allow users to choose from a palette of available shapes and change a shape's attributes once placed.

object-oriented programming Using objects in the design of a program.

object state The current values of the data of an object.

open a file Obtain a file reference for reading or writing a file.

outliers Data points in a set that are clearly different from the rest.

P

parallel lists Multiple lists where the same indexes refer to different aspects of the same data.

parameter passing The process by which a function's formal parameter receives an actual parameter value.

parameters Tell a function specifically what it should do.

parent class A general class; also called a superclass.

pattern matching Checking to see if a set of characters exists in a text.

Pearson correlation coefficient A formula that produces a number between −1 and 1 to measure how two variables are related.

pixels The smallest amounts of picture information that can be referenced.

plaintext Messages that are readable.

polymorphism A powerful feature that allows a method to operate on any object in a hierarchy.

positional parameters The values given in the order listed in a function's header.

predator–prey relationships Relationships between two or more life-forms that must coexist and depend on one another in some fashion, which suggests that one life-form hunts the other to survive.

priming statement A statement before a `while` loop that sets the condition to `True`.

production rules Rules that specify how one symbol can be replaced by one or more other symbols; made up of a left side that specifies a symbol and a right side that specifies the symbols that can replace the symbol on the left side.

program Set of step-by-step instructions to be performed by a computer.

programming languages Set of operators, keywords, and structures used to create a program.

prompt Helpful text describing what is needed as the user is asked for input.

proper prefixes of a string All the substrings that start with the first character of the string and are shorter than the original string.

pseudorandom numbers Numbers that appear to be random but in reality are not random.

Python A computer programming language.

Q

queue A data structure that allows a user to remove the first things on a to-do list and add things at the end; enforces a first-come, first-served strategy for handling events.

R

rail fence cipher Encryption algorithm that results in a string with the characters shuffled to new positions. Also known as a transposition cipher.

ranges Sequence collections created using the `range` built-in function.

readability An attribute of code that makes the logic easy to see and maintain.

recursion A common programming technique where a function calls itself.

recursive process In computer science and mathematics, a program that uses recursion to solve a problem.

recursive step Simplifies a problem in a way that moves it closer to the base case.

reference A name used to locate an object.

register To tell the event handler which callback function should be called when a specific event occurs, such as a mouse click or key press.

regular expressions Special characters used to check text for patterns.

relational expressions Statements that compare two data values using a relational or equality operator, yielding a result of either `True` or `False`.

relational operators Operators used to test how two values compare.

repetition A method of defining statements so they can be repeated some number of times. A `for` loop is an example of repetition.

representation A visual model.

resizing an image Increasing or decreasing the dimensions (width and height) of an image.

resolution The amount of detail in an image measured by the number of pixels in the image.

RGB color model Defining colors as a combination of red, blue, and green.

row-major order Mapping method that stores all values on the first row, one after the other, and then stores all the values on the second row, then the third row, and so on.

S

selection A technique a program uses to make decisions.

selection statement Also known as an `if` statement; contains a condition and groups of statements that may or may not be executed, depending on whether the condition evaluates to `True` or `False`.

sentinel value A special value at which point repeated operations stop.

sequence A collection of objects stored in some predefined order; strings, lists, tuples, and range objects are examples of sequences.

sequence object An object that allows the storage of multiple values.

Series `Pandas` data type that contains a row or column from a DataFrame.

setter methods Mutator methods that involve changes to one or more instance variables and that usually include the word "set."

short-circuit evaluation Python evaluates only as much of an expression, from left to right, as necessary to determine whether the expression is `True` or `False`.

Sierpinski triangle Self-similar fractal that draws triangles.

simplification A strategy that reduces a problem to a trivial size.

simulation Attempting to recreate real-life interactions in software.

single-threaded Capable of only executing one thread.

slice operator Operator used to extract multiple characters from a string.

Sobel operators Developed by Irwin Sobel for use in edge detection.

software model A set of classes and methods that implement a simulation.

sorting Placing the values in a list of data in order.

standard deviation A statistical measure of dispersion that shows how much the data items of a collection differ from the mean.

statement An executable line of Python code.

static method A method independent of objects; useful when a method belongs inside a class but does not apply to any instance of the class.

static variable A variable that is shared by all instances of a class and is available to all the methods in the class.

stepwise refinement A software engineering technique in which a program is developed in a series of steps.

string Python sequential data type for handling characters or text rather than numbers.

string concatenation operator Operator used to convert two strings into one string; it works only on strings.

string iterator A `for` loop in which the loop variable takes on the value of each character in the string.

subclass A class that inherits from a superclass.

substitution cipher A cipher that substitutes one letter for another throughout a message.

substrings Parts of a string.

superclass The class from which a subclass inherits data and methods.

T

tail nesting A common pattern for nested selection in which all of the nested selection occurs in the `else` of the previous `if-else` statement.

text file A file containing characters.

thread An executable piece of code.

top-down design Similar to decomposition, a problem-solving technique where a large problem is solved by solving smaller problems.

transposition cipher Encryption algorithm that results in a string with the characters shuffled to new positions.

tuples Immutable heterogeneous sequences of data, written as comma-delimited values enclosed in parentheses.

types Classifications of objects.

U

Unicode encoding system System used for assigning a value to each character in a string.

Unified Modeling Language (UML) A graphical tool for communicating relationships among classes.

Uniform Resource Locator (URL) The address of a web resource, such as a web page or a downloadable item.

user-friendly Characteristic of a program that is easy to use.

V

variables Named references to data objects; names that allow us to locate Python objects.

visualization The process of interpreting data in visual terms or of putting it into visible form.

W

`while` **loop** A loop that executes until an event occurs.

whitespace Spaces, tabs, and line feeds.

wildcard Character that specifies a code block to execute if the subject does not equal any of the `case` values in a `match` statement.



© Yuri Hoyda/Shutterstock.

Index

A

Abstract Base Class (ABC), 368
abstract class, 361, 376, 427–428
abstraction, 19–28, 31
  turtle module, 19–24
  writing own functions, 24–28
abstract method, 362, 427–428
@abstractmethod, 362
acceleration, 323–324
accessor methods, 304–307
accumulator pattern, 61–65, 87, 89, 92, 95, 96, 97, 105, 132, 140, 291, 299
accumulator variable, 62, 63
actual parameters, 197
__add__, 299, 300, 302, 310
addEvent, 384
addition operator, 414
addPlanet, 316
addThing method, 338, 343
algorithm, 1, 297
_allTurtles, 397
alphabet, 84, 85
alphabetString, 91
American Standard Code for Information Interchange (ASCII), 82
and, 424, 425
AnimatedTurtle, 394–397
append, 127, 132, 316
applications, of computer science, 2–3
applyProduction, 291
approximation, 10, 11, 32
architecture programs, 353
arithmetic operators, 8, 14, 414

artificial intelligence (AI), 39, 40, 54, 239
ASCII. *See* American Standard Code for Information Interchange (ASCII)
`ascii_lowercase`, 83, 84
assignment operator, 103
assignment statement, 13–16, 202, 203
`aString.find(ch)`, 91
attributes, 19
average, 121, 190, 231
axiom, 288

B

Babbage, Charles, 239
backslash continuation character, 89
bar chart, 136
base case, 277, 279, 280, 281
base class, 368
base index, 216
`__bases__`, 367
`Bear`, 344–346
Berners-Lee, Tim, 2
binary search, 55
binary selection, 425
black box, 19
block, 24, 25, 28
`Bomb`, 405–406
`bool`, 298, 299
Boolean expression, 42–44, 169, 424–425
bouncing turtles, 393–398
boundary values, 85
`BoundedTurtle`, 401–402
brute force, 99, 239, 242–243
built-in collections, 108
built-in function, 299, 414–415
  `input`, 57–58
  `zip`, 269, 270
`__builtins__`, 199, 200, 201
`builtins` namespace, 198, 199, 200, 204
`bye`, 441

C

Caesar cipher, 99
callback function, 381
call by assignment, 198, 389
calltips, 24
call tree, 285
  for Sierpinski triangle, 285

camelCase, 18
Cantor dust, 290
Canvas, 358, 360–361, 371, 372, 373
capture group, 267
case, 293
case block, 289
center, 79, 80
central tendency, 121–133
centroid, 225
character, 73, 82–85
character-by-character approach, 102
chatbot, 40–41, 51–54
chatbot2, 53, 54
chatbot function, 40, 52
checkWord, 265, 266, 267, 268
child class, 356
chr, 82
cImage module, 180–186, 317, 411
  EmptyImage objects, 182, 185–186
  FileImage objects, 182–185
  ImageWin object, 181–182
  Pixel object, 180–181
ciphertext, 85, 102
  frequency analysis, 251–260
cipherText, 87
circle, 321
circle, drawing, 31–35
circumference, 32, 34
class, 298
_ _class_ _, 314, 368
class definition object, 314
clear, 115, 129, 420
clearscreen, 441
close files, 146
close method, 146
cluster analysis, 221, 222–223
  example of, 222–232
  implementation, 233–237
  K-means algorithm, 226
  shortcomings and solutions, 237
  on simple data, 222–232
clusterAnalysis, 232, 233
clusters, 221
  and centroids, 225
collection, 108, 145
collection classes, 299
color intensity, 180
column, 184, 211–213

column-major order, 244
comma-delimited, 109, 126
comma-separated values (CSV), 155, 156, 166–168, 427
comment, 8
common error messages, 433–436
complex number, 11
composition, 355, 390
compound Boolean expressions, 44–45, 424–425
computer-aided design (CAD) programs, 353
computer science, 1–3, 122, 297
  applications of, 2–3
computer scientist, 1, 2
computer simulation, 331. *See also* simulation
concatenation, 74
concrete class, 363
condition, 46, 228
constructor, 22, 301–304
`contains` method, 80
`continue` statement, 60
convolution, 215
`convolve`, 216
coordinate, 23, 176, 224, 231, 319–321, 334, 336, 340, 341, 343
Copernicus, 297
copy list, 124
`copy method`, 115
copy string, 77, 88
correlation, 175–177
correlation coefficient, 175
counter, 63
`countList`, 279
`count` method, 79, 80, 115
`countPassed`, 63
`countTests`, 63
`createCentroids`, 235
`createClusters`, 229, 230, 235
`createWordList`, 241
cryptanalysis, 239–273
  rail fence cipher, 240–245
  substitution cipher, 245–273
cryptography, 73
Cryptoquip, 90, 99
`ctLetters`, 269
`current`, 135–136
current character, 86, 87, 97, 103
current namespace, 201, 203, 204, 314

D

data, 107–108. *See also specific data*
  correlation, 175–177
  reading/manipulating with `pandas`, 165–171
  statistics calculation, 118–121
  storage, 108–118
DataFrame, 165, 166, 168, 169
data mining, 221
data model, 298
data reading, from Internet, 155–165
  CSV files, 155–157
  JSON, 171–175
  list comprehension, 160–165
  `while` loop, 157–160
data science, 3
decomposition, 96
decorator, 362
decryption, 85, 88–90
  rail fence algorithm, 243–245
`def`, 24
define, 24
delimiter, 116
`del` statement, 111, 420
`delThing` method, 339, 342
denominator, 324
dereferencing, 21, 314, 369
design, `Planet` class, 300–315
`_ _dict_ _`, 368
dictionary, 126, 240–242
  to compute frequency table, 133–134
  frequency distribution, 133–139
  frequency table without, 134–136
  keys, 126
  methods, 129–130, 419–420
  Python, 126–131
  `setdefault` method, 255
  view object, 126
digital image, 179
digital image processing, 179–186
Dijkstra, Edsger, 1
`dir` function, 200, 201
dispersion, 119
  standard deviation, 139–142
`distance`, 397, 406
division operator, 8, 10, 12
`dogWeight`, 42
`done`, 441
dot operator, 21, 51
`doubleImage`, 207, 208

drawAll method, 372, 374
drawCircle, 34
drawLS, 289, 293
_draw method, 360, 361, 362, 364, 370, 372, 374, 376, 377
drawOctagon, 31
drawPolygon, 32, 33, 34
drawSpiral, 30
drawSquare, 25–28, 31, 277
drawTriangle, 31, 284
Drone class, 402–405
dunder methods, 308

E

earthquake, 107
  cluster analysis, 233–237
  correlating data, 175, 177
  CSV data, 155–156
  JSON data, 171–175
  list comprehensions, 162–164
  reading and manipulating with pandas, 166–169
  statistics on, 118–121
  while loop, 157–160
edgeDetect, 217, 218
edge detection, 214–218
elif statement, 425
else statement, 47, 49, 51, 60
EmptyImage objects, 182, 185–186
empty list, 109
encapsulation, 303
encryption, 85–88, 91
encryptVigenere, 103, 104
enlarged image, 206, 208, 209
equality operators, 42
error messages, 433–436
Etch, 390
euclidD, 224, 227, 230, 233
Euclidean distance, 223, 225
evaluation, 9
evenChars, 87, 89
event-driven program, 381–382
  turtle module, 388–398
EventHandler, 385–386
_ _eventKeeper, 382, 383
event loop, 381, 382
  multithreaded, 385–388
  simulation, 382–384
exitonclick, 441

experiment, 107
exponentiation operator, 9, 414
expression, 8
  mathematical, 6, 13
  Python, 8, 10, 14, 15, 17, 23, 29
  regular, 264, 272–273, 428–429
extends, 368

F

factorial, 91, 280
`False`, 44, 46, 47
Fibonacci production rules, 291
Fibonacci sequence, 288
file, 420–421
  close, 146, 421
  iterating over lines, 147–148
  open, 146, 421
  pointer, 153
  read, 153–155, 421
  text, 145–147
  write, 149
`FileImage` objects, 182–185
`fileRef`, 146
filtering, 161, 214
`findall` function, 268, 269
`findLetters`, 268, 270, 271
`find` method, 79, 80, 84, 91
`Fish`, 339–344
flip axis, 211
flipping, 211–213
`flcat`, 12, 41, 57, 298
floating-point division, 414
floating-point numbers, 10–11
force, 324, 325. *See also* brute force
`for` loop, 28, 29, 30, 62, 86, 95, 147, 148, 153, 422
formal parameters, 197, 198, 202, 203, 204
formatted string, 150–153, 421
formatting output, 421–422
fractals, 275–296
`freeze`, 319
`freezeWorld` method, 339
frequency analysis, 251
  ciphertext, 251–260
  word, 260–263
frequency distribution, 133–139
frequency of letters, 246–251, 259
frequency table

using dictionary, 133–134
without dictionary, 134–136
frequencyTable, 134, 162
frequencyTableAlt, 135, 136
from math import, 201
function, 12, 19, 23–27, 202, 208, 211

G

games
interactive, 381–407
Nim, 65–70
playing by rules, 39–71
video, 54, 381–407
generalization, 4
generalTransform, 196
genKeyFromPass, 98, 99
GeoJSON, 171
GecmetricObject, 356, 358, 359, 360, 361–363, 371, 374, 376
get, 129, 419
getDensity, 305
getFreq, 250
_ _getitem_ _, 300
getMass, 315
getMax, 119, 120
getName, 304, 307, 312, 314
getPixel method, 184, 189
getRange, 119, 120
getter methods, 304
getVolume, 305
getWidth, 369
getXPos, 321
getYPos, 321
Global Positioning System (GPS), 2
Go Fish, 54
grammars, 285–295
gravitational constant, 325, 326
gravity, 324–325
grayPixel, 190
grayscale, 189–192
grcups function, 267
_ _gt_ _, 310, 311
Guess a Number game, 54–55, 57, 58–59, 60, 64, 65

H

HAS-A relationship, 355, 356, 357
height, 319

help command, 66
heterogeneous collections, 110
hideturtle method, 360
hierarchy, 356
  inheritance, 356, 359, 361, 367, 368, 376, 378
  of shapes, 353–378
histogram, 136, 137, 138, 258
homogeneous collections, 109
hypotenuse, 197
hypotenuse, 198, 199, 200, 202, 204, 205

I

IDLE, 5
Idle, Eric, 5
if-elif-else statement, 49, 50, 51, 53, 84, 289
if-else statement, 46, 47, 49, 59
if statement, 46, 48, 49, 84, 97, 161, 264, 433
image processing
  advanced, 205–218
  basic, 186–196
  digital, 179–186
  flipping, 211–213
  functions and parameter passing, 196–205
ImageWin object, 181–182
immutable collections, 112
import, 19, 201
import math, 425
in, 78, 419
indexing, 75
index method, 79, 80, 115, 420
index operator, 75, 126
indexToLetter, 84, 85, 102
infinite loop, 56, 228
information, 25, 107, 184
information hiding, 304
inheritance, 350–351, 355, 367–369, 390, 391
inheritance hierarchy, 356, 359, 361, 367, 368, 376, 378
_ _init_ _, 301, 302, 314, 315, 372, 382, 390, 392, 397
initialization, 61, 228
in-place sort, 250
input, 57–58
installation
  cImage, 411
  Jupyter Notebook IDE, 409–410
  Matplotlib libraries, 410
  pandas libraries, 410
  Python, 409, 411

Python Image Library (PIL), 411
instance, 22, 298
instance data, 298, 300
instance variable, 301, 303, 304, 355, 357, 358, 359, 360, 361, 362, 363, 364, 367, 368, 370, 371, 372, 376
`int`, 12, 57, 298
integer, 6–10
integer division, 8, 10, 211, 414
Intel, 3
intensity, 180, 187, 189, 190, 214, 215, 216, 217
interactive games, 381–407
interpreter, 5, 7, 8, 14, 23, 24
`ints`, 41
invocation, 198, 205
IS-A relationship, 355, 356
`isinstance`, 341, 342, 361
`items`, 126, 129, 249, 419
iteration, 28, 126, 422–424
list comprehensions, 423–424
over collections of data, 422–423
`while` loop, 423
iterator, 156

J

JavaScript Object Notation (JSON), 155, 171–175, 427
`join`, 243
`jscn` module, 173
Jupyter Notebook, 5, 6, 24, 25, 28, 136, 156, 165, 168, 409–410

K

kernel, 214
key(s), 90, 126, 129, 249, 419
creation, 93–99
dictionary, 126
substitution cipher, 90–91
Vigenère cipher, 100–104
`keyGen`, 95, 272
key–value pairs, 126, 249
keyword parameter, 103
keywords, 17
K-means algorithm, 226–232
Koch snowflake curve, 286, 287

L

`LaserCannon`, 400–401
latitude, 233

left-hand side, 14, 202, 203
`len`, 75, 77, 121, 253
length, 75
letter frequency, 246–251, 259
`letterFrequency`, 248
letter pair analysis, 251–260
`letterToIndex`, 84, 85, 102
libraries, 5
  `Matplotlib`, 410
  `pandas`, 410, 432–433
  Python Image Library (PIL), 411
Lindenmayer, Aristid, 287
`Line`, 358, 359, 361, 364–365, 368
`_ _lineColcr`, 358, 359, 362, 364, 368, 370, 372
`_ _lineWidth`, 358, 359, 362, 364, 368, 370
list, 109–118, 134, 417–418
list comprehension, 160–165, 404–405, 423–424
`listen`, 390, 441
listings, 6
`list` method, 114–116, 231, 249, 299, 417
list of lists, 215, 335
`liveALittle` method, 339, 340, 341, 342, 345, 346, 347, 349
`ljust` method, 79, 80
`loads` function, 173
local namespace, 202, 203, 204, 205, 312, 313, 314
local variables, 202
logic, 3, 158
logical operators, 44–45
longitude, 233
loop body, 28
loop update, 58, 60, 158, 160
loop variable, 29, 30
`lower` method, 52, 53, 79, 80
`lSystem`, 294
L-system, 287–290, 292–295
`_ _lt_ _`, 310, 311
lunar eclipse, 297

M

machine learning, 3
magic methods, 308
`mainloop`, 390, 441
`main` namespace, 200, 201, 312, 314
`mainSimulation` function, 346–349
`makeDataList` function, 160, 161, 162
`makeGrayScale`, 192
`makeNegative`, 192

mangling, 394
mapper.py, 102
mapping, 91, 92, 99, 100, 192, 195, 206, 207, 208, 209, 210, 211, 213, 216
mask, 214
mass, 324–325
match, 264, 287–290, 293, 425
math, 201, 204, 205, 305, 317, 425–426
mathematical expressions, 6, 13
matplotlib, 258, 410, 431–432
matplotlib.pyplot, 138
max, 118, 119, 132
maybeAdd, 253, 254, 256
mean, 121–122, 139
mean, 169, 426
median, 122–125
median, 124, 426
medical imaging, 2
megapixels, 179
|(merge), 420
metadata, 155, 157, 172
method(s), 19, 298, 357
  accessor, 304–307
  constructor, 301–304
  dictionary, 129–130, 419–420
  dunder, 308
  edge detection, 214–218
  file-reading, 153–155
  getter, 304
  integer addition and indexing, 300
  magic, 308
  mutator, 307–308, 340
  setter, 307
  special, 308–312
  of statistics module, 118
  string, 79–82
  of turtle, 19–24
midPoint, 284
min, 118, 119, 231
minmax pattern, 243
mirroring, 213
mixing numeric types, 11–13
mode, 125–133
mode, 426
module, 19, 201, 202, 425–428
  and namespaces, 204–205
modulo, 414
modulo arithmetic, 102, 103
modulo operator, 8, 86, 102, 124

movePlanets, 327
msg variable, 57
multimode, 426
multithreaded event loop, 385–388
multithreaded execution, 385
multithreaded program, 385
multithreading, 385, 428
mutable collections, 112
mutator methods, 307–308, 340
_ _myCanvas, 372
myList, 250
myList._ _getitem_ _, 300
myMatch.groups(), 269
myNeighbors, 253
myPlanet, 302, 303, 304, 312, 314
myString, 97

N

name, 13–18, 202–203, 300
  calling functions and finding, 202–204
name, 301, 438
_ _name_ _, 304, 307, 312
namespace, 190, 198–201
  builtins, 198, 199, 200, 204
  current, 201, 203, 204, 314
  local, 202, 203, 204, 205, 312, 313, 314
  main, 199, 200, 201, 312, 314
  modules and, 204–205
negativePixel, 187, 188, 189, 190
negatives, 187–189
neighborCount, 254, 255, 256, 258
nestedBox, 277, 278
nested iteration, 188, 189, 191, 196, 206, 209, 212, 217
nested selection, 49, 425
newline, 148, 153
newName, 307
newStr, 97
Newton, Sir Isaac, 319
next built-in function, 158
nextEvent, 383
Nim game, 65–70
None, 128, 319
not, 424, 425
not in, 78, 97, 419
nouns, 333
  characteristic properties for, 333–334
null object, 128

numAttempts, 64
numeric type, 11–13, 413–414
numTimePeriods, 329

O

object, 368
object(s), 6, 298
  characteristics, 19
  class definition, 314
  deleting, 430–431
  distance between, 323
  EmptyImage, 182, 185–186
  FileImage, 182–185
  hierarchy, 356
  ImageWin, 181–182
  naming, 13–18
  null, 128
  Pixel, 180–181
  Planet, 297–330
  sequence, 29
object-oriented drawing programs, 353
object-oriented programming, 298
object state, 304
oddChars, 87
OmniGraffle, 353
onclick, 388, 392, 393, 440–441
onkey, 388, 389, 441
ontimer, 388, 393, 394, 395, 441
open files, 146
operand, 8, 10, 12
operator, 10
  arithmetic, 8, 14, 414
  assignment, 103
  division, 8, 10, 12
  dot, 21, 51
  equality, 42
  exponentiation, 9, 414
  floating-point division, 414
  index, 75, 126
  integer division, 8, 10, 211, 414
  logical, 44–45
  modulo, 8, 86, 102, 124
  multiplication, 8–9, 414
  relational, 42, 424
  repetition, 74, 113
  sequence, 110, 111, 415–418
  slice, 76, 77, 89, 94, 112

string, 78, 79
string concatenation, 56
`or`, 424, 425
`ord`, 82
ordinal value, 100, 102
outliers, 237

P

palindrome, 228
`pandas`
data manipulation with, 168–171
data structures, 165–166
libraries, 410, 432–433
reading CSV data with, 166–168
parallel lists, 230
parameter(s), 19, 24
actual, 197
call by assignment, 198
formal, 197, 198, 202, 203, 204
keyword, 103
passing, 198
positional, 103
parameter passing, 196–205
parent class, 356, 368
partial words, pattern matching with, 264–272
`pass` statement, 362
password, 95–96
pattern matching, 264
with partial words, 264–272
patterns
accumulator, 61–65, 87, 89, 92, 95, 96, 97, 105, 132, 140, 291, 299
defined, 4
minmax, 243
spiral, 29–31
Pearson correlation coefficient, 175, 177
Persians, 73
pixel, 179
pixel mapper, 192–196
`pixelMapper`, 192, 195, 196, 217
`Pixel` method, 185, 187, 189
`Pixel` object, 180–181
plaintext, 85, 87
`Planet`, 297–330, 300–315, 321, 326, 327
accessor methods, 304–307
constructor method, 301–304
design, 300–315
methods, 312–314

mutator methods, 307–308
`self`, 312–314
special methods, 308–312
storage and lookup mechanism, 314–315
planetary orbits, 323–326
`Plant`, 349–350
`Point` class, 359, 363–364
polygon, 31, 32, 33, 34, 355, 356, 376–378
polymorphism, 361, 374
`pop`, 114
positional parameters, 103
precision, 11, 13
predator–prey relationships, 332
prediction, 107
`previous`, 135–136
prime meridian, 233
priming statement, 56, 158
primitive classes, 298
primitive elements, 6–13
complex numbers, 11
floating-point numbers, 10–11
integers, 6–10
mixing numeric types, 11–13
print, 7, 13, 14, 17
`print` function, 51, 52, 56, 103, 204, 308, 317
problem-solving, 61, 297, 298
strategies, 3–5
production rule, 287, 288, 289, 290–292
program, 1
programming language, 5, 46, 264, 297, 298
Project Gutenberg, 248
prompt, 57, 103
proper prefixes of a string, 77
pseudorandom numbers, 66
Ptolemy, 297
Pythagorean theorem, 224
Python, 5–6
abstraction, 19–28
arithmetic operators, 8
arithmetic operators in, 414
built-in functions, 414–415
common error messages, 433–436
dictionaries, 126–131
functions for converting between numeric types, 13
Image Library (PIL), 411
installation, 409, 411
modules, 425–428
naming objects, 13–18

numeric data types, 413–414
order of operations, 9
primitive elements, 6–13
read–eval–print loop in, 7
repetition, 28–35
reserved keywords, 18
reserved words, 413

Q

quadratic Koch island, 292
`quakeT`, 235
queue, 382

R

`radians`, 365, 439
radius, 15, 34
`railBreak`, 243
`railDecrypt`, 242, 243, 245
`railEncrypt`, 245
rail fence cipher, 86, 240–245
`rainfallInCM.txt`, 149
`rainfall.txt`, 146, 147
`randint`, 66, 67, 426
`random`, 66, 142, 426
random number, 66–67, 331, 426
random number generator, 228
`randrange`, 426
`range`, 31, 417
ranges, 114
raw data, 107
read, 153–175
`read`, 153, 154, 248
readability, 289
`readEarthquakeFile` function, 233
`reader` method, 156, 157
read–eval–print loop, 7, 13, 17, 23, 382, 387
`readFile`, 227
`readline`, 153, 156, 158, 228
real numbers, 10
recursion, 276
  classic functions, 278–280
  drawing tree, 280–281
  Sierpinski triangle, 282–285
  squares, 277–278
recursive process, 276, 277
recursive step, 277

reference, 14
reference diagram, 15–17, 27, 112, 368
register, 382
`registerCallback` method, 382
regular expression, 264, 272–273
  patterns, 428–429
relational expression, 42, 43
relational operator, 42, 424
remainder, 86, 124, 136
`re.match`, 264
`removeChar`, 94–95
`removeDupes`, 97, 98
`removeMatches`, 97, 98, 246, 251
`remove method`, 115
`removeNonletters`, 246, 247
`removeString`, 97
repetition, 28–35, 74–75, 110, 112, 113
  drawing a circle, 31–35
  drawing a spiral, 29–31
repetition operator, 74, 113
`replace` method, 80, 260
representation, 4
reserved words, 17, 413
`resetscreen`, 441
resizing, 206–207
resolution, 179
`re.sub` function, 266, 267
`return`, 67, 202
`return self._ _name`, 312
`reverse`, 115, 249
`reversed` method, 129
`rfind` method, 80
RGB color model, 179–180
Richter scale, 157
`right`, 31
right-hand side, 14–15
`rindex` method, 80
`rjust` method, 79, 80
rosemary, 294, 295
rotate, 213
row-major order, 244, 336
RSA algorithm, 239
`rules.get`, 291
`run`, 385, 386, 387, 428
running sum, 140, 231

S

`save` method, 186
scientific notation, 10–11
`scramble2Decrypt`, 89
`scramble2Encrypt`, 87
`screensize`, 441
security, 99
selection, 41, 46
selection statement, 46–51, 425
`self`, 301, 302, 303, 304, 305, 307, 310, 312–314, 314
`self.getVolume`, 305
`self.name`, 301
`self._ _name`, 312
`self._ _sTurtle`, 321
sentinel value, 55
sepia tone, 196
sequence, 29, 30, 73, 108, 110, 111, 113, 114, 148, 415
sequence object, 29
sequence operators, 110, 111, 415–418
  lists, 417–418
  string, 415–416
  tuples, 418
sequential collection, 109, 126
Series, 165
sessions, 6
`setdefault`, 255, 256, 419
`setName`, 307
`setPixel` method, 186, 189
setter method, 307, 374
`setworldcoordinates`, 319, 441
shapes, hierarchy of, 353–378
short-circuit evaluation, 44
`showPlanets`, 317
`sideLength`, 30
`sierpinski`, 282
Sierpinski triangle, 282–285
  call tree for, 285
simplification, 4, 102, 277
simulation, 319, 326–330, 331–351
single-threaded execution, 385
`sleep`, 387
slice, 76
slice operator, 76, 77, 89, 94
snowflakes, 285–295
Sobel, Irwin, 214
Sobel operators, 214
software model, 297
solar system
  animating, 319–330

designing and implementing, 316–319
SolarSystem, 316, 317, 318, 319, 320, 321, 327
sort, 115, 124, 249, 260
sortByLen, 260
sorting, 122, 124
Space Invaders, 398
special methods, 300, 308–312
spiral patterns, 29–31
split, 116, 117, 148, 243
sqrt function, 205
square, 25, 29, 31
recursion, 277–278
square bracket, 77, 109, 128
squareIt, 193, 194
_ _ssScreen, 319
_ _ssTurtle, 319
standardDev, 140, 141
standard deviation, 139–142
start, 58, 385, 386, 387, 428
startswith method, 51, 79, 80
state, 301
statement, 13
stateSaver, 293
static method, 403
@staticmethod, 403
static variable, 396
statistics, 107, 108, 118–121, 163, 174, 175, 426
statistics module, 118
stdev, 426
stepwise construction, 191
stepwise refinement, 65
sticky label, 15
storage
data, 108–118
and lookup mechanism, 314–315
str, 56, 57, 82, 299, 310
stretching, 208–211
string, 41, 73, 108, 415–416
printing numbers in, 56–57
revisited, 108–109
string concatenation operator, 56
string data type, 41–42, 51, 73–85
character functions, 82–85
concatenation, 74
indexing, 75
repetition, 74
string methods, 79–82
string searching, 78–79

string slicing, 76–77
string formatting, 150–153, 421
string iterator, 86
string methods, 79–82, 116, 243, 416
string operators, 78, 79
string searching, 78–79
string slicing, 76–77
`stringVariable`, 86
`__str__` method, 309, 310, 315, 317
subclass, 356
substitution cipher, 90–93, 99, 100, 245–273
ciphertext frequency analysis, 251–260
letter frequency, 246–251
letter pair analysis, 251–260
pattern matching with partial words, 264–272
regular expression, 272–273
word frequency analysis, 260–263
`substituticnDecrypt`, 93
`substituticnEncrypt`, 92, 93
substring, 76
subtraction operator, 414
`sum`, 121
sum of squares, 140
`Sun`, 315–316, 320, 321
`super`, 364
superclass, 356
symbol, 287, 288

T

tail nesting, 49
`test`, 194
testing
boundary values, 85
helper functions, 84
`removeChar` function, 94
`removeDupes` and `remcveMatches`, 97, 98
text file, 145–147
`__theSun`, 319
thread, 385
`Thread`, 385–386
`time`, 387
`time.sleep` method, 386
TKinter, 371
top-down design, 98
transposition cipher, 85–90
decryption, 88–90
encryption, 85–88

`tree`, 280, 281
`True`, 44, 46, 47, 298
truncation, 12
`tryToBreed` method, 342, 343, 345
`tryToMove` method, 343, 346
tuples, 126–131, 130, 418
Turing, Alan, 239
`turtle.mainloop`, 390
`turtle` module, 19–24, 51, 298, 302, 317, 319–323, 334, 339, 340, 358, 360, 365, 388–398
  event-driven program, 388–398
`TurtlePlace`, 392, 393, 395, 396
`turtle` reference, 437–441
  basic move and draw, 437
  color, 440
  controlling shape and appearance, 438–439
  drawing control, 438
  drawing speed, 440
  drawing state, 438
  events, 440–441
  filling, 438
  measurement settings, 439
type, 6
`type`, 299, 303
type conversion, 13

U

unary selection, 425
Unicode, 82
Unicode encoding system, 44
Unified Modeling Language (UML), 355
Uniform Resource Locator (URL), 173, 427
universal gravitation, 325
`upper` method, 53, 79, 80
`urllib.request` module, 173, 427
`urlopen` function, 173
user-friendly, 52

V

value(s), 19, 128, 172, 173
  boundary, 85
  correlational, 176
  dictionary, 126, 255
  index, 75, 77, 94, 216
  instance variables, 303
  maximum, 120
  ordinal, 100, 102

returning, 23, 250
returning from function, 67–70
sentinel, 55
`values` method, 129, 132, 419
variable, 14
vector, 323
velocity, 323, 324, 326
Verily Life Sciences, 3
`verticalFlip`, 212, 213
video game, 54, 381–407
Vigenère cipher, 100–104, 239
`vigenereIndex`, 102
Vigenère square, 100–102
`_ _visible`, 372
`_ _visibleObjects`, 372, 374
Visio, 353
visualization, 235
`Planet` class with, 321
`Sun` class with, 320–321
`visualizeQuakes`, 237

W

The War of the Worlds, 248, 250, 261
web page, 173
Wells, H. G., 248
`while` loop, 55–56, 58, 59, 60, 63, 157–160, 228, 229, 343, 397, 423
`whitePixel` method, 186
whitespace, 116
`width`, 319
wildcard, 264, 289
word frequency analysis, 260–263
`wordlist.txt`, 241
`World`, 334–339, 349
`write` method, 149

X

Xerxes, 73

Y

Yellowstone National Park, 331

Z

`zip`, 269, 270, 418

A row of circles representing the addition of 1, 2, 3, and 4. It consists of four rows and four columns resembling a lower triangular matrix. Row 1 has a circle. Row 2 has two circles. Row 3 has three circles. Row 4 has four circles.

Back to Figure

A row of circles represents the addition of numbers. It consists of four rows and five columns resembling a rectangle. Each row has five circles. It represents a lower triangular matrix and an upper triangular matrix.

Back to Figure

The window displays the following program code. Input 1: 2, plus, 2. Output 1: 4. Input 2: 100, minus, 75. Output 2: 25. Input 3: 7, asterisk, 9. Output 3: 63. Input 4: 14, forward slash, forward slash, 2. Output 4: 7. Input 5: 15, forward slash, forward slash, 2, hash, integer division truncates the remainder. Output 5: 7. Input 6: 15, percentage, 2, hash, to get remainder, use modulo operator. Output 6: 1.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 12, asterisk, 6, plus, 2, hash, multiply, then add. Output 1: 74. Input 2: 2, plus, 12, asterisk, 6, hash, multiply, then add. Output 2: 74. Input 3: left parenthesis, 2, plus, 12, right parenthesis, asterisk, 6, hash, add then, multiply. Output 3: 84. Input 4: 12, asterisk, left parenthesis, 6, plus, 2, right parenthesis, hash, add, then multiply.

Output 4: 96. Input 5: 2, asterisk, asterisk, 3, hash, 2 to the third power. Output 5: 8.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 2.0, plus, 2.0. Output 1: 4.0. Input 2: 15.3, forward slash, 2.5. Output 2: 6.12. Input 3: 6.0, asterisk, 1.5. Output 3: 9.0. Input 4, 2.5, asterisk, asterisk, 25.0. Output 4: 8881784197.001253. Input 5: 2.0, asterisk, asterisk, 500.0. Output 5: 3.273390607896142 e plus 150. Input 6: 1.33 e plus 5, plus 1.0. Output 6: 133001.0.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, 2, asterisk, asterisk, 500. Output 1: 327339060789614187001318969682759915221664204604306478948329136809613379640467455488327009232590415715088668412756007100921725654588539305332852758937 6. Input 2: 1, 100000000, asterisk, 500, hash, 100 million, asterisk, 500. Output 2: 50000000000. Input 3: 1, 100, underscore, 000, underscore, 000, asterisk, 500, hash, underscore, can be a separator for large numbers. Output 3: 50000000000.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, 100, asterisk, 3.4, hash, mix of integers and float, implies, float result. Output 1: 340.0. Input 2: 1, 100, plus, 3.4. Output 2: 103.4. Input 3: 1, 100, minus, 3.4. Output 3: 96.6. Input 4: 1, 100, asterisk, asterisk,

3.4. Output 4: 6309573.44480193. Input 5: 1, 1000, forward slash, 10.2, hash, division is a special case. Output 5: 98.03921568627452. Input 6: 1, 1000, forward slash, forward slash, 10.2. Out, left square bracket, 6, right square bracket, colon 98.0.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: float, left parenthesis, 5, right parenthesis. Output 1: 5.0. Input 2: int, left parenthesis, 3.99999, right parenthesis. Output 2: 3. Input 4: round, left parenthesis, 3.99999, right parenthesis. Output 4: 4.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: pi equals, 3.14159. Input 2: radius, equals, 8.0. Input 3: height, equals 16. Input 4: base Area equals pi, asterisk, radius, asterisk, asterisk, 2. Input 5: base Area. Output 5: 201.06176. Input 6: cylinder Volume, equals, base Area asterisk height. Input 7: cylinder Volume. Output 7: 3216.98816.

Back to Session

The diagram consists of two columns separated by a dashed vertical line. The left column is labeled Name space and the right is labeled Object space. The namespaces point to the respective object spaces. Namespace reads pi, radius, height, base Area, and cylinder Volume. Pi points to 3.14159 under the object space. Radius points to 8.0 under the object space. Height points to 16 under the object space. Base Area points to 201.06176 under the

object space. Cylinder Volume points to 3216.98816 under the object space.

Back to Figure

The diagram consists of two columns separated by a dashed vertical line. The left column is labeled Name space and the right is labeled Object space. The namespaces point to the respective object spaces. Namespace reads pi, radius, height, base Area, and cylinder Volume. Pi points to 3.14159 under the object space. Radius points to 8.0 under the object space. Height points to 16 under the object space. Base Area points to 201.06176 under the object space. Cylinder Volume points to 3216.98816 under the object space. A label x is added between pi and radius under the Namespace and it points to 8.0 under the object space.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 2: 1, a, equals 10. Input 3: 1, b, equals 20. Input 4: 1, a, equals b. Input 5: 1, a. Output 5: 20. Input 6: 1, b. Output 6: 20.

Back to Session

The diagram consists of two columns separated by a dashed vertical line. The left column is labeled Name space and the right is labeled Object space. The namespaces point to the respective object spaces. Namespace reads a and b. Object space reads 10 and 20. The names a and b under the namespace point to the integer 20 under the object space.

Back to Figure

The table lists the following text. And, and, assert, async, await, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield.

Back to Table

The window displays the following program code. In the code, the words in the variables are merged. Input 1: import turtle. Input 2: turtle. Output 2: left angular bracket, module, left single quotation mark, turtle, right single quotation mark, from, left single quotation mark, C, colon, backslash, backslash, Users, backslash, backslash, Julie, backslash, backslash, App Data, backslash, backslash, Local, backslash, backslash, Programs, backslash, backslash, Python, backslash, backslash, Python 310, backslash, backslash, lib, backslash, backslash, turtle, dot, p y, right single quotation mark, right angular bracket. Input 3: gertrude equals turtle, dot, Turtle, left parenthesis, right parenthesis. Input 4: Gertrude. Output 4: left angular bracket, turtle, dot, Turtle at 0x17dbf297730, right angular bracket. Input 5: Gertrude, dot, forward, left parenthesis, 100, right parenthesis. Input 6: Gertrude, dot, right, left parenthesis, 90, right parenthesis. Input 7: Gertrude, dot, forward, left parenthesis, 50, right parenthesis. Input 8: Gertrude, dot, position, left parenthesis, right parenthesis. Output 8: left parenthesis, 100.00, negative 50.00, right parenthesis. Input 9: Gertrude, dot, heading, left parenthesis, right parenthesis. Output 9: 270.0.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def function Name, left parenthesis, param 1,

comma, param 2, comma, ellipses, right parenthesis, colon. Line 2, indented once: statement 1. Line 3, indented once: statement 2. Line 4, indented once: ellipses.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def draw Square, left parenthesis, my Turtle, comma, side Length, right parenthesis, colon. Line 2, indented once: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis, hash, side 1. Line 3, indented once: my Turtle, dot, right, left parenthesis, 90, right parenthesis. Line 4, indented once: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis, hash, side 2. Line 5, indented once: my Turtle, dot, right, left parenthesis, 90, right parenthesis. Line 6, indented once: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis, hash, side 3. Line 7, indented once: my Turtle, dot, right, left parenthesis, 90, right parenthesis. Line 8, indented once: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis, hash, side 4. Line 9, indented once: my Turtle, dot, right, left parenthesis, 90, right parenthesis.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from d s import asterisk. Input 2: import turtle. Input 3: t, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Input 4: draw Square, left parenthesis, lowercase t, comma, 150, right parenthesis.

Back to Session

The diagram consists of two columns separated by a dashed vertical line. The left column is labeled Name space and the right is labeled Object space. The namespaces point to the respective object spaces. Namespace reads three names such as turtle, t, and draw square. The turtle points to the module under the object space. The turtle inside the module points to the function. The name t under the namespace points to the turtle object under the object space. The draw square points to the function module under the object space. The function module has two additional names such as my Turtle and side length with my Turtle pointing to the turtle object and side length pointing to 150.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: for, lowercase i, in range, left parenthesis, lowercase n, right parenthesis, colon. Line 2, indented once: statement 1. Line 3, indented once: statement 2. Line 4, indented once: ellipses.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def draw Square, left parenthesis, my Turtle, comma, side Length, right parenthesis, colon. Line 2, indented once: for, lowercase I, in range, left parenthesis, 4, right parenthesis, colon. Line 3, indented twice: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis. Line 4, indented twice: my Turtle, dot, right, left parenthesis, 90, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def draw Spiral, left parenthesis, my Turtle, comma, max Side, right parenthesis, colon. Line 2, indented once: for, side Length in range, left parenthesis, 1, comma, max Side, plus, 1, comma, 5, right parenthesis, colon. Line 3, indented twice: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis. Line 4, indented twice: my Turtle, dot, right, left parenthesis, 90, right parenthesis.

Back to Listing

A horizontal stack with twelve cells reads the following which are labeled, sequence produced by range (0, 60, 5). 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55. The fourth item 15 is labeled, for items in range (0, 60, 5) and the fourth time through the loop.

Back to Figure

In the first iteration, the turtle draws a horizontal line that is of side length 1 unit long, followed by a vertical line of side length of 6 units long in the second iteration. From that, the turtle turns horizontally left and draws a line that is of side length 11 units long in the third iteration. In the fourth iteration, the turtle continues vertically upwards and draws a line of side length of 16 units, thus resembling a spiral.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def draw Triangle, left parenthesis, my Turtle, comma, side Length, right parenthesis, colon. Line 2, indented once: for, lowercase i, in range, left parenthesis, 3, right parenthesis, colon. Line 3, indented twice: my Turtle, dot,

forward, left parenthesis, side Length, right parenthesis. Line 4, indented twice: my Turtle, dot, right, left parenthesis, 120, right parenthesis.

Back to Listing

The table consists of four columns: Polygon, Sides, range left parenthesis right parenthesis, and Angle. Row entries are as follows. Row 1. triangle, 3, 3, 360 over 3 equals 120. Row 2. square, 4, 4, 360 over 4 equals 90. Row 3. pentagon, 5, 5, 360 over 5 equals 72. Row 4. octagon, 8, 8, 360 over 8 equals 45.

Back to Table

Program code. In the code, the words in the variables are merged. Line 1: def draw Polygon, left parenthesis, my Turtle, comma, side Length, num Sides, right parenthesis, colon. Line 2, indented once: turn Angle equals, 360, forward slash, num Sides. Line 3, indented once: for, lowercase i, in range, left parenthesis, num Sides, right parenthesis, colon. Line 4, indented twice: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis. Line 5, indented twice: my Turtle, dot, right, left parenthesis, turn Angle, right parenthesis.

Back to Listing

The functions are listed as follows: from d p import asterisk slash contains drawPolygon. import turtle. t = turtle dot Turtle left parenthesis right parenthesis. hash move turtle to left and face up. t dot up left parenthesis right parenthesis. t dot backward 200 within parenthesis. t dot left 90 within parenthesis. t dot down left parenthesis right parenthesis. hash call draw Polygon with different sizes and sides. draw Polygon left parenthesis t, 100, 4

right parenthesis. draw Polygon left parenthesis t, 100, 8 right parenthesis. draw Polygon left parenthesis t, 50, 20 right parenthesis. draw Polygon left parenthesis t, 20, 20 right parenthesis.

Back to Figure

The arrow in the middle of the screen represents the turtle and is moved to the left and make it face upward. A large circle is drawn from the turtle and resembles a polygon. A 10-sided polygon, a square, and a small circle are drawn inside it from the same point.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def draw Circle, left parenthesis, my Turtle, comma, radius, right parenthesis, colon. Line 2, indented once: circumference equals, 2, asterisk, 3.1415, asterisk, radius. Line 3, indented once: side Length equals, circumference, forward slash, 360. Line 4, indented once: draw Polygon, left parenthesis, my Turtle, comma, side Length, comma, 360, right parenthesis.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from d p import asterisk. Input 2: import turtle. Input 3: t equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Input 4: t, dot, up, left parenthesis, right parenthesis. Input 5: t, dot, backward, left parenthesis, 200, right parenthesis. Input 6: t, dot, left, left parenthesis, 90, right parenthesis. Input 6: t, dot, down, left parenthesis, right parenthesis. Input 8: draw Circle, left

parenthesis, t, comma, 20, right parenthesis. Input 9: draw Circle, left parenthesis, t, comma, 200, right parenthesis.

Back to Session

The arrow in the middle of the screen represents the turtle and is moved to the left and makes it face upward. A large circle and a small circle are drawn from the turtle.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from chatbot import, asterisk. Input 2: chatbot, left parenthesis, left double quotation mark, Who brings gifts for lost teeth, question mark, right double quotation mark, right parenthesis. The tooth fairy. Input 3: chatbot, left parenthesis, left double quotation mark, Where is the tooth fairy, question mark, right double quotation mark, right parenthesis. At the dentist's office. Input 4: chatbot, left parenthesis, left double quotation mark, What is the meaning of life, question mark, right double quotation mark, right parenthesis. To be kind to one another. Input 5: chatbot, left parenthesis, left double quotation mark, Why is the sky blue, question mark, right double quotation mark, right parenthesis. You'll understand when you're older. Input 6: chatbot, left parenthesis, left double quotation mark, Who is the best student in this class, question mark, right double quotation mark, right parenthesis. The tooth fairy. Input 7: chatbot, left parenthesis, left double quotation mark, Where am I, question mark, right double quotation mark, right parenthesis. At the dentist's office. Input 8: chatbot, left parenthesis, left double quotation mark, How did I

get here, question mark, right double quotation mark, right parenthesis. You stumped me on that, exclamatory mark.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: left double quotation mark, Hello, right double quotation mark, hash, use double quotes. Output 1: left single quotation mark, Hello, right single quotation mark. Input 2: left single quotation mark, world, right single quotation mark, hash, or single quotes. Output 2: left single quotation mark, world, right single quotation mark. Input 3: greeting equals, left double quotation mark, Hello world, right double quotation mark, hash, name a string. Input 4: greeting. Input 4: left single quotation mark, Hello world, right single quotation mark. Input 5: left double quotation mark, Let's go, right single quotation mark, hash, embed, single quotation mark, inside, double quotation mark. Output 5: left double quotation mark, Let's go, right single quotation mark. Input 6: left single quotation mark, She said, comma, left double quotation mark, How are you question mark, right double quotation mark, comma, then left, dot, right single quotation mark, hash, embed, double quotation mark, inside, single quotation mark. Output 6: left single quotation mark, She said, comma, left double quotation mark, How are you, question mark, right double quotation mark, comma, then left, dot, right single quotation mark.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: dog Weight equals

25, hash, assign 25 to dog Weight. Input 2: dog Weight. Output 2: 25. Input 3: dog Weight, equals, equals, 25, hash is dog Weight equal to 25, question mark. Output 3: True. Input 4: dog Weight, exclamatory equals, 25, hash is dog Weight not equal to 25, question mark. Output 4: False. Input 5: dog Weight less than 25, hash, is dog Weight less than 25, question mark. Output 5: False. Input 6: dog Weight less than, equals, 25, hash is dog Weight less than or equal to 25, question mark. Output 6: True. Line 12: Input 7: dog Weight greater than 25, hash, is dog Weight greater than 25, question mark. Line 13: Output 7: False. Line 14: Input 8: dog Weight greater than, equals, 25, hash is dog Weight greater than or equal to 25, question mark. Line 15: Output 8: True. Line 16: Input 9: dog Weight, equals, equals, 5, asterisk, 5, hash is dog Weight equal to result of calculation, question mark. Line 17: Output 9: True. Line 18: Input 10: cat Weight equals 18, hash, assign 18 to cat Weight. Line 19: Input 11: dog Weight, equals, equals, cat Weight, hash, is dog Weight equal to another variable, question mark. Line 20: Output 11: False. Line 21: Input 12: today equals, left double quotation mark, Monday, right double quotation mark, hash, assign, left double quotation mark, Monday, right double quotation mark, to today. Line 22: Input 13: today, equals, equals, left double quotation mark, Monday, right double quotation mark, hash, strings can also be compared. Line 23: Out, left square bracket, 13, right square bracket, colon, True. Line 24: In, left square bracket, 14, right square bracket, colon, today, equals, equals, left double quotation mark, monday, right double quotation mark, hash, comparing strings is case sensitive. Line 25: Out, left square bracket, 14, right square bracket, colon, False.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 6 less than 10 and 3 less than 7. Output 1: True. Input 2: 6 less than 10 and 10 less than 6. Output 2: False. Input 3: 4 exclamatory mark 4 or 5 less than 8. Output 3: True. Input 4: 4 exclamatory mark, equals, 4 or 8 less than 5. Output 4: False. Input 5: not 6 less than 10. Output 5: False. Input 6: not 6 greater than 10. Line 12: Output 6: True.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: if, left angular bracket, condition, right angular bracket, colon. Line 2, indented once: left angular bracket, statements, right angular bracket, hash, execute only if condition is true. else, colon. Line 4, indented once: left angular bracket, statements, right angular bracket, hash, execute only if condition is false.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: a equals 5. Input 2: b equals 3. Input 3: if a greater than b, colon. C equals 10, hash, executed only if a greater than b. else, colon. C equals 20, hash, executed only if a lesser than or equals b. Input 4: c. Output 4: 10. Input 5: b equals 6. Input 6: if a greater than b, colon. c equals 10. Else, colon. c equals 20. Input 7: c. Output 7: 20.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: if, left angular bracket, condition, right angular bracket, colon. Line 2, indented once: left angular bracket,

statements, right angular bracket, hash, execute only if condition is true.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: a equals 5. Input 2: b equals 3. Input 3: if a greater than b, colon, hash, condition is True. c equals 10, hash, this statement is executed. Input 4: c. Output 4: 10. Input 5: if b greater than a, colon, hash condition is False. c equals 25, hash, this statement is not executed. Input 6: c. Input 6: 10.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: if, left angular bracket, condition 1, right angular bracket, colon. Line 2, indented once: if, left angular bracket, condition 2, right angular bracket, colon. Line 3, indented twice: left angular bracket, statements, right angular bracket, hash, condition 1 is true and condition 2 is true. Line 4, indented once: else, colon. Line 5, indented twice: left angular bracket, statements, right angular bracket, hash, condition 1 is true and condition 2 is false. Line 6 else, colon. Line 7, indented once: if, left angular bracket condition 3, right angular bracket, colon. Line 8, indented twice: left angular bracket, statements, right angular bracket, hash, condition 1 is false and condition 3 is true. Line 9, indented once: else, colon. Line 10, indented twice: left angular bracket, statements, right angular bracket, hash, condition 1 is false and condition 3 is false.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: if, left angular bracket, condition 1, right angular bracket, colon. Line 2, indented once: left angular bracket, statements, right angular bracket, hash, condition 1 is true. else, colon. Line 4, indented once: if, left angular bracket, condition 2, right angular bracket, colon. Line 5, indented twice: left angular bracket, statements, right angular bracket, hash, condition 1 is false and condition 2 is true. Line 6, indented once: else, colon. Line 7, indented twice: if, left angular bracket, condition 3, right angular bracket, colon. Line 8, indented thrice: left angular bracket, statements, right angular bracket, hash, condition 1 and condition are false. Line 9: hash, and condition 3 is true. Line 10: ellipses.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: if, left angular bracket, condition 1, right angular bracket, colon. Line 2, indented once: left angular bracket, statements, right angular bracket, hash, condition 1 is true. Line 3: elif, left angular bracket, condition 2, right angular bracket, colon. Line 4, indented once: left angular bracket, statements, right angular bracket, hash, condition 1 is false and condition 2 is true. Line 5: ellipses. Line 6: elif, left angular bracket, condition N, right angular bracket, colon. Line 7, indented once: left angular bracket, statements, right angular bracket, hash, conditions 1 minus, left parenthesis, N minus 1, right parenthesis, are false, hash, and condition N is true. else, colon. Line 9, indented once: left angular bracket, statement, right angular bracket, hash, conditions 1 minus N are all false.

Back to Figure

Program code. In the code, the words in the variables are merged. Input 1: score equals 75. Input 2: if score greater than, equals 90, colon. grade equals, left single quotation mark, uppercase A, right single quotation mark. elif score, greater than, equals, 80, colon. grade equals, left single quotation mark, uppercase B, right single quotation mark. elif score, greater than, equals, 70, colon. grade equals, left single quotation mark, uppercase C, right single quotation mark. elif score, greater than, equals, 60, colon. grade equals, left single quotation mark, uppercase D, right single quotation mark. else, colon. grade equals, left single quotation mark, uppercase F, right single quotation mark. Input 3: grade. Output 3: left single quotation mark, uppercase C, right single quotation mark.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def chatbot, left parenthesis, question, right parenthesis, colon. Line 2, indented once: if question, dot, starts with, left parenthesis, left double quotation mark, Who, right double quotation mark, right parenthesis, colon. Line 3, indented twice: print, left parenthesis, left double quotation mark, The tooth fairy, right double quotation mark, right parenthesis. Line 4, indented once: elif question, dot, starts with, left parenthesis, left double quotation mark, What, right double quotation mark, right parenthesis, colon. Line 5, indented twice: print, left parenthesis, left double quotation mark, To be kind to one another, right double quotation mark, right parenthesis. Line 6, indented once: elif question, dot, starts with, left parenthesis, left double quotation mark, Where, right double quotation mark, right parenthesis, colon. Line 7, indented twice: print, left parenthesis, left double quotation mark, At the dentist's office, right double

quotation mark, right parenthesis. Line 8, indented once: elif question, dot, starts with, left parenthesis, left double quotation mark, Why, right double quotation mark, right parenthesis, colon. Line 9, indented twice: print, left parenthesis, left double quotation mark, You'll understand when you're older, dot, right double quotation mark, right parenthesis. Line 10, indented once: elif question, dot, starts with, left parenthesis, left double quotation mark, When, right double quotation mark, right parenthesis, colon. Line 11, indented twice: print, left parenthesis, left double quotation mark, This afternoon, right double quotation mark, right parenthesis. Line 12, indented once: else, colon. Line 13, indented twice: print, left parenthesis, left double quotation mark, You stumped me on that, exclamatory mark, right double quotation mark, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def chatbot 2, left parenthesis, question, right parenthesis, colon. Line 2, indented once: question equals question, dot, lower, left parenthesis, right parenthesis. blank. Line 4, indented once: if question, dot, starts with, left parenthesis, left double quotation mark, who, right double quotation mark, right parenthesis, or, left double quotation mark, Who name, right double quotation mark, in question, colon. Line 5, indented twice: print, left parenthesis, left double quotation mark, The tooth fairy, right double quotation mark, right parenthesis. Line 6, indented once: elif question, dot, starts with, left parenthesis, left double quotation mark, what, right double quotation mark, right parenthesis, colon. Line 7, indented twice: print, left parenthesis, left double quotation mark, To be kind to one another, right double quotation mark, right parenthesis. Line

8, indented once: elif question, dot, starts with, left parenthesis, left double quotation mark, how, right double quotation mark, right parenthesis, and, left double quotation mark, feeling, right double quotation mark, in question, colon. Line 9, indented twice: print, left parenthesis, left double quotation mark, I don't feel anything. I'm a computer, exclamatory mark, right double quotation mark, right parenthesis. Line 10, indented once: elif question, dot, starts with, left parenthesis, left double quotation mark, where, right double quotation mark, right parenthesis, colon. Line 11, indented twice: print, left parenthesis, left double quotation mark, At the dentist's office, right double quotation mark, right parenthesis. Line 12, indented once: elif question, dot, starts with, left parenthesis, left double quotation mark, why, right double quotation mark, right parenthesis, colon. Line 13, indented twice: print, left parenthesis, left double quotation mark, You'll understand when you're older, dot, right double quotation mark, right parenthesis. Line 14, indented once: elif question, dot, starts with left parenthesis, left double quotation mark, when, right double quotation mark, right parenthesis, colon. Line 15, indented twice: print, left parenthesis, left double quotation mark, This afternoon, right double quotation mark, right parenthesis. Line 16, indented once: else, colon. Line 17, indented twice: print, left parenthesis, left double quotation mark, You stumped me on that, exclamatory mark, right double quotation mark, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Input 1: from chatbot 2 import, asterisk. Input 2: chatbot 2, left parenthesis, left double quotation mark, WHO is at the door, question mark, right double quotation mark, right

parenthesis. The tooth fairy. Input 3: chatbot 2, left parenthesis, left double quotation mark, How are you feeling, question mark, right double quotation mark, right parenthesis. I don't feel anything, dot, I'm a computer, exclamatory mark. Input 4: chatbot 2, left parenthesis, left double quotation mark, What is your name, question mark, right double quotation mark, right parenthesis. The tooth fairy. Input 5: chatbot 2, left parenthesis, left double quotation mark, What is the meaning of life, question mark, right double quotation mark, right parenthesis. To be kind to one another.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: find and guess the middle number, left parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 2: while the guess is not correct, colon. Line 3, indented once: If guess is too high, comma. Line 4, indented twice: set end to guess, negative 1. Line 5, indented once: Else if guess is too low, comma. Line 6, indented twice: set start to guess, positive 1. Line 7, indented once: Find and guess the middle number, left parenthesis end, plus, start, right parenthesis, forward slash, forward slash, 2.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: priming statement. while, left angular bracket, condition, right angular bracket, colon. Line 3, indented once: statement 1. Line 4, indented once: statement 2. Line 5, indented once: ellipses. Line 6, indented once: statement to update the condition.

Back to Figure

Program code. In the code, the words in the variables are merged. Input 1: sum equals 1, plus, 1. left single quotation mark, 1 plus 1 equals, right single quotation mark, plus, str, left parenthesis, sum, right parenthesis. Output 1: left single quotation mark, 1 plus 1 equals 2, right single quotation mark. Input 2: left single quotation mark, 1 plus 1 equals, right single quotation mark, plus, str, left parenthesis, 1 plus 1, right parenthesis, plus, in parentheses means addition. Output 2: left single quotation mark, 1 plus 1 equals 2, right single quotation mark. Input 3: left single quotation mark, 1 plus 1 DOES NOT equal, right single quotation mark, plus, str, left parenthesis, 1, right parenthesis, plus, str, left parenthesis, 1, right parenthesis. Output 3: left single quotation mark, 1 plus 1 DOES NOT equal 11, right single quotation mark. Input 4: voting Age equals 18. print, left parenthesis, left double quotation mark, You can vote if you are, right double quotation mark, plus, str, left parenthesis, voting Age, right parenthesis, plus, left double quotation mark, years old, dot, right double quotation mark, right parenthesis. You can vote if you are 18 years old, dot.

Back to Session

Program code. In the code, the words in the variables are merged. Input 1: Line 1: name equals input, left parenthesis, left double quotation mark, What is your name, question mark, right double quotation mark, right parenthesis. Line 2: print, left parenthesis, left double quotation mark, Hello, comma, right double quotation mark, plus, name, right parenthesis. Line 3: What is your name, question mark, Monica. Line 4: Hello, comma, Monica. Input 2: Line 1: test 1 equals, input, left

parenthesis, left double quotation mark, Enter the first test score, colon, right double quotation mark, right parenthesis. Line 2: test 2 equals input, left parenthesis, left double quotation mark, Enter the second test score, colon, right double quotation mark, right parenthesis. Line 3: test 3 equals input, left parenthesis, left double quotation mark, Enter the third test score, colon, right double quotation mark, right parenthesis. Line 4: average equals, left parenthesis, int, left parenthesis, test 1, right parenthesis, plus, int, left parenthesis, test 2, right parenthesis, plus, int, left parenthesis, test 3, right parenthesis, right parenthesis, forward slash, 3. Line 5: print, left parenthesis, left double quotation mark, The average test score is, right double quotation mark, plus, str, left parenthesis, average, right parenthesis, right parenthesis. Enter the first test score, colon, 92. Enter the second test score, colon, 85. Enter the third test score, colon, 88. The average test score is 88.33333333333333.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def guess A Number, left parenthesis, right parenthesis, colon. Line 2, indented once: start equals 1. Line 3, indented once: end equals 100. blank. Line 5, indented once: print, left parenthesis, left double quotation mark, Pick a number between, right double quotation mark, plus, str, left parenthesis, start, right parenthesis, plus, left double quotation mark, and, right double quotation mark, plus, str, left parenthesis, end, right parenthesis, plus, left double quotation mark, dot, right double quotation mark, right parenthesis. Line 6, indented once: print, left parenthesis, left double quotation mark, I'll try to guess it in as few guesses as possible, dot, right double quotation mark, right parenthesis. blank. Line 8, indented once: guess equals, left

parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 9, indented once: response equals input, left parenthesis, left double quotation mark, Is, right double quotation mark, plus, str, left parenthesis, guess, right parenthesis, plus, left double quotation mark, too high, comma, too low, comma, or correct, question mark, right double quotation mark, right parenthesis. Line 10, indented once: response equals response, dot, lower, left parenthesis, right parenthesis. Lime 11: blank. Line 12, indented once: while response, exclamatory equals, left single quotation mark, correct, right single quotation mark, colon. Line 13, indented twice: if response, equals, equals, left single quotation mark, too high, right single quotation mark, colon. Line 14, indented thrice: end equals guess minus 1. Line 15, indented twice: else, colon, hash, response, equals, equals, left single quotation mark, too low, right single quotation mark, colon. Line 16, indented thrice: start equals guess plus 1. Line 17: blank. Line 18, indented twice: guess equals, left parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 19, indented twice: response equals input, left parenthesis, left double quotation mark, Is, right double quotation mark, plus, str, left parenthesis, guess, right parenthesis, plus, left double quotation mark, too high, comma, too low, comma, or correct, question mark, right double quotation mark, right parenthesis. Line 20, indented twice: response equals response, dot, lower, left parenthesis, right parenthesis. Line 21: blank. Line 22, indented once: print, left parenthesis, left double quotation mark, Your number is, plus, str, left parenthesis, guess, right parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Input 1: from Guess A Number Pass 1 import, asterisk. Input 2: guess A Number, left parenthesis, right parenthesis, hash, secret number is 73. Pick a number between 1 and 100. I'll try to guess it in as few guesses as possible, dot. Is 50 too high, comma, too low, comma, or correct, question mark, too low. Is 75 too high, comma, too low, comma, or correct, question mark, too high. Is 62 too high, comma, too low, comma, or correct, question mark, too low. Is 68 too high, comma, too low, comma, or correct, question mark, too low. Is 71 too high, comma, too low, comma, or correct, question mark, too low. Is 73 too high, comma, too low, comma, or correct, question mark, correct. Your number is 73.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def guess A Number, left parenthesis, right parenthesis, colon. Line 2, indented once: start equals 1. Line 3, indented once: end equals 100. blank. Line 5, indented once: print, left parenthesis, left double quotation mark, Pick a number between, right double quotation mark, plus, str, left parenthesis, start, right parenthesis, plus, left double quotation mark, and, right double quotation mark, plus, str, left parenthesis, end, right parenthesis, plus, left double quotation mark, dot, right double quotation mark, right parenthesis. Line 6, indented once: print, left parenthesis, left double quotation mark, I'll try to guess it in as few guesses as possible, dot, right double quotation mark, right parenthesis. blank. Line 8, indented once: guess equals, left parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 9, indented once: response equals input, left parenthesis, left double quotation mark, Is, right double

quotation mark, plus, str, left parenthesis, guess, right parenthesis, plus, left double quotation mark, too high, comma, too low, comma, or correct, question mark, right double quotation mark, right parenthesis. Line 10, indented once: response equals response, dot, lower, left parenthesis, right parenthesis. Lime 11: blank. Line 12, indented once: while response, exclamatory equals, left single quotation mark, correct, right single quotation mark, colon. Line 13, indented twice: if response, equals, equals, left single quotation mark, too high, right single quotation mark, colon. Line 14, indented thrice: end equals guess minus 1. Line 15, indented twice: elif response, equals, equals, left single quotation mark, too low, right single quotation mark, colon. Line 16, indented thrice: start equals guess plus 1. Line 17, indented twice: else, colon, hash, invalid response. Line 18, indented thrice: response equals input I'll try to guess it in as few guesses as possible, dot, right double quotation mark, Please enter, left single quotation mark, too high, right single quotation mark, comma, left single quotation mark, too low right single quotation mark, comma, or, left single quotation mark, correct, right single quotation mark, right double quotation mark, right parenthesis. Line 19, indented thrice: response equals response, dot, lower, left parenthesis, right parenthesis. Line 20, indented thrice: continue, hash, skip the remainder of the loop. Line 21: blank. Line 22, indented twice: guess equals, left parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 23, indented twice: response equals input, left parenthesis, left double quotation mark, Is, right double quotation mark, plus, str, left parenthesis, guess, right parenthesis, plus, left double quotation mark, too high, comma, too low, comma, or correct, question mark, right double quotation mark, right parenthesis. Line 24, indented twice: response equals response, dot, lower, left parenthesis, right

parenthesis. Line 25: blank. Line 26, indented once: print, left parenthesis, left double quotation mark, Your number is, plus, str, left parenthesis, guess, right parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Input 1: from Guess A Number Pass 2 import, asterisk. Input 2: guess A Number, left parenthesis, right parenthesis, hash, secret number is 31. Pick a number between 1 and 100. I'll try to guess it in as few guesses as possible, dot. Is 50 too high, comma, too low, comma, or correct, question mark, too high. Is 25 too high, comma, too low, comma, or correct, question mark, too low. Is 37 too high, comma, too low, comma, or correct, question mark, a little high. Please enter, left single quotation mark, too high, right single quotation mark, comma, left single quotation mark, too low, right single quotation mark, or, left single quotation mark, correct, right single quotation mark, hello. Please enter, left single quotation mark, too high, right single quotation mark, comma, left single quotation mark, too low, right single quotation mark, or, left single quotation mark, correct, right single quotation mark, too high. Is 31 too high, comma, too low, comma, or correct, question mark, Correct. Your number is 31.

Back to Session

Program code. In the code, the words in the variables are merged. Input 1: acc equals 0. Input 2: for x in range, left parenthesis, 1, comma, 6, right parenthesis, colon. acc equals acc plus x. Input 3: acc. Output 4: 15.

Back to Session

The numbers at the top are labeled X. The numbers at the bottom are labeled accumulator. X reads 1, 2, 3, 4, and 5. The accumulators read 0, 1, 3, 6, 10, and 15. 0 plus 1 gives 1, 1 plus 2 gives 3, 3 plus 3 gives 6, 6 plus 4 gives 10, 10 plus 5 gives 15.

Back to Figure

Program code. In the code, the words in the variables are merged. Input 1: Line 1: count Passed equals 0, hash, initialize both counting variables. Line 2: count Tests equals 0. Line 3: blank. Line 4: score equals input, left parenthesis, left double quotation mark, Enter a test score, negative 1, to stop, colon, right double quotation mark, right parenthesis. Line 5: score equals int, left parenthesis, score, right parenthesis. Line 6: blank. Line 7: while score, exclamatory mark, equals negative 1, colon. Line 8, indented once: if score greater than equals 60, colon. Line 9, indented twice: count Passed equals count Passed plus 1, hash, only count if passing grade. Line 10, indented once: count Tests equals count Tests plus 1, hash, count every score. Line 11, indented once: score equals input, left parenthesis, left double quotation mark, Enter another score, negative 1 to stop, colon, right double quotation mark, right parenthesis, hash, loop update. Line 12, indented once: score equals int, left parenthesis, score, right parenthesis. print, left parenthesis, left double quotation mark, You passed, right double quotation mark, plus, str, left parenthesis, count Passed, right parenthesis, plus, left double quotation mark, out of, right double quotation mark, plus, str, left parenthesis, count Tests, right parenthesis, plus, left double quotation mark, tests, dot, right double quotation mark, right parenthesis. Line 13: Enter a test score, comma, negative 1, to

stop, colon, 75. Line 14: Enter another score, comma, negative 1, to stop, colon, 60. Line 15: Enter another score, comma, negative 1, to stop, colon, 59. Line 16: Enter another score, comma, negative 1, to stop, colon, 82. Line 17: Enter another score, comma, negative 1, to stop, colon, negative 1. Line 18: You passed 3 out of 4 tests.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def guess A Number, left parenthesis, right parenthesis, colon. Line 2, indented once: start equals 1. Line 3, indented once: end equals 100. blank. Line 5, indented once: print, left parenthesis, left double quotation mark, Pick a number between, right double quotation mark, plus, str, left parenthesis, start, right parenthesis, plus, left double quotation mark, and, right double quotation mark, plus, str, left parenthesis, end, right parenthesis, plus, left double quotation mark, dot, right double quotation mark, right parenthesis. Line 6, indented once: print, left parenthesis, left double quotation mark, I'll try to guess it in as few guesses as possible, dot, right double quotation mark, right parenthesis. blank. Line 8, indented once: guess equals, left parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 9, indented once: num Attempts equals 1. Line 10, indented once: response equals input, left parenthesis, left double quotation mark, Is, right double quotation mark, plus, str, left parenthesis, guess, right parenthesis, plus, left double quotation mark, too high, comma, too low, comma, or correct, question mark, right double quotation mark, right parenthesis. Line 11, indented once: response equals response, dot, lower, left parenthesis, right parenthesis. Line 12: blank. Line 13, indented once: while response, exclamatory equals, left single quotation

mark, correct, right single quotation mark, colon. Line 14, indented twice: if response, equals, equals, left single quotation mark, too high, right single quotation mark, colon. Line 15, indented thrice: end equals guess minus 1. Line 16, indented twice: elif response, equals, equals, left single quotation mark, too low, right single quotation mark, colon. Line 17, indented thrice: start equals guess plus 1. Line 18, indented twice: else, colon, hash, invalid response. Line 19, indented thrice: response equals input I'll try to guess it in as few guesses as possible, dot, right double quotation mark, Please enter, left single quotation mark, too high, right single quotation mark, comma, left single quotation mark, too low right single quotation mark, comma, or, left single quotation mark, correct, right single quotation mark, right double quotation mark, right parenthesis. Line 20, indented thrice: response equals response, dot, lower, left parenthesis, right parenthesis. Line 21, indented thrice: continue, hash, skip the remainder of the loop. Line 22: blank. Line 23, indented twice: guess equals, left parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 24, indented twice: num Attempts equals num Attempts plus 1. Line 25, indented twice: response equals input, left parenthesis, left double quotation mark, Is, right double quotation mark, plus, str, left parenthesis, guess, right parenthesis, plus, left double quotation mark, too high, comma, too low, comma, or correct, question mark, right double quotation mark, right parenthesis. Line 26, indented twice: response equals response, dot, lower, left parenthesis, right parenthesis. Line 27: blank. Line 28, indented once: print, left parenthesis, left double quotation mark, Your number is, plus, str, left parenthesis, guess, right parenthesis, right parenthesis. Line 29, indented once: print, left parenthesis, left double quotation mark, I guessed it in, right double quotation mark, plus, str, left parenthesis, num Attempts,

right parenthesis, plus, left double quotation mark, tries, dot, right double quotation mark, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Input 2: from Guess A Number Pass 3 import, asterisk. Input 3: guess A Number, left parenthesis, right parenthesis, hash, secret number is 33. Pick a number between 1 and 100. I'll try to guess it in as few guesses as possible. Is 50 too high, comma, too low, comma, or correct, question mark, too high. Is 25 too high, comma, too low, comma, or correct, question mark, too low. Is 37 too high, comma, too low, comma, or correct, question mark, too high. Is 31 too high, comma, too low, comma, or correct, question mark, too low. Is 34 too high, comma, too low, comma, or correct, question mark, too high. Is 32 too high, comma, too low, comma, or correct, question mark, too low. Is 33 too high, comma, too low, comma, or correct, question mark, correct. Your number is 33. I guessed it in 7 tries.

Back to Session

Program code. In the code, the words in the variables are merged. Input 1: import random. Input 2: random, dot, rand int, left parenthesis, 1, comma, 100, right parenthesis. Output 2: 67. Input 3: for i in range, left parenthesis, 5, right parenthesis, colon. print, left parenthesis, random, dot, rand int, left parenthesis, 1, comma, 10, right parenthesis, right parenthesis. 7. 5. 8. 1. 5. Input 4: for i in range, left parenthesis, 5, right parenthesis, colon. print, left parenthesis, random, dot, rand int, left parenthesis, 20, comma, 50, right parenthesis, right parenthesis. 35. 35. 48. 24. 24.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def function Name, left parenthesis, param 1, comma, param 2, comma, ellipses, right parenthesis, colon. Line 2, indented once: statement 1. Line 3, indented once: statement 2. Line 4, indented once: ellipses. Line 5, indented once: return expression, hash, exit and return expression's value.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: while there are more stones. Line 2, indented once: get Player 1's play, comma, make the play, comma. Line 3, indented twice: if pile is 0, comma, winner is Player 1. Line 4, indented once: get Player 2's play, comma, make the play, comma. Line 5, indented twice: if pile is 0, comma, winner is Player 2.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def nim, left parenthesis, right parenthesis, colon. Line 2, indented once: pile equals 15. Line 3, indented once: print, left parenthesis, left double quotation mark, The pile has, right double quotation mark, plus, str, left parenthesis, pile, right parenthesis, plus, left double quotation mark, stones, dot, right double quotation mark, right parenthesis. Line 4, indented once: blank. Line 5, indented once: winner equals, left double quotation mark, right double quotation mark. Line 6, indented once: while pile, exclamatory mark, equals, 0, colon. Line 7, indented twice: c 1 Move equals random Move, left parenthesis,

pile, right parenthesis. Line 8, indented twice: pile equals pile minus c 1 Move. Line 9, indented twice: print, left parenthesis, left double quotation mark, Random Randy takes, right double quotation mark, plus str, left parenthesis, c 1 Move, right parenthesis, plus, left double quotation mark, stone, left parenthesis, lowercase s, right parenthesis, Pile is now, right double quotation mark, plus, str, left parenthesis, pile, right parenthesis, right parenthesis. Line 10, indented twice: if pile, equals, equals, 0, colon. Line 11, indented thrice: winner equals, left double quotation mark, Random Randy, right double quotation mark. Line 12, indented twice: else, colon. Line 13, indented thrice: c 2 Move equals nimble Move, left parenthesis, pile, right parenthesis. Line 14, indented thrice: pile equals pile minus c 2 Move. Line 15, indented thrice: print, left parenthesis, left double quotation mark, Nimble Nancy takes, right double quotation mark, plus, str left parenthesis, c 2 Move, right parenthesis, plus, left double quotation mark, stone, left parenthesis, lowercase s, right parenthesis, Pile is now, right double quotation mark, plus, str, left parenthesis, pile, right parenthesis, right parenthesis. Line 16, indented thrice: if pile equals, equals, 0, colon. Line 17, indented quarce: winner equals, left double quotation mark, Nimble Nancy, right double quotation mark. Line 18, indented once: print, left parenthesis, left double quotation mark, The winner is, right double quotation mark, plus, winner, plus, left double quotation mark, exclamatory mark, right double quotation mark, right parenthesis. Line 19: blank. Line 20: def random Move, left parenthesis, pile, right parenthesis, colon. Line 21, indented once: import random. Line 22, indented once: move equals random, dot, rand int, left parenthesis, 1, comma, 3, right parenthesis. Line 23, indented once: if move greater than pile, colon. Line 24, indented twice: move equals pile. Line 25: blank. Line 26, indented once: return

move. Line 27: blank. Line 28: def nimble Move, left parenthesis, pile, right parenthesis, colon. Line 29, indented once: if pile percentage 4, exclamatory, equals, 0, colon. Line 30, indented twice: move equals pile percentage 4. Line 31, indented once: else colon. Line 32, indented twice: move equals 1. Line 33: blank. Line 34, indented once: return move.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 2: nim, left parenthesis, right parenthesis. The pile has 15 stones. Random Randy takes 3 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 12. Random Randy takes 3 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 8. Random Randy takes 1 stones, lowercase s, right parenthesis. Pile is now 6. Random Randy takes 2 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 2. Nimble Nancy takes 1 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 11. Nimble Nancy takes 1 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 7. Nimble Nancy takes 2 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 4. Nimble Nancy takes 2 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 0. The winner is Nimble Nancy, exclamatory mark. Input 3: nim, left parenthesis, right parenthesis. The pile has 15 stones. Random Randy takes 3 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 12. Random Randy takes 1 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 10. Random Randy takes 1 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 7. Random Randy takes 2 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 2. Nimble Nancy takes 1 stones, left

parenthesis, lowercase s, right parenthesis. Pile is now 11. Nimble Nancy takes 2 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 8. Nimble Nancy takes 3 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 4. Nimble Nancy takes 2 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 0. The winner is Nimble Nancy, exclamatory mark. Input 4: nim, left parenthesis, right parenthesis. The pile has 15 stones. Random Randy takes 1 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 14. Random Randy takes 1 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 11. Random Randy takes 1 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 7. Random Randy takes 1 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 3. Nimble Nancy takes 2 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 12. Nimble Nancy takes 3 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 8. Nimble Nancy takes 3 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 4. Nimble Nancy takes 3 stones, left parenthesis, lowercase s, right parenthesis. Pile is now 0. The winner is Nimble Nancy, exclamatory mark.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: left double quotation mark, Hip, right double quotation mark, asterisk, 2, plus, left double quotation mark, Hooray, exclamatory mark, right double quotation mark. Output 1: left single quotation mark, Hip Hip Hooray, exclamatory mark, right single quotation mark. Input 2: def row Your Boat, left parenthesis, right parenthesis, colon. print, left parenthesis, left double quotation

mark, Row, comma, right double quotation mark, asterisk, 3, plus, left single quotation mark, your boat, right double quotation mark, right parenthesis. print, left parenthesis, left double quotation mark, Gently down the stream, right double quotation mark, right parenthesis. print, left parenthesis, left double quotation mark, Merrily, comma, right double quotation mark, asterisk, 4, right parenthesis. print, left parenthesis, left double quotation mark, Life is but a dream, right double quotation mark, right parenthesis. Input 3: row Your Boat, left parenthesis, right parenthesis. Row, comma, Row, comma, Row, comma, your boat. Gently down the stream. Merrily, comma, Merrily, comma, Merrily, comma, Merrily, comma. Life is but a dream.

Back to Session

The table has three rows and twelve columns. The row headers are Positive indexes, String, Negative indexes. The table entries are as follows. Row 1: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. Row 2: P, Y, T, H, O, N, blank, R, O, C, K, S. Row 3: negative 12, negative 11, negative 10, negative 9, negative 8, negative 7, negative 6, negative 5, negative 4, negative 3, negative 2, negative 1.

Back to Figure

Line 1 reads: Input 1 within bracket colon mary within double quotes 0 colon 3 within bracket. Line 2 reads: Out 1 within bracket colon mar within single quotes.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: name equals, left double quotation mark, Roy G Biv, right double quotation mark.

Input 2: first Char equals name, left square bracket, 0, right square bracket. Input 3: first Char. Output 3: left single quotation mark, R, right single quotation mark. Input 4: middle Index equals len, left parenthesis, name, right parenthesis, forward slash, forward slash, 2. Input 5: middle Index. Output 5: 4. Input 6: middle Char equals name, left square bracket, middle Index, right square bracket. Input 7: middle Char. Output 7: left single quotation mark, G, right single quotation mark. Input 8: name, left square bracket, negative 1, right square bracket. Output 8: left single quotation mark, V, right single quotation mark. Input 9: name, left square bracket, len, left parenthesis, name, right parenthesis, negative 1, right square bracket. Output 9: left single quotation mark, v, right single quotation mark. Input 10: for i in range, left parenthesis, len, left parenthesis, name, right parenthesis, right parenthesis, colon. print, left parenthesis, name, left square bracket, i, right square bracket, right parenthesis. R. o. y. G. B. i. v.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: name equals, left double quotation mark, Roy G Biv, right double quotation mark. Input 2: name, left square bracket, 0, colon, 3, right square bracket. Output 2: left single quotation mark, Roy, right single quotation mark. Input 3: name, left square bracket, colon, 3, right square bracket, hash, start index omitted, comma, start at 0. Output 3: left single quotation mark, Roy, right single quotation mark. Input 4: name, left square bracket, 6, colon, 9, right square bracket. Output 4: left single quotation mark, Biv, right single quotation mark. Input 5: name, left square bracket, 6, colon, right square bracket, hash, end index omitted, comma, continue to

string end. Output 5: left single quotation mark, Biv, right single quotation mark. Input 6: for i in range, left parenthesis, 1, comma, len, left parenthesis, name, right parenthesis, right parenthesis, colon. print, left parenthesis, name, left square bracket, 0, colon, i, right square bracket, right parenthesis. R. Ro. Roy. Roy. Roy G. Roy G. Roy G B. Roy G Bi.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input blank: Line 1: Left double quotes, Sub, right double quotes, in, left double quotes, string, right double quotes. Line 2: Left double quotes, sub, right double quotes, not in, left double quotes, string, right double quotes.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: name equals, left double quotation mark, Roy G Biv, right double quotation mark. Input 2: left double quotation mark, Biv, right double quotation mark, in name. Output 2: True. Input 3: left double quotation mark, biv, right double quotation mark, in name, hash, b does not match B. Output 3: False. Input 4: left double quotation mark, v, right double quotation mark, not in name. Output 4: False. Input 5: left double quotation mark, 3, right double quotation mark, in name. Output 5: False. Input 6: left double quotation mark, Bv, right double quotation mark, in name, hash, B and v are both in name, comma, but not together. Output 6: False. Input 8: if, left double quotation mark, y, right double quotation mark, in name, colon. Line 1, indented once: print, left

parenthesis, left double quotation mark, The letter y is in name, dot, right double quotation mark, right parenthesis. Line 2: else, colon. Line 3, indented once: print, left parenthesis, left double quotation mark, The letter y is not in name, dot, right double quotation mark, right parenthesis. Line 4: The letter y is in name.

Back to Session

The table consists of four columns: Operator or Function, Description, Example, and Result. Row entries are as follows. Row 1. Operator or Function: Plus sign. Description: Concatenation: joins two strings together. Example: Double quotes hello double quotes plus double quotes world double quotes. Result: Double quotes hello world double quotes. Row 2. Operator or Function: Asterisk. Description: Repetition: repeats the string. Example: Double quotes a b c double quotes asterisk 3. Result: Double quotes a b c a b c a b c double quotes. Row 3. Operator or Function: Left square bracket i right square bracket. Description: Index: returns the character at position i; indexes start at 0. Example: Double quotes a b c double quotes left square bracket 1 right square bracket. Result: Single quote b single quote. Row 4. Operator or Function: Left square bracket i colon j right square bracket. Description: Slice: returns the substring from i to j minus 1. Example: Double quotes a b c d e f double quotes left square bracket 0 colon 2 right square bracket. Result: Double quotes a b double quotes. Row 5. Operator or Function: in. Description: in operator: determines whether one string contains another. Example: Single quote b c single quote in single quote a b c single quote. Result: True. Row 6. Operator or Function: not in. Description: not in operator: determines whether one string does not contain another. Example: Single

quote d single quote not in single quote a b c single quote. Result: True.

Back to Table

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, left double quotation mark, hello, right double quotation mark, dot, l, just, left parenthesis, 10, right parenthesis. Output 1: left single quotation mark, hello, space, right single quotation mark. Input 2: 1, left double quotation mark, hello, right double quotation mark, dot, r, just, left parenthesis, 10, right parenthesis. Output 2: left single quotation mark, space, hello, right single quotation mark. Input 3: 1, team Sport equals, left double quotation mark, Golden Gopher football, right double quotation mark. Input 4: 1, team Sport, dot, count, left parenthesis, left single quotation mark, o, right single quotation mark, right parenthesis. Output 4: 4. Input 5: 1, team Sport, dot, count, left parenthesis, left single quotation mark, o o, right single quotation mark, right parenthesis. Output 5: 1. Input 6: 1, team Sport, dot, lower, left parenthesis, right parenthesis. Output 6: left single quotation mark, golden gopher football, right single quotation mark. Input 7: 1, team Sport, hash, original string is not affected. left single quotation mark, Golden Gopher Football, right single quotation mark. Input 8: 1, team Sport equals team Sport, dot, upper, left parenthesis, right parenthesis, hash, assign result to original string. Input 9: 1, team Sport. Output 9: left single quotation mark, GOLDEN GOPHER FOOTBALL, right single quotation mark. Input 10: 1, team Sport, dot, find, left parenthesis, left single quotation mark, G, right single quotation mark, right parenthesis, hash, finds first G at index 0. Output 10: 0. Input 11: 1, team Sport, dot, r find, left parenthesis, left single quotation

mark, G, right single quotation mark, right parenthesis, hash, finds first G at index 7. Output 11: 7. Input 12: 1, team Sport, dot, index, left parenthesis, left single quotation mark, G, right single quotation mark, right parenthesis. Output 12: 0. Input 13: 1, team Sport, dot, r index, left parenthesis, left single quotation mark, G, right single quotation mark, right parenthesis. Output 13: 7. Input 15: 1, team Sport, dot, find, left parenthesis, left single quotation mark, g, right single quotation mark, right parenthesis, hash, not found. Output 15: negative 1. Input 16: 1, team Sport, dot, index, left parenthesis, left single quotation mark, g, right single quotation mark, right parenthesis, hash, not found, comma, index method throws an error. Value Error Trace back, left parenthesis, most recent call last, right parenthesis. Input In, left square bracket, 16, right square bracket, comma, in, left angular bracket, cell line, colon, 1, right angular bracket, left parenthesis, right parenthesis. right angular bracket, 1, team Sport, dot, index, left parenthesis, left single quotation mark, g, right single quotation mark, right parenthesis. Value Error, colon, substring not found. Input 17: 1, left single quotation mark, a b, c d, e f, right single quotation mark, dot, replace, left parenthesis, left single quotation mark, c d, right single quotation mark, comma, left single quotation mark, x y, right single quotation mark, right parenthesis, hash, replace c d with x y. Output 17: left single quotation mark, a b, space, x y, space, e f, right single quotation mark.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: ord, left parenthesis, left single quotation mark, a, right single quotation mark, right parenthesis. Line 2: Output 1: 97. Line 3: Input 2: ord,

left parenthesis, left single quotation mark, c, right single quotation mark, right parenthesis. Line 4: Output 2: 99. Line 5: Input 3: ord, left parenthesis, left single quotation mark, A, right single quotation mark, right parenthesis. Line 6: Output 3: 65. Line 7: Input 4: chr, left parenthesis, 104, right parenthesis. Line 8: Output 4: left single quotation mark, h, right single quotation mark. Line 9: Input 5: chr, left parenthesis, 97, plus, 13, right parenthesis. Line 10: Output 5: left single quotation mark, n, right single quotation mark. Line 11: Input 6: str, left parenthesis, 10960, right parenthesis. Line 12: Output 6: left single quotation mark, 10960, right single quotation mark.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def letter To Index, left parenthesis, letter, right parenthesis, colon. Line 2, indented once: import string. Line 3, indented once: alphabet equals string, dot, ascii, underscore, lowercase, plus, left single quotation mark, space, right single quotation mark. Line 4, indented once: idx equals alphabet, dot, find, left parenthesis, letter, right parenthesis. Line 5, indented once: if idx equals, equals, negative 1, colon, hash, not found. Lime 6, indented twice: print, left parenthesis, left double quotation mark, error, colon, right double quotation mark, plus, letter, plus, left double quotation mark, is not in the alphabet, right double quotation mark, right parenthesis. Line 7, indented once: return idx. Line 8: Blank. Line 9: def index To Letter, left parenthesis, idx, right parenthesis, colon. Line 10, indented once: import string. Line 11, indented once: alphabet equals string, dot, ascii, underscore, lowercase, plus, left single quotation mark, space, right single quotation mark. Line 12, indented once: letter equals, left single quotation mark, space, right single quotation

mark. Line 13, indented once: if idx greater than, equals, len, left parenthesis, alphabet, right parenthesis, colon. Line 14, indented twice: print, left parenthesis, left double quotation mark, error, colon, right double quotation mark, plus, str, left parenthesis, idx, right parenthesis, plus, left double quotation mark, is too large, right double quotation mark, right parenthesis. Line 15, indented once: elif idx less than 0, colon. Line 16, indented twice: print, left parenthesis, left double quotation mark, error, colon, right double quotation mark, plus, str, left parenthesis, idx, right parenthesis, plus, left double quotation mark, is less than 0, right double quotation mark, right parenthesis. Line 17, indented once: else, colon. Line 18, indented twice: letter equals alphabet, left square bracket, idx, right square bracket. Line 19, indented once: return letter.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from mapper import, asterisk. Input 2: 1, letter To Index, left parenthesis, left single quotation mark, a, right single quotation mark, right parenthesis. Output 2: 0. Input 3: 1, letter To Index, left parenthesis, left single quotation mark, space, right single quotation mark, right parenthesis, hash, space character. Output 3: 26. Input 4: 1, letter To Index, left parenthesis, left single quotation mark, 5, right single quotation mark, right parenthesis. error, colon, 5 is not in the alphabet. Output 4: negative 1. Input 5: 1, index To Letter, left parenthesis, 0, right parenthesis. Output 5: left single quotation mark, a, right single quotation mark. Input 6: 1, index To Letter, left parenthesis, 26, right parenthesis. Output 6: left single quotation mark, space, right single quotation mark. Input 7: 1, index To Letter, left parenthesis, negative 1,

right parenthesis. error, colon, negative 1 is less than 0. Output 7: left single quotation mark, space, right single quotation mark. Input 8: 1, index To Letter, left parenthesis, 27, right parenthesis. error, colon, negative 1 is less than 0. Output 8: left single quotation mark, space, right single quotation mark.

Back to Session

The figure defines the breakup of the plaintext into even and odd characters in three rows. The row headers are Original, Even, Odd. Row 1 reads the following. It was a dark and stormy night. Row 2 reads the following. I, underscore, a, underscore, underscore, a, k, a, d, s, o, m, underscore, i, h. Row 3 reads the following. t, w, s, a, d, r, underscore, n, underscore, t, r, y, n, g, t. The bottom defines how the even and odd parts are combined to make the cipher text. It reads the following. t, w, s, a, d, r, underscore, n, underscore, t, r, y, n, g, t, plus, I, underscore, a, underscore, underscore, a, k, a, d, s, o, m, underscore, i, h.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def scramble 2 Encrypt, left parenthesis, plaintext, right parenthesis, colon. Line 2, indented once: even Chars equals, left double quotation mark, space, right double quotation mark. Line 3, indented once: odd Chars equals, left double quotation mark, space, right double quotation mark. Line 4, indented once: char Count equals 0. Line 5, indented once: for ch in plaintext, colon. Line 6, indented twice: if char Count percentage 2 equals, equals 0, colon. Line 7, indented thrice: even Chars equals even Chars plus ch. Line 8, indented twice: else, colon. Line 9, indented thrice: odd Chars equals odd Chars plus

ch. Line 10, indented twice: char Count equals char Count plus 1. Line 11: blank. Line 12, indented once: cipher Text equals odd Chars plus even Chars. Line 13, indented once: return cipher Text.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from transposition import asterisk. Input 2: 1, scramble 2 Encrypt, left parenthesis, left single quotation mark, a b a b a b a b, right single quotation mark, right parenthesis. Output 2: left single quotation mark, a b a b a b a b, right single quotation mark. Input 3: 1, scramble 2 Encrypt, left parenthesis, left single quotation mark, a b a b a b a b c, right single quotation mark, right parenthesis. Output 3: left single quotation mark, a b a b a b a b c, right single quotation mark. Input 4: 1, scramble 2 Encrypt, left parenthesis, left single quotation mark, I do not like green eggs and ham, right single quotation mark, right parenthesis. Output 4: left single quotation mark, o n t l k, space, r e, space, g s a d h m I d, space, o, space, i e g e n e g, space, n a, right single quotation mark. Input 5: 1, scramble 2 Encrypt, left parenthesis, left single quotation mark, a, right single quotation mark, right parenthesis. Out, left square bracket, 5, right square bracket, colon left single quotation mark, a, right single quotation mark. Input 6: 1, scramble 2 Encrypt, left parenthesis, left single quotation mark, space, right single quotation mark, right parenthesis. Output 6: left single quotation mark, space, right single quotation mark.

Back to Session

The figure consists of three rows. Row 1 reads the following. I, underscore, a, underscore, underscore, a, k, a, d, s, o, m, underscore, i, h. Row 2 reads the following. It was. Row 3 reads the following. t, w, s, a, d, r, underscore, n, underscore, t, r, y, n, g, t. An arrow labeled 0 from the letter I in the first row goes to the letter I in the second row. An arrow labeled 2 from the underscore after the letter I in the first row goes to the letter t in the second row. An arrow labeled 4 from the underscore after the letter a in the first row goes to the letter a, in the second row. An arrow labeled 1 from the letter t in the third row goes to the letter t in the second row. An arrow labeled 3 from the letter w in the third row goes to the letter w in the second row.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def scramble 2 Decrypt, left parenthesis, cipher text, right parenthesis, colon. Line 2, indented once: half Length equals len, left parenthesis, cipher text, right parenthesis, forward slash, forward slash, 2. Line 3, indented once: even Chars equals cipher Text, left square bracket, half Length, colon, right square bracket, hash, half Length to end. Line 4, indented once: odd Chars equals cipher Text left square bracket, colon, half Length, right square bracket, hash, 0 to half Length, minus 1. Line 5, indented once: plain Text equals, left double quotation mark, space, right double quotation mark. Line 6: blank. Line 7, indented once: for i in range, left parenthesis, half Length, right parenthesis, colon. Line 8, indented twice: plain Text equals plain Text plus even Chars, left square bracket, i, right square bracket. Line 9, indented twice: plain Text equals plain Text plus odd Chars, left square bracket, i, right square bracket. Line 10: blank. Line 11, indented once: if len, left parenthesis, odd Chars, right

parenthesis, less than, len, left parenthesis, even Chars, right parenthesis, colon. Line 12, indented twice: plain Text equals plaint Text plus even Chars, left square bracket, negative 1, right square bracket, hash, Last even character. Line 13: blank. Line 14, indented once: return plain Text.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from transposition import asterisk. Input 2: 1, scramble 2 Decrypt, left parenthesis, scramble 2 Encrypt, left parenthesis, left single quotation mark, a b a b a b a b, right single quotation mark, right parenthesis. Output 2: left single quotation mark, a b a b a b a b, right single quotation mark. Input 3: 1, scramble 2 Decrypt, left parenthesis, scramble 2 Encrypt, left parenthesis, left single quotation mark, a, right single quotation mark, right parenthesis, right parenthesis. Output 3: left single quotation mark, a, right single quotation mark. Input 4: 1, scramble 2 Decrypt, left parenthesis, scramble 2 Encrypt, left single quotation mark, space, right single quotation mark, right parenthesis, right parenthesis. Output 4: left single quotation mark, space, right single quotation mark. Input 6: 1, scramble 2 Decrypt, left parenthesis, scramble 2 Encrypt, left parenthesis, left single quotation mark, I do not like green eggs, backslash, and ham, right single quotation mark, right parenthesis, right parenthesis. Out, left square bracket, 6, right square bracket, colon left single quotation mark, I do not like green eggs and ham, right single quotation mark.

Back to Session

The plain text undergoes an encryption algorithm with a key, where the cipher text is extracted. The cipher text undergoes a decryption algorithm with a key, where the plain text is extracted.

Back to Figure

The process involves four rows. Ro w 1 reads the word flow. Row 2 reads the alphabet and lists the following. a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z. The letters f, l, o, and w are highlighted. Row 3 reads the keys and lists the following. b, p, z, h, g, o, c, v, j, d, q, s, w, k, i, m, l, u, t, n, e, r, y, a, x, f. The letters o, s, i, and y are highlighted. An arrow from the letter f in the word flow from the first row goes to the letter f in row 2. An arrow from the letter l in the word flow from the first row goes to the letter l in row 2. An arrow from the letter o in the word flow from the first row goes to the letter o in row 2. An arrow from the letter w in the word flow from the first row goes to the letter w in row 2. An arrow from the letter o in row 3 goes to the letter o in row 4. An arrow from the letter s in row 3 goes to the letter s in row 4. An arrow from the letter i in row 3 goes to the letter i in row 4. An arrow from the letter y in row 3 goes to the letter y in row 4.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: alphabet String equals, left double quotation mark, a b c d e f g h i j k lm n o p q r s t u v w x y z, right double quotation mark. Input 2: key equals, left double quotation mark, z y x w v u t s r q p o n m l k j i h g f e d c b a, right double quotation mark. Input 3: i equals alphabet String, dot, find, left parenthesis, left single quotation

mark, h, right single quotation mark, right parenthesis. Input 4: print, left parenthesis, i, right parenthesis. 7. Input 5: print, left parenthesis, key, left square bracket, i, right square bracket, right parenthesis. t.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def substitution Encrypt, left parenthesis, plain Text, comma, key, right parenthesis, colon. Line 2, indented once: import string. Line 3, indented once: alphabet equals string, dot, ascii, underscore, lowercase, plus, left double quotation mark, space, right double quotation mark. Line 4, indented once: plain Text equals plain Text, dot, lower, left parenthesis, right parenthesis. Line 5: blank. Line 6, indented once: cipher Text equals, left double quotation mark, space, right double quotation mark. Line 7, indented once: for ch in plain Text, colon. Line 8, indented twice: idx equals alphabet, dot, find, left parenthesis, ch, right parenthesis. Line 9, indented twice: cipher Text equals cipher Text plus key, left square bracket, idx, right square bracket. Line 10: blank. Line 11, indented once: return cipher Text.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from substitution import asterisk. Input 2: test Key 1 equals, left double quotation mark, z y x w v u t s r q p o n m l, space, k j i h g f e d c b a, right double quotation mark. Input 3: test Key 2 equals, left double quotation mark, o u w c k b j m p z y e x a, space, v r l t s f g d q i h n, right double quotation mark. Input 4: plain Text equals

input, left parenthesis, left double quotation mark, Enter the message to encrypt, colon, right double quotation mark, right parenthesis. Enter the message to encrypt, colon, The quick brown fox. Input 5: cipher Text equals substitution Encrypt, left parenthesis, plain Text, test Key 1, right parenthesis. Input 6: print, left parenthesis, cipher Text, right parenthesis. h s v a k g r x p a y j l e m a u l d. Input 7: cipher Text equals substitution Encrypt, left parenthesis, plain Text, test Key 2, right parenthesis. Input 8: print, left parenthesis, cipher Text, right parenthesis. s m k n r f p w y n u l, space, d a n b, space, q.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def remove Char, left parenthesis, string, comma, dx, right parenthesis, colon. Line 2, indented once: return string, left square bracket, colon, idx, right square bracket, plus, string, left square bracket, idx plus 1, colon, right square bracket. Line 3: blank.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from key Generation import asterisk. Input 2: remove Char, left parenthesis, left single quotation mark, a b c d e f g, right single quotation mark, comma, 0, right parenthesis, hash, remove first character. Output 2: left single quotation mark, b c d e f g, right single quotation mark. Input 3: remove Char, left parenthesis, left single quotation mark, a b c d e f g, right single quotation mark, comma, 6, right parenthesis, hash, remove last character. Output 3: left single quotation mark, a b c d e f, right single quotation

mark. Input 4: remove Char, left parenthesis, left single quotation mark, a b c d e f g, right single quotation mark, comma, 3, right parenthesis, hash, remove from the middle. Output 4: left single quotation mark, a b c e f g, right single quotation mark. Input 5: remove Char, left parenthesis, left single quotation mark, a b c d e f g, right single quotation mark, comma, negative 2, right parenthesis, hash, remove second to last character. Output 5: left single quotation mark, a b c d e g, right single quotation mark. Input 6: remove Char, left parenthesis, left single quotation mark, a b c d e f g, right single quotation mark, comma, negative 1, right parenthesis, hash, try to remove last character, question mark, question mark, question mark. Output 6: left single quotation mark, a b c d e f a b c d e f g, right single quotation mark

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def key Gen, left parenthesis, right parenthesis, colon. Line 2, indented once: import string, comma, random. Line 3, indented once: alphabet equals string, dot, ascii, underscore, lowercase, plus, left double quotation mark, space, right double quotation mark. Line 4, indented once: key equals, left double quotation mark, space, right double quotation mark. Line 5: blank. Line 6, indented once: for i in range, left parenthesis, len, left parenthesis, alphabet, right parenthesis, negative 1, negative 1, negative 1, right parenthesis, colon. Line 7, indented twice: idx equals random, dot, rand int, left parenthesis, 0, comma i, right parenthesis. Line 8, indented twice: key equals key plus alphabet, left square bracket, idx, right square bracket. Line 9, indented twice: alphabet equals remove

Char, left parenthesis alphabet, comma, idx, right parenthesis. Line 10: blank. Line 11, indented once: return key.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from key Generation import asterisk. Input 2: key Gen, left parenthesis, right parenthesis. Output 2: left single quotation mark, n t o d, space, v x p e f m a q u k b g c w s l r y z j i h, right single quotation mark. Input 3: key Gen, left parenthesis, right parenthesis. Output 3: left single quotation mark, m r, space, z a p d k e q u n l o f w x c h j b g s v i y t, right single quotation mark. Input 4: key Gen, left parenthesis, right parenthesis. Output 4: left single quotation mark, p v, space, w l k y d i m c a z h x b f e u t o r s j g n q, right single quotation mark. Input 5: len, left parenthesis, key Gen, left parenthesis, right parenthesis, right parenthesis. Output 5: 27.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def remove Dupes, left parenthesis, my String, right parenthesis, colon. Line 2, indented once: new Str equals, left double quotation mark, space, right double quotation mark. Line 3: blank. Line 4, indented once: for ch in my String, colon. Line 5, indented twice: if ch not in new Str, colon. Line 6, indented thrice: new Str equals new Str plus ch. Line 7: blank. Line 8, indented once: return new Str.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def remove Matches, left parenthesis, my String, comma remove String, right parenthesis, colon. Line 2, indented once: new Str equals, left double quotation mark, space, right double quotation mark. Line 3: blank. Line 4, indented once: for ch in my String, colon. Line 5, indented twice: if ch not in remove String, colon. Line 6, indented thrice: new Str equals new Str plus ch. Line 7: blank. Line 8, indented once: return new Str.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from key Generation import asterisk. Input 2: import string. Input 3: alphabet equals string, dot, ascii, underscore, lowercase, plus, left double quotation mark, space, right double quotation mark. Input 4: remove Dupes, left parenthesis, left double quotation mark, top secret, right double quotation mark, right parenthesis. Output 4: left single quotation mark, top secr, right single quotation mark. Input 5: remove Dupes, left parenthesis, left double quotation mark, wonder woman, right double quotation mark, right parenthesis. Output 5: left single quotation mark, wonder ma, right single quotation mark. Input 6: remove Matches, left parenthesis, alphabet, comma, left single quotation mark, top secr, right single quotation mark, right parenthesis. Output 6: left single quotation mark, a b d f g h i j k l m n q u v w x y z, right single quotation mark. Input 8: remove Matches, left parenthesis, alphabet, comma, remove Dupes, left parenthesis, left single quotation mark, wonder woman, right single quotation mark, right parenthesis, right parenthesis. Output 8: left single quotation mark, b c f g h i j k l p q s t u v x y z, right single quotation mark.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def gen Key From Pass, left parenthesis, password, right parenthesis, colon. Line 2, indented once: import string. Line 3, indented once: alphabet equals string, dot, ascii, underscore, lowercase, plus, left double quotation mark, space, right double quotation mark. Line 4, indented once: password equals password, dot, lower, left parenthesis, right parenthesis. Line 5: blank. Line 6, indented once: password equals remove Dupes, left parenthesis, password, right parenthesis. Line 7: blank. Line 8, indented once: last Char equals password, left square bracket, negative 1, right square bracket. Line 9, indented once last Idx equals alphabet, dot, find, left parenthesis, last Char, right parenthesis. Line 10, indented once after String equals remove Matches, left parenthesis, alphabet, left square bracket, last Idx plus 1, colon, right square bracket, password, right parenthesis. Line 11, indented once before String equals remove Matches, left parenthesis, alphabet, left square bracket, colon, last Idx, right square bracket, password, right parenthesis. Line 12: blank. Line 13, indented once: key equals password plus after String plus before String. Line 14: blank. Line 15, indented once: return key.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from key Generation import asterisk. Input 2: gen Key From Pass, left parenthesis, left single quotation mark, top secret, right single quotation mark, right parenthesis. Output 2: left single quotation mark, top s e c r u v w x y z, space, a b d f g h i j k l m n q, right

single quotation mark. Input 3: gen Key From Pass, left parenthesis, left single quotation mark, wonder woman, right single quotation mark, right parenthesis. Output 3: left single quotation mark, wonder, space, m a b c f g h i j k l p q s t u v x y z, right single quotation mark.

Back to Session

The column is titled Plaintext and divided into 26 columns labeled through a to z. The row is titled Key and divided into 26 rows labeled through a to z. Row 1. a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z. Row 2. b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a. Row 3. c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b. Row 4. d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c. Row 5. e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d. Row 6. f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e. Row 7. g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f. Row 8. h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g. Row 9. i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h. Row 10. j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i. Row 11. k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j. Row 12. l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k. Row 13. m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l. Row 14. n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m. Row 15. o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n. Row 16. p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o. Row 17. q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p. Row 18. r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q. Row 19. s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q, r. Row 20. t, u, v, w, x, y, z, a, b, c, d, e, f,

g, h, I, j, k, l, m, n, o, p, q, r, s. Row 21. u, v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q, r, s, t. Row 22. v, w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q, r, s, t, u. Row 23. w, x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q, r, s, t, u, v. Row 24. x, y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q, r, s, t, u, v, w. Row 25. y, z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x. Row 26. z, a, b, c, d, e, f, g, h, I, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y.

Back to Table

The table consists of two rows. The column entries are as follows. Column1. d, t. Column 2. a, h. Column 3. v, e. Column 4. i, e. Column 5. n, a. Column 6. c, g. Column 7. i, l. Column 8. d, e. Column 9. a, h. Column 10. v, a. Column 11. i, s. Column 12. n, l. Column 13. c, a. Column 14. i, n. Column 15. d, d. Column 16. a, e. Column 17. v, d.

Back to Table

Program code. In the code, the words in the variables are merged. Line 1: from mapper import asterisk. Line 2: blank. Line 3: def Vigenere Index, left parenthesis, key Letter, comma, plain Text Letter, right parenthesis, colon. Line 4, indented once: key Index equals letter To Index, left parenthesis, key Letter, right parenthesis. Line 5, indented once: pt Index equals letter To Index, left parenthesis, plain Text Letter, right parenthesis. Line 6: blank. Line 7, indented once: new Index equals, left parenthesis, pt Index plus key Index, right parenthesis, percentage, 26. Line 8: blank. Line 9, indented once: return index To Letter, left parenthesis, new Index, right parenthesis.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: for i in range, left parenthesis, 100, right parenthesis, colon. print, left parenthesis, i percentage 7, comma, end equals, left single quotation mark, space, right single quotation mark, right parenthesis. 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6. 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6. 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6. 0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def encrypt Vigenere, left parenthesis, right parenthesis, colon. Line 2, indented once: key equals input, left parenthesis, left single quotation mark, Enter the key, colon, right single quotation mark, right parenthesis. Line 3, indented once: plain Text equals input, left parenthesis, left single quotation mark, Enter the message to be encrypted, colon, right single quotation mark, right parenthesis. Line 4, indented once: cipher Text equals, left double quotation mark, space, right double quotation mark. Line 5, indented once: key Len equals len, left parenthesis, key, right parenthesis. Line 6, indented once: k equals 0, hash, this represents which key row should be used in the table. Line 7, indented once: for ch in plain Text, colon. Line 8, indented twice: if ch equals, equals, left single quotation mark, right single quotation mark, colon. Line 9, indented thrice: hash, remember spaces are not used in the encryption. Line 10, indented thrice: cipher Text equals cipher Text plus ch. Line 11, indented twice: else, colon. Line 12, indented thrice: cipher Text equals cipher Text, backslash. Line 13, indented quarce: plus,

vigenere Index, left parenthesis, key, left square bracket, k percentage, key Len, right square bracket, comma, ch, right parenthesis. Line 14, indented thrice: k equals k plus 1. Line 15, indented once: return cipher Text.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: from vigenere import asterisk. Input 2: encrypt Vigenere, left parenthesis, right parenthesis. Enter the key, colon, davinci. Enter the message to be encrypted, colon, the eagle has landed. Output 2: left single quotation mark, w h z, space, m n i t h, space, h v a, space, y c v g e y, right single quotation mark. Input 3: encrypt Vigenere, left parenthesis, right parenthesis. Enter the key, colon, calendar. Enter the message to be encrypted, colon, one small step for man. Output 3: left single quotation mark, q n p, space, w z d l c, space, u t p t, space, s r r, space, d c n, right single quotation mark.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: my Answers equals, left double quotation mark, A B E A D B B A C E, right double quotation mark. Line 2: for answer in my Answers, colon. Line 3, indented once: print, left parenthesis, answer, right parenthesis. Line 4: A. Line 5: B. Line 6: E. Line 7: A. Line 8: D. Line 9: B. Line 10: B. Line 11: A. Line 12: C. Line 13: E.

Back to Figure

The answer sheet lists the following. Line 1: 1, dot, 34. Line 2: 2, dot, 56. Line 3: 3, dot, 2. Line 4: 4, dot, 652. Line 5: 5, dot, 26. Line 6: 6, dot, 1. Line 7: 7, dot, 99. Line 8: 8, dot, 865. Line 9: 9, dot, 22. Line 10: 10, dot, 16.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: left square bracket, 3, comma, left double quotation mark, cat, right double quotation mark, comma, 6.5, comma, 2, right square bracket, hash, define an unnamed list. Output 1: left square bracket, 3, comma, left double quotation mark, cat, right double quotation mark, comma, 6.5, comma, 2, right square bracket. Input 2: my List equals left square bracket, 3, comma, left double quotation mark, cat, right double quotation mark, comma, 6.5, comma, 2, right square bracket. Input 3: my List, hash, assign the list to a name. Output 3: left square bracket, 3, comma, left single quotation mark, cat, right single quotation mark, comma, 6.5, comma, 2, right square bracket.

Back to Session

A horizontal stack has four cells. Each reads the following. Cell 1 is labeled 0 and reads 3. Cell 2 is labeled 1 and reads left single quotation mark, cat, right single quotation mark. Cell 3 is labeled 2 and reads 6.5. Cell 4 is labeled 3 and reads 2.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: my List equals left square bracket, 3, comma, left single quotation mark, cat, right

single quotation mark, comma, 6.5, comma, 2, right square bracket. Input 2: my List left square bracket, 2, right square bracket, hash, item 2 using indexing. Output 2: 6.5. Input 3: my List plus left square bracket, left double quotation mark, read right double quotation mark, 7, right square bracket, hash, concatenate list with another list. Output 3: left square bracket, 3, comma, left single quotation mark, cat right single quotation mark, comma, 6.5, comma, 2, comma, left single quotation mark, read right single quotation mark, comma, 7, right square bracket. Input 4: my List, asterisk, 3, hash, repeat list 3 times. Output 4: left square bracket, 3, comma, left single quotation mark, cat right single quotation mark, comma, 6.5, comma, 2, comma, 3, left single quotation mark, cat right single quotation mark, 6.5, comma, 2, comma, 3, left single quotation mark, cat right single quotation mark, 6.5, comma, 2, comma, 3, right square bracket. Input 6: len, left parenthesis, my List, right parenthesis, hash, number of items in the list. Output 6: 4. Input 7: my List, left square bracket, 1, colon, 3, right square bracket, hash, slice items 1 to 2. Output 7: left square bracket, left single quotation mark, cat right single quotation mark, comma, 6.5, right square bracket. Input 8: in 3 my List, hash, test whether item is in list. Output 8: True. Input 9: left double quotation mark, dog, right double quotation mark, in my List. Output 9: False. Input 10: left double quotation mark, dog, right double quotation mark, is not in my List. Output 10: True. Input 11: del my List left square bracket, 2, right square bracket, hash, delete item 2. Input 12: my List. Output 12: left square bracket, 3, comma, left single quotation mark, cat right single quotation mark, comma, 2, right square bracket.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: change List equals, left square bracket, 1, comma, 2, comma, left double quotation mark, buckle my shoe, right double quotation mark, comma, 3, comma, 4, comma, left double quotation mark, shut the door, right double quotation mark, right square bracket. Input 2: change List. Line 3: Output 2: left square bracket, 1, comma, 2, comma, left single quotation mark, buckle my shoe, right single quotation mark, comma, 3, comma, 4, comma, left single quotation mark, shut the door, right single quotation mark, right square bracket. Input 3: change List, left square bracket, 2, right square bracket equals, left double quotation mark, the sky is blue, right double quotation mark, hash, change item 2. Input 4: change List. Output 4: left square bracket, 1, comma, 2, comma, left single quotation mark, the sky is blue, right single quotation mark, comma, 3, comma, 4, comma, left single quotation mark, shut the door, right single quotation mark, right square bracket. Input 5: name equals, left double quotation mark, Monte, right double quotation mark. Input 6: name, left square bracket, 2, right square bracket, equals, left double quotation mark, x, right double quotation mark, hash, attempt to change character in a string. Type Error Trace back, left parenthesis, most recent call last, right parenthesis. Input In, left square bracket, 6, right square bracket, in, left angular bracket, cell line, colon, 1, right angular bracket, left parenthesis, right parenthesis. Right angular bracket, 1 name left square bracket, 2, right square bracket, equals, left double quotation mark, x, right double quotation mark. Type Error, colon, left single quotation mark, str, right single quotation mark, object does not support item assignment.

Back to Session

A box labeled Change List is at the left. The change list includes a horizontal stack of six cells on the right. Cell 1 is labeled 0 and reads 1. Cell 2 is labeled 1 and reads 2. Cell 3 is labeled 2 and reads left double quotation mark, the sky is blue, right double quotation mark. A dashed arrow from cell 3 reads left double quotation mark, buckle my shoe, right double quotation mark. Cell 4 is labeled 3 and reads 3. Cell 5 is labeled 4 and reads 4. Cell 6 is labeled 5 and reads left double quotation mark, shut the door, right double quotation mark.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 3: my List equals, left square bracket, 1, comma, 2, comma, 3, comma, 4, right square bracket. Input 4: my List. Line 3: Output 4: left square bracket, 1, comma, 2, comma, 3, comma, 4, right square bracket. Input 5: list Of My List equals, left square bracket, my List, right square bracket, asterisk, 3. Input 6: list Of My List. Output 6: left square bracket, left square bracket, 1, comma, 2, comma, 3, comma, 4, right square bracket, comma, left square bracket, 1, comma, 2, comma, 3, comma, 4, right square bracket, comma, left square bracket, 1, comma, 2, comma, 3, comma, 4, right square bracket, right square bracket. Input 7: my List, left square bracket, 2, right square bracket, equals, 45, hash, change one item in my List. Line 8: Input 8: list Of My List. Output 8: left square bracket, left square bracket, 1, comma, 2, comma, 45, comma, 4, right square bracket, comma, left square bracket, 1, comma, 2, comma, 45, comma, 4, right square bracket, comma, left square bracket, 1, comma, 2, comma, 45, comma, 4, right square bracket, right square bracket.

Back to Session

A box labeled List Of My List is at the left. The List Of My List includes a horizontal stack of three cells on the right. Each cell is labeled 0, 1, and 2. Each cell points to my List which has four cells labeled 0, 1, 2, and 3. Each cell in my List points to respective cells labeled 1, 2, 3, and 4. An arrow from the cell labeled 2 of my List points to a cell reading 45.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: range, left parenthesis, 10, right parenthesis, hash, range is a sequence. Output 1: range, left parenthesis, 0, comma, 10, right parenthesis. Line 3: Input 2: list, left parenthesis, range, left parenthesis, 10, right parenthesis, right parenthesis, hash, create a list from a range. Output 2: left square bracket, 0, comma, 1, comma, 2, comma, 3, comma, 4, comma, 5, comma, 6, comma, 7, comma, 8, comma, 9, right square bracket. Input 3: list, left parenthesis, range, left parenthesis, 10, comma, 2, comma, negative 2, right parenthesis, right parenthesis. Output 3: left square bracket, 10, comma, 8, comma, 6, comma, 4, right square bracket. Input 4: list, left parenthesis, left double quotation mark, the quick fox, right left double quotation mark, right parenthesis, hash, create a list from a string. Line 8: Output 4: left square bracket, left single quotation mark, t, right single quotation mark, comma, left single quotation mark, h, right single quotation mark, comma, left single quotation mark, e, right single quotation mark, comma, left single quotation mark, space, right single quotation mark, comma, left single quotation mark, q, right single quotation mark, comma, left single quotation mark, u, right single

quotation mark, comma, left single quotation mark, i, right single quotation mark, comma, left single quotation mark, c, right single quotation mark, comma, left single quotation mark, k, right single quotation mark, comma, left single quotation mark, space, right single quotation mark, comma, left single quotation mark, f, right single quotation mark, comma, left single quotation mark, o, right single quotation mark, comma, left single quotation mark, x, right single quotation mark, right square bracket.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, my List equals list, left parenthesis, left parenthesis, 1024, comma, 3, comma, True, comma, 6.5, right parenthesis, right parenthesis. Input 2: 1, my List. Line 3: Output 2: left square bracket, 1024, comma, 3, comma, True, comma, 6.5, right square bracket. Input 3: 1, my List, dot, append, left parenthesis, False, right parenthesis, hash, add False to end of list. Input 4: 1, my List. Output 4: left square bracket, 1024, comma, 3, comma, True, comma, 6.5, comma, False, right square bracket. Input 5: 1, my List, dot, insert, left parenthesis, 2, comma, 4.5, right parenthesis, hash, insert 4.5 at index 2. Line 8: Input 6: 1, my List. Output 6: left square bracket, 1024, comma, 3, comma, 4.5, comma, True, comma, 6.5, comma, False, right square bracket. Input 7: 1, my List, dot, pop, left parenthesis, right parenthesis, hash, remove and return last item. Output 7: False. Input 8: 1, my List. Output 9: left square bracket, 1024, comma, 3, comma, 4.5, comma, True, comma, 6.5, right square bracket. Input 9: 1, my List, dot, pop, left parenthesis, 3, right parenthesis, hash, remove and return item at index 3. Output 9: True.

The window displays the following program code. In the code, the words in the variables are merged. Input 10: 1, my List. Output 10: left square bracket, 1024, comma, 3, comma, 4.5, comma, 6.5, right square bracket. Input 11: 1, my List, dot, sort, left parenthesis, right parenthesis, hash, sort the list. Input 12: 1, my List. Output 12: 1, left square bracket, 3, comma, 4.5, comma, 6.5, comma, 1024, right square bracket. Input 13: 1, my List, dot, reverse, left parenthesis, right parenthesis, hash, reverse the order of the list. Input 14: 1, my List. Output 14: 1, left square bracket, 1024, comma, 6.5, comma, 4.5, comma, 3, right square bracket. Input 15: 1, my List, dot, count, left parenthesis, 6.5, right parenthesis, hash, count the occurrences of 6.5. Output 15: 1. Input 16: 1, my List, dot, index, left parenthesis, 4.5, right parenthesis, hash, return index of 4.5. Output 16: 2. Input 17: 1, my List, dot, remove, left parenthesis, 6.5, right parenthesis, hash, delete 6.5. Input 18: my List. Output 18: 1, left square bracket, 1024, comma, 4.5, comma, 3, right square bracket. Input 19: my List, dot, clear, left parenthesis, right parenthesis, hash, delete all items. Input 20: my List. Output 20: left square bracket, right square bracket.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, team equals, left double quotation mark, Minnesota Vikings, right left double quotation mark. Input 2: 1, team, dot, split, left parenthesis, right parenthesis, hash, split at spaces or tabs. Output 2: left square bracket, left single quotation mark, Minnesota, right single quotation mark, comma, left single quotation mark, Vikings, right single quotation mark, right square bracket. Input 3: 1, team, dot, split, left parenthesis, left single quotation mark, i, right single

quotation mark, right parenthesis, hash, split using i as delimiter. Output 3: left square bracket, left single quotation mark, M, right single quotation mark, comma, left single quotation mark, nnesota V, right single quotation mark, comma, left single quotation mark, k, right single quotation mark, comma, left single quotation mark, ngs, right single quotation mark, right square bracket. Input 4: 1, team, dot, split, left parenthesis, left single quotation mark, n n, right single quotation mark, right parenthesis, hash, split using n n as delimiter. Output 4: left square bracket, left single quotation mark, Mi, right single quotation mark, comma, left single quotation mark, esota Vikings, right single quotation mark, right square bracket.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, earthquakes equals, left square bracket, 37, comma, 32, comma, 46, comma, 28, comma, 37, comma, 41, comma, 31, right square bracket. Input 2: 1, max, left parenthesis, earthquakes, right parenthesis. Line 3: Output 2: 46. Input 3: 1, min, left parenthesis, earthquakes, right parenthesis. Output 3: 28. Input 4: 1, max, left parenthesis, earthquakes, right parenthesis, minus min, left parenthesis, earthquakes, right parenthesis. Output 4: 18. Input 5: 1, max, left parenthesis, left double quotation mark, house, right double quotation mark, right parenthesis, hash, max, forward slash, min also work on characters in strings. Output 5: left single quotation mark, u, right single quotation mark. Input 6: 1, min, left parenthesis, left double quotation mark, house, right double quotation mark, right parenthesis. Output 6: left single quotation mark, e, right single quotation mark.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def get Range, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: return max, left parenthesis, a List, right parenthesis, minus min, left parenthesis, a List, right parenthesis.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from get Range import asterisk. Input 2: 1, get Range, left parenthesis, left square bracket, 2, comma, 5, right square bracket, right parenthesis. Output 2: 3. Input 3: 1, earthquakes equals, left square bracket, 37, comma, 32, comma, 46, comma, 28, comma, 37, comma, 41, comma, 31, right square bracket. Input 4: get Range, left parenthesis, earthquakes, right parenthesis. Output 4: 18.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def get Max By Index, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: max So Far equals a List, left square bracket, 0, right square bracket. Line 3, indented once: for pos in range, left parenthesis, 1, comma, len, left parenthesis, a List, right parenthesis, right parenthesis, colon. Line 4, indented twice: if a List, left square bracket, pos, right square bracket, greater than, max So Far, colon. Line 5, indented thrice: max So Far equals a List, left square bracket, pos, right square bracket. Line 6: blank. Line 7, indented once: return max So Far.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def get Max By Index, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: max So Far equals a List, left square bracket, 0, right square bracket. Line 3, indented once: for item in a List, left square bracket, 1, colon, right square bracket, colon. Line 4, indented twice: if item greater than max So Far, colon. Line 5, indented thrice: max So Far equals item. Line 6: blank. Line 7, indented once: return max So Far.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def mean, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: mean equals sum, left parenthesis, a List, right parenthesis, forward slash, len, left parenthesis, a List, right parenthesis. Line 3, indented once: return mean.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from stats functions import asterisk, hash, our mean function is in statsfunctions, dot, py. Input 2: 1, earthquakes equals, left square bracket, 37, comma, 32, comma, 46, comma, 28, comma, 37, comma, 41, comma, 31, right square bracket. Line 3: Input 3: 1, mean, left parenthesis, earthquakes, right parenthesis, hash, call our mean function. Output 3: 36.0. Input 4: 1, import statistics. Input 5: 1, statistics, dot, mean, left parenthesis, earthquakes,

right parenthesis, hash, call the statistics mean function. Output 5: 36.

Back to Session

The diagram has two horizontal stacks one below the other. The stack on the top has seven cells. Cell 1 is labeled 0 and reads 2. Cell 2 is labeled 1 and reads 5. Cell 3 is labeled 2 and reads 8. Cell 4 is labeled 3 and reads 9. Cell 5 is labeled 4 and reads 10. Cell 6 is labeled 5 and reads 15. Cell 7 is labeled 6 and reads 24. Cell 3 which reads 9 is highlighted. The right reads The median is 9. The stack on the bottom has eight cells. Cell 1 is labeled 0 and reads 2. Cell 2 is labeled 1 and reads 5. Cell 3 is labeled 2 and reads 8. Cell 4 is labeled 3 and reads 9. Cell 5 is labeled 4 and reads 10. Cell 6 is labeled 5 and reads 15. Cell 7 is labeled 6 and reads 24. Cell 8 is labeled 7 and reads 54. Cells 3 and 4 which read 9 and 10 are highlighted. The right reads The median is 9.5.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, a List equals, left square bracket, 24, comma, 5, comma, 8, comma, 2, comma, 9, comma, 15, comma, 10, right square bracket, hash, odd number if items. Input 2: 1, a List, dot, sort, left parenthesis, right parenthesis, hash, arrange list in order. Input 3: 1, a List. Output 3: left square bracket, 2, comma, 5, comma, 8, comma, 2, comma, 9, comma, 10, comma, 15, comma, 24, right square bracket. Input 4: 1, len, left parenthesis, a List, right parenthesis, hash, number of items in a List. Output 4: 7. Input 5: 1, len, left parenthesis, a List, right parenthesis, forward slash, forward slash, 2, hash, find middle item's index. Output 5: 3. Input 6: 1, a

List left square bracket, 3, right square bracket, hash, item at 3 is median. Output 6: 9. Input 7: 1, b List equals, left square bracket, 2, comma, 5, comma, 8, comma, 2, comma, 9, comma, 10, comma, 15, comma, 24, comma, 54, right square bracket, hash, even number of sorted items. Input 8: 1, len, left parenthesis, b List, right parenthesis. Output 8: 8. Input 9: 1, len, left parenthesis, b List, right parenthesis, forward slash, forward slash, 2, hash, index of right middle item. Output 9: 4. Input 10: 1, b List left square bracket, 4, right square bracket, hash, higher of two middle items. Output 10: 10. Input 11: 1, b List left square bracket, 3, right square bracket, hash, lower of two middle items. Output 11: 9. Input 12: 1, average equals, left parenthesis, b List, left square bracket, 4, right square bracket, plus, b List, left square bracket, 3, right square bracket, right parenthesis, forward slash, 2, hash, median is average. Input 13: 1, average. Output 13: 9.5.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def median, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: copy List equals a List, dot, copy, left parenthesis, right parenthesis. Line 3, indented once: copy List, dot, sort, left parenthesis, right parenthesis, hash, sort the copy. Line 4, indented once: if len, left parenthesis, copy List, right parenthesis, percentage 2 equals, equals, 0, colon, hash, even length, question mark. Line 5, indented twice: right Mid equals, len, left parenthesis, copy List, right parenthesis, forward slash, forward slash, 2. Line 6, indented twice: left Mid equals right Mid minus 1. Line 7, indented twice: median equals, left parenthesis, copy List, left square bracket, left Mid, right square bracket, plus copy List, left square bracket, right Mid,

right square bracket, right parenthesis, forward slash, 2. Line 8, indented once: else, colon, hash, odd length. Line 9, indented twice: mid equals len, left parenthesis, copy List, right parenthesis, forward slash, forward slash, 2. Line 10, indented twice: median equals copy List, left square bracket mid, right square bracket. Line 11, indented once: return median.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from stats functions import asterisk, hash, import our median function. Input 2: 1, import statistics. Input 3: 1, earthquakes 7 equals, left square bracket, 37, comma, 32, comma, 46, comma, 28, comma, 37, comma, 41, comma, 31, right square bracket, hash 7 items. Input 4: 1, earthquakes 7 equals, left square bracket, 37, comma, 32, comma, 46, comma, 28, comma, 37, comma, 41, comma, 31, comma, 29, right square bracket, hash 8 items. Input 5: median, left parenthesis, earthquakes 7, right parenthesis. Output 5: 37. Input 6: 1, statistics, dot, median, left parenthesis, earthquakes 7, right parenthesis. Output 6: 37. Input 7: 1, statistics, dot, median, left parenthesis, earthquakes 8, right parenthesis. Output 7: 34.5. Input 8: 1, statistics, dot, median, left parenthesis, earthquakes 8, right parenthesis. Output 8: 34.5.

Back to Session

Python collections are split into Sequential collections and Non-sequential collections. Sequential collections are split into Strings and Lists. Non-sequential collections list Dictionary.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, parents equals, left brace, left single quotation mark, David, right single quotation mark, colon, 45, comma, left single quotation mark, Brenda, right single quotation mark, colon, 46, right brace, hash, create dict with 2 items. Input 2: 1, parents. Output 2: left brace, left single quotation mark, David, right single quotation mark, colon, 45, comma, left single quotation mark, Brenda, right single quotation mark, colon, 46, right brace. Input 3: 1, parents, left square bracket, left single quotation mark, David, right single quotation mark, right square bracket, equals, 46, hash, change value using key. Input 4: 1, parents, left square bracket, left single quotation mark, David, right single quotation mark, right square bracket. Output 4: 46. Line 7: Input 5: 1, children equals, left brace, left single quotation mark, Kelsey, right single quotation mark, colon, 19, comma, left single quotation mark, Hannah, right single quotation mark, colon, 16, right brace, hash, create a second dict. Input 6: 1, children, left square bracket, left single quotation mark, Rylea, right single quotation mark, right square bracket, equals, 7. Input 7: 1, children. Output 7: left brace, left single quotation mark, Kelsey right single quotation mark, colon, 19, comma, left single quotation mark, Hannah, right single quotation mark, colon, 16, comma, left single quotation mark, Rylea right single quotation mark, colon, 7, right brace. Input 8: 1, family equals parents vertical bar children, hash, merge 2 dicts. Input 9: 1, family. Output 9: left brace, left single quotation mark, David, right single quotation mark, colon, 46, comma, left single quotation mark, Brenda, right single quotation mark, colon, 46, comma, 'Kelsey': 19, left single quotation mark, Kelsey right single quotation mark, colon, 19, comma, left single quotation mark, Hannah, right single quotation mark, colon, 16, comma, left single quotation mark, Rylea right single quotation mark,

colon, 7, right brace. Input 10: 1, len, left parenthesis, family, right parenthesis, hash, len is number of pairs. Output 10: 5. Input 11: del family, left square bracket, left single quotation mark, David, right single quotation mark, right square bracket. Input 12: family. Output 12: left brace, left single quotation mark, Brenda, right single quotation mark, colon, 46, comma, 'Kelsey': 19, left single quotation mark, Kelsey right single quotation mark, colon, 19, comma, left single quotation mark, Hannah, right single quotation mark, colon, 16, comma, left single quotation mark, Rylea right single quotation mark, colon, 7, right brace. Input 13: 1, left single quotation mark, David, right single quotation mark in family. Output 13: False.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, family equals, left brace, left single quotation mark, David, right single quotation mark, colon, 45, comma, left single quotation mark, Brenda, right single quotation mark, colon, 46, comma, left single quotation mark, Kelsey, right single quotation mark, colon, 19, comma, left single quotation mark, Hannah, right single quotation mark, colon, 16, comma, left single quotation mark, Rylea, right single quotation mark, colon, 7, right brace. Input 2: 1, family. Output 2: left brace, left single quotation mark, David, right single quotation mark, colon, 45, comma, left single quotation mark, Brenda, right single quotation mark, colon, 46, comma, left single quotation mark, Kelsey, right single quotation mark, colon, 19, comma, left single quotation mark, Hannah, right single quotation mark, colon, 16, comma, left single quotation mark, Rylea, right single quotation mark, colon, 7, right brace. Input 3: 1, family, dot, keys, left parenthesis, right parenthesis, hash,

request keys. Output 3: dict, underscore, keys, left parenthesis, left square bracket, left single quotation mark, David, right single quotation mark, comma, left single quotation mark, Brenda, right single quotation mark, comma, left single quotation mark, Kelsey, right single quotation mark, comma, left single quotation mark, Hannah, right single quotation mark, comma, left single quotation mark, Rylea, right single quotation mark, right square bracket, right parenthesis. Input 4: 1, list, left parenthesis, family, dot, keys, left parenthesis, right parenthesis, right parenthesis, hash, create list of keys. Output 4: left square bracket, left single quotation mark, David, right single quotation mark, comma, left single quotation mark, Brenda, right single quotation mark, comma, left single quotation mark, Kelsey, right single quotation mark, comma, left single quotation mark, Hannah, right single quotation mark, comma, left single quotation mark, Rylea, right single quotation mark, right square bracket. Input 5: 1, list, left parenthesis, family, dot, values, left parenthesis, right parenthesis, right parenthesis, hash, create list of values. Output 5: left square bracket, 45, comma, 46, comma, 19, comma, 16, comma, 7, right square bracket. Input 6: 1, list, left parenthesis, family, dot, items, left parenthesis, right parenthesis, right parenthesis, hash, create list of items. Output 6: left square bracket, left parenthesis, left single quotation mark, David, right single quotation mark, comma, 45, right parenthesis, comma, left parenthesis, left single quotation mark, Brenda, right single quotation mark, comma, 46, right parenthesis, comma, left parenthesis, left single quotation mark, Kelsey, right single quotation mark, comma, 19, right parenthesis, comma, left parenthesis, left single quotation mark, Hannah, right single quotation mark, comma, 16, right parenthesis, comma, left parenthesis, left single quotation mark, Rylea, right single quotation mark, comma, 7, right parenthesis, right square bracket. Input 7: 1, family, dot, get, left parenthesis,

left single quotation mark, Lena, right single quotation mark, comma, left single quotation mark, No item, right single quotation mark, right parenthesis, hash, get key with default value. Output 7: left single quotation mark, No item, right single quotation mark. Input 8: 1, for name in family, dot, keys, left parenthesis, right parenthesis, colon, hash, iterate through keys. 2, print, left parenthesis, name, right parenthesis. 3, blank. David. Brenda. Kelsey. Hannah. Rylea. Input 9: 1, for name in family, colon, hash, also iterate through keys. 2, print, left parenthesis, name, right parenthesis. 3, blank. David. Brenda. Kelsey. Hannah. Rylea. Input 10: 1, for item in reversed, left parenthesis, family, dot, items, left parenthesis, right parenthesis, right parenthesis, colon, hash, iterate through items. 2, print, left parenthesis, item, right parenthesis. 3, blank. left parenthesis, left single quotation mark, Rylea, right single quotation mark, comma, 7, right parenthesis. left parenthesis, left single quotation mark, Hannah, right single quotation mark, comma, 16, right parenthesis. left parenthesis, left single quotation mark, Kelsey, right single quotation mark, comma, 19, right parenthesis. left parenthesis, left single quotation mark, Brenda, right single quotation mark, comma, 46, right parenthesis. left parenthesis, left single quotation mark, David, right single quotation mark, comma, 45, right parenthesis. Input 11: 1, family, dot, clear, left parenthesis, right parenthesis, hash, delete all items. Input 12: family. Output 12: left brace, right brace.

Back to Session

The list reads: names equals open bracket joe within single quotes, tom within single quotes, barb within single quotes, sue within single quotes, sally within single quotes close bracket. scores equals open bracket 10, 23, 13, 18, 12 close bracket.

Back to Code

Program code. In the code, the words in the variables are merged. Line 1: def mode, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: count Dict equals, left brace, right brace. Line 3, indented once: for item in a List, colon. Line 4, indented twice: if item in count Dict, colon. Line 5, indented thrice: count Dict, left square bracket, item, right square bracket equals count Dict, left square bracket, item, right square bracket, plus 1. Line 6, indented twice: else, colon. Line 7, indented trice: count Dict, left square bracket, item, right square bracket, equals 1. Line 8, indented once: hash, more code to come.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def mode, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: count Dict equals, left brace, right brace. Line 3, indented once: for item in a List, colon. Line 4, indented twice: if item in count Dict, colon. Line 5, indented thrice: count Dict, left square bracket, item, right square bracket equals count Dict, left square bracket, item, right square bracket, plus 1. Line 6, indented twice: else, colon. Line 7, indented trice: count Dict, left square bracket, item, right square bracket, equals 1. Line 8: blank. Line 9, indented once: count List equals count Dict, dot, values, left parenthesis, right parenthesis. Line 10, indented once: max Count equals max, left parenthesis, count List, right parenthesis, hash, find the maximum count. Line 11: blank. Line 12, indented once: mode List equals, left square bracket, right square bracket. Line 13, indented once: for item in count Dict, colon. Line 14, indented twice: if count Dict, left

square bracket, item, right square bracket, equals, equals, max Count, colon. Line 15, indented thrice: mode List, dot, append, left parenthesis, item, right parenthesis. Line 16, indented once: return mode List.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from stats functions import, asterisk, hash, import our mode function. Input 2: 1, mode, left parenthesis, left square bracket, 1, comma, 1, comma, 4, comma, 5, comma, 6, comma, 2, comma, 4, comma, 7, comma, 1, comma, 4, comma, 6, comma, 1, right square bracket, right parenthesis. Output 2: left square bracket, 1, right square bracket. Input 3: 1, mode, left parenthesis, left square bracket, 1, comma, 1, comma, 1, comma, 2, comma, 2, comma, 2, comma, 3, comma, 3, comma, 3, comma, 4, comma, 4, comma, 4, comma, 4, comma, 5, comma, 5, comma, 5, comma, 5, comma, 6, comma, 6, right square bracket, right parenthesis. Output 3: left square bracket, 4, comma, 5, right square bracket. Input 4: 1, equals, left square bracket, 37, comma, 32, comma, 46, comma, 28, comma, 37, comma, 41, comma, 31, right square bracket. Input 5: 1, mode, left parenthesis, earthquakes, right parenthesis. Output 5: left square bracket, 37, right square bracket. Input 6: 1, import statistics, hash, use multimode function in statistics module. Input 7: 1, statistics, dot, multimode, left parenthesis, left square bracket, 1, comma, 1, comma, 4, comma, 5, comma, 6, comma, 2, comma, 4, comma, 7, comma, 1, comma, 4, comma, 6, comma, 1, right square bracket, right parenthesis. Output 7: left square bracket, 1, right square bracket. Input 8: 1, statistics, dot, multimode, left parenthesis, left square bracket, 1, comma, 1, comma, 1, comma,

2, comma, 2, comma, 2, comma, 3, comma, 3, comma, 3, comma, 4, comma, 4, comma, 4, comma, 4, comma, 5, comma, 5, comma, 5, comma, 5, comma, 6, comma, 6, right square bracket, right parenthesis. Input 8: left square bracket, 4, comma, 5, right square bracket.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def frequency Table, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: count Dict equals, left brace, right brace. Line 3: blank. Line 4, indented once: for item in a List, colon. Line 5, indented twice: if item in count Dict, colon. Line 6, indented thrice: count Dict, left square bracket, item, right square bracket, equals count Dict, left square bracket, item, right square bracket, plus 1. Line 7, indented twice: else, colon. Line 8, indented thrice: count Dict, left square bracket, item, right square bracket, equals 1. Line 9: blank. Line 10, indented once: item List equals list, left parenthesis, count Dict, dot, keys, left parenthesis, right parenthesis, right parenthesis. Line 11, indented once: item List, dot, sort, left parenthesis, right parenthesis. Line 12: blank. Line 13, indented once: print, left parenthesis, left double quotation mark, ITEM, right double quotation mark, comma, left double quotation mark FREQUENCY, right double quotation mark, right parenthesis. Line 14: blank. Line 15, indented once: for item in item List, colon. Line 16, indented twice: print, left parenthesis, item, comma, left double quotation mark, space, right double quotation mark, comma, count Dict, left square bracket, item, right square bracket, right parenthesis.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from freqTables import asterisk. Input 2: 1, frequency Table, left parenthesis, left square bracket, 3, comma, 1, comma, 1, comma, 5, comma, 3, comma, 1, comma, 2, comma, 2, comma, 3, comma, 5, comma, 3, comma, 5, comma, 4, comma, 4, comma, 6, comma, 7, comma, 6, comma, 7, comma, 5, comma, 7, comma, 8, comma, 3, comma, 8, comma, 2, comma, 3, comma, 4, comma, 1, comma, 5, comma, 6, comma, 7, right square bracket, right parenthesis. ITEM FREQUENCY. 1, space, 4. 2, space, 3. 3, space, 6. 4, space, 3. 5, space, 5. 6, space, 3. 7, space, 4. 8, space, 2.

Back to Session

A horizontal stack has fourteen cells and reads move current and previous from the left to right. Cell 1 reads 2. Cell 2 reads 2. Cell 3 reads 4 and is labeled previous. Cell 4 reads 4 and is labeled current. Cell 5 reads 4. Cell 6 reads 5. Cell 7 reads 6. Cell 8 reads 6. Cell 9 reads 6. Cell 10 reads 6 and is labeled previous. Cell 11 reads 9 and is labeled current. Cell 12 reads 9. Cell 13 reads 9. Cell 14 reads 10.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def frequency Table Alt, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: print, left parenthesis, left double quotation mark, ITEM, right double quotation mark, comma, left double quotation mark, FREQUENCY, right double quotation mark, right parenthesis, hash, print table headers. Line 3: blank. Line 4, indented once:

sorted List equals a List, dot, copy, left parenthesis, right parenthesis. Line 5, indented once: sorted List, dot, sort, left parenthesis, right parenthesis. Line 6: blank. Line 7, indented once: previous equals sorted List, left square bracket, 0, right square bracket. Line 8, indented once: group Count equals 0. Line 9, indented once: for current in sorted List, colon. Line 10, indented twice: if current equals, equals, previous, colon, hash, same value as group. Line 11, indented thrice: group Count equals group Count plus 1, hash, count the item. Line 12, indented twice: else, colon, hash, new group detected. Line 13, indented thrice: print, left parenthesis, previous, comma, left double quotation mark, space, right double quotation mark, comma, group Count, right parenthesis. Line 14, indented thrice: previous equals current. Line 15, indented thrice: group Count equals 1. Line 16, indented once: print, left parenthesis, current, comma, left double quotation mark, space, right double quotation mark, comma, group Count, right parenthesis, hash, print last group.

Back to Listing

The x-axis is labeled value and ranges from 0 to 8. The y-axis is labeled frequency and ranges from 0 to 6. Data from the graph are approximately as follows. Bar 1: (1, 4). Bar 2: (2, 3). Bar 3: (3, 6). Bar 4: (4, 3), Bar 5: (5, 5). Bar 6: (6, 3), Bar 7: (7, 4). Bar 8: (8, 2.5).

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: import mat plot lib, dot, pyplot as plt. Line 2: blank. Line 3: def frequency Chart, left parenthesis, a List, right

parenthesis, colon. Line 4: blank. Line 5, indented once: count Dict equals, left brace, right brace. Line 26 blank. Line 7, indented once: for item in a List, colon. Line 8, indented twice: if item in count Dict, colon. Line 9, indented thrice: count Dict, left square bracket, item, right square bracket, equals count Dict, left square bracket, item, right square bracket, plus 1. Line 10, indented twice: else, colon. Line 11, indented thrice: count Dict, left square bracket, item, right square bracket, equals, 1. Line 12: blank. Line 13, indented once: keys equals list, left parenthesis, count Dict, dot, keys, left parenthesis, right parenthesis, right parenthesis, hash, the x-axis. Line 14, indented once: values equals list, left parenthesis, count Dict, dot, values, left parenthesis, right parenthesis, right parenthesis, hash, the y-axis. Line 15: blank. Line 16, indented once: plt, dot, bar, left parenthesis, x equals keys, comma, height equals values, right parenthesis. Line 17, indented once: plt, dot, x label, left parenthesis, left double quotation mark, Number, right double quotation mark, right parenthesis. Line 18, indented once: plt, dot, y label, left parenthesis, left double quotation mark, Frequency, right double quotation mark, right parenthesis. Line 19, indented once: plt, dot, show, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def standard Dev, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: import statistics as stats, comma, math. Line 3, indented once: the Mean equals stats, dot, mean, left parenthesis, a List, right parenthesis. Line 4: blank. Line 5, indented once: total equals 0. Line 6, indented once: for item in a List, colon. Line 7, indented twice: difference

equals item minus the Mean, hash, subtract from mean. Line 8, indented twice: diffSq equals difference asterisk, asterisk, 2, hash, square the difference. Line 9, indented twice: total equals total plus diffSq, hash, add to running total. Line 10: blank. Line 11, indented once: s Dev equals math, dot, sqrt, left parenthesis, total forward slash, left parenthesis, len, left parenthesis, a List, right parenthesis minus 1, right parenthesis, right parenthesis. Line 12, indented once: return s Dev.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: 1, from stats functions import standard Dev, comma, mean. Input 2: 1, data List equals, left square bracket, 7, comma, 11, comma, 9, comma, 18, comma, 15, comma, 12, right square bracket. 2, mean, left parenthesis, data List, right parenthesis. Output 2: 12.0. Input 3: 1, standard Dev, left parenthesis, data List, right parenthesis. Output 3: 4.0. Input 4: 1, data List 2 equals, left square bracket, 2, comma, 10, comma, 6, comma, 24, comma, 18, comma, 12, right square bracket. 2, mean, left parenthesis, data List 2, right parenthesis. Output 4: 12.0. Input 5: 1, standard Dev, left parenthesis, data List 2, right parenthesis. Output 5: 8.0. Input 6: 1, earthquakes equals, left square bracket, 37, comma, 32, comma, 46, comma, 28, comma, 37, comma, 41, comma, 31, right square bracket. 2, standard Dev, left parenthesis, earthquakes, right parenthesis. Output 6: 6.2182527020592095. Input 7: 1, import statistics. 2, statistics, dot, stdev, left parenthesis, earthquakes, right parenthesis. Output 7: 6.2182527020592095. Input 8: 1, statistics, dot, stdev, left parenthesis, earthquakes, comma, statistics, dot, mean, left

parenthesis, earthquakes, right parenthesis, right parenthesis. Output 8: 6.2182527020592095.

Back to Session

Line 1. p l t dot scatter open parenthesis x equals x Data, y equals y Data close parenthesis. Line 2. p l t dot a x h line open parenthesis y = theMean close parenthesis. Line 3. p l t dot show open parenthesis close parenthesis.

Back to Figure

The screen displays the following. Akron 25.81. Albia 37.65. Algona 30.69. Allison 33.64. Alton 27.43. Ames W 34.07. Ames S E 33.95. Anamosa 35.33. Ankeny 33.38. Atlantic 34.77. Audubon 33.41. Beaconsfield 35.27. Bedford 36.35. Belle Plaine 35.81. Bellevue 34.35. Blockton 36.28. Bloomfield 38.02. Boone 36.30. Brighton 33.59. Britt 31.54. Buckeye 33.66. Burlington K B U R 37.94. Burlington 36.94. Carroll 33.33. Cascade 33.48.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: Line 1. Hash, option 1, colon, open, comma, then close file. Line 2. file R e f, equals, open, left parenthesis, left double quotation mark, rainfall, dot, t x t, right double quotation mark, comma, left double quotation mark, r, right double quotation mark, right parenthesis. Line 3. Blank. Line 4. Hash, read and process the data in the file. Line 5. Blank. Line 6. file R e f, dot, close, left parenthesis, right parenthesis. Line 1. Hash, option 2, colon, use with, Forward slash, as block. Line 2. with open, left parenthesis, left double quotation mark, rainfall dot t x t, right double quotation mark, comma, left double quotation

mark, r, right double quotation mark, right parenthesis, as file R e f, colon. Line 3. Blank. Line 4. hash, read and process data in the file. Line 5. hash, file is automatically closed when block finishes.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: for line in inFile, colon. Line 2, indented once: statement 1. Line 3, indented once: statement 2. Line 4, indented once: ellipses.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1. with open, left parenthesis, left double quotation mark, rainfall dot t x t, right double quotation mark, comma, left double quotation mark, r, right double quotation mark, right parenthesis, as rain File, colon. Line 2, indented once: for a Line in rain File, colon. Line 3, indented twice: values equals a Line dot split, left parenthesis, right parenthesis. Line 4, indented twice: print, left parenthesis, values, left square bracket, 0, right square bracket, comma, left double quotation mark, had, right double quotation mark, comma, values, left square bracket, 1, right square bracket, comma, left double quotation mark, inches of rain, right double quotation mark, right parenthesis. Line 5. Blank. The following lines display the output. Akron had 25.81 inches of rain. Albia had 37.65 inches of rain. Algona had 30.69 inches of rain. Allison had 33.64 inches of rain. Alton had 27.43 inches of rain. Ames W had 34.07 inches of rain. Ames S E had 33.95 inches of rain. Anamosa had 35.33 inches of rain. Ankeny had 33.38 inches of rain. Atlantic had 34.77 inches of rain.

Audubon had 33.41 inches of rain. Beaconsfield had 35.27 inches of rain. Bedford had 36.35 inches of rain. Belle Plaine had 35.81 inches of rain. Bellevue had 34.35 inches of rain. Blockton had 36.28 inches of rain. Bloomfield had 38.02 inches of rain. Boone had 36.30 inches of rain. Brighton had 33.59 inches of rain. Britt had 31.54 inches of rain. Buckeye had 33.66 inches of rain. Burlington K B U R had 37.94 inches of rain. Burlington had 36.94 inches of rain. Carroll had 33.33 inches of rain. Cascade had 33.48 inches of rain.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: with open, left parenthesis, left double quotation mark, rainfall dot t x t, right double quotation mark, comma, left double quotation mark, r, right double quotation mark, right parenthesis, as rain File, colon. Line 2, indented once: with open, left parenthesis, left double quotation mark, rainFall In C M.t x t, right double quotation mark, comma, left double quotation mark, w, right double quotation mark, right parenthesis, as out File, colon. Line 3, indented twice: for a Line in rain File, colon. Line 4, indented thrice: values, equals a Line dot split, left parenthesis, right parenthesis. Line 5: Blank. Line 6, indented thrice: inches, equals, float, left parenthesis, values, left square bracket, 1, right square bracket, right parenthesis. Line 7, indented thrice: c m, equals, 2.54, asterisk, inches, Hash, convert to C Ms. Line 8: Blank. Line 9, indented thrice: n Chars, equals, out File dot write, left parenthesis, values, left square bracket, 0, right square bracket, plus, left double quotation mark, space, right double quotation mark, backslash, plus, s t r, left parenthesis, c m, right parenthesis, plus, left double quotation mark, backslash, n, right double quotation mark, right parenthesis.

Back to Listing

The screen displays the following. Akron 65.5574. Albia 95.631. Algona 77.9526. Allison 85.4456. Alton 69.6722. Ames W 86.5378. Ames S E 86.233. Anamosa 89.73819999999999. Ankeny 84.7852. Atlantic 88.31580000000001. Audubon 84.86139999999999. Beaconsfield 89.5858. Bedford 92.32900000000001. Belle Plaine 90.9574. Bellevue 87.24900000000001. Blockton 92.1512. Bloomfield 96.5708. Boone 92.202. Brighton 85.3186. Britt 80.1116. Buckeye 85.4964. Burlington K B U R 96.3676. Burlington 93.82759999999999. Carroll 84.6582. Cascade 85.0392.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 6: Line 1: Hash, format item price to 2 decimal points. Line 2: item, Equals, left single quotation mark, a dozen eggs, right single quotation mark. Line 3: item Price, Equals, 2.4. Line 4: print, left parenthesis, left double quotation mark, The price of, left brace, 0, right brace, is, dollar, left brace, 1, colon, .2 f, right brace, right double quotation mark, dot, format, left parenthesis, item, comma, item Price, right parenthesis, right parenthesis. Line 5: The price of a dozen eggs is, dollar, 2.40. Input 8: Line 1: Hash, format dict values using keys, note, asterisk, asterisk. Line 2: my Dict, Equals, left brace, left single quotation mark, name, right single quotation mark, colon, left single quotation mark, candy bar, right single quotation mark, comma, left single quotation mark, price, right single quotation mark, colon, 95, right brace. Line 3: print, left parenthesis, left double quotation mark, The, left brace, name, right brace, costs, left brace, price, right brace, cents, right

double quotation mark, dot format, left parenthesis, asterisk, asterisk, my Dict, right parenthesis, right parenthesis. Line 4: The candy bar costs 95 cents. Input 10: Line 1: Hash, align center, right, and left. Line 2: print, left parenthesis, left double quotation mark, left brace, 0, colon, caret, 40 s, right brace, right double quotation mark, dot format, left parenthesis, left double quotation mark, This text is centered, right double quotation mark, right parenthesis, right parenthesis. Line 3: print, left parenthesis, left double quotation mark, left brace, 0, colon, right angle bracket, 40 s, right brace, right double quotation mark, dot format, left parenthesis, left double quotation mark, This text is right justified, right double quotation mark, right parenthesis, right parenthesis. Line 4: print, left parenthesis, left double quotation mark, left brace, 0, colon, left angle bracket, 40 s, right brace, right double quotation mark, dot format, left parenthesis, left double quotation mark, This text is left justified, right double quotation mark, right parenthesis, right parenthesis. Line 5, indented twice: This text is centered. Line 6, indented thrice: This text is right justified. Line 7: This text is left justified. Input 11: Line 1: Hash, format percentage. Line 2: quiz Grade, Equals, 7.5. Line 3: total Points, Equals, 12. Line 4: print, left parenthesis, left double quotation mark, left brace, 0, colon, .1 F, right brace, is, left brace, 1, colon, .2, percent, right brace, of, left brace, 2, colon, d, right brace, total points, right double quotation mark, dot format, left parenthesis, quiz Grade, comma, quiz Grade, Forward slash, total Points, comma, total Points, right parenthesis, right parenthesis. Line 5: blank. Line 6: 7.5 is 62.50, percent, of 12 total points.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: with open, left parenthesis, left single quotation mark, rainfall dot t x t, right single quotation mark, comma, left single quotation mark, r, right single quotation mark, right parenthesis, as in File, colon. Line 2, indented once: a Line, Equals, in File dot read line, left parenthesis, right parenthesis, Hash, read one line. Input 2: Line 1: a Line. Output 2: left single quotation mark, Akron 25.81, backslash, n, right single quotation mark. Input 3: with open, left parenthesis, left single quotation mark, rainfall dot t x t, right single quotation mark, comma, left single quotation mark, r, right single quotation mark, right parenthesis, as in File, colon. Line 2, indented once: line List, Equals, in File dot read lines, left parenthesis, right parenthesis, Hash, read all lines as a list. Input 5: Line 1: print, left parenthesis, line List, right parenthesis. Line 2: left square bracket, left single quotation mark, Akron 25.81, backslash, n, right single quotation mark, comma, left single quotation mark, Albia 37.65, backslash, n, right single quotation mark, comma, left single quotation mark, Algona 30.69, backslash, n, right single quotation mark, comma, left single quotation mark, Allison 33.64, backslash, n, right single quotation mark, comma, left single quotation mark, Alton 27.43, backslash, n, right single quotation mark, comma, left single quotation mark, Ames W 34.07, backslash, n, right single quotation mark, comma, left single quotation mark, Ames S E 33.95, backslash, n, right single quotation mark, comma, left single quotation mark, Anamosa 35.33, backslash, n, right single quotation mark, comma, left single quotation mark, Ankeny 33.38, backslash, n, right single quotation mark, comma, left single quotation mark, Atlantic 34.77, backslash, n, right single quotation mark, comma, left single quotation mark, Audubon 33.41, backslash, n, right single quotation mark, comma, left

single quotation mark, Beaconsfield 35.27, backslash, n, single quote, comma, left single quotation mark, Bedford 36.35, backslash, n, right single quotation mark, comma, left single quotation mark, Belle Plaine 35.81, backslash, n, right single quotation mark, comma, left single quotation mark, Bellevue 34.35, backslash, n, right single quotation mark, comma, left single quotation mark, Blockton 36.28, backslash, n, right single quotation mark, comma, left single quotation mark, Bloomfield 38.02, backslash, n, right single quotation mark, comma, left single quotation mark, Boone 36.30, backslash, n, right single quotation mark, comma, left single quotation mark, Brighton 33.59, backslash, n, right single quotation mark, comma, left single quotation mark, Britt 31.54, backslash, n, right single quotation mark, comma, left single quotation mark, Buckeye 33.66, backslash, n, right single quotation mark, comma, left single quotation mark, Burlington K B U R 37.94, backslash, n, right single quotation mark, comma, left single quotation mark, Burlington 36.94, backslash, n, right single quotation mark, comma, left single quotation mark, Carroll 33.33, backslash, n, right single quotation mark, comma, left single quotation mark, Cascade 33.48, right single quotation mark, right square bracket. Input 6: Line 1: with open, left parenthesis, left single quotation mark, rainfall dot t x t, right single quotation mark, comma, left single quotation mark, r, right single quotation mark, right parenthesis, as in File, colon. Line 2, indented once: file String, Equals, in File dot read, left parenthesis, right parenthesis, Hash, read entire file as a string. Input 7: Line 1: file String. Output 7: left single quotation mark, Akron 25.81, backslash, n Albia 37.65, backslash, n Algona 30.69, backslash, n Allison 33.64, backslash, n Alton 27.43, backslash, n Ames W 34.07, backslash, n Ames S E 33.95, backslash, n Anamosa 35.33, backslash, n Ankeny 33.38, backslash, n Atlantic 34.77, backslash, n Audubon 33.41,

backslash, n Beaconsfield 35.27, backslash, n Bedford 36.35, backslash, n Belle Plaine 35.81, backslash, n Bellevue 34.35, backslash, n Blockton 36.28, backslash, n Bloomfield 38.02 n Boone 36.30, backslash, n Brighton 33.59, backslash, n Britt 31.54, backslash, n Buckeye 33.66, backslash, n Burlington K B U R 37.94, backslash,n Burlington 36.94, backslash, n Carroll 33.33, backslash, n Cascade 33.48, right single quotation mark.

Back to Session

The screen displays four sets of lines in two colors. The first line, colored in blue in the figure, provides the labels and relative positions of the data in the succeeding lines. Each succeeding line has information about one earthquake. The metadata line plus data for the first three earthquakes in the Philippine Islands region, southern East Pacific Rise, and Hachinohe, Japan. The places are enclosed in double quotes. For each earthquake, the first value of the metadata indicates the date and time followed by a comma, then the latitude followed by a comma, then the longitude followed by a comma, and place. There is a line break after each earthquake's data.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: import c s v. Input 2: Line 1: with open, left parenthesis, left double quotation mark, earthquakes dot c s v, right double quotation mark, comma, left double quotation mark, r, right double quotation mark, right parenthesis, as in File, colon. Line 2, indented once: c s v Reader equals c s v dot reader, left parenthesis, in File, right parenthesis, hash, pass file to c s v

reader. Line 3, indented once: for line in c s v Reader, solon, hash, read each line. Line 4, indented twice: print, left parenthesis, line, right parenthesis. Line 5: blank. The output displays the earthquake data in the Philippine Islands region, southern East Pacific Rise, Hachinohe of Japan, Bandar-e Lengeh of Iran, Mindanao of the Philippines, and so on in the C S V format. The earthquake details are listed as follows. left square bracket, left single quotes, time, right single quotes, comma, left single quotes, latitude, right single quotes, comma, left single quotes, longitude, right single quotes, comma, left single quotes, depth, right single quotes, comma, left single quotes, mag, right single quotes, comma, left single quotes, mag Type, right single quotes, comma, left single quotes, n s t, right single quotes, comma, left single quotes, gap, right single quotes, comma, left single quotes, dmin, right single quotes, comma, left single quotes, r m s, right single quotes, comma, left single quotes, net, right single quotes, comma, left single quotes, id, right single quotes, comma, left single quotes, updated, right single quotes, comma, left single quotes, place, right single quotes, comma, left single quotes, type, right single quotes, comma, left single quotes, horizontal Error, right single quotes, comma, left single quotes, depth Error, right single quotes, comma, left single quotes, mag Error, right single quotes, comma, left single quotes, mag N s t, right single quotes, comma, left single quotes, status, right single quotes, comma, left single quotes, location Source, right single quotes, comma, left single quotes, mag Source, right single quotes, right square bracket.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: read a line. Line 2: while not end of file, colon.

Line 3, indented once: process the line. Line 4, indented once: red another line.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: import c s v. Input 2: Line 1: with open, left parenthesis, left double quotation mark, earthquakes dot c s v, right double quotation mark, comma, left double quotation mark, r, right double quotation mark, right parenthesis, as in File, colon. Line 2, indented once: c s v Reader equals c s v dot reader, left parenthesis, in File, right parenthesis, hash, feed file to c s v reader. Line 3: blank. Line 4, indented once: titles equals next, left parenthesis c s v Reader, right parenthesis, hash, read first line with titles. Line 5, indented once: col Num equals 0. Line 6, indented once: while titles, left square bracket, col Num, right square bracket, exclamatory mark equals, left double quotation mark mag, right double quotation mark, colon. Line 7, indented twice: col Num equals col Num plus 1. Line 8: blank. Line 9, indented once: print, left parenthesis, left double quotation mark, The magnitude is found in col, right double quotation mark, comma, col Num, right parenthesis. Output. The magnitude is found in col 4.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: while col Num, less than, len, left parenthesis, titles, right parenthesis, and titles, left square bracket, col Num, right square bracket, exclamatory mark equals, data Name, colon.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 4: Line 1: def make Data List, left parenthesis, data Name, right parenthesis, colon. Line 2, indented once: import c s v. Line 3, indented once: with open, left parenthesis, left double quotes, earthquakes dot c s v, right double quotes, comma, left double quotes, r, right double quotes, right parenthesis, as in File, colon. Line 4, indented twice: data List, Equals, left square bracket, right square bracket. Line 5: Blank. Line 6, indented twice: c s v Reader, Equals, c s v dot reader, left parenthesis, in File, right parenthesis, Hash, get iterator. Line 7, indented twice: titles, Equals, next, left parenthesis, s v Reader, right parenthesis, Hash, read titles line. Line 8: Blank. Line 9, indented twice: c o l N u m, Equals, 0. Line 10, indented twice: while c o l N u m, less than, l e n, left parenthesis, titles, right parenthesis, and titles, left square bracket, c o l N u m, right square bracket, exclamatory mark, equals, data Name, colon. Line 11, indented thrice: c o l N u m, Equals, c o l N u m, plus, 1. Line 12: Blank. Line 13, indented twice: if c o l N u m, Equals, Equals, l e n, left parenthesis, titles, right parenthesis, colon, Hash, was end of titles reached, Question mark. Line 14, indented thrice: Print, left parenthesis, left double quotes, Error, colon, right double quotes, comma, data Name, comma, left double quotes, not found, right double quotes, right parenthesis. Line 15, indented twice: else, colon. Line 16, indented thrice: for line in c s v Reader, colon. Line 17, indented quarce: data List dot append, left parenthesis, float, left parenthesis, line, left square bracket, c o l N u m, right square bracket, right parenthesis, right parenthesis, Hash, create list of values. Line 18, indented once: return data List.

Back to Listing

The statement reads the following. Line 1: Left square bracket, left angular bracket, expression, right angular bracket, for, left angular bracket, item 1, right angular bracket, in, left angular bracket, sequence 1, right angular bracket. Line 2, indented thrice: for left angular bracket, item 2, right angular bracket, in, left angular bracket, sequence 2, right angular bracket. Line 3, indented thrice: ellipses. Line 4, indented once: if, left angular bracket, condition, right angular bracket, right square bracket.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: cubes, equals, left square bracket, right square bracket. Line 2: for x in range, left parenthesis, 1, comma, 11, right parenthesis, colon. Line 3, indented once: cubes, dot, append, left parenthesis, x, asterisk, x, asterisk, x, right parenthesis. Line 4: blank. Input 2: Line 1: cubes. Out 2: left square bracket, 1, comma, 8, comma, 27, comma, 64, comma, 125, comma, 216, comma, 343, comma, 512, comma, 729, comma, 1000, right square bracket.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: cubes, equals, left square bracket, x, asterisk, x, asterisk, x for x in range, left parenthesis, 1, comma, 11, right parenthesis, right square bracket. Input 2: Line 1: cubes. Output 2: left square bracket, 1, comma, 8, comma, 27, comma, 64, comma, 125,

comma, 216, comma, 343, comma, 512, comma, 729, comma, 1000, right square bracket.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: cubes, equals, left square bracket, x, asterisk, x, asterisk, x for x in range, left parenthesis, 1, comma, 11, right parenthesis, right square bracket. Line 2: even Cubes, equals, left square bracket, x for x in cubes if x percentage 2, equals, equals, 0, right square bracket. Input 2: Line 1: even Cubes. Output 2: left square bracket, 8, comma, 64, comma, 216, comma, 512, comma, 1000, right square bracket.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def make Data List, left parenthesis, data Name, right parenthesis, colon. Line 2, indented once: import c s v. Line 3, indented once: with open, left parenthesis, left double quotes, earthquakes dot c s v, right double quotes, comma, left double quotes, r, right double quotes, right parenthesis, as in File, colon. Line 4, indented twice: data List, Equals, left square bracket, right square bracket. Line 5: Blank. Line 6, indented twice: c s v Reader, Equals, c s v dot reader, left parenthesis, in File, right parenthesis, Hash, get iterator. Line 7, indented twice: titles, Equals, next, left parenthesis, s v Reader, right parenthesis, Hash, read titles line. Line 8: Blank. Line 9, indented twice: c o l N u m, Equals, 0, hash, search for data Name column. Line 10, indented twice: while c o l N u m, less than, l e n, left parenthesis, titles, right parenthesis, and titles, left square bracket, c o l N u m, right

square bracket, exclamatory mark, equals, data Name, colon. Line 11, indented thrice: c o l N u m, Equals, c o l N u m, plus, 1. Line 12: Blank. Line 13, indented twice: if c o l N u m, Equals, Equals, l e n, left parenthesis, titles, right parenthesis, colon, Hash, was end of titles reached, Question mark. Line 14, indented thrice: Print, left parenthesis, left double quotes, Error, colon, right double quotes, comma, data Name, comma, left double quotes, not found, right double quotes, right parenthesis. Line 15, indented twice: else, colon. Line 16, indented thrice: data List, equals, left square bracket, left parenthesis, float, left parenthesis, line, left square bracket, col Num, right square bracket, right parenthesis, right parenthesis, for line in c s v Reader, right square bracket. Line 17, indented once: return data List.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from make Data List 2 import, asterisk. Input 2: Line 1: mag List, Equals, make Data List, left parenthesis, left double quotes, mag, right double quotes, right parenthesis. Input 3: Line 1: l e n, left parenthesis, mag List, right parenthesis. Output 3: 500. Input 4: Line 1: for i in range, left parenthesis, 10, right parenthesis, colon. Line 2, indented once: print, left parenthesis, mag List, left square bracket, i, right square bracket, comma, end, Equals, left double quotes, space, right double quotes, right parenthesis. Line 3: Blank. Line 4: 4.6, space, 4.7, space, 4.5, space, 4.6, space, 4.8, space, 4.5, space, 4.9, space, 5.7, space, 4.6, space, 4.6. Input 5: Line 1: depth List, Equals, make Data List, left parenthesis, left double quotes, depth, right double quotes, right parenthesis. Input 6: l e n, left parenthesis, depth List, right parenthesis. Output 6: 500. Input 7: Line 1: for i in range, left parenthesis, 10,

right parenthesis, colon. Line 2, indented once: print, left parenthesis, depth List, left square bracket, i, right square bracket, comma, end, Equals, left double quotes, space, right double quotes, right parenthesis. Line 3: Blank. Line 4: 150.253, space, 10.0, space, 58.173, space, 16.891, space, 10.0, space, 10.0, space, 10.0, space, 12.077, space, 14.685, space, 35.0. Input 8: Line 1: make Data List, left parenthesis, left double quotes, not A Title, right double quotes, right parenthesis. Line 2: Error, colon, not A Title not found. Output 8: left square bracket, right square bracket.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from make Data List 2 import, asterisk. Input 2: Line 1: import statistics. Input 3: Line 1: mag List, Equals, make Data List, left parenthesis, left double quotes, mag, right double quotes, right parenthesis. Input 4: Line 1: max, left parenthesis, mag List, right parenthesis. Output 4: 7.3. Input 5: Line 1: min, left parenthesis, mag List, right parenthesis. Output 5: 4.5. Input 6: Line 1: Statistics dot mean, left parenthesis, mag List, right parenthesis. Output 6: 4.8302. Input 7: Line 1: statistics dot median, left parenthesis, mag List, right parenthesis. Output 7: 4.7. Input 8: Line 1: statistics dot multimode, left parenthesis, mag List, right parenthesis. Output 8: left square bracket, 4.5, right square bracket. Input 9: Line 1: statistics dot s t d e v, left parenthesis, mag List, right parenthesis. Output 9: 0.406051370433424 04.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: def frequency Table, left parenthesis, List, right parenthesis, colon. Line 2, indented once: count Dict, Equals, left brace, right brace. Line 3, indented once: for item in a List, colon. Line 4, indented twice: if item in count Dict, colon. Line 5, indented thrice: count Dict, left square bracket, item, right square bracket, Equals, count Dict, left square bracket, item, right square bracket, plus, 1. Line 6, indented twice: else, colon. Line 7, indented thrice: count Dict, left square bracket, item, right square bracket, Equals, 1. Line 8: Blank. Line 9, indented once: item List, Equals, list, left parenthesis, count Dict dot keys, left parenthesis, right parenthesis, right parenthesis. Line 10, indented once: item List dot sort, left parenthesis, right parenthesis. Line 11: Blank. Line 12, indented once: print, left parenthesis, left double quotes, ITEM, right double quotes, comma, left double quotes, FREQUENCY, right double quotes, right parenthesis. Line 13, indented once: for item in item List, colon. Line 14, indented twice: print, left parenthesis, left double quotes, left brace, 0, colon, 4.1 f, right brace, left brace, 1, colon, d, right brace, right double quotes, dot format, left parenthesis, item, comma, count Dict, left square bracket, item, right square bracket, right parenthesis, right parenthesis. Line 15: Blank. Input 2: Line 1: from make Data List import asterisk. Input 3: Line 1: mag List equals make Data List, left parenthesis, left double quotes, mag, right double quotes, right parenthesis. Input 4: Line 1: frequency Table, left parenthesis, mag List, right parenthesis. It displays the item and frequency in two columns as follows. Column 1: ITEM, 4.5, 4.6, 4.7, 4.8, 4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7.0, 7.1, 7.2, 7.3. Column 2: FREQUENCY, 115, 96, 60, 45, 59, 34, 26, 16, 8, 6, 12, 4, 3, 2, 1, 1, 3, 2, 1, 1, 1, 3, 1.

Back to Session

The code lines are as follows. Line 1: left square bracket left square bracket date 1 comma magnitude 1 comma magnitude 2 comma magnitude comma 3 comma ellipsis right square bracket comma. Line 2: left square bracket date 2 comma magnitude 1 comma magnitude 2 comma right square bracket. Line 3: ellipsis right square bracket.

Back to Figure

The code lines are as follows. Line 1: product comma color comma price. Line 2: suit comma black comma 250. Line 3: suit comma gray comma 275. Line 4: shoes comma brown comma 75. Line 5: shoes comma blue comma 68. Line 6: shoes comma tan comma 65.

Back to Figure

The pandas data frame resembles a table with six columns and five rows. The column headers are Latitude, Longitude, Place, Depth, and Mag. The first column indicates the Row indices. The data from the pandas data frame are as follows. Row 1: 0, 5.5917, 127.1591, Philippine Islands, 148.005, 4.6. Row 2: 1, 40.2538, 142.1337, Honshu, Japan, 57.997, 4.6. Row 3: 2, 26.8592, 55.1776, Bandar-e Lengeh, Iran, 17.467, 4.6. Row 4: 3, 8.8248, 126.8707, Aras-asan, Philippines, 10.000, 4.8. Row 5: 4, negative 15.9117, 176.9719, Fiji region. 10.000, 4.8. The pandas series resembles two tables. Table 1 reads by column and Table 2 reads by rows. The data from the pandas series by column are as follows. It has two columns and five rows. The first column reads the row indices and the second column reads the depth.

Column 1: 0, 1, 2, 3, 4. Column 2: 148.005, 57.997, 17.467, 10.000, 10.000. The data from the pandas series by rows are as follows. It has six columns and a row. The first column reads the row indices. The column headers are atitude, Longitude, Place, Depth, and Mag. Row 1: 2, 2, 26.8592, 55.1776, Bandar-e Lengeh, Iran, 17.467, 4.6.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: import pandas as p d. Input 2: Line 1: d f equals, p d, dot, read, underscore, c s v, left parenthesis, left double quotation mark, earthquakes, dot, c s v, right double quotation mark, right parenthesis, hash, read entire c s v file. Line 2: d f, hash, display the result. Output 2 displays a list of earthquake files with their time, latitude, longitude, depth, mag, mag Type, n s t, gap, d min, r m s, ellipses, updated place, t y, horizontal Error with a total of 500 rows and 22 columns.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: import pandas as p d. Input 2: Line 1: d f equals, p d, dot, read, underscore, c s v, left parenthesis, left double quotation mark, earthquakes, dot, c s v, right double quotation mark, right parenthesis. Input 2: Line 1: hash, get a list of the column names in the Data Frame. Line 2: d f, dot, columns. Output 2: Index, left parenthesis, left square bracket, left single quotes, time, right single quotes, comma, left single quotes, latitude, right single quotes, comma, left single quotes, longitude, right single quotes,

comma, left single quotes, depth, right single quotes, comma, left single quotes, mag, right single quotes, comma, left single quotes, mag Type, right single quotes, comma, left single quotes, n s t, right single quotes, comma, left single quotes, gap, right single quotes, comma, left single quotes, d min, right single quotes, comma, left single quotes, r m s, right single quotes, comma, left single quotes, net, right single quotes, comma, left single quotes, id, right single quotes, comma, left single quotes, updated right single quotes, comma, left single quotes, place, right single quotes, comma, left single quotes, type, right single quotes, comma, left single quotes, horizontal Error, right single quotes, comma, left single quotes, depth Error, right single quotes, comma, left single quotes, mag Error, right single quotes, comma, left single quotes, mag Nst, right single quotes, comma, left single quotes, status, right single quotes, comma, left single quotes, location Source, right single quotes, comma, left single quotes, mag Source, right single quotes, right square bracket, comma, d type, equals, left single quotes, object, right single quotes, right parenthesis. Input 3: Line 1: hash, get only the magnitude column data as a Series. Line 2: magnitudes, equals, d f, left square bracket, left double quotes, mag, right double quotes, right square bracket. Line 3: hash, calculate the mean of the magnitude Series. Line 4: magnitudes, dot, mean, left parenthesis, right parenthesis. Output 3: 4.8302. Input 4: Line 1: hash, get the fifth row, left parenthesis, index 4, right parenthesis, as a Series. Line 2: fifth Row equals, d f, dot, i l o c, left square bracket, 4, right square bracket. Line 3: hash, retrieve the, left double quotes, place, right double quotes, column value of the fifth row. Line 4: fifth Row, left square bracket, left double quotes, place, right square bracket. Output 4: left single quotes, Mindanao, comma, Philippines, right single quotes.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 5: Line 1: hash, create a new column based on a calculation using an existing column. Line 2: d f, left square bracket, left double quotation mark, depth, underscore, m, right double quotation mark, right square bracket, greater, equals, d f, left square bracket, left double quotation mark, depth, underscore, m, right double quotation mark, right square bracket, asterisk, 1000.0. Input 6: Line 1: hash, subset the Data Frame to show only a few columns. Line 2: my Columns, equals, left square bracket, left double quotation mark, latitude, right double quotation mark, comma, left double quotation mark, longitude, right double quotation mark, comma, left double quotation mark, place, right double quotation mark, comma, left double quotation mark, depth, underscore, m, right double quotation mark, comma, left double quotation mark, mag, right double quotation mark, right square bracket. Line 3: subset D f, equals, d f, left square bracket, my Columns, right square bracket. Line 4: subset D f, dot, head, left parenthesis, 5, right parenthesis. Output 6: A table of six columns and five rows displays the following. The column headers are as follows. Blank, latitude, longitude, place, depth underscore m, mag. Row 1: 0, 5.5812, 127.1411, Philippine Islands region, 150253.0, 4.6. Row 2: 1, negative 35.5874, negative 110.1823, southern East Pacific Rise, 10000.0, 4.7. Row 3: 2, 40.2398, 142.1853, 64 kilometers E S E of Hachinohe Japan, 58173.0, 4.5. Row 4: 3, 26.8456, 55.1789, 43 kilometers N E of Bandar-e Lengeh Iran, 16891.0, 4.6. Row 5: 4, 8.8228, 126.8539, Mindanao Philippines, 10000.0, 4.8. Input 7: Line 1: hash, retrieve the first 7 rows. Line 2: subset D f, dot, i loc, left square bracket, 0, colon, 7, right square bracket. Output 7: A table of six columns and

seven rows displays the following. The column headers are as follows. Blank, latitude, longitude, place, depth underscore m, mag. Row 1: 0, 5.5812, 127.1411, Philippine Islands region, 150253.0, 4.6. Row 2: 1, negative 35.5874, negative 110.1823, southern East Pacific Rise, 10000.0, 4.7. Row 3: 2, 40.2398, 142.1853, 64 kilometers E S E of Hachinohe Japan, 58173.0, 4.5. Row 4: 3, 26.8456, 55.1789, 43 kilometers N E of Bandar-e Lengeh Iran, 16891.0, 4.6. Row 5: 4, 8.8228, 126.8539, Mindanao Philippines, 10000.0, 4.8. Row 6: 5, negative 27.9013, 63.6965, Southwest Indian Ridge, 10000.0, 4.5. Row 7: 6, negative 16.1908, 176.8334, Fiji region, 10000.0, 4.9. Input 8: Line 1: hash, retrieve only the even rows using list comprehension. Line 2: subset D f, dot, i loc, left square bracket, left square bracket, left parenthesis i percentage 2, equals, equals, 0, right parenthesis, for i in subset D f, dot, index, right square bracket, right square bracket. Output 8: A table of six columns and eleven rows displays the following. The column headers are as follows. Blank, latitude, longitude, place, depth underscore m, mag. Row 1: 0, 5.5812, 127.1411, Philippine Islands region, 150253.0, 4.6. Row 2: 2, 40.2398, 142.1853, 64 kilometers E S E of Hachinohe Japan, 58173.0, 4.5. Row 3: 4, 8.8228, 126.8539, Mindanao Philippines, 10000.0, 4.8. Row 4: 6, negative 16.1908, 176.8334, Fiji region, 10000.0, 4.9. Row 5: 8, 21.3009, 121.1190, 87 kilometers S S E of Hengchun Taiwan, 14685.0, 4.6. Row 6: ellipses. Row 7: 490, negative 15.6096, negative 173.0045, 93 kilometers E N E of Hihifo Tonga, 10000.0, 4.7. Row 8: 492, negative 26.0355, 178.4719, south of the Fiji Islands, 656429.0, 4.9. Row 9: 494, negative 10.9543, 113.3845, south of Java Indonesia, 10000.0, 4.8. Row 10: 496, negative 25.5781, 178.2612, south of the Fiji Islands, 624464.0, 6.6. Row 11: 498, negative 25.6944, 178.3159, south of the Fiji Islands, 634472.0, 4.7. Input 9: Line 1: hash, retrieve all rows where the magnitude

is greater than 6.5. Line 2: subset D f, left square bracket, subset D f, left square bracket, left double quotation mark, mag, right double quotation mark, right square bracket, greater than, 6.5, right square bracket. Output 9: A table of six columns and seven rows displays the following. The column headers are as follows. Blank, latitude, longitude, place, depth underscore m, mag. Row 1: 261, negative 9.8078, 159.5927, Solomon Islands, 14000.0, 7.0. Row 2: 332, negative 4.9043, 100.7862, 204 kilometers S W of Bengkulu Indonesia, 25000.0, 6.9. Row 3: 446, negative 20.1112, negative 178.3453, Fiji region, 579000.0, 7.0. Row 4: 474, negative 19.2881, negative 172.1471, 205 kilometers E S E of Neiafu Tonga, 37000.0, 7.3. Row 5: 496, negative 25.5781, 178.2612, south of the Fiji Islands, 624464.0, 6.6. Row 6: 497, negative 26.0901, 178.3427, south of the Fiji Islands, 660000.0, 7.0. Row 7: 499, negative 26.0077, 178.2776, south of the Fiji Islands, 630379.0, 6.8. Input 10: Line 1: subset D f left square bracket, left parenthesis, subset D f, left square bracket, left double quotation mark, place, right double quotation mark, right square bracket, dot, str, dot, contains, left parenthesis, left double quotation mark, Fiji, right double quotation mark, right parenthesis, right parenthesis, ampersand, subset D f, left parenthesis, subset D f, left square bracket, left double quotation mark, mag, right double quotation mark, right square bracket, greater than, 6.5, right parenthesis, right square bracket. Output 10: A table of six columns and four rows displays the following. The column headers are as follows. Blank, latitude, longitude, place, depth underscore m, mag. Row 1: 446, negative 20.1112, negative 178.3453, Fiji region, 579000.0, 7.0. Row 2: 496, negative 25.5781, 178.2612, south of the Fiji Islands, 624464.0, 6.6. Row 3: 497, negative 26.0901, 178.3427, south of the Fiji Islands, 660000.0, 7.0. Row 4: 499, negative 26.0077, 178.2776, south of the Fiji Islands, 630379.0, 6.8.

Back to Session

The screen displays six rows with all the earthquake details. The blue-colored data illustrates the main description information of the earthquake along with the metadata and the specific data for the first three earthquakes in J S O N formatting. The last field in the metadata is count with the value of 500, which is the number of earthquakes for which data is provided. For each earthquake, the type, feature, and properties are listed. The data for each earthquake is provided as name-value pairs separated by commas. The ellipsis in the final gray section indicates information about the remaining earthquakes. The magnitude of the earthquake is represented by mag and is highlighted in red. The magnitudes of the first three of the 500 earthquakes are 4.6, 4.7, and 4.5, respectively. The U R L is listed on the second line of the screen for downloading the J S O N data.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: import j s o n. Line 2: import u r l lib, dot, request. Input 2: Line 1: handle equals u r l lib, dot, request, dot, u r l open, left parenthesis, left double quotation mark, h t t p s, colon, forward slash, forward slash, earthquake, dot, u s g s, dot, g o v, forward slash, f d s n w s, forward slash, event, forward slash, 1, forward slash, query, dot, geo j s o n, question mark, backslash, start time, equals, 2022, hyphen, 11, hyphen, 01 percent 2000, colon, 00, colon, 00, ampersand, end time, equals, 2022, hyphen, 11, hyphen, 30 percent, 2023, colon, 59, colon, 59, backslash, ampersand, min magnitude, equals, 4.5, ampersand, Order by, equals, time, ampersand, limit, equals, 500, right double quotation mark, right

parenthesis. Line 2: data equals handle, dot, read, left parenthesis, right parenthesis. Line 3: e Data equals j s o n, dot, loads, left parenthesis, data, right parenthesis. Input 3: Line 1: e Data, dot, keys, left parenthesis, right parenthesis. Output 3: dict, underscore, keys, left parenthesis, left square bracket, left single quotes, type, right single quotes, comma, left single quotes, metadata, right single quotes, comma, left single quotes, features, right single quotes, comma, left single quotes, b box, right single quotes, right square bracket, right parenthesis. Input 4: Line 1: earthquake List equals, e Data, dot, get, left parenthesis, left single quotes, features, right single quotes, right parenthesis. Input 5: Line 1: len, left parenthesis, earthquake List, right parenthesis. Output 5: 500.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def make Mag List, left parenthesis, earthquake Data, right parenthesis, colon. Line 2, indented once: mag List equals, left square bracket, right square bracket. Line 3, indented once: earthquakes equals earthquake Data, dot, get, left parenthesis, left single quotes, features, right single quotes, right parenthesis. Line 4, indented once: for i in range, left parenthesis, len, left parenthesis, earthquakes, right parenthesis, right parenthesis, colon. Line 5, indented twice: earthquake equals earthquakes, left square bracket, i, right square bracket. Line 6, indented twice: properties equals earthquake, dot, get, left parenthesis, left single quotes, properties, right single quotes, right parenthesis. Line 7, indented twice: mag equals properties, dot get, left parenthesis, left single quotes, mag, right single quotes, right parenthesis. Line 8, indented twice: mag List, dot

append, left parenthesis, mag, right parenthesis. Line 9: blank. Line 10, indented once: return mag List.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from make J s o n Mag List import asterisk. Line 2: import j s o n, comma, statistics. Input 2: Line 1: with open, left parenthesis, left double quotation mark, earthquakes, dot, j s o n, right double quotation mark, comma, encoding, equals, left single quotes, u t f-8, right single quotes, right parenthesis, as handle, colon. Line 2, indented once: data equals handle, dot, read, left parenthesis, right parenthesis. Line 3, indented once: e Data equals, j s o n, dot, loads, left parenthesis, data, right parenthesis. Input 3: Line 1: mag List equals make Mag List, left parenthesis, e Data, right parenthesis. Input 4: Line 1: statistics, dot, mean, left parenthesis, mag List, right parenthesis. Output 4: 4.8302. Input 5: Line 1: statistics, dot, median, left parenthesis, mag List, right parenthesis. Output 5: 4.7. Input 6: Line 1: statistics, dot, multimode, left parenthesis, mag List, right parenthesis. Output 6: left square bracket, 4.5, right square bracket. Input 7: Line 1: statistics, dot, stdev, left parenthesis, mag List, right parenthesis. Output 7: 0.406051370433424 04

Back to Session

The screen displays four graphs in two rows. The x-axis of all the graphs ranges from 0.0 to 1.0 in increments of 0.2. Similarly, the y-axis of all the graphs ranges from 0.0 to 1.0 in increments of 0.2. The data from the graphs are approximated as follows. The graph at the top left is labeled correlation equals 0.1. A cluster of

data points is plotted at the center, horizontally throughout the graph between 0.2 and 0.8 on the y-axis. The graph at the top right is labeled correlation equals negative 1.0. A cluster of data points is plotted slantingly from the top left to the bottom right resembling a decreasing slope. The graph at the bottom left is labeled correlation equals 1.0. A cluster of data points is plotted slantingly from the origin to the top right resembling an increasing slope. The graph at the bottom right is labeled correlation equals 0.8. A cluster of data points is plotted slantingly from the top left to the bottom right.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def correlation, left parenthesis, x List, comma, y List, right parenthesis, colon. Line 2, indented once: import statistics as s t. Line 3, indented once: x Bar equals st, dot, mean, left parenthesis, x List, right parenthesis. Line 4, indented once: y Bar equals st, dot, mean, left parenthesis, y List, right parenthesis. Line 5, indented once: x Std v st, dot, stdev, left parenthesis, x List, right parenthesis. Line 6, indented once: y Std equals st, dot, stdev, left parenthesis, y List, right parenthesis. Line 7: blank. Line 8, indented once: num equals 0.0. Line 9, indented once: for i in range, left parenthesis, len, left parenthesis, x List, right parenthesis, right parenthesis, colon. Line 10, indented twice: num equals num plus, left parenthesis, x List, left square bracket, i, right square bracket, minus, x Bar, right parenthesis, asterisk, left parenthesis, y List, left square bracket, i, right square bracket, minus, y Bar, right parenthesis. Line 11: blank. Line 12, indented once: corr equals num, forward slash, left parenthesis, left parenthesis, len, left parenthesis, x List, right parenthesis, minus,

1, right parenthesis, asterisk, x Std, asterisk, y Std, right parenthesis. Line 13, indented once: return corr.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from correlation import asterisk. Line 2: from make Data List import asterisk. Input 2: Line 1: mag List equals make Data List, left parenthesis, left double quotation mark, mag, right double quotation mark, right parenthesis. Line 2: depth List equals make Data List, left parenthesis, left double quotation mark, depth, right double quotation mark, right parenthesis. Input 3: Line 1: correlation, left parenthesis, mag List, comma, depth List, right parenthesis. Output 3: 0.1392354238478085.

Back to Session

The table has four columns: Color, Red, Green, and Blue. Row entries are as follows. Row 1. Red, 255, 0, 0. Row 2. Green, 0, 255, 0. Row 3. Blue, 0, 0, 255. Row 3. Magenta, 255, 0, 255. Row 4. Yellow, 255, 255, 0. Row 5. Cyan, 0, 255, 255. Row 6. White, 255, 255, 255. Row 7. Black, 0, 0, 0.

Back to Table

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from c Image import, asterisk. Input 2: Line 1: p, equals, Pixel, left parenthesis, 200, comma, 100, comma, 150, right parenthesis, hash, create a Pixel object. Input 3: Line 1: p. Output 3: left parenthesis, 200, comma, 100, comma, 150, right parenthesis. Input 4: Line 1: p, dot, get Red, left parenthesis, right parenthesis,

hash, get the red component. Output 4: 200. Input 6: Line 1. P, dot, set Blue, left parenthesis, 20, right parenthesis, hash, set the blue component to 20. Input 7: Line 1: p. Output 7: left parenthesis, 200, comma, 100, comma, 20, right parenthesis.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: from c Image import, asterisk. Line 2: my Win, equals, Image Win, left parenthesis, left double quotes, Image Processing, right double quotes, comma, 600, comma, 400, right parenthesis.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from c Image import, asterisk. Input 2: Line 1: my Win, equals, Image Win, left parenthesis, left double quotes, Butterfly, right double quotes, comma, 300, comma, 224, right parenthesis. Input 3: Line 1: butterfly, equals, File Image, left parenthesis, left double quotes, butterfly, dot, p n g, right double quotes, right parenthesis. Input 4: Line 1: butterfly, dot, draw, left parenthesis, my Win, right parenthesis.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from c Image import, asterisk. Input 2: Line 1: butterfly, equals, File Image, left parenthesis, left double quotes, butterfly, dot, p n g, right double quotes, right parenthesis. Input 3: Line 1: butterfly, dot, get Width, left parenthesis, right parenthesis. Output 3: 300.

Input 4: Line 1: butterfly, dot, get Height, left parenthesis, right parenthesis. Output 4: 224. Input 5: Line 1: butterfly, dot, get Pixel, left parenthesis, 124, comma, 165, right parenthesis. Output 5: left parenthesis, 86, comma, 38, comma, 25, right parenthesis.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 2: Line 1: from c Image import, asterisk. Input 4: Line 1: my I m Win, equals, Image Win, left parenthesis, left double quotes, Empty Image, right double quotes, comma, 300, comma, 300, right parenthesis. Input 5: Line 1: empty I m, equals, Empty Image, left parenthesis, 300, comma, 300, right parenthesis. Input 6: Line 1: empty I m, dot, draw, left parenthesis, my I m Win, right parenthesis.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from c Image import, asterisk. Input 2: Line 1: my I m Win, equals, Image Win, left parenthesis, left double quotes, Line Image, right double quotes, comma, 300, comma, 300, right parenthesis. Input 3: Line 1: Line Image, equals, Empty Image, left parenthesis, 300, comma, 300, right parenthesis. Input 4: Line 1: white Pixel, equals, Pixel, left parenthesis, 255, comma, 255, comma, 255, right parenthesis. Input 5: Line 1: for i in range, left parenthesis, line Image dot get Height, left parenthesis, right parenthesis, right parenthesis, colon. Line 2, indented once: line Image dot set Pixel, left parenthesis, i, comma, i, comma, white Pixel, right parenthesis. Input 6: Line 1: line Image, dot, draw, left parenthesis, my I m Win, right parenthesis. Input 8: Line 1: line

Image, dot, save, left parenthesis, left double quotes, line Image dot gif, right double quotes, right parenthesis.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def negative Pixel, left parenthesis, old Pixel, right parenthesis, colon. Line 2, indented once: new Red, equals, 255, minus, old Pixel dot get Red, left parenthesis, right parenthesis. Line 3, indented once: new Green, equals, 255, minus, old Pixel dot get Green, left parenthesis, right parenthesis. Line 4, indented once: New Blue, equals, 255, minus old Pixel dot get Blue, left parenthesis, right parenthesis. Line 5, indented once: new Pixel, equals, Pixel, left parenthesis, new Red, comma, new Green, comma, new Blue, right parenthesis. Line 6: return new Pixel.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from negative import, asterisk. Input 2: Line 1: a Pixel, equals, Pixel, left parenthesis, 155, comma, 23, comma, 255, right parenthesis. Input 3: Line 1: negative Pixel, left parenthesis, a Pixel, right parenthesis. Output 3: left parenthesis, 100, comma, 232, comma, 0, right parenthesis.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: for num in range, left parenthesis, 3, right parenthesis, colon. Line 2, indented once: for ch in, left double quotes, cat, right double

quotes, colon. Line 3, indented twice: print, left parenthesis, num, comma, ch, right parenthesis. The output displays the following in nine rows. Row 1: 0, space, c. Row 2: 0, space, a. Row 3: 0, space, t. Row 4: 1, space, c. Row 5: 1, space, a. Row 6: 1, space, t. Row 7: 2, space, c. Row 8: 2, space, a. Row 9: 2, space, t.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def make Negative, left parenthesis, image File, right parenthesis, colon. Line 2, indented once: old Image, equals, File Image, left parenthesis, image File, right parenthesis. Line 3, indented once: width, equals, old Image dot get Width, left parenthesis, right parenthesis. Line 4, indented once: height, equals, old Image dot get Height, left parenthesis, right parenthesis. Line 5: Blank. Line 6, indented once: my Image Window, equals, Image Win, left parenthesis, left double quotes, Negative Image, right double quotes, comma, width, asterisk, 2, comma, height, right parenthesis. Line 7, indented once: old Image dot draw, left parenthesis, my Image Window, right parenthesis. Line 8, indented once: new I m, equals, Empty Image, left parenthesis, width, comma, height, right parenthesis. Line 9: Blank. Line 10, indented once: for row in range, left parenthesis, height, right parenthesis, colon. Line 11, indented twice: for col in range, left parenthesis, width, right parenthesis, colon. Line 12, indented thrice: old Pixel, equals, old Image dot get Pixel, left parenthesis, col, comma, row, right parenthesis. Line 13, indented thrice: new Pixel, equals, negative Pixel, left parenthesis, old Pixel, right parenthesis. Line 14, indented thrice: new I m dot set Pixel, left parenthesis, col, comma, row, comma, new Pixel, right parenthesis. Line 15: Blank. Line 16. new I m dot set Position, left parenthesis, width, plus, 1, comma, 0, right

parenthesis. Line 17: new I m dot draw, left parenthesis, my Image Window, right parenthesis. Line 18: my Image Window dot exit On Click, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: for row in range, left parenthesis, height, right parenthesis, colon. Line 2, indented twice: for col in range, left parenthesis, width, right parenthesis, colon.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def gray Pixel, left parenthesis, old Pixel, right parenthesis, colon. Line 2: Intensity Sum, equals, old Pixel dot get Red, left parenthesis, right parenthesis, plus, old Pixel dot get Green, left parenthesis, right parenthesis, backslash, plus, old Pixel dot get Blue, left parenthesis, right parenthesis. Line 3: Ave R G B, equals, intensity Sum, forward slash, forward slash, 3 Line 4: New Pixel, equals, Pixel, left parenthesis, ave R G B, comma, ave R G B, comma, ave R G B, right parenthesis. Line 5: return new Pixel.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from grayscale import, asterisk. Input 2: Line 1: gray Pixel, left parenthesis, Pixel, left parenthesis, 34, comma, 128, comma, 74, right parenthesis, right parenthesis. Output 2: left parenthesis, 78, comma, 78, comma, 78, right parenthesis. Input 3: Line 1: gray Pixel, left parenthesis, Pixel, left parenthesis, 200, comma, 234,

comma, 165, right parenthesis, right parenthesis. Output 3: left parenthesis, 199, comma, 199, comma, 199, right parenthesis. Input 4: Line 1: gray Pixel, left parenthesis, Pixel, left parenthesis, 23, comma, 56, comma, 77, right parenthesis, right parenthesis. Output 4: left parenthesis, 52, comma, 52, comma, 52, right parenthesis.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: def make Gray Scale, left parenthesis, image File, right parenthesis, colon. Line 2, indented once: Old Image, equals, File Image, left parenthesis, image File, right parenthesis. Line 3, indented once: width, equals, old Image dot get Width, left parenthesis, right parenthesis. Line 4, indented once: height, equals, old Image dot get Height, left parenthesis, right parenthesis. Line 5: Blank. Line 6, indented once: My Image Window, equals, Image Win, left parenthesis, double quotes, Grayscale, double quotes, comma, width, asterisk, 2, comma, height, right parenthesis. Line 7, indented once: Old Image dot draw, left parenthesis, my Image Window, right parenthesis. Line 8, indented once: New I m, equals, Empty Image, left parenthesis, width, comma, height, right parenthesis. Line 9: Blank. Line 10, indented once: for row in range, left parenthesis, height, right parenthesis, colon. Line 11, indented twice: for col in range, left parenthesis, width, right parenthesis, colon. Line 12, indented thrice: Old Pixel, equals, old Image dot get Pixel, left parenthesis, col, comma, row, right parenthesis. Line 13, indented thrice: New Pixel, equals, gray Pixel, left parenthesis, old Pixel, right parenthesis. Line 14, indented thrice: New I m dot set Pixel, left parenthesis, col, comma, row, comma, new Pixel, right parenthesis. Line 15: Blank. Line 16, indented once: New I m dot

set Position, left parenthesis, width, plus, 1, comma, 0, right parenthesis. Line 17, indented once: New I m dot draw, left parenthesis, my Image Window, right parenthesis. Line 18, indented once: My Image Window dot exit On Click, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def square It, left parenthesis, n, right parenthesis, colon. Line 2, indented once: return n, asterisk, n.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: def square It, left parenthesis, n, right parenthesis, colon. Line 2, indented once: return n, asterisk, n. Input 2: square It, left parenthesis, 3, right parenthesis. Output 2: 9. Input 3: Line 1: square It, left parenthesis, square It, left parenthesis, 3, right parenthesis, right parenthesis. Output 3: 81. Input 4: Line 1: square It, hash, evaluate name without parameters. Output 4: left angle bracket, function, double underscore, main, double underscore, dot, square It, left parenthesis, n, right parenthesis, right angle bracket. Input 5: Line 1: z, equals, square It, hash, give square t another name. Input 6: Line 1: z, left parenthesis, 3, right parenthesis. Output 6: 9. Input 7: Line 1: z. Output 7: left angle bracket, function, double underscore, main, double underscore, dot, square It, left parenthesis, n, right parenthesis, right angle bracket.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: def square It, left parenthesis, n, right parenthesis, colon. Line 2, indented once: return n, asterisk, n. Input 2: Line 1: def test, left parenthesis, function Param, comma, n, right parenthesis, colon. Line 2, indented once: return function Param, left parenthesis, n, right parenthesis. Input 3: Line 1: test, left parenthesis, square It, comma, 3, right parenthesis, hash, pass function name without parentheses. Output 3: 9. Input 4: Line 1: test, left parenthesis, square It, comma 5, right parenthesis. Output 4: 25. Input 5: Line 1: test, left parenthesis, square It, left parenthesis, 3, right parenthesis, comma, 5, right parenthesis, hash, actually passing, left parenthesis, 9, comma, 5, right parenthesis. The output displays the following. Line 1: Type Error Traceback, left parenthesis, most recent call last, right parenthesis. Line 2: Input In, left square bracket, 5, right square bracket, comma, in, left angle bracket, cell line, colon, 1, right angle bracket, left parenthesis, right parenthesis. Line 3: right angle bracket, 1 test, left parenthesis, square It, left parenthesis, 3, right parenthesis, comma, 5, right parenthesis. Line 4: Input In, left square bracket, 2, right square bracket, comma, in test, left parenthesis, function Param, comma, n, right parenthesis. Line 5, indented once: def test, left parenthesis, function Param, comma, n, right parenthesis, colon. Line 6: right angle bracket, 2 return function Param, left parenthesis, n, right parenthesis. Line 7. Type Error, colon, single quote, int, single quote, object is not callable.

Back to Session

Line 1 reads: def double Image 2 left parenthesis old image right parenthesis colon. Line 2 reads: Old W equals old Image dot get Width left parenthesis right parenthesis. Line 3 reads: Old H

equals old Image dot get Height left parenthesis right parenthesis. Line 4 is blank. Line 5 reads: new Im equals Empty Image left parenthesis old W asterisk 2 comma old H asterisk 2 right parenthesis. Line 6 is blank. Line 7 reads: for row in range left parenthesis new Im dot get Height left parenthesis right parenthesis right parenthesis. Line 8 reads: for col in range left parenthesis new Im dot get Width left parenthesis right parenthesis right parenthesis. Line 9 is blank. Line 10 reads: original Col equals col double slash 2. Line 11 reads: original Row equals col double slash 2. Line 12 reads: old Pixel equals old Image dot get Pixel left parenthesis original Col comma original Row comma right parenthesis. Line 13 is blank. Lin 14 reads: new Im dot set Pixel left parenthesis original col comma row comma old Pixel right parenthesis. Line 15 is blank. Lin 16 reads: return new Im.

Back to Listing

A square block on the top left is labeled square It. A square block on the bottom left is labeled function Param. An arrow from both blocks goes to a cloud on the top right that is labeled square It definition. Another block labeled n is at the bottom right. An arrow from the n block goes to a sphere on the right and is labeled 3.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1. def pixel Mapper, left parenthesis, file Image, comma, r g b Function, right parenthesis, colon. Line 2: Blank. Line 3, indented once: width, equals, file Image, dot, get Width, left parenthesis, right parenthesis. Line 4, indented once: height,

equals, file Image dot get Height, left parenthesis, right parenthesis. Line 5, indented once: New I m, equals, Empty Image, left parenthesis, width, comma, height, right parenthesis. Line 6: Blank. Line 7, indented once: for row in range, left parenthesis, height, right parenthesis, colon. Line 8, indented twice: for col in range, left parenthesis, width, right parenthesis, colon. Line 9, indented thrice: Old Pixel, equals, file Image dot get Pixel, left parenthesis, col, comma, row, right parenthesis. Line 10, indented thrice: New Pixel, equals, r g b Function, left parenthesis, old Pixel, right parenthesis. Line 11, indented thrice: New I m dot set Pixel, left parenthesis, col, comma, row, comma, new Pixel, right parenthesis. Line 12: Blank. Line 13: return new I m.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def general Transform, left parenthesis, image File, right parenthesis, colon. Line 2, indented once: Old Image, equals, File Image, left parenthesis, image File, right parenthesis. Line 3, indented once: width, equals, old Image dot get Width, left parenthesis, right parenthesis. Line 4, indented once: height, equals, old Image dot get Height, left parenthesis, right parenthesis. Line 5: Blank. Line 6, indented once: My Image Window, equals, Image Win, left parenthesis, double quotes, Grayscale, double quotes, comma, width, asterisk, 2, comma, height, right parenthesis. Line 7, indented once: Old Image dot draw, left parenthesis, my Image Window, right parenthesis. Line 8: Blank. Line 9, indented once: New Image, equals, pixel Mapper, left parenthesis, old Image, comma, gray Pixel, right parenthesis. Line 10: Blank. Line 11, indented once: New Image dot set Position, left parenthesis, old Image dot get Width, left

parenthesis, right parenthesis, plus, 1, comma, 0, right parenthesis. Line 12, indented once: New Image dot draw, left parenthesis, my Image Window, right parenthesis. Line 13, indented once: My Image Window dot exit On Click, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1. import math Line 2. def hypotenuse, left parenthesis, a, comma, b, right parenthesis, colon. Line 3, indented once: c, equals, math dot s q r t, left parenthesis, a, asterisk, asterisk, 2, plus, b, asterisk, asterisk, 2, right parenthesis. Line 4, indented once: return c.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from hypotenuse import, asterisk. Input 2: Line 1: hypotenuse, left parenthesis, 3, comma, 4, right parenthesis, hash, receives 3, receives 4. Output 2: 5.0. Input 3: Line 1: side 1 equals 3. Input 4: Line 1: side 2 equals 4. Input 6: Line 1: hypotenuse, left parenthesis, side 1, comma, side 2, right parenthesis, hash, receives 3, receives 4. Output 6: 5.0. Input 7: Line 1: hypotenuse, left parenthesis, side 1, asterisk, 2, comma, side 2, asterisk, 2, right parenthesis, hash, receives 6, receives 8. Output 7: 10.0. Input 8: hypotenuse, hash, evaluate the function itself. Output 8: left angle bracket, function hypotenuse dot hypotenuse, left parenthesis, a, comma, b, right parenthesis, right angle bracket.

Back to Session

A rectangular box reads builtins at the top left. It reads s t r, i n t, and range in a column. An arrow from the namespace s t r points to a cloud at the left of the box that reads, Type, single quote, s t r, single quote. An arrow from the namespace i n t points to a cloud at the left of the box that reads, Type, single quote, i n t, single quote. An arrow from the namespace range points to a cloud at the left of the box that reads, Built-in function range.

Back to Figure

A rectangular box reads builtins at the top left. It reads s t r, i n t, and range in a column. An arrow from the namespace s t r points to a cloud at the left of the box that reads, Type, single quote, s t r, single quote. An arrow from the namespace i n t points to a cloud at the left of the box that reads, Type, single quote, i n t, single quote. An arrow from the namespace range points to a cloud at the left of the box that reads, Built-in function range. Another box inside the builtins box reads main at the top left. It lists side 1, side 2, and hypotenuse in a column. An arrow from side 1 points to a circle that reads 3. An arrow from side 2 points to a circle that reads 4. An arrow from hypotenuse points to a cloud that reads, left angle bracket, function, Hypotenuse dot hypotenuse, left parenthesis, a, comma, b, right parenthesis, right angle bracket.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: import math. Input 2: Line 1: def hypotenuse, left parenthesis, a, comma, b, right parenthesis, colon. Line 2, indented once: c, equals, math dot s q r t, left parenthesis, a, asterisk, asterisk, 2, plus, b,

asterisk, asterisk, 2, right parenthesis. Line 3, indented once: return c. Line 4: Blank. Input 3: Line 1: side 1, equals, 3. Input 4: Line 1: Side 2, equals, 4. Input 5: Line 1: double underscore, name, double underscore. Output 5: Line 1: left single quote, double underscore, main, double underscore, right single quote. Input 6: Line 1: print, left parenthesis, dir, left parenthesis, right parenthesis, right parenthesis. The output displays the following. left square bracket, left single quote, In, right single quote, comma, left single quote, Out, right single quote, comma, left single quote, underscore, right single quote, comma, left single quote, underscore, 5, right single quote, comma, left single quote, double underscore, right single quote, comma, left single quote, double underscore, underscore, right single quote, comma, left single quote, double underscore, builtin, double underscore, right single quote comma, left single quote, double underscore, builtins, double underscore, right single quote, left single quote, double underscore, doc, double underscore, right single quote, comma, left single quote, double underscore, loader, double underscore, right single quote, comma, left single quote, double underscore, name, double underscore, right single quote, comma, left single quote, double underscore, package, double underscore, right single quote, comma, left single quote, double underscore, spec, double underscore, right single quote, comma, left single quote, underscore, d h, right single quote, comma, left single quote, underscore, i, right single quote, comma, left single quote, underscore, i 1, right single quote, comma, left single quote, underscore, i 2, right single quote, comma, left single quote, underscore, i 3, right single quote, comma, left single quote, underscore, i 4, right single quote, comma, left single quote, underscore, i 5, right single quote, comma, left single quote, underscore, i 6, right single quote, comma, left single quote, underscore, i h, right single quote, comma, left single

quote, underscore, i i, right single quote, comma, left single quote, underscore, i i i, right single quote, comma, left single quote, underscore, oh, right single quote, comma, left single quote, exit, right single quote, comma, left single quote, get, underscore, i python, right single quote, comma, left single quote, hypotenuse, right single quote, comma, left single quote, math, right single quote, comma, left single quote, quit, right single quote, comma, left single quote, side 1, right single quote, comma, left single quote, side 2, right single quote, right square bracket. Input 7: Line 1: math. Output 7: left angle bracket, module, left single quote, math, right single quote, left parenthesis, built-in, right parenthesis, right angle bracket. Input 8: Line 1: print, left parenthesis, dir, left parenthesis, math, right parenthesis, right parenthesis. The output displays the following. left square bracket, left single quote, double underscore, doc, double underscore, right single quote, comma, left single quote, double underscore, loader, double underscore, right single quote, comma, left single quote, double underscore, name, double underscore, right single quote, comma, left single quote, double underscore, package, double underscore, right single quote, comma, left single quote, double underscore, spec, double underscore, right single quote, comma, left single quote, a c o s, right single quote, comma, left single quote, a c o s h, right single quote, comma, left single quote, asin, right single quote, comma, left single quote, asinh, right single quote, comma, left single quote, atan, right single quote, comma, left single quote, atan 2, right single quote, comma, left single quote, atanh, right single quote, comma, left single quote, ceil, right single quote, comma, left single quote, comb, right single quote, comma, left single quote, copy sign, right single quote, comma, left single quote, cos, right single quote, comma, left single quote, cos h, right single quote, comma, left single quote, degrees, right single

quote, comma, left single quote, dist, right single quote, comma, left single quote, e, right single quote, comma, left single quote, e r f, right single quote, comma, left single quote, e r f c, right single quote, comma, left single quote, exp, right single quote, comma, left single quote, e x p m 1, right single quote, left single quote, fabs, right single quote, comma, left single quote, factorial, right single quote, comma, left single quote, floor, right single quote, comma, left single quote, f mod, right single quote, comma, left single quote, f r exp, right single quote, comma, left single quote, f sum, right single quote, comma, left single quote, gamma, right single quote, comma, left single quote, g c d, right single quote, comma, left single quote, hypot, right single quote, comma, left single quote, inf, right single quote, comma, left single quote, is close, right single quote, comma, left single quote, is finite, right single quote, comma, left single quote, is inf, right single quote, comma, left single quote, is nan, right single quote, comma, left single quote, l s q r t, right single quote, comma, left single quote, l c m, right single quote, comma, left single quote, l d e x p, right single quote, comma, left single quote, l gamma, right single quote, comma, left single quote, log, right single quote, comma, left single quote, log 10, right single quote, comma, left single quote, log 1 p, right single quote, left single quote, log 2, right single quote, comma, left single quote, mod f, right single quote, comma, left single quote, nan, right single quote, comma, left single quote, next after, right single quote, comma, left single quote, perm, right single quote, comma, left single quote, pi, right single quote, comma, left single quote, pow, right single quote, comma, left single quote, prod, right single quote, comma, left single quote, radians, right single quote, comma, left single quote, remainder, right single quote, comma, left single quote, sin, right single quote, comma, left single quote, sin h, right single quote, comma, left single

quote, s q r t, right single quote, comma, left single quote, tan, right single quote, comma, left single quote, tan h, right single quote, left single quote, tau, right single quote, comma, left single quote, trunc, right single quote, comma, left single quote, u l p, right single quote, right square bracket. Input 9: Line 1: print, left parenthesis, math, dot, double underscore, doc, double underscore, right parenthesis. The bottom reads the following. This module provides access to the mathematical functions defined by the C standard.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: import math, asterisk. Input 2: Line 1: print, left parenthesis, dir, left parenthesis, right parenthesis, right parenthesis. The output displays the following. left square bracket, left single quote, In, right single quote, comma, left single quote, Out, right single quote, comma, left single quote, underscore, right single quote, comma, left single quote, double underscore, right single quote, comma, left single quote, double underscore, underscore, right single quote, comma, left single quote, double underscore, builtin, double underscore, right single quote comma, left single quote, double underscore, builtins, double underscore, right single quote, left single quote, double underscore, doc, double underscore, right single quote, comma, left single quote, double underscore, loader, double underscore, right single quote, comma, left single quote, double underscore, name, double underscore, right single quote, comma, left single quote, double underscore, package, double underscore, right single quote, comma, left single quote, double underscore, spec, double underscore, right single quote, comma, left single quote,

underscore, d h, right single quote, comma, left single quote, underscore, i, right single quote, comma, left single quote, underscore, i 1, right single quote, comma, left single quote, underscore, i 2, right single quote, comma, left single quote, underscore, i h, right single quote, comma, left single quote, underscore, i i, right single quote, comma, left single quote, underscore, i i i, right single quote, comma, left single quote, underscore, oh, right single quote, comma, left single quote, a c o s, right single quote, comma, left single quote, a c o s h, right single quote, comma, left single quote, asin, right single quote, comma, left single quote, asinh, right single quote, comma, left single quote, atan, right single quote, comma, left single quote, atan 2, right single quote, comma, left single quote, atanh, right single quote, comma, left single quote, ceil, right single quote, comma, left single quote, comb, right single quote, comma, left single quote, copy sign, right single quote, comma, left single quote, cos, right single quote, comma, left single quote, cos h, right single quote, comma, left single quote, degrees, right single quote, comma, left single quote, dist, right single quote, comma, left single quote, e, right single quote, comma, left single quote, e r f, right single quote, comma, left single quote, e r f c, right single quote, comma, left single quote, exit, right single quote, left single quote, exp, right single quote, comma, left single quote, e x p m 1, right single quote, left single quote, fabs, right single quote, comma, left single quote, factorial, right single quote, comma, left single quote, floor, right single quote, comma, left single quote, f mod, right single quote, comma, left single quote, f r exp, right single quote, comma, left single quote, f sum, right single quote, comma, left single quote, gamma, right single quote, comma, left single quote, g c d, right single quote, comma, left single quote, get underscore i python, right single quote, comma, left single quote, hypot, right single quote, comma, left

single quote, inf, right single quote, comma, left single quote, is close, right single quote, comma, left single quote, is finite, right single quote, comma, left single quote, is inf, right single quote, comma, left single quote, is nan, right single quote, comma, left single quote, I s q r t, right single quote, comma, left single quote, l c m, right single quote, comma, left single quote, l d e x p, right single quote, comma, left single quote, l gamma, right single quote, comma, left single quote, log, right single quote, comma, left single quote, log 10, right single quote, comma, left single quote, log 1 p, right single quote, left single quote, log 2, right single quote, comma, left single quote, mod f, right single quote, comma, left single quote, nan, right single quote, comma, left single quote, next after, right single quote, comma, left single quote, perm, right single quote, comma, left single quote, pi, right single quote, comma, left single quote, pow, right single quote, comma, left single quote, prod, right single quote, comma, left single quote, quit, right single quote, comma, left single quote, radians, right single quote, comma, left single quote, remainder, right single quote, comma, left single quote, sin, right single quote, comma, left single quote, sin h, right single quote, comma, left single quote, s q r t, right single quote, comma, left single quote, tan, right single quote, comma, left single quote, tan h, right single quote, left single quote, tau, right single quote, comma, left single quote, trunc, right single quote, comma, left single quote, u l p, right single quote, right square bracket. Input 4: Line 1: s q r t, left parenthesis, 9, right parenthesis. Output 4: 3.0.

Back to Session

A rectangular box reads builtins at the top left. It reads s t r, i n t, and range in a column. An arrow from the namespace s t r points

to a cloud at the left of the box that reads, Type, single quote, s t r, single quote. An arrow from the namespace i n t points to a cloud at the left of the box that reads, Type, single quote, i n t, single quote. An arrow from the namespace range points to a cloud at the left of the box that reads, Built-in function range. Another box inside the builtins box reads main at the top left. It lists side 1, side 2, and hypotenuse in a column. An arrow from side 1 points to a circle that reads 3. An arrow from side 2 points to a circle that reads 4. An arrow from hypotenuse points to a cloud that reads, left angle bracket, function, Hypotenuse dot hypotenuse, left parenthesis, a, comma, b, right parenthesis, right angle bracket. Another box inside the main box reads local, left parenthesis, hypotenuse, right parenthesis with namespaces a, b, and c. It points to a circle labeled 5.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: def test 1, left parenthesis, a, right parenthesis, colon. Line 2, indented once: a, equals, a, plus, 5, hash, refers to a in local namespace Line 3, indented once: print, left parenthesis, a, right parenthesis. Input 2: Line 1: a equals 6. Input 3: Line 1: test 1, left parenthesis, a, right parenthesis. The output displays 11. Input 4: Line 1: a. Output 4: 6. Input 5: Line 1: def test 2, left parenthesis, b, right parenthesis, colon. Line 2, indented once: print, left parenthesis, b, right parenthesis, hash, refers to b in the local namespace Line 3, indented once: print, left parenthesis, a, right parenthesis, hash, refers to a in the main namespace. Input 6: Line 1: test 2, left parenthesis, 14, right parenthesis. The output displays 14 and 6. Input 7: Line 1: a, hash, refers to a in main namespace. Output 7: 6. Input 8: Line 1: b, hash, test 2 local namespace has been

destroyed. The bottom reads the following. Line 1: Name Error Traceback, left parenthesis, most recent call last, right parenthesis. Line 2: Input In, left square bracket, 8, right square bracket, comma, in, left angle bracket, cell line, colon, 1, right angle bracket, left parenthesis, right parenthesis. Line 3: right angle bracket, 1 b. Line 4: Name Error, colon, name, single quote, b, single quote, is not defined.

Back to Session

A rectangular box reads builtins at the top left. It reads s t r, i n t, and range in a column. An arrow from the namespace s t r points to a cloud at the left of the box that reads, Type, single quote, s t r, single quote. An arrow from the namespace i n t points to a cloud at the left of the box that reads, Type, single quote, i n t, single quote. An arrow from the namespace range points to a cloud at the left of the box that reads, Built-in function range. Another box inside the builtins box reads main at the top left. It lists side 1, side 2, and hypotenuse in a column. An arrow from side 1 points to a circle that reads 3. An arrow from side 2 points to a circle that reads 4. An arrow from hypotenuse points to a cloud that reads, left angle bracket, function, Hypotenuse dot hypotenuse, left parenthesis, a, comma, b, right parenthesis, right angle bracket. Another box is next to the main box lists math at the top and s q r t at the bottom. The namespace s q r t points to a cloud that reads math function s q r t.

Back to Figure

A rectangular box reads builtins at the top left. It reads s t r, i n t, and range in a column. An arrow from the namespace s t r points to a cloud at the left of the box that reads, Type, single quote, s t

r, single quote. An arrow from the namespace i n t points to a cloud at the left of the box that reads, Type, single quote, i n t, single quote. An arrow from the namespace range points to a cloud at the left of the box that reads, Built-in function range. Another box inside the builtins box reads main at the top left with the namespaces side 1, side 2, and hypotenuse. The namespaces local and hypotenuse are inside the main box. Another box is next to the main box lists the namespaces math and s q r t with respective local functions.

Back to Figure

Four expressions are listed in two rows and two columns. Row 1: left parenthesis, 2, cross, col, comma, 2, cross, row, right parenthesis, comma, left parenthesis, 2, cross, col, plus, 1, comma, 2, cross, row, right parenthesis. Row 2: left parenthesis, 2, cross, col, comma, 2, cross, row, plus, 1, right parenthesis, comma, left parenthesis, 2, cross, col, plus, 1, 2, cross, row, plus, 1, right parenthesis.

Back to Figure

On the left is the original image, which resembles a grid of 3 pixels wide by 4 pixels high. A plus sign is formed by shading the second column and the second row. When the image is enlarged, by a factor of 2, the new image will be 6 pixels wide by 8 pixels high. The third and fourth columns as well as the third and fourth rows are shaded which represents the pixel construction.

Back to Figure

On the left is the original image, which resembles a grid of 3 pixels wide by 4 pixels high. A plus sign is formed by shading the second column and the second row. When the image is enlarged, by a factor of 2, the new image will be 6 pixels wide by 8 pixels high. The third and fourth columns as well as the third and fourth rows are shaded which represents the pixel construction. The pixels are labeled 0, 1, 2 in the columns and 0, 1, 2, 3 in the rows of the original image. Similarly, the pixels are labeled 0, 1, 2, 3, 4, 5 in the columns and 0, 1, 2, 3, 4, 5, 6, 7 in the rows of the enlarged image. Each pixel of the original is mapped into 4 pixels in the enlarged image. Every 1-by-1 block of pixels in the original image is mapped into a 2-by-2 block in the enlarged image. The pixel (0, 0) in the original image is mapped to pixels (0, 0), (1, 0), (0, 1), (1, 1) of the enlarged image. Likewise, pixel (2, 2) in the original image is mapped to pixels (4, 4), (5, 4), (4, 5), (5, 5) of the enlarged image.

Back to Figure

A square block at the left reads col, row which is the original image. A 2-cross-2 grid is at the right which represents the new image. Arrows from the original image are mapped to the pixels (2, asterisk, col, comma, 2, asterisk, row), (2, asterisk, col, plus, 1, comma, 2, asterisk, row), (2, asterisk, col, comma, 2, asterisk, row, plus, 1), and (2, asterisk, col, plus, 1, comma, 2, asterisk, row, plus, 1) of the new image.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: Double Image, left parenthesis, old Image, right parenthesis, colon. Line 2, indented once: Old W, equals, old

Image dot get Width, left parenthesis, right parenthesis. Line 3, indented once: Old H, equals, old Image dot get Height, left parenthesis, right parenthesis. Line 4: Blank. Line 5, indented once: new I m, equals, Empty Image, left parenthesis, old W, asterisk, 2, comma, old H, asterisk, 2, right parenthesis. Line 6: Blank. Line 7, indented once: for row in range, left parenthesis, old H, right parenthesis, colon. Line 8, indented twice: for col in range, left parenthesis, old W, right parenthesis, colon. Line 9, indented thrice: Old Pixel, equals, old Image dot get Pixel, left parenthesis, col, comma, row, right parenthesis. Line 10: Blank. Line 11, indented thrice: new I m dot set Pixel, left parenthesis, 2, asterisk, col, comma, 2, asterisk, row, comma, old Pixel, right parenthesis. Line 12 indented thrice: new I m dot set Pixel, left parenthesis, 2, asterisk, col, plus, 1, comma, 2, asterisk, row, comma, old Pixel, right parenthesis. Line 13 indented thrice: new I m dot set Pixel, left parenthesis, 2, asterisk, col, comma, 2, asterisk, row, plus, 1, comma, old Pixel, right parenthesis. Line 14 indented thrice: new I m dot set Pixel, left parenthesis, 2, asterisk, col, plus, 1, comma, 2, asterisk, row, plus, 1, comma, old Pixel, right parenthesis. Line 15: Blank. Line 16 indented once: return new I m.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def make Double Image, left parenthesis, image File, right parenthesis, colon. Line 2, indented once: Old Image, equals, File Image, left parenthesis, image File, right parenthesis. Line 3, indented once: width, equals, old Image dot get Width, left parenthesis, right parenthesis. Line 4, indented once: height, equals, old Image dot get Height, left parenthesis, right parenthesis. Line 5: Blank. Line 6, indented once: my Win,

equals, Image Win, left parenthesis, double quotes, Double Size, double quotes, comma, width, asterisk, 2, comma, height, asterisk, 3, right parenthesis. Line 7, indented once: old Image dot draw, left parenthesis, my Win, right parenthesis. Line 8: Blank. Line 9, indented once: new Image, equals, double Image, left parenthesis, old Image, right parenthesis. Line 10, indented once: new Image dot set Position, left parenthesis, 0, comma, old Image dot get Height, left parenthesis, right parenthesis, plus, 1, right parenthesis. Line 11, indented once: new Image dot draw, left parenthesis, my Win, right parenthesis. Line 12: Blank. Line 13, indented once: my Win dot exit On Click, left parenthesis, right parenthesis

Back to Listing

On the left is the original image, which resembles a grid of 3 pixels wide by 4 pixels high. A plus sign is formed by shading the second column and the second row. When the image is enlarged, by a factor of 2, the new image will be 6 pixels wide by 8 pixels high. The third and fourth columns as well as the third and fourth rows are shaded which represents the pixel construction. 4 pixels in the enlarged image are mapped into each pixel of the original image. Each 2-by-2 block in the enlarged image is mapped into every 1-by-1 block of pixels in the original image.

Back to Figure

A square block at the left reads col, row which is the original image. A 2-cross-2 grid is at the right which represents the new image. Arrows from the pixels (2, asterisk, col, comma, 2, asterisk, row), (2, asterisk, col, plus, 1, comma, 2, asterisk, row),

(2, asterisk, col, comma, 2, asterisk, row, plus, 1), and (2, asterisk, col, plus, 1, comma, 2, asterisk, row, plus, 1) of the new image are mapped to the original image. The arrows are labeled, divided by 2.

Back to Figure

On the left is the original image, which resembles a grid of 4 pixels wide by 4 pixels high. A vertical axis divides the original image in its center. On the right is the flipped image, which resembles a grid of 4 pixels wide by 4 pixels high. Arrows from each pixel of the original image go to the flipped image. On flipping, each row will maintain its position relative to every other row. However, within each row, the pixels will move to the opposite side of the axis, as shown by the arrows. The first pixel of the original image will be the last pixel in the flipped image. The last pixel of the original image will be the first pixel in the flipped image.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def vertical Flip, left parenthesis, old Image, right parenthesis, colon. Line 2, indented once: Old W, equals, old Image dot get Width, left parenthesis, right parenthesis. Line 3, indented once: Old H, equals, old Image dot get Height, left parenthesis, right parenthesis. Line 4: Blank. Line 5, indented once: new I m, equals, Empty Image, left parenthesis, old W, comma, old H, right parenthesis. Line 6: Blank. Line 7, indented once: max P, equals, old W minus 1. Line 8, indented once: for row in range, left parenthesis, old H, right parenthesis, colon. Line 9, indented twice: for col in range, left parenthesis, old W,

right parenthesis, colon. Line 10: Blank. Line 11, indented thrice: old Pixel, equals, old Image dot get Pixel, left parenthesis, max P minus col, comma, row, right parenthesis. Line 12: Blank. Line 13, indented thrice: new I m dot set Pixel, left parenthesis, col, comma, row, comma, old Pixel, right parenthesis. Line 14: Blank. Line 15, indented once: return new I m.

Back to Listing

The window is titled Vertical Flip and displays a colorful butterfly on a flower bud at the left panel. The head of the butterfly is towards the left. The panel at the bottom displays its flipped view with the head of the butterfly towards the right.

Back to Figure

On the left is a white square labeled the original image. A white star is enclosed inside a dark cloud inside the square. On the right is a white square labeled the edges. A white star is enclosed inside a white cloud inside the square.

Back to Figure

On the left is a 3-cross-3 kernel labeled X mask. The weights of the kernels are as follows. Row 1: negative 1, 0, 1. Row 2: negative 2, 0, 2. Row 3: negative 1, 0, 1. The leftmost column of values is negative and the rightmost column is positive. On the right is a 3-cross-3 kernel labeled Y mask. The weights of the kernels are as follows. Row 1: 1, 2, 1. Row 2: 0, 0, 0. Row 3: negative 1, negative 2, negative 1. The topmost row of values is positive and the bottommost row is negative.

Back to Figure

On the left is the original grayscale image, which resembles a grid of 6 pixels wide by 8 pixels high. The columns are represented by c and the rows are represented by r. A dark dot is at the pixel (0, 0). On the right is the new image, labeled Placing the X Mask on pixel c, r, which resembles a grid of 6 pixels wide by 8 pixels high. The columns are represented by c and the rows are represented by r. A 3-cross-3 mask is at the center with integer weights. The weights are as follows. (negative 1, 0, 1), (negative 2, 0, 2), and (negative 1, 0, 1).

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def convolve, left parenthesis, an Image, comma, pixel Row, comma, pixel Col, comma, kernel, right parenthesis, colon. Line 2, indented once: Kernel Column Base, equals, pixel Col minus 1. Line 3, indented once: Kernel Row Base, equals, pixel Row minus 1. Line 4: Blank. Line 5, indented once: sum, equals, 0. Line 6, indented once: for row in range, left parenthesis, kernel Row Base, comma, kernel Row Base, plus, 3, right parenthesis, colon. Line 7, indented twice: for col in range, left parenthesis, kernel Column Base, comma, kernel Column Base, plus, 3, right parenthesis, colon. Line 8, indented thrice: k Col Index, equals, col minus kernel Column Base. Line 9, indented thrice: k Row Index, equals, row minus kernel Row Base. Line 10: Blank. Line 11, indented thrice: a Pixel, equals, an Image dot get Pixel, left parenthesis, col, comma, row, right parenthesis. Line 12, indented thrice: intensity, equals, a Pixel dot get Red, left parenthesis, right parenthesis. Line 13: Blank. Line 14, indented thrice: sum, equals, sum, plus, intensity, asterisk, kernel, left square bracket, k Row Index, right square bracket,

left square bracket, k Col Index, right square bracket. Line 15: Blank. Line 16, indented once: return sum.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: import math. Line 2: Blank. Line 3: def edge Detect, left parenthesis, the Image, right parenthesis, colon. Line 4, indented once: gray Image, equals, pixel Mapper, left parenthesis, the Image, comma, gray Pixel, right parenthesis. Line 5, indented once: new I m, equals, Empty Image, left parenthesis, gray Image dot get Width, left parenthesis, right parenthesis, comma, gray Image dot get Height, left parenthesis, right parenthesis, right parenthesis. Line 6, indented once: black, equals, Pixel, left parenthesis, 0, comma, 0, comma, 0, right parenthesis. Line 7, indented once: white, equals, Pixel, left parenthesis, 255, comma, 255, comma, 255, right parenthesis. Line 8, indented once: X Mask, equals, left square bracket, left square bracket, negative 1, comma, negative 2, comma, negative 1, right square bracket, comma, left square bracket, 0, comma, 0, comma, 0, right square bracket, comma, left square bracket, 1, comma, 2, comma, 1, right square bracket, right square bracket. Line 9, indented once: Y Mask, equals, left square bracket, left square bracket, 1, comma, 0, comma, negative 1, right square bracket, comma, left square bracket, 2, comma, 0, comma, negative 2, right square bracket, comma, left square bracket, 1, comma, 0, comma, negative 1, right square bracket, right square bracket. Line 10: Blank. Line 11, indented once: for row in range, left parenthesis, 1, comma, gray Image dot get Height, left parenthesis, right parenthesis, minus, 1, right parenthesis, colon. Line 12, indented twice: for col in range, left parenthesis, 1, comma, gray Image dot get Width, left parenthesis, right

parenthesis, minus, 1, right parenthesis, colon. Line 13, indented thrice: g X, equals, convolve, left parenthesis, gray Image, comma, row, comma, col, X Mask, right parenthesis. Line 14, indented thrice: g Y, equals, convolve, left parenthesis, gray Image, comma, row, comma, col, comma, Y Mask, right parenthesis. Line 15, indented thrice: g, equals, math dot s q r t, left parenthesis, g X, asterisk, asterisk, 2, plus, g Y, asterisk, asterisk, 2, right parenthesis. Line 16: Blank. Line 17, indented thrice: if g, greater than, 175, colon. Line 18, indented quarce: new I m dot set Pixel, left parenthesis, col, comma, row, comma, black, right parenthesis. Line 19, indented thrice: else, colon. Line 20, indented quarce: new I m dot set Pixel, left parenthesis, col, comma, row, comma, white, right parenthesis. Line 21: Blank. Line 22, indented once: return new I m.

Back to Listing

The window is titled Edge Detection and displays a colorful butterfly on a flower bud at the left panel. The panel at the right displays black pixels forming the shape of the butterfly on the bud.

Back to Figure

The vertical axis represents the exam percentage, and the horizontal axis represents the homework percentage. Several points are scattered all over the plane, but the maximum concentrations are restricted to groups of points at the top right and the bottom left.

Back to Figure

The vertical axis represents the exam percentage, and the horizontal axis represents the homework percentage. Several points are scattered all over the plane. Ten points are clustered at the top right of the graph, and the cluster is labeled as A. Four points are clustered at the bottom right of the graph, near the horizontal axis, and the cluster is labeled as C. Ten points are clustered at the bottom left of the graph, and the cluster is labeled as C.

Back to Figure

Row 1 reads: 34, 56, 12, 44, 87, 45, 76. Row 2 reads: 98, 25, 34, 76, 12, 78, 98. Row 3 reads: 78, 90, 89, 45, 77, 22, 11.

Back to Table

Two points are plotted on either end of a double-arrow number line. Point A corresponds to variable X subscript 1 and is on the left end of the line. Point B corresponds to variable X subscript 2 and is on the right end of the line. The horizontal distance between the points is d.

Back to Figure

The vertical axis represents the Y-direction, and the horizontal axis represents the X-direction. Point 1 is labeled as A, at the coordinate X subscript 1, Y subscript 1, and is located at the bottom left of the graph. Point 1 is labeled as B at the coordinates X subscript 2 and Y subscript 2 and is located at the top right of the graph. A line passes through the points, and the distance of the line between the points is d. Perpendicular from point A moves right, and from point B, moves down and meets at a point.

Back to Figure

Nine code lines. Line 1. def Euclid D, left parenthesis, point 1, point 2, right parenthesis, colon, hash, points are equal-length lists or tuples. Line 2. Indented once. import math Line 3. Indented once. total, equals, 0. Line 4. Indented once. for index in range, left parenthesis, l e n, left parenthesis, point 1, right parenthesis, right parenthesis, colon. Line 5. Indented twice. diff, equals, left parenthesis, point 1, left square bracket, index, right square bracket, to point 2, left square bracket, index, right square bracket, right parenthesis, asterisk, asterisk, 2. Line 6. Indented twice. total, equals, total, plus, diff. Line 7. Empty. Line 8. Indented once. Euclid Distance, equals, math period s q r t, left parenthesis, total, right parenthesis. Line 8. Indented once. return Euclid Distance.

Back to Listing

The vertical axis represents the Y-direction, and the horizontal axis represents the X-direction. Point 1 is at the left of the graph, positioned at a grid intersection in the middle section. Point 2 is at the top of the graph, positioned at a grid intersection border. Point 3 is at the right of the graph, positioned at a grid intersection in the top-right section. A star is at a grid intersection below point 2.

Back to Figure

Line 1. def read File, left parenthesis, filename, right parenthesis, colon. Line 2. Indented once. with open, left parenthesis, filename, single quote, r, single quote, right parenthesis, as data File, colon. Line 3. Blank. Line 4. Indented twice. data List,

equals, left square bracket, right square bracket, hash, start with an empty list. Line 5. Blank. Line 6. Indented twice. for a Line in data File, colon. Line 7. Indented thrice. score, equals, int, left parenthesis, a Line, right parenthesis, hash, convert string to int. Line 8. Indented thrice. data List period append, left parenthesis, left square bracket, score, right square bracket, right parenthesis, hash, add to data List as a single-element list. Line 9. Blank. Line 10. return data List.

Back to Listing

Input 1. Line 1. from read file import read File Line 2. read File, left parenthesis, double quotes, c s 150 exams dot txt, double quotes, right parenthesis. Output 1. Line 1. left square bracket, left square bracket, 34, right square bracket comma. Line 2. left square bracket, 56, right square bracket comma. Line 3. left square bracket, 12, right square bracket comma. Line 4. left square bracket, 44, right square bracket comma. Line 5. left square bracket, 87, right square bracket comma. Line 6. left square bracket, 45, right square bracket comma. Line 7. left square bracket, 76, right square bracket comma. Line 8. left square bracket, 98, right square bracket comma. Line 9. left square bracket, 25, right square bracket comma. Line 10. left square bracket, 34, right square bracket comma. Line 11. left square bracket, 76, right square bracket comma. Line 12. left square bracket, 12, right square bracket comma. Line 13. left square bracket, 78, right square bracket comma. Line 14. left square bracket, 98, right square bracket comma. Line 15. left square bracket, 78, right square bracket comma. Line 16. left square bracket, 90, right square bracket comma. Line 17. left square bracket, 89, right square bracket comma. Line 18. left square bracket, 45, right square bracket comma. Line 19. left

square bracket, 77, right square bracket comma. Line 20. left square bracket, 22, right square bracket comma. Line 21. left square bracket, 11, right square bracket, right square bracket.

Back to Session

Line 1. def create Centroids, left parenthesis, k, data List, right parenthesis, colon. Line 2. Indented once. import random. Line 3. Indented once. centroids, equals, left square bracket, right square bracket, hash, list of unique values. Line 4. Indented once. centroid Indices, equals, left square bracket, right square bracket, hash, list of unique indices. Line 5. Indented once. centroid Count, equals, 0. Line 6. Blank. Line 7. Indented once. while centroid Count, left angle bracket, k, colon. Line 8. Indented twice. r Index, equals, random dot r and i n t, left parenthesis, 0, len, left parenthesis, data List, right parenthesis, negative 1, right parenthesis. Line 9. Indented twice. if r Index not in centroid Indices, colon, hash, if index not already selected. Line 10. Indented thrice. centroids dot append, left parenthesis, data List, left square bracket, r Index, right square bracket, right parenthesis, hash, add to centroids. Line 11. Indented thrice. centroid Indices dot append, left parenthesis, r Index, right parenthesis, hash, add index to the selected centroid indices. Line 12. Indented thrice. centroid Count, equals, centroid Count, plus, 1. Line 13. Blank. Line 14. Indented twice. return centroids.

Back to Listing

Line 1. def create Clusters, left parenthesis, k, centroids, data List, repeats, right parenthesis, colon. Line 2. Blank. Line 3. Indented once. for a Pass in range, left parenthesis, repeats, right parenthesis, colon. Line 4. Indented twice. print, left parenthesis,

double quotes, asterisk, asterisk, asterisk, asterisk, PASS, double quotes, a Pass, plus, 1, double quotes, asterisk, asterisk, asterisk, asterisk, double quotes, right parenthesis. Line 5. Blank. Line 6. Indented twice. clusters, equals, left square bracket, right square bracket, hash, create list of k empty lists. Line 7. Indented twice. for i in range, left parenthesis, k, right parenthesis, colon. Line 8. Indented thrice. clusters dot append, left parenthesis, left square bracket, right square bracket, right parenthesis. Line 9. Blank. Line 10. Indented thrice. for data Index in range, left parenthesis, len, left parenthesis, data List, right parenthesis, right parenthesis, colon, hash, calculate distance to centroid. Line 11. Indented thrice. distances, equals, left square bracket, right square bracket. Line 12. Indented thrice. for cluster Index in range, left parenthesis, k, right parenthesis, colon. Line 13. Indented quarce. d to C, equals, Euclid D, left parenthesis, data List, left square bracket, data Index, right square bracket, centroids, left square bracket, cluster Index, right square bracket, right parenthesis. Line 14. Indented quarce, hash, d to C, equals, math dot dist, left parenthesis, tuple, left parenthesis, data Dict, left square bracket, a Key, right square bracket, right parenthesis, tuple, left parenthesis, centroids, left square bracket, cluster Index, right square bracket, right parenthesis, right parenthesis. Line 15. Indented quarce. distances dot append, left parenthesis, to C, right parenthesis. Line 16. Blank. Line 17. Indented thrice. min Dist, equals, min, left parenthesis, distances, right parenthesis, hash, find minimum distance. Line 18. Indented thrice. closest Cluster, equals, distances dot index, left parenthesis, min Dist, right parenthesis, hash, find index of the min distance. Line 19. Blank. Line 20. Indented thrice. clusters, left square bracket, closest Cluster, right square bracket, .append, left parenthesis, data Index, right parenthesis, hash, add to cluster. Line 21. Indented twice. dimensions, equals, len, left parenthesis, data

List, left square bracket, 0, right square bracket, right parenthesis, hash, recompute the clusters. Line 22. Indented once. for cluster Index in range, left parenthesis, k, right parenthesis, colon. Line 23. Indented twice. sums, equals, left square bracket, 0, right square bracket, asterisk, dimensions, hash, init sum for each dimension. Line 24. Indented twice. for data Index in clusters, left square bracket, cluster Index, right square bracket, colon. Line 25. Indented thrice. data Points, equals, data List, left square bracket, data Index, right square bracket. Line 26. Indented thrice. for i n d in range, left parenthesis, len, left parenthesis, datapoints, right parenthesis, right parenthesis, colon, hash, calculate sums. Line 27. Indented quarce. sums, left square bracket, i n d, right square bracket, equals, sums, left square bracket, ind, right square bracket, plus, data Points, left square bracket, ind, right square bracket. Line 28. Indented twice. for ind in range, left parenthesis, len, left parenthesis, sums, right parenthesis, right parenthesis, colon, hash, calculate average. Line 29. Indented thrice. cluster Len, equals, len, left parenthesis, clusters, left square bracket, cluster Index, right square bracket, right parenthesis. Line 30. Indented thrice. if cluster Len, not equal to, 0, colon, hash, do not divide by 0. Line 31. Indented quarce. sums, left square bracket, ind, right square bracket, equals, sums, left square bracket, ind, right square bracket, forward slash, cluster Len. Line 32. Blank. Line 33. Indented thrice. centroids, left square bracket, cluster Index, right square bracket, equals, sums, hash, assign avg to centroids. Line 34. Blank. Line 35. Indented once. for c in range, left parenthesis, k, right parenthesis, colon, hash, output the clusters. Line 36. Indented twice. print, left parenthesis, double quotes, CLUSTER, hash, double quotes, plus, str, left parenthesis, c, plus, 1, right parenthesis, right parenthesis. Line 37. Indented twice. for data Index in clusters, left square bracket, c, right square bracket,

colon. Line 38. Indented thrice. print, left parenthesis, data List, left square bracket, data Index, right square bracket, end, equals, double quotes, double quotes, right parenthesis. Line 39. Indented twice. print, left parenthesis, right parenthesis. Line 40. Blank. Line 41. Indented once. return clusters.

Back to Listing

List 1: Distance. The values are listed in a grid of five unit squares. Unit 0: 66.0. Unit 1: 78.0. Unit 2: 76.0. Unit 3: 87.0. Unit 4: 45.0. List 2: Centroids. Unit 0: left square bracket, 77, right square bracket. Unit 1: left square bracket, 89, right square bracket. Unit 2: left square bracket, 87, right square bracket. Unit 3: left square bracket, 98, right square bracket. Unit 4: left square bracket, 56, right square bracket. List 3: Clusters. Unit 0: left square bracket, 7, 11, 13, 15, 19, right square bracket. Unit 1: left square bracket, 16, 17, right square bracket. Unit 2: left square bracket, 5, right square bracket. Unit 3: left square bracket, 8, 14, right square bracket. Unit 4: left square bracket, 1, 2, 3, 4, 6, 9, 10, 12, 18, 20, 21, right square bracket.

Back to Figure

Line 1. def cluster Analysis, left parenthesis, data File, right parenthesis, colon. Line 2. Indented once. exam Dict, equals, read File, left parenthesis, data File, right parenthesis. Line 3. Indented once. exam Centroids, equals, create Centroids, left parenthesis, 5, exam Dict, right parenthesis. Line 4. Indented once. exam Clusters, equals, create Clusters, left parenthesis, 5, exam Centroids, exam Dict, 3, right parenthesis.

Back to Listing

Input 1. Line 1. asterisk, asterisk, asterisk, asterisk, PASS 1, asterisk, asterisk, asterisk, asterisk. Line 2. CLUSTER, hash. Line 3. left square bracket, 34, right square bracket, left square bracket, 12, right square bracket, left square bracket, 44, right square bracket, left square bracket, 45, right square bracket, left square bracket, 25, right square bracket, left square bracket, 34, right square bracket, left square bracket, 12, right square bracket, left square bracket, 45, right square bracket, left square bracket, 22, right square bracket, left square bracket, 11, right square bracket. Line 4. CLUSTER, hash. Line 5. left square bracket, 56, right square bracket. Line 6. CLUSTER, hash. Line 7. left square bracket, 98, right square bracket, left square bracket, 98, right square bracket. Line 8. CLUSTER, hash. Line 9. left square bracket, 76, right square bracket, left square bracket, 76, right square bracket, left square bracket, 78, right square bracket, left square bracket, 78, right square bracket, left square bracket, 77, right square bracket. Line 10. CLUSTER, hash. Line 11. left square bracket, 87, right square bracket, left square bracket, 90, right square bracket, left square bracket, 89, right square bracket. Line 12. asterisk, asterisk, asterisk, asterisk, PASS 2, asterisk, asterisk, asterisk, asterisk. Line 13. CLUSTER, hash. Line 14. left square bracket, 34, right square bracket, left square bracket, 12, right square bracket, left square bracket, 25, right square bracket, left square bracket, 34, right square bracket, left square bracket, 12, right square bracket, left square bracket, 22, right square bracket, left square bracket, 11, right square bracket. Line 15. CLUSTER, hash. Line 16. left square bracket, 56, right square bracket, left square bracket, 44, right square bracket, left square bracket, 45, right square bracket, left square bracket, 45, right square bracket. Line 17. CLUSTER, hash. Line 18. left square bracket, 98, right square bracket, left square bracket, 98, right square bracket. Line 19. CLUSTER, hash. Line 20. left square

bracket, 76, right square bracket, left square bracket, 76, right square bracket, left square bracket, 78, right square bracket, left square bracket, 78, right square bracket, left square bracket, 77, right square bracket. Line 21. CLUSTER, hash. Line 22. left square bracket, 87, right square bracket, left square bracket, 90, right square bracket, left square bracket, 89, right square bracket. Line 23. asterisk, asterisk, asterisk, asterisk, PASS 3, asterisk, asterisk, asterisk, asterisk. Line 24. CLUSTER, hash. Line 25. left square bracket, 34, right square bracket, left square bracket, 12, right square bracket, left square bracket, 25, right square bracket, left square bracket, 34, right square bracket, left square bracket, 12, right square bracket, left square bracket, 22, right square bracket, left square bracket, 11, right square bracket. Line 26. CLUSTER, hash. Line 27. left square bracket, 56, right square bracket, left square bracket, 44, right square bracket, left square bracket, 45, right square bracket, left square bracket, 45, right square bracket. Line 28. CLUSTER, hash. Line 29. left square bracket, 98, right square bracket, left square bracket, 98, right square bracket. Line 30. CLUSTER, hash. Line 31. left square bracket, 76, right square bracket, left square bracket, 76, right square bracket, left square bracket, 78, right square bracket, left square bracket, 78, right square bracket, left square bracket, 77, right square bracket. Line 32. CLUSTER, hash. Line 33. left square bracket, 87, right square bracket, left square bracket, 90, right square bracket, left square bracket, 89, right square bracket. Line 34. from cluster analysis import cluster Analysis Line 35. cluster Analysis, left parenthesis, double quotes, c s 150 exams dot t x t, double quotes, right parenthesis.

Back to Session

Line 1. time, latitude, longitude, depth, mag, mag Type, n s t, gap, dmin, r m s, net, i d, updated, place. Line 2. type, horizontal Error, depth Error, mag Error, mag N s t, status, location Source, mag Source.

Back to Code

Line 1. 2022 dash 11 dash 30T22 colon 30 colon 36 dot 483Z,5 dot 5812, 127 dot 1411, 150 dot 253, 4 dot 6, m b, 103, 113 comma. Line 2. 2 dot 144, 0 dot 54, u s, u s 6000j5w0, 2022 dash 12 dash 11T03 colon 42 colon 29 dot 040Z, left double quotes Philippine Islands. Line 3. region right double quotes, earthquake, 6 dot 82,5 dot 018, 0 dot 06, 81, reviewed, u s, u s.

Back to Code

Input 1. Line 1. import c s v. Line 2. with open, left parenthesis, double quotes, earthquakes dot c s v, double quotes, double quotes, r, double quotes, right parenthesis, as data File, colon. Line 3. Indented once. c s v Reader, equals, c s v dot reader, left parenthesis, data File, right parenthesis, hash, get iterator. Line 4. Indented once. titles, equals, next, left parenthesis, s v Reader, right parenthesis, hash, read titles line. Line 5. Indented once. print, left parenthesis, double quotes, titles, colon, double quotes, titles, right parenthesis, hash, output titles. Line 6. Indented once. print, left parenthesis, right parenthesis. Line 7. Indented once. earthquake Line, equals, next, left parenthesis, s v Reader, right parenthesis, hash, read first earthquake line. Line 8. Indented once. print, left parenthesis, double quotes, earthquake, colon, double quotes, earthquake Line, right parenthesis, hash, output all data. Line 9. Indented once. print, left parenthesis, right parenthesis. Line 10. Indented once. print, left parenthesis,

double quotes, latitude, colon, double quotes, earthquake Line, left square bracket, 1, right square bracket, right parenthesis, hash, output latitude. Line 11. Indented once. print, left parenthesis, double quotes, longitude, colon, double quotes, earthquake Line, left square bracket, 2, right square bracket, right parenthesis, hash, output longitude. Outside the code bracket. Line 1. titles, colon, left square bracket, single quote, time, single quote,, single quote, latitude, single quote,, single quote, longitude, single quote,, single quote, depth, single quote,, single quote, mag, single quote,, single quote, mag Type, single quote,, single quote, s t, single quote,, single quote, gap, single quote,, single quote, d min, single quote,, single quote, m s, single quote,, single quote, net, single quote,, single quote, d, single quote,, single quote, updated, single quote,, single quote, place, single quote,, single quote, type, single quote,, single quote, horizontal Error, single quote,, single quote, depth Error, single quote,, single quote, mag Error, single quote,, single quote, mag N s t, single quote,, single quote, status, single quote,, single quote, location Source, single quote,, single quote, mag Source, single quote, right square bracket. Line 2. Blank. Line 3. earthquake, colon, left square bracket, single quote, 2022negative 11negative 30T22, colon, 30, colon, 36.483Z, single quote,, single quote, 5.5812, single quote,, single quote, 127.1411, single quote,, single quote, 150.253, single quote,, single quote, 4.6, single quote,, single quote, b, single quote,, single quote, 103, single quote,, single quote, 113, single quote,, single quote, 2.144, single quote,, single quote, 0.54, single quote,, single quote, us, single quote,, single quote, s 6000 j 5 w 0, single quote,, single quote, 2022negative 12negative 11T03, colon, 42, colon, 29.040Z, single quote,, single quote, Philippine Islands region, single quote,, single quote, earthquake, single quote,, single quote, 6.82, single quote,, single quote, 5.018, single quote,, single quote, 0.06,

single quote,, single quote, 81, single quote,, single quote, reviewed, single quote,, single quote, us, single quote,, single quote, us, single quote, right square bracket. Line 4. Blank. Line 5. latitude, colon, 5.5812. Line 6. longitude, colon, 127.1411.

Back to Session

Line 1. def read Earthquake File, left parenthesis, filename, right parenthesis, colon. Line 2. Indented once. import c s v. Line 3. Blank. Line 4. Indented once. data List, equals, left square bracket, right square bracket. Line 5. Blank. Line 6. Indented once. with open, left parenthesis, double quotes, earthquakes.csv, double quotes, double quotes, r, double quotes, right parenthesis, as data File, colon. Line 7. Indented twice. c s v Reader, equals, c s v dot reader, left parenthesis, data File, right parenthesis. Line 8. Indented once. titles, equals, next, left parenthesis, s v Reader, right parenthesis, hash, read and skip the titles. Line 9. Blank. Line 10. Indented once. for a Line in c s v Reader, colon. Line 11. Indented twice. lat, equals, float, left parenthesis. Line, left square bracket, 1, right square bracket, right parenthesis, hash, extract latitude. Line 12. Indented twice. long, equals, float, left parenthesis, a Line, left square bracket, 2, right square bracket, right parenthesis, hash, extract longitude. Line 13. Indented twice. data List dot append, left parenthesis, left square bracket, long, lat, right square bracket, right parenthesis. Line 14. Blank. Line 15. Indented once. return data List. Outside the code box. Line 1. CLUSTER, hash. Line 2. left square bracket, 55.1789, 26.8456, right square bracket, left square bracket, 55.241, 26.8872, right square bracket, left square bracket, 24.2549, 38.2977, right square bracket, left square bracket, 24.2734, 38.3262, right square bracket. Line 3. left square bracket, 31.01, 40.8448, right square bracket, left square bracket, 27.7429, 35.3543, right square

bracket, left square bracket, 3.2541, 72.5191, right square bracket, left square bracket, 30.9831, 40.8356, right square bracket. Line 4. left square bracket, 14.1232, 36.1568, right square bracket, left square bracket, 26.4358, 35.841, right square bracket, left square bracket, 13.3283, 43.8974, right square bracket, left square bracket, 13.7911, negative 52.3065, right square bracket. Line 5. left square bracket, negative 16.4832, negative 58.8209, right square bracket, left square bracket, 55.2174, 26.9637, right square bracket, left square bracket, 56.0137, 27.7191, right square bracket, left square bracket, negative 11.0345, negative 20.1783, right square bracket. Line 6. left square bracket, 26.5301, 34.3958, right square bracket, left square bracket, 3.0727, 36.3127, right square bracket, left square bracket, 56.1092, 27.5471, right square bracket, left square bracket, negative 13.3391, negative 15.6013, right square bracket. Line 7. left square bracket, negative 13.2234, negative 15.5834, right square bracket, left square bracket, negative 13.1191, negative 15.5765, right square bracket, left square bracket, negative 13.2489, negative 15.6171, right square bracket.

Back to Listing

The cluster points are as follows: Cluster number 1 left square bracket 55.1789, 26.8456 right square bracket left square bracket 55.241, 26.8872 right square bracket left square bracket 24.2549, 38.2977 right square bracket left square bracket 24.2734, 38.3262 right square bracket left square bracket 31.01, 40.8448 right square bracket left square bracket 27.7429, 35.3543 right square bracket left square bracket 3.2541, 72.5191 right square bracket left square bracket 30.9831, 40.8356 right square bracket left square bracket 14.1232, 36.1568 right square bracket left square bracket 26.4358, 35.841 right square bracket left square bracket

13.3283, 43.8974 right square bracket left square bracket 13.7911, negative 52.3065 right square bracket left square bracket negative 16.4832, negative 58.8209 right square bracket left square bracket 55.2174, 26.9637 right square bracket left square bracket 56.0137, 27.7191 right square bracket left square bracket negative 11.0345, negative 20.1783 right square bracket left square bracket 26.5301, 34.3958 right square bracket left square bracket 3.0727, 36.3127 right square bracket left square bracket 56.1092, 27.5471 right square bracket left square bracket negative 13.3391, negative 15.6013 right square bracket left square bracket negative 13.2234, negative 15.5834 right square bracket left square bracket negative 13.1191, negative 15.5765 right square bracket left square bracket negative 13.2489, negative 15.6171 right square bracket, ellipsis.

Back to Equation

Line 1. def visualize Quakes, left parenthesis, data File, right parenthesis, colon. Line 2. Indented once. data List, equals, read Earthquake File, left parenthesis, data File, right parenthesis. Line 3. Indented once. quake Centroids, equals, create Centroids, left parenthesis, 6, data List, right parenthesis. Line 4. Indented once. clusters, equals, create Clusters, left parenthesis, 6, quake Centroids, data List, 7, right parenthesis. Line 5. Blank. Line 6. Indented once. quake T, equals, turtle dot Turtle, left parenthesis, right parenthesis. Line 7. Indented once. quake Win, equals, turtle dot Screen, left parenthesis, right parenthesis. Line 8. Indented once. quake Win dot b g pic, left parenthesis, double quotes, worldmap.gif, double quotes, right parenthesis. Line 9. Indented once. quake Win dot screensize, left parenthesis, 448, 252, right parenthesis. Line 10. Blank. Line 11. Indented once. w Factor, equals, left parenthesis, quake Win dot screensize, left parenthesis, right parenthesis, left square bracket, 0, right square

bracket, forward slash, 2, right parenthesis, forward slash, 180. Line 12. Indented once. h Factor, equals, left parenthesis, quake Win dot screensize, left parenthesis, right parenthesis, left square bracket, 1, right square bracket, forward slash, 2, right parenthesis, forward slash, 90. Line 13. Blank. Line 14. Indented once. quake T dot hide turtle, left parenthesis, right parenthesis. Line 15. Indented once. quake T dot up, left parenthesis, right parenthesis. Line 16. Blank. Line 17. Indented once. color List, equals, left square bracket, double quotes, red, double quotes, double quotes, lawn green, double quotes,, double quotes, blue, double quotes,, double quotes, orange, double quotes,, double quotes, cyan, double quotes,, double quotes, yellow, double quotes, right square bracket. Line 18. Blank. Line 19. Indented once. for cluster Index in range, left parenthesis, 6, right parenthesis, colon. Line 20. Indented twice. quake T dot color, left parenthesis, color List, left square bracket, cluster Index, right square bracket, right parenthesis, hash, choose cluster color. Line 21. Indented twice. for a Index in clusters, left square bracket, cluster Index, right square bracket, colon. Line 22. Indented thrice. lon, equals, data List, left square bracket, Index, right square bracket, left square bracket, 0, right square bracket. Line 23. Indented thrice. lat, equals, data List, left square bracket, Index, right square bracket, left square bracket, 1, right square bracket. Line 24. Indented thrice. quake T dot go to, left parenthesis, lon, asterisk, w Factor, lat, asterisk, h Factor, right parenthesis. Line 25. Indented thrice. quake T dot, left parenthesis, right parenthesis. Line 26. Indented twice. quake Win dot exit on click, left parenthesis, right parenthesis. Code lines to visualize earthquake clusters.

Back to Listing

The table has three rows and nine columns. Row entries are as follows. Row 1. rail 1, n, blank, a, c, I, blank, m, r. Row 2. rail 2, e, i, d, o, n, t, o, o. Row 3. rail 3, w, p, blank, m, g, o, r, w.

Back to Table

Input 1. Line 1. cipher Text, equals, double quotes, a c i m r e I d o n t o o w p m g o r w, double quotes. Line 2. for i in range, left parenthesis, 2, len, left parenthesis, cipher Text, right parenthesis, plus, 1, right parenthesis, colon. Line 3. Indented once. print, left parenthesis, rail Decrypt, left parenthesis, cipher Text, i, right parenthesis, right parenthesis. Outside the code box. Line 1. left square bracket, single quote, n, single quote, single quote, a o c o i w, single quote, single quote, m, single quote, single quote, m e g i o d r o w, single quote, right square bracket. Line 2. left square bracket, single quote, new, single quote, single quote, pad, single quote, single quote, coming, single quote, single quote, tomorrow, single quote, right square bracket. Line 3. left square bracket, single quote, m n, single quote, single quote, t m a e o g c i o o I d w r, single quote, single quote, p w, single quote, right square bracket. Line 4. left square bracket, single quote, i e n w, single quote, single quote, t p a m d o, single quote, single quote, r o o m, single quote, right square bracket. Line 5. left square bracket, single quote, i e n w g, single quote, single quote, t p o a m d o, single quote, single quote, c r o o m w, single quote, right square bracket. Line 6. left square bracket, ‘n c m i n o’, single quote, r d t w m a, single quote, single quote, o o p g, single quote, right square bracket. Line 7. left square bracket, ‘n c m i n o’, single quote, o, single quote, single quote, r d t w m r a, single quote, single quote, o o p g w, single quote, right square bracket. Line 8. left square bracket, ‘naimed now’, single quote, c, single quote, single quote, i o t o p, single quote, right square bracket.

Line 9. left square bracket, ‘naimed now’, single quote, c, single quote, single quote, i o t o p m, single quote, right square bracket. Line 10. left square bracket, ‘naimed now’, single quote, g, single quote, single quote, c, single quote, single quote, i o t o p m o, single quote, right square bracket. Line 11. left square bracket, ‘naimed now’, single quote, gr, single quote, single quote, c, single quote, single quote, i o t o p m o w, single quote, right square bracket. Line 12. left square bracket, single quote, n, single quote, ‘a c i’, single quote, r e i d o n, single quote, right square bracket. Line 13. left square bracket, single quote, n, single quote, ‘a c i’, single quote, r e i d o n t, single quote, right square bracket. Line 14. left square bracket, single quote, n, single quote, ‘a c i’, single quote, r e i d o n t o, single quote, right square bracket. Line 15. left square bracket, single quote, n, single quote, ‘a c i’, single quote, r e i d o n t o o, single quote, right square bracket. Line 16. left square bracket, single quote, n, single quote, ‘a c i’, single quote, r e i d o n t o o w, single quote, right square bracket. Line 17. left square bracket, single quote, n, single quote, ‘a c i’, ‘m r e i d o n t o o w p’, right square bracket. Line 18. left square bracket, single quote, n, single quote, ‘a c i’, ‘m r e i d o n t o o w p’, right square bracket. Line 19. left square bracket, single quote, n, single quote, ‘a c i’, ‘m r e i d o n t o o w p’, single quote, m, single quote, right square bracket. Line 20. left square bracket, single quote, n, single quote, ‘a c i’, ‘m r e i d o n t o o w p’, single quote, g, single quote, right square bracket. Line 21. left square bracket, single quote, n, single quote, ‘a c i’, ‘m r e i d o n t o o w p’, single quote, g o, single quote, right square bracket. Line 22. left square bracket, single quote, n, single quote, ‘a c i’, ‘m r e i d o n t o o w p’, single quote, g o r, single quote, right square bracket. Line 23. left square bracket, single quote, n, single quote, ‘a c i’, ‘m r e i d o n t o o w p’, single quote, g o r w, single quote, right square bracket.

Back to Session

Line 1. def create Word Dict, left parenthesis, Name, right parenthesis, colon. Line 2. Indented once. my Dict, equals, left brace, right brace. Line 3. Indented once. with open, left parenthesis, Name, single quote, r, single quote, right parenthesis, as my File, colon. Line 4. Indented twice. for line in my File, colon. Line 5. Indented thrice. my Dict, left square bracket, line, left square bracket, colon, negative 1, right square bracket, right square bracket, equals, True, hash, set all values to True. Line 6. Indented twice. return my Dict.

Back to Listing

Line 1. def rail Break, left parenthesis, cipher Text, right parenthesis, colon. Line 2. Indented once. word Dict, equals, create Word Dict, left parenthesis, single quote, word list dot t x t, single quote, right parenthesis. Line 3. Indented once. cipher Len, equals, len, left parenthesis, cipher Text, right parenthesis. Line 4. Indented once. max Good So Far, equals, 0. Line 5. Indented once. best Guess, equals, double quotes, No words found in dictionary, double quotes, hash, default response. Line 6. Indented once. for i in range, left parenthesis, 2, cipher Len, plus, 1, right parenthesis, colon. Line 7. Indented twice. words, equals, rail Decrypt, left parenthesis, cipher Text, i, right parenthesis. Line 8. Indented twice. good Count, equals, 0, hash, reset for new list. Line 9. Indented twice. for w in words, colon. Line 10. Indented thrice. if w in word Dict, colon. Line 11. Indented quarce. good Count, equals, good Count, plus, 1. Line 12. Indented twice. if good Count, right angle bracket, max Good So Far, colon, hash, if more words in this list. Line 13. Indented thrice. max Good So Far, equals, good Count. Line 14. Indented

thrice. best Guess, equals, double quotes, double quotes, .join, left parenthesis, words, right parenthesis, hash, join words in list with space. Line 15. Indented twice. return best Guess.

Back to Listing

Input 1. Line 1. cipher Text, equals, double quotes, a c i m r e i d o n t o o w p m g o r w, double quotes. Line 2. rail Break, left parenthesis, cipher Text, right parenthesis. Output 1: single quote, new i pad coming tomorrow, single quote.

Back to Session

The table has two rows and 24 columns. Row entries are as follows. Row 1. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24. Row 2. N, blank, a, c, i, blank, m, r, e, i, d, o, n, t, O, o, w, p, blank, m, g, o, r, w.

Back to Table

The table has two columns: Row-Major Order and Column-Major Order. Each column is further subdivided into 6 columns. Row entries are as follows. Row 1. 1, 2, 3, 4, 5, 6, 1, 4, 7, 10, 13, 16. Row 2. 7, 8, 9, 10, 11, 12, 2, 5, 8, 11, 14, 17. Row 3. 13, 14, 15, 16, 17, 18, 3, 6, 9, 12, 15, 18.

Back to Table

Line 1. def rail Decrypt, left parenthesis, cipher Text, num Rails, right parenthesis, colon. Line 2. Indented once. rail Len, equals, len, left parenthesis, cipher Text, right parenthesis, forward slash, forward slash, num Rails. Line 3. Indented once. solution, equals, single quote, single quote. Line 4. Indented once. for col

in range, left parenthesis, rail Len, right parenthesis, colon. Line 5. Indented twice. for rail in range, left parenthesis, num Rails, right parenthesis, colon. Line 6. Indented thrice. next Letter, equals, col, plus, left parenthesis, rail, asterisk, rail Len, right parenthesis. Line 7. Indented thrice. solution, equals, solution, plus, cipher Text, left square bracket, next Letter, right square bracket. Line 8. Indented once. return solution dot split, left parenthesis, right parenthesis.

Back to Listing

The vertical axis represents the relative frequency in percentage, ranging from 0.0 to 13.0 in increments of 0.5. The horizontal axis represents the letters a through z. The data inferred is presented as the letter: frequency percentage. a: 8; b: 1.2; c: 2.5; d: 4; e: 12.5; f: 2.3; g: 2; h: 6; i: 7; j: 0.3; k: 0.7; l: 4; m: 2; n: 6.3; o: 7; p: 1.5; q: 0.2; r: 6; s: 6.5; t: 9; u: 3; v: 0.7; w: 2; x: 0.3; y: 1.5; z: 0.2. All data is estimated.

Back to Figure

Input 1. Line 1. def remove Non letters, left parenthesis, my String, right parenthesis, colon. Line 2. Indented once. my String, equals, my String dot lower, left parenthesis, right parenthesis. Line 3. Indented once. new S t r, equals, double quotes, double quotes. Line 4. Indented once. letters, equals, single quote, b c d e f g h i j k l m n o p q r s t u v w x y z, single quote. Line 5. Indented once. for c h in my String, colon. Line 6. Indented twice. if c h in letters, colon. Line 7. Indented thrice. new S t r, equals, new S t r, plus, c h. Line 8. Indented once. return new S t r. Input 2. Line 1. text, equals, double quotes, Are there 25, 26, or 27 non-letters to remove, question mark, double quotes. Line 2.

remove Non letters, left parenthesis, text, right parenthesis.
Output 2: Single quote, are there or non letters to remove, single quote.

Back to Session

Line 1. def remove Non letters, left parenthesis, my String, right parenthesis, colon. Line 2. Indented once. my String, equals, my String dot lower, left parenthesis, right parenthesis. Line 3. Indented once. new S t r, equals, double quotes, double quotes. Line 4. Indented once. letters, equals, single quote, b c d e f g h i j k l m n o p q r s t u v w x y z, single quote. Line 5. Indented once. for c h in my String, colon. Line 6. Indented twice. if c h in letters, colon. Line 7. Indented thrice. new S t r, equals, new S t r, plus, c h. Line 8. Indented once. return new S t r. Line 9. Blank. Line 10. def letter Frequency, left parenthesis, text, right parenthesis, colon. Line 11. Indented once. text, equals, remove Non letters, left parenthesis, text, right parenthesis. Line 12. Indented once. l Count, equals, left brace, right brace. Line 13. Indented once. total, equals, len, left parenthesis, text, right parenthesis. Line 14. Indented once. for c h in text, colon, hash, count each letter, single quote, occurrence. Line 15. Indented twice. l Count, left square bracket, h, right square bracket, equals, l Count dot get, left parenthesis, h, 0, right parenthesis, plus, 1. Line 16. Indented once. for c h in l Count, colon, hash, calculate percentages. Line 17. Indented twice. l Count, left square bracket, h, right square bracket, equals, l Count, left square bracket, h, right square bracket, forward slash, total. Line 18. Indented once. return l Count.

Back to Listing

Input 1. Line 1. with open, left parenthesis, double quotes, wells dot t x t, double quotes,, single quote, r, single quote, encoding, equals, single quote, utf-8, single quote, right parenthesis, as wells, colon. Line 2. text, equals, wells dot read, left parenthesis, right parenthesis. Input 2. Line 1. lf, equals, letter Frequency, left parenthesis, text, right parenthesis. Line 2. for letter in, single quote, b c d e f g h i j k l m n o p q r s t u v w x y z, single quote, colon. Line 3. Indented once. print, left parenthesis, letter, lf dot get, left parenthesis, letter, right parenthesis, right parenthesis. Output. Line 1. a, colon, 0.083208914. Line 2. b, colon, 0.015231739. Line 3. c, colon, 0.023973586. Line 4. d, colon, 0.04755707. Line 5. e, colon, 0.125098856. Line 6. f, colon, 0.022874206. Line 7. g, colon, 0.024200555. Line 8. h, colon, 0.06027442. Line 9. i, colon, 0.068158041. Line 10. j, colon, 0.000978803. Line 11. k, colon, 0.007674385. Line 12. l, colon, 0.037960543. Line 13. m, colon, 0.025601379. Line 14. n, colon, 0.071995234. Line 15. o, colon, 0.07110509. Line 16. p, colon, 0.01806885. Line 17. q, colon, 0.000673814. Line 18. r, colon, 0.059877224. Line 19. s, colon, 0.060150296. Line 20. t, colon, 0.096678098. Line 21. u, colon, 0.02681779. Line 22. v, colon, 0.008844693. Line 23. w, colon, 0.023047979. Line 24. x, colon, 0.001464658. Line 25. y, colon, 0.018104313. Line 26. z, colon, 0.000379464.

Back to Session

Input 1. list, left parenthesis, lf dot items, left parenthesis, right parenthesis, right parenthesis, left square bracket, colon, 3, right square bracket. Output 1. Line 1. left square bracket, left parenthesis, single quote, t, single quote, 0.096678097859045 24, right parenthesis. Line 2. left parenthesis, single quote, h, single quote, 0.060274419544856496, right parenthesis. Line 3. left

parenthesis, single quote, e, single quote, 0.1250988555804197, right parenthesis, right square bracket.

Back to Figure

Input 1. Line 1. x List, equals, left square bracket, 3, 7, 4, 9, right square bracket. Line 2. x List dot sort, left parenthesis, reverse, equals, True, right parenthesis. Line 3. x List. Output 1. Left square bracket, 9, 7, 4, 3, right square bracket.

Back to Session

Line 1. def get Freq, left parenthesis, t, right parenthesis, colon. Line 2. Indented once. return t, left square bracket, 1, right square bracket, hash, return second item in the tuple.

Back to Listing

Input 1. Line 1. lf List, equals, list, left parenthesis, lf dot items, left parenthesis, right parenthesis, right parenthesis. Line 2. lf List dot sort, left parenthesis, key, equals, get Freq, reverse, equals, True, right parenthesis. Line 3. for entry in lf List, colon, hash, each entry is a tuple. Indented once. Line 4. print, left parenthesis, double quotes, left brace, 0, right brace, left brace, 1, colon, dot 3 f, right brace, double quotes, dot format, left parenthesis, entry, left square bracket, 0, right square bracket, entry, left square bracket, 1, right square bracket, right parenthesis, right parenthesis. Output. Line 1. e, colon, 0.125. Line 2. t, colon, 0.097. Line 3. a, colon, 0.083. Line 4. n, colon, 0.072. Line 5. o, colon, 0.071. Line 6. i, colon, 0.068. Line 7. h, colon, 0.06. Line 8. s, colon, 0.06. Line 9. r, colon, 0.06. Line 10. d, colon, 0.048. Line 11. l, colon, 0.038. Line 12. u, colon, 0.027. Line 13. m, colon, 0.026. Line 14. g, colon, 0.024. Line 15. c,

colon, 0.024. Line 16. w, colon, 0.023. Line 17. f, colon, 0.023. Line 18. y, colon, 0.018. Line 19. p, colon, 0.018. Line 20. b, colon, 0.015. Line 21. v, colon, 0.009. Line 22. k, colon, 0.008. Line 23. x, colon, 0.001. Line 24. j, colon, 0.001. Line 25. q, colon, 0.001. Line 26. z, colon, 0.

Back to Session

Line 1. my List equals open bracket 4, 5, 3, 9, 1, 6, 7 close bracket. Line 2. my List dot sort open parenthesis close parenthesis.

Back to Code

The table has two columns: Frequency and Letter. Row entries are as follows. Row 1. 0.134, l. Row 2. 0.095, b. Row 3. 0.086, k. Row 4. 0.086, h. Row 5. 0.077, v. Row 6. 0.070, i. Row 7. 0.059, j. Row 8. 0.053, u. Row 9. 0.051, e. Row 10. 0.046, c. Row 11. 0.042, z. Row 12. 0.031, a. Row 13. 0.029, t. Row 14. 0.024, q. Row 15. 0.022, w. Row 16. 0.022, o. Row 17. 0.018, s. Row 18. 0.018, m. Row 19. 0.013, d. Row 20. 0.011, n. Row 21. 0.009, g. Row 22. 0.002, x. Row 23. 0.002, r. Row 24. 0.000, y. Row 25. 0.000, p. Row 26. 0.000, f.

Back to Table

Line 1. def maybe Add, left parenthesis, h, toList, right parenthesis, colon. Line 2. Indented once. if c h in, single quote, b c d e f g h i j k l m n o p q r s t u v w x y z, single quote, and c h not in toList, colon. Line 3. Indented twice. to List dot append, left parenthesis, h, right parenthesis.

Back to Listing

Input 1. Line 1. my List, equals, left square bracket, right square bracket. Line 2. maybe Add, left parenthesis, single quote, a, single quote, my List, right parenthesis, hash, a is a letter and not in the list Line 3. my List. Output 1. Left square bracket, single quote, a, single quote, right square bracket. Input 2. Line 1. maybe Add, left parenthesis, single quote, dash, single quote, my List, right parenthesis, hash, is not a letter Line 2. my List. Output 2. Left square bracket, single quote, a, single quote, right square bracket. Input 3. Line 1. maybe Add, left parenthesis, single quote, b, single quote, my List, right parenthesis, hash, b is a letter and not in the list Line 2. my List. Output 3. Left square bracket, single quote, a, single quote, single quote, b, single quote, right square bracket. Input 4. Line 1. maybe Add, left parenthesis, single quote, a, single quote, my List, right parenthesis, hash, a is already in the list Line 2. my List. Output 4. Left square bracket, single quote, a, single quote, single quote, b, single quote, right square bracket.

Back to Session

Line 1. def neighbor Count, left parenthesis, text, right parenthesis, colon. Line 2. Indented once. n b Dict, equals, left brace, right brace. Line 3. Indented once. text, equals, text dot lower, left parenthesis, right parenthesis. Line 4. Indented once. for i in range, left parenthesis, len, left parenthesis, text, right parenthesis, - 1, right parenthesis, colon. Line 5. Indented twice. n b List, equals, n b Dict dot set default, left parenthesis, text, left square bracket, i, right square bracket, left brace, right brace, right parenthesis. Line 6. Indented twice. n b List, equals, n b Dict dot set default, left parenthesis, text, left square bracket, i, plus, 1, right square bracket, left brace, right brace, right parenthesis. Line 7. Indented twice. maybe Add, left parenthesis,

text, left square bracket, i, plus, 1, right square bracket, b List, right parenthesis. Line 8. Indented twice. maybe Add, left parenthesis, text, left square bracket, i, right square bracket, b List, right parenthesis. Line 9. Indented once. for key in n b Dict dot keys, left parenthesis, right parenthesis, colon, hash, replace letters list with count. Line 10. Indented twice. n b Dict, left square bracket, key, right square bracket, equals, len, left parenthesis, b Dict dot get, left parenthesis, key, right parenthesis, right parenthesis. Line 11. Indented once. return n b Dict.

Back to Listing

Line 1. n b List, equals, n b Dict dot get, left parenthesis, text, left square bracket, 1, right square bracket, right parenthesis. Line 2. if n b List, equals, equals, None, colon. Line 3. Indented once. n b Dict, left square bracket, text, left square bracket, i, right square bracket, right square bracket, equals, left square bracket, right square bracket. Line 4. Indented once. n b List, equals, n b Dict, left square bracket, text, left square bracket, i, right square bracket, right square bracket.

Back to Listing

Input 1. Line 1. freq Dict, equals, neighbor Count, left parenthesis, text, right parenthesis. Line 2. for i in, single quote, ent, single quote, colon. Line 3. Indented once. print, left parenthesis, i, freq Dict, left square bracket, i, right square bracket, right parenthesis. Output. Line 1. e 26. Line 2. n 25. Line 3. t 21.

Back to Session

Line 1. def maybe Add, left parenthesis, h, to Dict, right parenthesis, colon. Line 2. Indented once. if c h in, single quote, b c d e f g h i j k l m n o p q r s t u v w x y z, single quote, colon. Line 3. Indented twice. to Dict, left square bracket, h, right square bracket, equals, to Dict dot set default, left parenthesis, h, 0, right parenthesis, plus, 1. Line 4. Blank. Line 5. def neighborCount, left parenthesis, text, right parenthesis, colon. Line 6. Indented once. n b Dict, equals, left brace, right brace. Line 7. Indented once. text, equals, text dot lower, left parenthesis, right parenthesis. Line 8. Indented once. for i in range, left parenthesis, len, left parenthesis, text, right parenthesis, negative 1, right parenthesis, colon. Line 9. Indented twice. n b List, equals, n b Dict dot set default, left parenthesis, text, left square bracket, i, right square bracket, left brace, right brace, right parenthesis. Line 10. Indented twice. maybe Add, left parenthesis, text, left square bracket, i, plus, 1, right square bracket, b List, right parenthesis. Line 11. Indented twice. n b List, equals, n b Dict dot set default, left parenthesis, text, left square bracket, i, plus, 1, right square bracket, left brace, right brace, right parenthesis. Line 12. Indented twice. maybe Add, left parenthesis, text, left square bracket, i, right square bracket, b List, right parenthesis. Line 13. Indented once. return n b Dict.

Back to Listing

The table has 26 rows and 26 columns. The row and column headers are a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, and z. Row entries are as follows. Row 1. 0, 0, 0, 3, 5, 1, 0, 4, 3, 0, 0, 5, 2, 8, 0, 2, 0, 10, 5, 8, 1, 1, 6, 0, 0, 0. Row 2. 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 1, 0. Row 3. 0, 0, 0, 0, 4, 0, 0, 1, 1, 0, 0, 1, 0, 2, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0. Row 4. 3, 0, 0, 0, 9, 0, 2, 0, 6, 0, 0, 0, 0, 9, 0, 0, 0, 2, 3, 0, 0, 0, 0,

0, 1, 0. Row 5. 5, 1, 4, 9, 4, 1, 3, 17, 2, 0, 0, 2, 7, 7, 0, 3, 0, 15, 3, 0, 0, 4, 2, 0, 1, 2. Row 6. 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 3, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0. Row 7. 0, 0, 0, 2, 3, 0, 0, 0, 1, 0, 0, 0, 0, 11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. Row 8. 4, 0, 1, 0, 17, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 16, 0, 0, 0, 0, 0, 0. Row 9. 3, 1, 1, 6, 2, 0, 1, 3, 0, 0, 0, 2, 1, 20, 0, 2, 0, 8, 2, 8, 0, 4, 1, 0, 0, 0. Row 10. 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0. Row 11. 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0. Row 12. 5, 0, 1, 0, 2, 2, 0, 0, 2, 0, 0, 6, 1, 2, 3, 0, 0, 2, 0, 0, 1, 0, 0, 0, 7, 0. Row 13. 2, 1, 0, 0, 7, 0, 0, 0, 1, 0, 0, 1, 2, 0, 7, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0. Row 14. 8, 0, 2, 9, 7, 0, 11, 0, 20, 0, 0, 2, 0, 0, 8, 0, 0, 2, 0, 2, 4, 0, 0, 0, 1, 0. Row 15. 0, 0, 2, 0, 0, 3, 0, 0, 0, 0, 0, 3, 7, 8, 0, 0, 0, 4, 2, 3, 2, 1, 2, 0, 0, 0. Row 16. 2, 0, 0, 0, 3, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. Row 17. 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. Row 18. 10, 2, 1, 2, 15, 2, 0, 0, 8, 0, 0, 2, 1, 2, 4, 0, 0, 4, 0, 3, 1, 0, 0, 0, 1, 0. Row 19. 5, 0, 0, 3, 3, 0, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 12, 4, 0, 0, 0, 0, 0. Row 20. 8, 0, 0, 0, 0, 0, 0, 16, 8, 0, 0, 0, 0, 2, 3, 0, 0, 3, 12, 0, 0, 0, 0, 0, 0, 0. Row 21. 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 4, 2, 0, 0, 1, 4, 0, 0, 0, 0, 0, 0, 0. Row 22. 1, 0, 0, 0, 4, 0, 0, 0, 4, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. Row 23. 6, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. Row 24. 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0. Row 25. 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 7, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0. Row 26. 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.

Back to Table

Input 1. d, equals, neighbor Count, left parenthesis, text, right parenthesis. Input 2. print, left parenthesis, d, left square bracket, single quote, e, single quote, right square bracket, right parenthesis. Output. Line 1. left brace, single quote, h, single

quote, colon, 8632, single quote, j, single quote, colon, 144, single quote, c, single quote, colon, 1663, single quote, t, single quote, colon, 3143, single quote, n, single quote, colon, 4541, single quote, b, single quote, colon, 1268, single quote, r, single quote, colon, 8255. Line 2. single quote, w, single quote, colon, 1436, single quote, l, single quote, colon, 2894, single quote, s, single quote, colon, 4139, single quote, d, single quote, colon, 4837, single quote, o, single quote, colon, 230, single quote, v, single quote, colon, 2087, single quote, f, single quote, colon, 660. Line 3. single quote, a, single quote, colon, 1928, single quote, g, single quote, colon, 1010, single quote, y, single quote, colon, 798, single quote, k, single quote, colon, 701, single quote, p, single quote, colon, 1333, single quote, m, single quote, colon, 2552, single quote, x, single quote, colon, 333. Line 4. single quote, e, single quote, colon, 2604, single quote, u, single quote, colon, 168, single quote, i, single quote, colon, 900, single quote, q, single quote, colon, 26, single quote, z, single quote, colon, 70, right brace. Input 3. print, left parenthesis, d, left square bracket, single quote, n, single quote, right square bracket, right parenthesis. Line 1. left brace, single quote, e, single quote, colon, 4541, single quote, b, single quote, colon, 113, single quote, a, single quote, colon, 5203, single quote, y, single quote, colon, 204, single quote, o, single quote, colon, 4017, single quote, i, single quote, colon, 6684, single quote, u, single quote, colon, 1257. Line 2. single quote, d, single quote, colon, 4040, single quote, s, single quote, colon, 915, single quote, c, single quote, colon, 576, single quote, l, single quote, colon, 226, single quote, t, single quote, colon, 1744, single quote, g, single quote, colon, 3185, single quote, h, single quote, colon, 32. Line 3. single quote, m, single quote, colon, 34, single quote, w, single quote, colon, 292, single quote, r, single quote, colon, 375, single quote, f, single quote, colon, 101, single quote, v, single quote,

colon, 65, single quote, n, single quote, colon, 278, single quote, k, single quote, colon, 342, single quote, q, single quote, colon, 7. Line 4. single quote, p, single quote, colon, 13, single quote, j, single quote, colon, 10, single quote, x, single quote, colon, 6, right brace. Input 4. print, left parenthesis, d, left square bracket, single quote, t, single quote, right square bracket, right parenthesis. Line 1. left brace, single quote, h, single quote, colon, 10434, single quote, c, single quote, colon, 705, single quote, u, single quote, colon, 1712, single quote, e, single quote, colon, 3143, single quote, i, single quote, colon, 4516, single quote, s, single quote, colon, 3095, single quote, a, single quote, colon. Line 3847, single quote, o, single quote, colon, 3163, single quote, r, single quote, colon, 1884, single quote, n, single quote, colon, 1744, single quote, l, single quote, colon, 586, single quote, f, single quote, colon, 282, single quote, w, single quote, colon, 215, single quote, t, single quote, colon. Line 864, single quote, y, single quote, colon, 298, single quote, m, single quote, colon, 80, single quote, p, single quote, colon, 194, single quote, x, single quote, colon, 84, single quote, b, single quote, colon, 34, single quote, g, single quote, colon, 29, single quote, d, single quote, colon, 2, right brace.

Back to Session

The vertical axis represents the number of neighbors in 10,000s. The horizontal axis represents the letters a through z. The data is represented as letter: pairing with e, a, t, and n. a: 5, 0, 8, and 8. b: 0, 4, 9, and 4. c: 4, 0, 0, and 2. d: 9, 3, 0, and 9. e: 4, 5, 0, and 7. f: 1, 1, 0, and 0. g: 3, 0, 0, and 2. h: 17, 4, 16, and 0. i: 2, 3, 8, and 0. j: 0, 0, 0, and 2. k: 0, 0, 0, and 7. l: 0, 0, 0, and 7. m: 15, 10, 3, and 0. n: 3, 5, 12, and 2. o: 0, 8, 0, and 0. p: 0, 1, 0, and 2. q: 2, 1, 0, and 4. r: 7, 6, 0, and 0. s: 7, 0, 2, and 8. t: 0, 2, 3, and 0.

u: 0, 0, 0, and 0. v: 0, 0, 0, and 0. w: 0, 0, 0, and 0. x: 2, 5, 0, and 2. y: 0, 8, 0, and 0. z: 0, 1, 0, and 2.

Back to Figure

The vertical axis represents the relative frequency in percentage. The horizontal axis represents the letters a through z. The data is represented as letters: percentage for a through z. b: 0, 0, 8, 0, 9, 0, 0, 8, 2, 0, 2, 7, 4, 0, 0, 0, 1, 0, 0, 11, 0, 20, 2, 0, 0, and 2. k: 0, 2, 3, 0, 0, 0, 0, 8, 3, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 8, 0, 0, 0, and 0. l: 7, 7, 0, 3, 9, 0, 1, 5, 15, 3, 0, 4, 0, 4, 2, 0, 1, 2, 1, 3, 17, 2, 4, 0, 0, and 2. h: 2, 8, 0, 2, 3, 0, 0, 0, 10, 5, 8, 5, 1, 1, 6, 0, 0, 0, 1, 0, 4, 3, 0, 0, 0, and 5.

Back to Figure

The table has 26 columns: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, and z. Row entries are as follows. Row 1. b: 0, 0, 8, 0, 9, 0, 0, 8, 2, 0, 2, 7, 4, 0, 0, 0, 1, 0, 0, 11, 0, 20, 2, 0, 0, 2. Row 2. k: 0, 2, 3, 0, 0, 0, 0, 8, 3, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 8, 0, 0, 0, 0. Row 3. l: 7, 7, 0, 3, 9, 0, 1, 5, 15, 3, 0, 4, 0, 4, 2, 0, 1, 2, 1, 3, 17, 2, 4, 0, 0, 2. Row 4. h: 2, 8, 0, 2, 3, 0, 0, 0, 10, 5, 8, 5, 1, 1, 6, 0, 0, 0, 1, 0, 4, 3, 0, 0, 0, 5.

Back to Table

Input 1. Line 1. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, l, single quote, single quote, E, single quote, right parenthesis. Line 2. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, h, single quote, single quote, A, single quote, right parenthesis. Line 3. cipher Text. Output. Single quote, u E i E a A v b E e j k A b e v b t A k k u E E e t E c s k u E d v k k u A k k u E k u v b t u A e a A e E s c I v

k j E z s j k A i v b t A k v k j j k i A b t E A d d E A i A b w E A j k c b v j u E e w u v E s z q A k v k j m b m j m A z j u A d E A b e w c z c m i A b e e v a z q d E i w E v n v b t E n E b k u E b j c a E E n v e E b w E c s e E j v t b v b v k j A i i v n A z k u E E A i z q a c i b v b t o A j o c b e E i s m z z q j k v z z A b e k u E j m b x m j k w z E A i v b t k u E d v b E k i E E j k c o A i e j o E q g i v e t E o A j A z i E A e q o A i a u E e v e b c k i E a E a g E i u E A i v b t A b q g v i e j k u A k a c i b v b t k u E i E o A j w E i k A v b z q b c g i E E r E j k v i i v b t A b e k u E c b z q j c m b e j o E i E k u E s A v b k a c n E a E b k j s i c a o v k u v b k u E w v b e E i q w q z v b e E i u E o A j A z z A z c b E c b k u E w c a a c b, single quote.

Back to Session

Line 1. def sort By Len, left parenthesis, w, right parenthesis, colon. Line 2. return len, left parenthesis, w, right parenthesis.

Back to Listing

Input 1. Line 1. cipher Words, equals, cipher Text dot split, left parenthesis, right parenthesis. Line 2. cipher Words, dot sort, left parenthesis, key, equals, sort By Len right parenthesis. Line 3. print left parenthesis, cipherWords right parenthesis. Output. Single quote, u E, A k, c s, A k, A k, v b, u E, b c, u E, c b, k u E, k u E, d v k, k u E, u A e, s c i, v k j, v k j, A b e, A b e, v k j, k u E, o A j, A b e, k u E, j m b, k u E, o A j, e v e, b c k, A b q, o A j, A b e, k u E, k u E, k u E, o A j, A z z, k u E, E e t E, k u A k, a A e E, E n E b, k u E b, j c a E, x m j k, d v b E, o A i a, k u A k, c b z q, o E i E, s i c a, k u v b t, j u A d E, e v a z q, E A i z q, j k v z z, k i E E j, g v i e j, k u E i E, s A v b k, A z c b E, v k j E z s, w c z c m i, g i E E r E, j c m b e j, o v k u v b, w c a a c b, j k A i

v b t, j k i A b t E, w u v E s z q, m b m j m A z, A i i v n A z, a c i b v b t, k c o A i e j, A z i E A e q, u E A i v b t, a c i b v b t, w v b e E i q, i E a A v b E e, j k A b e v b t, E n v e E b w E, c s e E j v t b, w z E A i v b t, i E a E a g E i, j k v i i v b t, w q z v b e E i, o E q g i v e t E, w E i k A v b z q, a c n E a E b k j, A d d E A i A b w E, A j k c b v j u E e, d E i w E v n v b t, o c b e E i s m z z q, single quote.

Back to Session

Input 1. Line 1. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, b, single quote, single quote, N, single quote, right parenthesis. Line 2. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, e, single quote, single quote, D, single quote, right parenthesis. Line 3. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, k, single quote, single quote, T, single quote, right parenthesis. Line 4. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, u, single quote, single quote, H, single quote, right parenthesis. Line 5. cipher Text. Output. Single quote, H E i E a A v N E D j T A N D v N t A T T H E E D t E c s T H E d v T T H A T T H E T H v N t H A D a A D E s c I v T j E z s j T A i v N t A T v T j j T i A N t E A d d E A i A N w E A j T c N v j H E D w H v E s z q A T v T j m N m j m A z j H A d E A N D w c z c m i A N D D v a z q d E i w E v n v N t E n E N T H E N j c a E E n v D E N w E c s D E j v t N v N v T j A i i v N A z T H E E A i z q a c i N v N t o A j o c N D E i s m z z q j T v z z A N D T H E j m N x m j T w z E A i v N t T H E d v N E T i E E j T c o A i D j o E q g i v D t E o A j A z i E A D q o A i a H E D v D N c T i E a E a g E i H E A i v N t A N q g v i D j T H A T a c i N v N t T H E i E o A j w E i T A v N z q N c g i E E r E j T v i i v N t A N D T H E c N z q j c m N D j o E i E T H E s A v N T a c n E a E N T j s i c a o

v T H v N T H E w v N D E i q w q z v N D E i H E o A j A z z A z c N E c N T H E w c a a c N, single quote.

Back to Session

The table has 27 columns: Plaintext, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, and Z. Row entry is as follows. Row 1. Ciphertext, h, blank, blank, e, l, s, t, u, v, blank, blank, z, blank, b, c, blank, blank, blank, j, k, blank, blank, blank, blank, blank, blank.

Back to Table

Input 1. import r e. Input 2. re dot match, left parenthesis, single quote, dot A D E, single quote, single quote, FADE, single quote, right parenthesis. Output 2, left angle bracket, re dot Match object, semi colon, span, equals, left parenthesis, 0, 4, right parenthesis, match, equals, single quote, FADE, single quote, right angle bracket. Input 3. re dot match, left parenthesis, single quote, dot ADE, single quote, single quote, FADER, single quote, right parenthesis, hash, matches FADE, but not FADER. Output 3, left angle bracket, re dot Match object, semi colon, span, equals, left parenthesis, 0, 4, right parenthesis, match, equals, single quote, FADE, single quote, right angle bracket. Input 4. re dot match, left parenthesis, single quote, dot A D E, dollar sign, single quote, single quote, FADER, single quote, right parenthesis, hash, no match, dollar sign, matches ADE at the end. Input 5. re dot match, left parenthesis, single quote, dot A D E, single quote, single quote, A D E, single quote, right parenthesis, hash, no match. Input 6. re dot match, left parenthesis, single quote, dot A D E, single quote, single quote, FUDE, single quote, right parenthesis, hash, no match.

Back to Session

Line 1. def check Word, left parenthesis, r e g e x, right parenthesis, colon. Line 2. Indented once. res List, equals, left square bracket, right square bracket. Line 3. Indented once. with open, left parenthesis, single quote, wordlist.txt, single quote, single quote, r, single quote, right parenthesis, as word File, colon. Line 4. Indented twice. for line in word File, colon. Line 5. Indented thrice. if re dot match, left parenthesis, r e g e x, line, left square bracket, colon, -1, right square bracket, right parenthesis, colon. Line 6. Indented quarce. res List dot append, left parenthesis, line, left square bracket, colon, negative 1, right square bracket, right parenthesis. Line 7. Indented twice. return res List.

Back to Listing

Input 1. check Word, left parenthesis, single quote, .o dot ning, single quote, right parenthesis. Output 1. Left square bracket, single quote, morning, single quote, right square bracket. Input 2. Line 1. hash, match only unmapped characters Line 2. check Word, left parenthesis, single quote, left square bracket, b c j k m p q r u v w x y z, right square bracket, o, left square bracket, b c j k m p q r u v w x y z, right square bracket, ning, single quote, right parenthesis. Output 2. left square bracket, single quote, morning, single quote, right square bracket. Input 3. Line 1. hash, match a, any two characters, i, any one character, al Line 2. hash, stop finding characters when pattern ends Line 3. check Word, left parenthesis, single quote, a dot dot i dot al, dollar sign, single quote, right parenthesis. Output 3. left square bracket, single quote, admiral, single quote,, single quote, arrival, single quote, right square bracket. Input 4. Line 1. hash, match only unmapped

characters, stop when pattern ends Line 2. check Word, left parenthesis, single quote, a, left square bracket, b c j k m p q r u v w x y z, right square bracket, left square bracket, b c j k m p q r u v w x y z, right square bracket, i, left square bracket, b c j k m p q r u v w x y z, right square bracket, al, dollar sign, single quote, right parenthesis Output 4. Left square bracket, single quote, arrival, single quote, right square bracket.

Back to Session

Line 1. def check Word, left parenthesis, unused, pattern, right parenthesis, colon. Line 2. Indented once. res List, equals, left square bracket, right square bracket. Line 3. Indented once. with open, left parenthesis, single quote, word list dot t x t, single quote, single quote, r, single quote, right parenthesis, as word File, colon. Line 4. Indented twice. re Pat, equals, single quote, left square bracket, single quote, plus, unused, plus, single quote, right square bracket, single quote. Line 5. Indented twice. r e g e x, equals, re dot sub, left parenthesis, single quote, left square bracket, a-z, right square bracket, single quote, re Pat, pattern, right parenthesis, plus, single quote, dollar sign, single quote. Line 6. Indented twice. r e g e x, equals, r e g e x dot lower, left parenthesis, right parenthesis. Line 7. Indented twice. print, left parenthesis, single quote, matching, single quote, e g e x, right parenthesis. Line 8. Indented twice. for line in word File, colon. Line 9. Indented thrice. if re dot match, left parenthesis, e g e x, line, left square bracket, colon, negative 1, right square bracket, right parenthesis, colon. Line 10. Indented quarce. res List dot append, left parenthesis, line, left square bracket, colon, -1, right square bracket, right parenthesis. Line 11. Indented twice. return res List.

Back to Listing

Input 1. Line 1. check Word, left parenthesis, single quote, b c j k m p q r u v w x y z, single quote, single quote, W O N D E i F m L L q, single quote, right parenthesis. Line 2. matching wonde, left square bracket, b c j k m p q r u v w x y z, right square bracket, f, left square bracket, b c j k m p q r u v w x y z, right square bracket, ll, left square bracket, b c j k m p q r u v w x y z, right square bracket, dollar sign. Output 1. Left square bracket, single quote, wonderfully, single quote, right square bracket. Input 2. Line 1. check Word, left parenthesis, single quote, b c j k m p q r u v w x y z, single quote,, single quote, A i i I n A L, single quote, right parenthesis. Line 2. matching a, left square bracket, b c j k m p q r u v w x y z, right square bracket, left square bracket, b c j k m p q r u v w x y z, right square bracket, i, left square bracket, b c j k m p q r u v w x y z, right square bracket, al, dollar sign. Output 2. Left square bracket, single quote, arrival, single quote, right square bracket. Input 3. Line 1. check Word, left parenthesis, single quote, b c j k m p q r u v w x y z, single quote, single quote, m N m S m A L, single quote, right parenthesis. Line 2. matching, left square bracket, b c j k m p q r u v w x y z, right square bracket, n, left square bracket, b c j k m p q r u v w x y z, right square bracket, s, left square bracket, b c j k m p q r u v w x y z, right square bracket, al, dollar sign. Output 3. Left square bracket, single quote, unusual, single quote, right square bracket.

Back to Session

Input 1. Line 1. import re Line 2. c g, equals, re dot match, left parenthesis, single quote, F, left parenthesis, ., right parenthesis, L, left parenthesis, dot, right parenthesis, single quote, single

quote, FOILED, single quote, right parenthesis. Line 3. c g. Output 1. Left angle bracket, r e dot Match object, semi colon, span, equals, left parenthesis, 0, 6, right parenthesis, match, equals, single quote, FOILED, single quote, right angle bracket. Input 2. c g dot group, left parenthesis, 1, right parenthesis. Output 2. Single quote, O I, single quote. Input 3. c g dot group, left parenthesis, 2, right parenthesis. Output 3. Single quote, E D, single quote. Input 4. Line 1. cg, equals, r e dot match, left parenthesis, single quote, F, left parenthesis, right parenthesis, L, left parenthesis, right parenthesis, single quote, single quote, FOOLER, single quote, right parenthesis. Line 2. c g groups groups, left parenthesis, right parenthesis. Output 4. Left parenthesis, single quote, O O, single quote, single quote, E R, single quote, right parenthesis.

Back to Session

Line 1. def find Letters, left parenthesis, unused, pattern, right parenthesis, colon. Line 2. Indented once. res List, equals, left square bracket, right square bracket. Line 3. Indented once. with open, left parenthesis, single quote, wordlist.txt, single quote, single quote, r, single quote, right parenthesis, as word File, colon. Line 4. Indented twice. c t Letters, equals, r e dot find all, left parenthesis, single quote, left square bracket, a-z, right square bracket, single quote, pattern, right parenthesis. Line 5. Indented twice. print, left parenthesis, t Letters, right parenthesis. Line 6. Indented twice. re Pat, equals, single quote, left parenthesis, left square bracket, single quote, plus, unused, plus, single quote, right square bracket, right parenthesis, single quote. Line 7. Indented twice. r e g e x, equals, r dot sub, left parenthesis, single quote, left square bracket, a-z, right square bracket, single quote, re Pat, pattern, right parenthesis, plus, single quote, dollar sign,

single quote. Line 8. Indented twice. r e g e x, equals, r e g e x dot lower, left parenthesis, right parenthesis. Line 9. Indented twice. for line in word File, colon. Line 10. Indented thrice. my Match, equals, r e dot match, left parenthesis, regex, line, left square bracket, colon, negative 1, right square bracket, right parenthesis. Line 11. Indented thrice. if my Match, colon. Line 12. Indented quarce. matching Letters, equals, my Match dot groups, left parenthesis, right parenthesis. Line 13. Indented quarce. match List, equals, left square bracket, right square bracket. Line 14. Indented quarce. for l in matching Letters, colon. Line 15. Indented five folds. match List dot append, left parenthesis, dot upper, left parenthesis, right parenthesis, right parenthesis. Line 16. Indented quarce. res List dot append, left parenthesis, line, left square bracket, colon, -1, right square bracket, right parenthesis. Line 17. Indented quarce. res List dot append, left parenthesis, list, left parenthesis, zip, left parenthesis, t Letters, match List, right parenthesis, right parenthesis, right parenthesis. Line 18. Indented twice. return res List.

Back to Listing

Input 1. Line 1. import re Line 2. r e dot find all, left parenthesis, single quote, left square bracket, 123, right square bracket, single quote, single quote, 1,234, single quote, right parenthesis, hash, find single characters. Output 1. Line 1. left square bracket, single quote, 1, single quote, single quote, 2, single quote, single quote, 3, single quote, right square bracket. Input 2. r e dot find all, left parenthesis, single quote, left square bracket, 1234, right square bracket, plus, single quote, single quote, 1,234, single quote, right parenthesis, hash, find multiple characters. Output 2. Left square bracket, single quote, 1, single quote, single quote, 234, single quote, right square bracket. Input 3. r e dot find all, left

parenthesis, single quote, left square bracket, A-Z, right square bracket, single quote, single quote, Hello World, single quote, right parenthesis, hash, find capital letters. Output 3. left square bracket, single quote, H, single quote, single quote, W, single quote, right square bracket.

Back to Session

Input 1. list, left parenthesis, zip, left parenthesis, left square bracket, 1,2,3, right square bracket, left square bracket, 4,5,6, right square bracket, right parenthesis, right parenthesis. Output 1. left square bracket, left parenthesis, 1, 4, right parenthesis, left parenthesis, 2, 5, right parenthesis, left parenthesis, 3, 6, right parenthesis, right square bracket. Input 2. list, left parenthesis, zip, left parenthesis, left square bracket, single quote, a, single quote, single quote, b, single quote, single quote, c, single quote, right square bracket, left square bracket, single quote, Z, single quote, single quote, Y, single quote, single quote, X, single quote, right square bracket, right parenthesis, right parenthesis. Output 2. left square bracket, left parenthesis, single quote, a, single quote, single quote, Z, single quote, right parenthesis, left parenthesis, single quote, b, single quote, single quote, Y, single quote, right parenthesis, left parenthesis, single quote, c, single quote, single quote, X, single quote, right parenthesis, right square bracket. Input 3. list, left parenthesis, zip, left parenthesis, left square bracket, single quote, a, single quote, single quote, b, single quote, single quote, c, single quote, right square bracket, left square bracket, single quote, Z, single quote, single quote, Y, single quote, single quote, X, single quote, right square bracket, left square bracket, 1,2,3, right square bracket, right parenthesis, right parenthesis. Output 3. left square bracket, left parenthesis, single quote, a, single quote, single quote, Z, single quote, 1, right

parenthesis, left parenthesis, single quote, b, single quote, single quote, Y, single quote, 2, right parenthesis, left parenthesis, single quote, c, single quote, single quote, X, single quote, 3, right parenthesis, right square bracket.

Back to Session

Input 1. Line 1. find Letters, left parenthesis, single quote, b c j k m p q r u v w x y z, single quote, single quote, A i i I n A L, single quote, right parenthesis, hash, i -, right angle bracket, r, dash, right angle bracket, V Line 2. left square bracket, single quote, i, single quote, single quote, i, single quote, single quote, n, single quote, right square bracket. Output 1. Left square bracket, single quote, arrival, single quote, left square bracket, left parenthesis, single quote, i, single quote, single quote, R, single quote, right parenthesis, left parenthesis, single quote, i, single quote, single quote, R, single quote, right parenthesis, left parenthesis, single quote, n, single quote, single quote, V, single quote, right parenthesis, right square bracket, right square bracket. Input 2. Line 1. find Letters, left parenthesis, single quote, c j k m p q r u v w x y z, single quote, ‘A L i E A D q’, right parenthesis, hash, q -, right angle bracket, Y Line 2. left square bracket, single quote, i, single quote, single quote, q, single quote, right square bracket. Output 2. left square bracket, single quote, already, single quote, left square bracket, left parenthesis, single quote, i, single quote, single quote, R, single quote, right parenthesis, left parenthesis, single quote, q, single quote, single quote, Y, single quote, right parenthesis, right square bracket, right square bracket. Input 3. Line 1. find Letters, left parenthesis, single quote, c j k m p q r u v w x y z, single quote, single quote, i E E r E, single quote, right parenthesis, hash, g -, right angle bracket, B, dash, right angle bracket, Z Line

2. left square bracket, single quote, g, single quote, single quote, i, single quote, single quote, r, single quote, right square bracket. Output 3. Left square bracket, single quote, breeze, single quote, left square bracket, left parenthesis, single quote, g, single quote, single quote, B, single quote, right parenthesis, left parenthesis, single quote, i, single quote, single quote, R, single quote, right parenthesis, left parenthesis, single quote, r, single quote, single quote, Z, single quote, right parenthesis, right square bracket, right square bracket. Input 4. Find Letters, left parenthesis, single quote, c j k m p q r u v w x y z, single quote, single quote, N m S m A L, single quote, right parenthesis, hash, m -, right angle bracket, U Line 2. left square bracket, single quote, m, single quote, single quote, m, single quote, single quote, m, single quote, right square bracket. Output 4. left square bracket, single quote, unusual, single quote, left square bracket, left parenthesis, single quote, m, single quote, single quote, U, single quote, right parenthesis, left parenthesis, single quote, m, single quote, single quote, U, single quote, right parenthesis, left parenthesis, single quote, m, single quote, single quote, U, single quote, right parenthesis, right square bracket, right square bracket. Input 5. Line 1. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, i, single quote, single quote, R, single quote, right parenthesis. Line 2. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, n, single quote, single quote, V, single quote, right parenthesis. Line 3. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, q, single quote, single quote, Y, single quote, right parenthesis. Line 4. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, g, single quote, single quote, B, single quote, right parenthesis. Line 5. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, r, single quote, single quote, Z, single quote, right parenthesis. Line 6. cipher Text, equals, cipher

Text dot replace, left parenthesis, single quote, m, single quote, single quote, U, single quote, right parenthesis.

Back to Session

Input 1. Line 1. find Letters, left parenthesis, single quote, c j k m p q w x, single quote, single quote, w O a a O N, single quote, right parenthesis, hash, w -, right angle bracket, C, question mark, -, right angle bracket, P, question mark, -, right angle bracket, M Line 2. left square bracket, single quote, w, single quote, single quote, a, single quote, single quote, a, single quote, right square bracket. Output 1. Line 1. left square bracket, single quote, common, single quote. Line 2. left square bracket, left parenthesis, single quote, w, single quote, single quote, C, single quote, right parenthesis, left parenthesis, single quote, a, single quote, single quote, M, single quote, right parenthesis, left parenthesis, single quote, a, single quote, single quote, M, single quote, right parenthesis, right square bracket. Line 3. single quote, pompon, single quote. Line 4. left square bracket, left parenthesis, single quote, w, single quote, single quote, P, single quote, right parenthesis, left parenthesis, single quote, a, single quote, single quote, M, single quote, right parenthesis, left parenthesis, single quote, a, single quote, single quote, P, single quote, right parenthesis, right square bracket, right square bracket. Input 2. Line 1. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, w, single quote, single quote, C, single quote, right parenthesis. Line 2. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, a, single quote, single quote, M, single quote, right parenthesis. Input 3. Line 1. find Letters, left parenthesis, single quote, j k p q x, single quote, single quote, d I T, single quote, right parenthesis, hash, no definitive substitution Line 2. left square bracket, single

quote, d, single quote, right square bracket. Output 3. Left square bracket, single quote, kit, single quote, left square bracket, left parenthesis, single quote, d, single quote, single quote, K, single quote, right parenthesis, right square bracket, single quote, pit, single quote, left square bracket, left parenthesis, single quote, d, single quote, single quote, P, single quote, right parenthesis, right square bracket, right square bracket. Input 4. find Letters, left parenthesis, single quote, j k p q x, single quote, single quote, Add E A R A N C E, single quote, right parenthesis, hash, d -, right angle bracket, P Line 2. left square bracket, single quote, d, single quote, single quote, d, single quote, right square bracket. Output 4. Left square bracket, single quote, appearance, single quote, left square bracket, left parenthesis, single quote, d, single quote, single quote, P, single quote, right parenthesis, left parenthesis, single quote, d, single quote, single quote, P, single quote, right parenthesis, right square bracket, right square bracket. Input 5. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, d, single quote, single quote, P, single quote, right parenthesis. Input 6. Line 1. find Letters, left parenthesis, single quote, j k q x, single quote, single quote, x U S T, single quote, right parenthesis, hash, x -, right angle bracket, J Line 2. left square bracket, single quote, x, single quote, right square bracket. Output 6. Left square bracket, single quote, just, single quote, left square bracket, left parenthesis, single quote, x, single quote, single quote, J, single quote, right parenthesis, right square bracket, right square bracket. Input 7. cipher Text, equals, cipher Text dot replace, left parenthesis, single quote, x, single quote, single quote, J, single quote, right parenthesis.

Back to Session

The table has 27 columns: Plain text, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, and Z. Row entries are as follows. Row 1. Ciphertext, H, g, w, e, l, s, t, u, v, x, y, z, a, b, c, d, f, i, j, k, m, n, o, p, q, r.

Back to Table

The Python Turtle Graphics screen shows a tree with two initial divisions from the parent node, and each division node has two further divisions. The process repeats itself until the divisible nodes are marginal.

Back to Figure

Line 1. def hello left parenthesis right parenthesis colon. Line 2, indented once: print left parenthesis double quotes Hello World double quotes right parenthesis. Line 3, indented once: hello left parenthesis right parenthesis.

Back to Listing

The Python Turtle Graphics screen shows a set of nested boxes. The boxes are arranged in an organizational structure where one box is placed inside another, creating a hierarchy relationship.

Back to Figure

Line 1. def draw Square left parenthesis a Turtle, side right parenthesis colon. Line 2, indented once: for i in range left parenthesis 4 right parenthesis colon. Line 3, indented twice: a Turtle dot forward left parenthesis side right parenthesis. Line 4, indented twice: a Turtle dot right left parenthesis 90 right parenthesis. Line 5. Blank. Line 6. def nested Box left parenthesis

a Turtle, side right parenthesis colon. Line 7, indented once: if side right angle bracket equals 1 colon hash check for base case colon side left angle bracket 1. Line 8, indented twice: draw Square left parenthesis a Turtle, side right parenthesis. Line 9, indented twice: nested Box left parenthesis a Turtle, side dash 5 right parenthesis.

Back to Listing

Input 1. Line 1. from draw Squares import asterisk hash import drawing functions. Line 2. import turtle. Line 3. t equals turtle dot Turtle left parenthesis right parenthesis hash create a turtle. Line 4. nested Box left parenthesis t, 150 right parenthesis hash initial side 150. Line 5. turtle dot done left parenthesis right parenthesis hash stop the turtle.

Back to Session

Line 1. def count List left parenthesis a List right parenthesis colon. Line 2, indented once: if a List equals equals left square bracket right square bracket colon hash base case, list is empty. Line 3, indented twice: return 0. Line 4, indented once: else colon. Line 5, indented twice: return 1 plus count List left parenthesis a List left square bracket 1 colon right square bracket right parenthesis.

Back to Listing

Input 1. from recursive Count List import asterisk. Input 2. print left parenthesis count List left parenthesis left square bracket 1, 20, 10, 30 right square bracket right parenthesis right parenthesis hash multiple elements. Output: 4. Input 3. print left parenthesis count List left parenthesis left square bracket right square bracket

right parenthesis right parenthesis hash empty list. Output: 0. Input 4. print left parenthesis count List left parenthesis left square bracket 20 right square bracket right parenthesis right parenthesis hash list with 1 element. Output: 1.

Back to Session

Line 1. def tree left parenthesis t, trunk Length right parenthesis colon. Line 2, indented once: if trunk Length left angle bracket 5 colon hash check for base case. Line 3, indented twice: return. Line 4, indented once: else colon. Line 5, indented twice:t dot forward left parenthesis trunk Length right parenthesis. Line 6, indented twice:t dot right left parenthesis 30 right parenthesis. Line 7, indented twice: tree left parenthesis t, trunk Length - 15 right parenthesis. Line 8, indented twice:t dot left left parenthesis 60 right parenthesis. Line 9, indented twice: tree left parenthesis t, trunk Length - 15 right parenthesis. Line 10, indented twice: t dot right left parenthesis 30 right parenthesis. Line 11, indented twice: t dot backward left parenthesis trunk Length right parenthesis.

Back to Listing

Input 1. Line 1. from recursive Tree import asterisk. Line 2. import turtle. Input 2. t equals turtle dot Turtle left parenthesis right parenthesis. Input 3. Line 1. t dot up left parenthesis right parenthesis. Line 2. t dot go to left parenthesis 0, dash 225 right parenthesis. Line 3. t dot down left parenthesis right parenthesis. Line 4. t dot left left parenthesis 90 right parenthesis hash face up. Line 5. t dot color left parenthesis double quotes green double quotes, double quotes green double quotes right parenthesis. Input 4. Line 1. tree left parenthesis t, 110 right parenthesi hash draw the tree. Line 2. t dot hide turtle left parenthesis right

parenthesis. Line 3. turtle dot done left parenthesis right parenthesis.

Back to Session

An intricate geometric pattern known as the Sierpinski Triangle. This fractal showcases a self-replicating structure by combining smaller equilateral triangles recursively nested within each other. The progression of triangles forms an intricate, self-similar design that converges towards the center. The top portion comprises numbers 9, 10, 11, and 12. The left portion comprises numbers 1, 2, 3, and 4. The right portion comprises numbers 6, 7, 8, and 9.

Back to Figure

The Python Turtle Graphics screen an elaborate Sierpinski Triangle of depth 5. It showcases a mesmerizing fractal pattern created by recursively dividing an equilateral triangle into smaller triangles. Each iteration reveals a finer level of detail as the self-replicating triangles nest within one another.

Back to Figure

An equilateral triangle with vertices A, B, and C is shown. Points 1, 2, and 3 lie on the following sides: A B, B C, and A C. Dotted sides connect points 1 3, 2 3, and 1 2 to form a triangle.

Back to Figure

Line 1. def draw Triangle, left parenthesis, t, 1, 2, 3, right parenthesis, colon. Line 2, indented once: t dot up, left parenthesis, right parenthesis. Line 3, indented once: t dot go to,

left parenthesis, 1, right parenthesis. Line 4, indented once: t dot down, left parenthesis, right parenthesis. Line 5, indented once: t dot go to, left parenthesis, 2, right parenthesis. Line 6. t dot go to, left parenthesis, 3, right parenthesis. Line 7. t dot go to, left parenthesis, 1, right parenthesis. Line 8. Blank. Line 9. def mid Point, left parenthesis, 1, 2, right parenthesis, colon. Line 10, indented once: return, left parenthesis, left parenthesis, 1, left square bracket, 0, right square bracket, plus, p 2, left square bracket, 0, right square bracket, right parenthesis, forward slash, 2.0, left parenthesis, 1, left square bracket, 1, right square bracket, plus, p 2, left square bracket, 1, right square bracket, right parenthesis, forward slash, 2.0, right parenthesis. Line 11. Blank. Line 12. def sierpinski, left parenthesis, t, 1, 2, 3, depth, right parenthesis, colon. Line 13, indented once: if depth, right angle bracket, 0, colon. Line 14, indented twice: sierpinski, left parenthesis, t, 1, mid Point, left parenthesis, 1, 2, right parenthesis, mid Point, left parenthesis, 1, 3, right parenthesis, depth negative 1, right parenthesis. Line 15, indented twice: sierpinski, left parenthesis, t, 2, mid Point, left parenthesis, 2, 3, right parenthesis, mid Point, left parenthesis, 2, 1, right parenthesis, depth negative 1, right parenthesis. Line 16, indented twice: sierpinski, left parenthesis, t, 3, mid Point, left parenthesis, 3, 1, right parenthesis, mid Point, left parenthesis, 3, 2, right parenthesis, depth negative 1, right parenthesis. Line 17, indented once: else, colon, hash, base case. Line 18, indented twice: draw Triangle, left parenthesis, t, 1, 2, 3, right parenthesis.

Back to Listing

Input 1. Line 1. from sierpinski import, asterisk. Line 2. import turtle. Input 2. Line 1. t, equals, turtle dot Turtle, left parenthesis, right parenthesis. Line 2. t dot color, left parenthesis, single

quote, dark orange, single quote, right parenthesis. Input 3. Line 1. sierpinski, left parenthesis, t, left square bracket, negative 225, negative 250, right square bracket, left square bracket, 225, negative 250, right square bracket, left square bracket, 0, 225, right square bracket, 5, right parenthesis. Line 2. t dot hide turtle, left parenthesis, right parenthesis. Line 3. turtle dot done, left parenthesis, right parenthesis.

Back to Session

At depth 2 of the Sierpinski triangle, the recursive structure unfolds as follows: The main function, initiates three branches, each corresponding to a sub-triangle with depth 1. The first branch involves drawing a Sierpinski triangle at depth 1, starting from the top vertex. This branch further divides into three sub-branches, each representing a Sierpinski triangle at depth 0, formed from the top-left, top-right, and bottom vertices of the original triangle. The second and third branches follow a similar pattern, starting from the top vertex of the top-right and bottom triangles, respectively. Each of these branches, in turn, leads to the drawing of smaller triangles at depth 0 within their boundaries.

Back to Figure

A snowflake forms symmetrical patterns. It has a hexagonal structure with six radial arms extending outward. Each arm showcases branching, resembling delicate fern-like fronds. These arms are covered in fine, feathery ice crystals.

Back to Figure

A screenshot of a Python Turtle Graphics window. The window shows the six radial arms that extends outward in a snowflake. Each arm has further branching with divisions.

Back to Figure

The Koch is created by iteratively adding smaller segments to each line segment of a triangle, forming a self-replicating pattern. Each iteration deepens the complexity of the curve. Describing the four levels of a Koch curve involves detailing the progression from the initial triangle to the fourth iteration. Level 1: It is an equilateral triangle. Each side of the triangle is divided into three equal segments, and the middle segment is replaced by two segments forming a V shape. Level 2: Moving to the second level involves applying the same process to each of the twelve line segments created in the first level. The V-shaped patterns on each segment are replaced with smaller segments, further enhancing the intricate, self-replicating nature of the curve. Level 3: The third level deepens the complexity even further. Each line segment generated in the second level is divided into three, and the middle segment is replaced with the V-shaped pattern. This recursive pattern continues, resulting in a more intricate and detailed curve. Level 4: At the fourth level, the process is repeated once more. Each of the line segments from the third level is divided into three, and the middle segment is replaced with the V-shaped pattern. This iteration enhances the intricate details of the curve, creating a highly complex and visually captivating structure.

Back to Figure

The instructions are as follows. Line 1. forward left parenthesis n right parenthesis. Line 2. left left parenthesis 60 right parenthesis. Line 3. forward left parenthesis n right parenthesis. Line 4. right left parenthesis 60 right parenthesis. Line 5. right left parenthesis 60 right parenthesis. Line 6. forward left parenthesis n right parenthesis. Line 7. left left parenthesis 60 right parenthesis. Line 8. forward left parenthesis n right parenthesis.

Back to Figure

Line 1. match, left angle bracket, subject, right angle bracket, colon. Line 2, indented once: case value 1, colon. Line 3. Indented twice, hash, statements to execute if subject, equals, equals, value 1. Line 4, indented once: case value 2, colon. Line 5. Indented twice, hash, statements to execute if subject, equals, equals, value 2. Line 6, indented once: Ellipses. Line 7, indented once: case, underscore, colon, hash, optional. Line 8. Indented twice, hash, statements to execute if subject does not match any case values.

Back to Figure

Line 1. def draw L S, left parenthesis, t, instructions, angle, distance, right parenthesis, colon. Line 2, indented once: for c m d in instructions, colon. Line 3, indented twice: match c m d, colon. Line 4, indented thrice: case, single quote, F, single quote, colon. Line 5, indented quarce: t dot forward, left parenthesis, distance, right parenthesis. Line 6, indented thrice: case, single quote, B, single quote, colon. Line 7, indented quarce: t dot backward, left parenthesis, distance, right parenthesis. Line 8, indented thrice: case, single quote, plus, single quote, colon. Line 9, indented quarce: t dot right, left parenthesis, angle, right

parenthesis. Line 10, indented thrice: case, single quote, dash, single quote, colon. Line 11, indented quarce: t dot left, left parenthesis, angle, right parenthesis. Line 12, indented thrice: case, underscore, colon, hash, wildcard matches any other value Line 13, indented quarce: print, left parenthesis, double quotes, Error, double quotes, m d, double quotes, is an unknown command, double quotes, right parenthesis.

Back to Listing

Line 1. def apply Production, left parenthesis, axiom, rules, n, right parenthesis, colon. Line 2, indented once: for i in range, left parenthesis, n, right parenthesis, colon. Line 3, indented twice: new String, equals, double quotes, double quotes. Line 4, indented twice: for c h in axiom, colon. Line 5, indented thrice: new String, equals, new String, plus, rules dot get, left parenthesis, h, h, right parenthesis. Line 6, indented twice: axiom, equals, new String Line 7, indented once: return axiom.

Back to Listing

Line 1 reads: production Rules equals left flower parenthesis A within single quotes colon B within single quotes comma. Line 2 reads: B within single quotes comma A B within single quotes right flower parenthesis.

Back to Figure

Input 1. from apply Production Rules import, asterisk. Input 2. Line 1. axiom, equals, single quote, A, single quote. Line 2. my Rules, equals, left brace, single quote, A, single quote, colon, single quote, B, single quote, single quote, B, single quote, colon, single quote, AB, single quote, right brace. Input 3. Line 1. for i

in range, left parenthesis, 10, right parenthesis, colon. Line 2, indented once: r e s, equals, apply Production, left parenthesis, axiom, my Rules, i, right parenthesis. Line 3, indented once: print, left parenthesis, double quotes, left brace, 0, colon, 2 d, right brace, left brace, 1, right brace, double quotes, .format, left parenthesis, e n, left parenthesis, e s, right parenthesis, e s, right parenthesis, right parenthesis. Output. Line 1. 1 A. Line 2. 1 B. Line 3. 2 A B. Line 4. 3 B A B. Line 5. 5 A B B A B. Line 6. 8 B A B A B B A B. Line 7. 13 A B B A B B A B A B B A B. Line 8. 21 B A B A B B A B A B B A B B A B A B B A B. Line 9. 34 A B B A B B A B A B B A B B A B A B B A B A B B A B B A B A B B A B. Line 10. 55 B A B A B B A B A B B A B B A B A B B A B A B B A B B A B A B B A B B A B A B B A B B A B A B B A B B A B A B B A B B A B A B B A B A B B A B B A B A B.

Back to Session

Line 1. def draw L S, left parenthesis, Turtle, instructions, angle, distance, right parenthesis, colon. Line 2, indented once: state Saver, equals, left square bracket, right square bracket, hash, list to save turtle state Line 3, indented once: for c m d in instructions, colon. Line 4, indented twice: match c m d, colon. Line 5, indented thrice: case, single quote, F, single quote, colon. Line 6, indented quarce: a Turtle dot forward, left parenthesis, distance, right parenthesis. Line 7, indented thrice: case, single quote, B, single quote, colon. Line 8, indented quarce: a Turtle dot backward, left parenthesis, distance, right parenthesis. Line 9, indented thrice: case, single quote, plus, single quote, colon. Line 10, indented quarce: a Turtle dot right, left parenthesis, angle, right parenthesis. Line 11, indented thrice: case, single quote, dash, single quote, colon. Line 12, indented quarce: a Turtle dot left, left parenthesis, angle, right parenthesis. Line 13, indented

thrice: case, single quote, left square bracket, single quote, colon. Line 14, indented quarce: p o s, equals, a Turtle dot position, left parenthesis, right parenthesis, hash, get turtle, single quote, position Line 15, indented quarce: head, equals, a Turtle dot heading, left parenthesis, right parenthesis, hash, and heading Line 16, indented quarce: state Saver dot append, left parenthesis, left parenthesis, pos, head, right parenthesis, right parenthesis, hash, save as tuple Line 17, indented thrice: case, single quote, right square bracket, single quote, colon. Line 18, indented quarce: pos, head, equals, state Saver dot pop, left parenthesis, right parenthesis, hash, get first items on list Line 19, indented quarce: a Turtle dot up, left parenthesis, right parenthesis. Line 20, indented quarce: a Turtle dot set position, left parenthesis, pos, right parenthesis, hash, restore position Line 21, indented quarce: a Turtle dot set heading, left parenthesis, head, right parenthesis, hash, and heading Line 22, indented quarce: a Turtle dot down, left parenthesis, right parenthesis.

Back to Listing

Line 1. def l System, left parenthesis, axiom, rules, depth, start Pos, heading, angle, length, right parenthesis, colon. Line 2, indented once: import turtle Line 3, indented once: a Turtle, equals, turtle dot Turtle, left parenthesis, right parenthesis. Line 4, indented once: a Turtle dot color, left parenthesis, single quote, GREEN, single quote, right parenthesis. Line 5, indented once: win, equals, turtle dot Screen, left parenthesis, right parenthesis. Line 6, indented once: a Turtle dot up, left parenthesis, right parenthesis. Line 7, indented once: a Turtle dot set position, left parenthesis, start Pos, right parenthesis. Line 8, indented once: a Turtle dot down, left parenthesis, right parenthesis. Line 9, indented once: a Turtle dot set heading, left parenthesis, heading,

right parenthesis. Line 10, indented once: new Rules, equals, apply Production, left parenthesis, axiom, rules, depth, right parenthesis. Line 11, indented once: draw L S, left parenthesis, Turtle, new Rules, angle, length, right parenthesis. Line 12, indented once: turtle dot done, left parenthesis, right parenthesis.

Back to Listing

Line 1. Input 1. Line 2. from draw l System import, asterisk. Line 3. import turtle Line 4. my Rules, equals, left brace, single quote, X, single quote, colon, single quote, F, left square bracket, -X, right square bracket, plus, X, single quote, single quote, F, single quote, colon, single quote, FF, single quote, right brace. Line 5. axiom, equals, single quote, X, single quote. Line 6. l System, left parenthesis, axiom, my Rules, 7, left parenthesis, 0, negative 200, right parenthesis, 90, 30, 2, right parenthesis.

Back to Session

The drawing has a slender stem with a bark-like texture. Extending from this central stem are needle-shaped leaves, arranged in an alternate pattern along the length of the stem. The leaves are narrow.

Back to Figure

Input 1: Line 1: count equals 1. Line 2: type, left parenthesis, count, right parenthesis. Output 1: i n t. Input 2: Line 1: radius equals 1 dot 57. Line 2: type left parenthesis, radius, right parenthesis. Output 2: float. Input 3: Line 1: is Raining equals False. Line 2: type, left parenthesis, is Raining, right parenthesis. Output 3: bool. Input 4: Line 1: my List equals left square bracket 23 comma 66 comma True right square bracket. Line 2: type left

parenthesis, my List, right parenthesis. Output 4: list. Input 5: Line 1: city equals double quotes Kingston double quotes. Line 2: type left parenthesis, city, right parenthesis. Output 5: s t r.

Back to Session

Input 1: Line 1: count equals 1. Line 2: count equals count plus 1. Line 3: count. Output 1: 2. Input 2: Line 1: count equals count dot double underscore add double underscore, left parenthesis, 1, right parenthesis, hash equivalent to count plus 1. Line 2: count. Output 2: 3. Input 3: Line 1: my List equals left square bracket, 23, comma, 66, comma, True, right square bracket. Line 2: my List left square bracket, 2, right square bracket hash get item at index 2. Output 3: True. Input 5: Line 1: my List dot double underscore get item double underscore left parenthesis, 2, right parenthesis, hash equivalent to my List left square bracket, 2, right square bracket. Output 5: True.

Back to Session

Line 1. class Class name colon. Line 2. d e f method 1 left parenthesis right parenthesis colon. Line 3.Ellipses. Line 4. d e f method 2 left parenthesis right parenthesis colon. Line 5.Ellipses. Line 6. Ellipses.

Back to Figure

Line 1: class Planet colon. Line 2: d e f double underscore in it double underscore left parenthesis, self, comma, i Name, comma, i Rad, comma, i M, comma i Dist, right parenthesis, colon. Line 3: self dot name equals i Name. Line 4: self dot radius equals i Rad. Line 5. self dot mass equals i M. Line 6: self dot distance equals i Dist.

Back to Listing

Input 1: Line 1: from planet class import asterisk. Input 2: Line 1: my Planet equals Planet, left parenthesis, double quotes X 25, double quotes, comma, 45, comma, 198, comma, 1000, right parenthesis. Line 2: my Planet. Output 2: left angle bracket, planet class dot Planet at 0 x 21 b 7 e 2179 a 0, right angle bracket. Input 3: Line 1: Type, left parenthesis, my Planet, right parenthesis. Output 3: planet class dot Planet. Input 4: Line 1: my Planet dot name hash retrieve the values of the instance variables. Output 4: single quotes X 25, single quotes. Input 5: Line 1: my Planet dot radius. Output 5: 45. Input 6: Line1: my Planet dot mass. Output 6: 198. Input 7: Line 1: my Planet dot distance. Output 7: 1000. Input 8: Line 1: my Planet dot mass equals 250 hash change the mass. Line 1: my Planet dot mass. Output 8: 250.

Back to Session

Line 1: class Planet colon. Line 2: d e f double underscore in it double underscore left parenthesis, self, comma, i Name, comma, i Rad, comma, i M, comma, i Dist, right parenthesis, colon. Line 3: self dot double underscore name equals i Name. Line 4: self dot double underscore radius equals I Rad. Line 5: self dot double underscore mass equals i M. Line 6: self dot double underscore distance equals i Dist.

Back to Listing

Input 1: Line 1: from planetclass2 import asterisk. Input 2: Line 1: my Planet equals Planet, left parenthesis, double quotes X 25 double quotes comma, 45, comma, 198, comma, 1000, right parenthesis. Line 2: my Planet dot double underscore name. The

output displays the following. Line 1: Attribute Error Trace back, left parenthesis, most recent call last, right parenthesis. Line 2: Input In, left square bracket, 2, right square bracket, comma, in, left angle bracket, cell line, colon, 2, right angle bracket, left parenthesis, right parenthesis. Line 3: indented once: 1 my Planet equals Planet, left parenthesis, double quotes X 25 double quotes comma, 45, comma, 198, comma, 1000, right parenthesis. Line 4: right angle bracket, 2 my Planet dot double underscore name. Line 5: Attribute Error colon single quotes, Planet, single quotes, object has no attribute single quotes double underscore name single quotes.

Back to Session

The My Planet object has a class with the name of "X 25", a radius of 45, a mass of 198, and a distance of 1000. The object is separated into two distinct layers. The inner layer, or object state, contains the instance variable names. The outer layer contains the names of the methods. The method is defined by in it underscore def.

Back to Figure

Line 1. d e f get Name left parenthesis, self, right parenthesis, colon. Line 2, indented once: return self dot double underscore name. Line 3: Blank. Line 4. d e f get Radius left parenthesis self, right parenthesis colon. Line 5, indented once: return self dot double underscore radius. Line 6: Blank. Line 7. d e f get Mass left parenthesis self, right parenthesis colon. Line 8, indented once: return self dot, double underscore, mass. Line 9: Blank. Line 10. d e f get Distance left parenthesis self, right parenthesis colon. Line 11, indented once: return self dot double underscore

distance. Line 12: Blank. Line 13: d e f get Volume left parenthesis self, right parenthesis colon. Line 14, indented once: v equals 4 forward slash 3 asterisk math dot pi asterisk self dot double underscore radius asterisk asterisk 2. Line 15, indented once: return v. Line 16: Blank. Line 17. d e f get Surface Area left parenthesis self, right parenthesis colon. Line 18, indented once: s a equals 4 asterisk math dot pi asterisk self dot double underscore radius asterisk asterisk 2. Line 19, indented once: return s a. Line 20: Blank. Line 21. d e f get Density left parenthesis self, right parenthesis colon. Line 22, indented once: d equals self dot double underscore mass forward slash self dot get Volume left parenthesis right parenthesis. Line 23, indented once: return d.

Back to Listing

Input 1: Line 1: from planet import asterisk. Line 2: my Planet equals Planet left parenthesis single quotes X 25 single quotes comma 45 comma 198 comma 1000 right parenthesis. Input 2: Line 1: my Planet dot get Name left parenthesis right parenthesis. Output 2: single quotes X 25 single quotes. Input 3: Line 1: my Planet dot get Radius left parenthesis right parenthesis. Output 3: 45. Input 4: Line 1: my Planet dot get Mass left parenthesis right parenthesis. Output 4: 198. Input 5: Line 1: my Planet dot get Distance left parenthesis right parenthesis. Output 5: 1000. Input 6: Line 1. my Planet dot get Volume left parenthesis right parenthesis. Output 6: 8482.30016469244. Input 7: Line 1: my Planet dot get Surface Area left parenthesis right parenthesis Output 7: 25446.900494077323. Input 8: Line 1: my Planet dot get Density left parenthesis right parenthesis. Output 8: 0.023342724986811 32.

Back to Session

Line 1: d e f set Name left parenthesis, self, comma, new Name, right parenthesis colon. Line 2, indented once: self dot double underscore name equals new name.

Back to Listing

Input 1: Line 1: from planet import asterisk. Line 2: my Planet equals Planet, left parenthesis, single quotes, X 25, single quotes, comma, 45, comma, 198, comma, 1000, right parenthesis. Input 2: Line 1: my Planet dot get Name, left parenthesis, right parenthesis. Output 2: single quotes X 25 single quotes. Input 3: Line 1: my Planet dot set Name left parenthesis, double quotes, Gamma Hydra double quotes, right parenthesis. Input 4: Line 1: my Planet dot get Name left parenthesis right parenthesis. Output 4: single quotes, Gamma Hydra, single quotes.

Back to Session

The table has four columns: Method, Operation, Example, and Use. Row entries are as follows. Row 1. double underscore getitem double underscore, Indexing, o b j 1 dot double underscore getitem double underscore open parentheses x close parentheses, o b j 1 left square bracket x right square bracket. Row 2. double underscore add double underscore, Addition, o b j 1 dot double underscore add double underscore open parentheses x close parentheses, o b j 1 equals o b j 1 plus x. Row 3. double underscore contains double underscore, Search, o b j 1 dot double underscore contains double underscore open parentheses s t r close parentheses, s t r in o b j 1. Row 4. double underscore s t r double underscore, Provide object as a string, o b j 1 dot double underscore s t r triple underscore open parentheses close parentheses, print open parentheses o b j 1

close parentheses. Row 5. double underscore e q double underscore, Equality, o b j 1 dot double underscore e q double underscore open parentheses o b j 2 close parentheses, o b j 1 double equal to o b j 2. Row 6. double underscore n e double underscore, Inequality, o b j 1 dot double underscore n e double underscore open parentheses o b j 2 close parentheses, o b j 1 exclamation equal to o b j 2. Row 7. double underscore l t double underscore, Less than, o b j 1 dot double underscore l t double underscore open parentheses o b j 2 close parentheses, o b j 1 less than o b j 2. Row 8. double underscore l e double underscore, Less than or equal to, o b j 1 dot double underscore l e double underscore open parentheses o b j 2 close parentheses, o b j 1 less than equal to o b j 2. Row 9. double underscore g t double underscore, Greater than, o b j 1 dot double underscore g t double underscore open parentheses o b j 2 close parentheses, o b j 1 greater than o b j 2. Row 10. double underscore g e double underscore, Greater than or equal to, o b j 1 dot double underscore g e double underscore open parentheses o b j 2 close parentheses, o b j 1 greater than equal to o b j 2. Row 11. double underscore l e n double underscore, length, o b j 1 dot double underscore l e n double underscore open parentheses close parentheses, returns length or number of elements.

Back to Table

Input 1: Line 1: from planet import asterisk. Line 2: my Planet equals Planet, left parenthesis, single quotes, X 25, single quotes, comma, 45, comma, 198, comma, 1000, right parenthesis. Input 2: Line 1: print, left parenthesis, my Planet, right parenthesis. Line 2: left angle bracket, planet dot Planet object at 0 x 00000207 D F 077 A 00, right angle bracket.

Back to Session

Line 1: d e f double underscore s t r double underscore left parenthesis, self, right parenthesis, colon. Line 2, indented once: return self dot double underscore name.

Back to Listing

Input 1: Line 1: from planet import asterisk. Line 2: my Home equals Planet, left parenthesis, single quotes, Earth, single quotes, comma, 6371, comma, 5.97 e 24, comma, 152097701, right parenthesis. Input 2: Line 1: print left parenthesis, my Home, right parenthesis. Line 2: Earth. Input 3: Line 1: my Home dot double underscore s t r double underscore left parenthesis right parenthesis. Output 3: single quotes, Earth, single quotes. Input 4: Line 1: s t r left parenthesis, my Home, right parenthesis. Output 4. single quotes, Earth, single quotes.

Back to Session

The My Planet object has a class with the name of “Earth”, a radius of 6371, a mass of 5.97 e 24, and a distance of 152097701. The object is separated into two distinct layers. The inner layer, or object state, contains the instance variable names. The outer layer contains the names of the methods. The in it method is defined by in it underscore def. The in it method is defined by in it underscore def. The s t r method is defined by s t r underscore def. The get Volume method is defined by get Volume underscore def. The get Surface Area method is defined by get Surface Area underscore def. The get Density method is defined by get Density underscore def. The get Name method is defined by get Name underscore def. The get Radius method is defined by get Radius

underscore def. The get Mass method is defined by get Mass underscore def. The get Distance method is defined by get Distance underscore def. The set Name method is defined by set Name underscore def.

Back to Figure

Line 1. d e f double underscore l t double underscore, left parenthesis, self, comma, other Planet right parenthesis colon. Line 2, indented once: return self dot double underscore distance left angle bracket other Planet dot double underscore distance. Line 3, Blank. Line 4: d e f double underscore g t double underscore left parenthesis self, comma, other Planet right parenthesis colon. Line 5, indented once: return self dot double underscore distance right angle bracket other Planet dot double underscore distance.

Back to Listing

Input 1: Line 1: from planet import asterisk. Line 2: earth equals Planet, left parenthesis, single quotes, Earth, single quotes, comma, 50, comma, 60, comma, 31, right parenthesis. Line 3: mercury equals Planet, left parenthesis, single quotes, Mercury, single quotes, comma, 19, comma, 10, comma, 25, right parenthesis. Line 4: mars equals Planet, left parenthesis, single quotes, Mars, single quotes, comma, 47, comma, 50, comma, 35, right parenthesis. Input 2: Line 1: mars, left angle bracket, earth, hash call the double underscore l t double underscore method. Output 2: False. Input 3: Line 1: earth, right angle bracket, mercury hash call the double underscore g t double underscore method. Output 3: True.

Back to Session

Input 1: Line 1: double underscore name double underscore. Line 2: single quotes, double underscore, main, double underscore, single quotes. Input 2: Line 1: from planet import asterisk. Line 2: my Planet equals Planet, left parenthesis, single quotes, X 25, single quotes, comma, 45, comma, 198, comma, 1000, right parenthesis. Input 3: Line 1: my Planet dot get Name, left parenthesis, right parenthesis. Output 3: single quotes X 25 single quotes.

Back to Session

The built Ins are, for s t r, type 's t r'; for i n t, type i n t; and range is a built-in function range. The My Planet object is derived from local (get Name) and self derivatives. The My Planet object has a class with the name of “X 25”, a radius of 45, a mass of 198, and a distance of 1000. The object is separated into two distinct layers. The inner layer, or object state, contains the instance variable names. The outer layer contains the names of the methods. The in it method is defined by in it underscore def. The get Name method is defined by get Name underscore def.

Back to Figure

Line 1: import turtle. Line 2: my turtle equals turtle dot Turtle left parenthesis right parenthesis. Line 3. my turtle dot forward left parenthesis, 10, right parenthesis.

Back to Figure

Line 1. my Planet equals Planet left parenthesis, single quotes, X 25, single quotes, comma, 45, comma, 198, comma, 1000, right

parenthesis.

Back to Figure

The built Ins are, for s t r, type 's t r'; for i n t, type i n t; and range is a built-in function range. The My Planet object is derived from local (get Name) and self derivatives. The Planet class definition object is defined by methods in it and get Name with definition codes: in it underscore def and get Name underscore def. The My Planet object has a class with the name of “X 25”, a radius of 45, a mass of 198, and a distance of 1000. The object is separated into two distinct layers. The inner layer, or object state, contains the instance variable names. The outer layer contains the names of the methods. The in it method is defined by in it underscore def. The get Name method is defined by get Name underscore def.

Back to Figure

Line 1: class Sun colon. Line 2, indented once: d e f double underscore in it double underscore left parenthesis self, comma i Name comma i Rad comma i M comma i Temp, right parenthesis, colon. Line 3, indented twice: self dot double underscore name equals i Name. Line 4, indented twice: self dot double underscore radius equals i Rad. Line 5, indented twice: self dot double underscore mass equals i M. Line 6, indented twice: self dot double underscore temp equals i Temp. Line 7. Blank. Line 8, indented once: d e f get Mass left parenthesis, self, right parenthesis, colon. Line 9, indented twice: return self dot double underscore mass. Line 10: Blank. Line 11, indented once: d e f double underscore s t r double underscore left parenthesis self,

right parenthesis, colon. Line 12, Indented twice: return self dot double underscore name.

Back to Listing

Line 1: class Solar System colon. Line 2, indented once: d e f double underscore in it double underscore left parenthesis self, comma, a Sun, right parenthesis, colon. Line 3, indented twice: self dot double underscore the Sun equals a Sun. Line 4, indented twice: self dot double underscore planets equals left square bracket right square bracket. Line 5: Blank. Line 6, indented once: d e f add Planet left parenthesis, self, comma, a Planet, right parenthesis, colon. Line 7, indented twice: self dot double underscore planets dot append left parenthesis, a Planet, right parenthesis. Line 8: Blank. Line 9, indented once: d e f show Planets, left parenthesis, self, right parenthesis, colon. Line 10, indented twice: for a Planet in self dot double underscore planets colon. Line 11, indented thrice: print left parenthesis a Planet right parenthesis. Line 12: Blank.

Back to Listing

Input 1: Line 1: from sun import asterisk. Line 2: from planet import asterisk. Line 3: from solar System import asterisk. Input 2: Line 1: sun equals Sun, left parenthesis, single quotes, Sun, single quotes, comma, 5000, comma, 1000, comma, 5800, right parenthesis. Line 2: s s equals Solar System, left parenthesis, sun, right parenthesis. Input 3: Line 1: p equals Planet, left parenthesis, single quotes, Mercury, single quotes, comma, 19, comma, 10, comma, 25, right parenthesis. Line 2: s s dot add Planet, left parenthesis, p, right parenthesis. Input 4: Line 4: p equals Planet left parenthesis single quotes Earth single quotes

comma 50 comma 60 comma 30, right parenthesis. Line 2: s s dot add Planet left parenthesis, p, right parenthesis. Input 5: Line 1: p equals Planet left parenthesis, single quotes, Mars, single quotes, comma, 47, comma, 50, comma, 35, right parenthesis. Line 2. s s dot add Planet left parenthesis, p right parenthesis. Input 6: Line 1: p equals Planet, left parenthesis, single quotes, Jupiter, single quotes, comma, 75, comma, 100, comma, 50, right parenthesis. Line 2. s s dot add Planet, left parenthesis, p, right parenthesis. Input 7: Line 1: s s dot show Planets left parenthesis right parenthesis. Mercury. Earth. Mars. Jupiter.

Back to Session

The s s class is defined using the in it method. The primary state comprises the Sun and four planets. The Sun is defined by the in it method and the state Sun. Planet 0 is defined by the in it method and the state Mercury. Planet 1 is defined by the in it method and the state Earth. Planet 2 is defined by the in it method and the state Mars. Planet 4 is defined by the in it method and the state Jupiter.

Back to Figure

Line 1: class Solar System colon. Line 2, Indented once: d e f double underscore in it double underscore left parenthesis, self, comma, width, comma, height, right parenthesis, colon. Line 3, indented twice: self dot double underscore the Sun equals None. Line 4, indented twice: self dot double underscore planets equals left square bracket right square bracket. Line 5, indented twice: self dot double underscore s s Turtle equals turtle dot Turtle left parenthesis right parenthesis. Line 6, indented twice: self dot double underscore s s Turtle dot hide turtle left parenthesis right

parenthesis. Line 7, indented twice: self dot double underscore s s Screen equals turtle dot Screen left parenthesis right parenthesis. Line 8, indented twice: self dot double underscore s s Screen dot set world coordinates left parenthesis hyphen width forward slash 2.0, comma, hyphen height forward slash 2 dot 0 comma. Line 9: width forward slash 2.0 comma height forward slash 2.0 right parenthesis. Line 10: Blank. Line 11, indented once: d e f add Planet left parenthesis self, comma a Planet right parenthesis colon. Line 12, indented twice: self dot double underscore planets dot append left parenthesis, a Planet, right parenthesis. Line 13. Blank. Line 14, indented once: d e f add Sun left parenthesis self, comma, a Sun right parenthesis colon. Line 15, indented twice: self dot double underscore the Sun equals a Sun. Line 16. Blank. Line 17, indented once: d e f show Planets left parenthesis self, right parenthesis colon. Line 18, indented twice: for a Planet in self dot double underscore planets colon. Line 19, Indented thrice: print left parenthesis a Planet right parenthesis. Line 20. Blank. Line 21, indented once: d e f freeze left parenthesis self, right parenthesis colon. Line 22, indented twice: self dot double underscore s s Screen dot exit on click left parenthesis right parenthesis.

Back to Listing

Line 1: class Sun colon. Line 2, indented once: d e f double underscore in it double underscore left parenthesis self comma i Name comma I Rad comma i M comma i Temp right parenthesis colon. Line 3, indented twice: self dot double underscore name equals I Name. Line 4, indented twice: self dot double underscore radius equals I Rad. Line 5, indented twice: self dot double underscore mass equals i M. Line 6, indented twice: self dot double underscore temp equals i Temp. Line 7. Blank. Line 8,

indented twice: self dot double underscore x equals 0. Line 9, indented twice: self dot double underscore y equals 0. Line 10, indented twice: self dot double underscore s Turtle equals turtle dot Turtle left parenthesis right parenthesis. Line 11, indented twice: self dot double underscore s Turtle dot shape left parenthesis single quotes circle single quotes right parenthesis. Line 12, indented twice: self dot double underscore s Turtle dot color left parenthesis single quotes yellow single quotes right parenthesis. Line 13, Blank. Line 14, indented once: d e f get X Pos left parenthesis self right parenthesis colon. Line 15, indented twice: return self dot double underscore x. Line 16: Blank. Line 17, indented once: d e f get Y Pos left parenthesis self, right parenthesis colon. Line 18, indented twice: return self dot double underscore y. Line 19: Blank. Line 20: hash other methods as before.

Back to Listing

Line 1: class Planet colon. Line 2, indented once: d e f double underscore in it double underscore left parenthesis, self comma i Name comma i Rad comma i M comma i Dist comma i V X comma i V y comma i C right parenthesis colon. Line 3, indented twice: self dot double underscore name equals i Name. Line 4, indented twice: self dot double underscore radius equals i Rad. Line 5, indented twice: self dot double underscore mass equals i M. Line 6, indented twice: self dot double underscore distance equals i Dist. Line 7. Blank. Line 8, indented twice: self dot double underscore x equals self dot double underscore distance. Line 9, indented twice: self dot double underscore y equals 0. Line 10, indented twice: self dot double underscore color equals i C. Line 11, indented twice: self dot double underscore p Turtle equals turtle dot Turtle left parenthesis right parenthesis. Line 12,

indented twice: self dot double underscore p Turtle dot color left parenthesis self dot double underscore color right parenthesis. Line 13, indented twice: self dot double underscore p Turtle dot shape left parenthesis double quotes circle double quotes right parenthesis. Line 14, indented twice: self dot double underscore p Turtle dot up left parenthesis right parenthesis. Line 15, indented twice: self dot double underscore p Turtle dot go to left parenthesis self dot double underscore x comma self dot double underscore y right parenthesis. Line 16, indented twice: self dot double underscore p Turtle dot down left parenthesis right parenthesis. Line 17. Blank. Line 18, indented once: d e f get X Pos left parenthesis self right parenthesis colon. Line 19, indented twice: return self dot double underscore x. Line 20. Blank. Line 21, indented once: d e f get Y Pos left parenthesis self right parenthesis colon. Line 22, indented twice: return self dot double underscore y. Line 23. Blank. Line 24, indented once: hash other methods as before.

Back to Listing

Input 1: Line 1: from solar System import asterisk. Line 2: from planet import asterisk. Line 3: from sun import asterisk. Input 2: Line 1: s s equals Solar System left parenthesis, 2, comma, 2, right parenthesis. Input 5 Line 1: sun equals Sun left parenthesis, single quotes, Sun, single quotes, comma, 5000, comma, 10, comma, 5800, right parenthesis. Line 2. s s dot add Sun left parenthesis, sun, right parenthesis. Input 7: Line 1: m equals Planet left parenthesis, single quotes, Mercury, single quotes, comma, 19.5, comma, 1000, comma, 0.25, comma, single quotes, blue, single quotes, right parenthesis Line 2: s s dot add Planet left parenthesis m right parenthesis. Input 8: Line 1: m equals Planet left parenthesis, single quotes, Earth, single quotes,

comma, 47.5, comma, 5000, comma, 0.3, comma, single quotes, green, single quotes, right parenthesis. Line 2: s s dot add Planet left parenthesis, m, right parenthesis. Input 9: Line 1: m equals Planet left parenthesis, single quotes, Mars, single quotes, comma, 50, comma, 9000, comma, 0.5, comma, single quotes, red, single quotes, right parenthesis. Line 2: s s dot add Planet left parenthesis, m, right parenthesis. Input 10: Line 1: m equals Planet left parenthesis, single quotes, Jupiter, single quotes, comma, 100, comma, 49000, comma, 0.7, comma, single quotes, black, single quotes, right parenthesis. Line 2: s s dot add Planet left parenthesis m, right parenthesis Input 11: Line 1: s s dot freeze left parenthesis right parenthesis.

Back to Session

The Python Turtle Graphics window has the yellow dot on the extreme left, followed by a blue and a green dot in proximity, then a red, and then ending with a black dot.

Back to Figure

An arrow from the origin moves to the top-right, labeled as V. Another arrow moves along the vertical y-axis, and it is labeled as V y. Another arrow moves along the horizontal x-axis, and it is labeled as V x.

Back to Figure

The slant connection between the Sun and the Planet is represented by the hypotenuse 'r'. The hypotenuse for the right-angled triangle comprises extensions from the vertical and horizontal components of the Planet, namely, F x and F y. The

perpendicular and base are labeled as y and x, respectively, and the force arrow from the Planet to the Sun is labeled as F.

Back to Figure

Line 1: class Planet colon. Line 2, indented once: d e f double underscore in it double underscore left parenthesis, self, comma, i Name, comma, i Rad, comma, i M, comma, i Dist, comma, i V x, comma, i V y, comma i C, right parenthesis, colon. Line 3. Blank. Line 4, indented twice: hash other instance variables as before. Line 5. Blank. Line 6, indented twice: self dot double underscore vel X equals i V x. Line 7, indented twice: self dot double underscore vel Y equals i V y.

Back to Listing

Line 1: d e f move To left parenthesis self, comma, new X, comma, new Y, right parenthesis colon. Line 2, indented once: self dot double underscore x equals new X. Line 3, indented once: self dot double underscore y equals new Y. Line 4, indented once: self dot double underscore p Turtle dot go to left parenthesis self dot double underscore x comma self dot double underscore y right parenthesis. Line 5. Blank. Line 6, indented once: d e f get X Vel left parenthesis self right parenthesis colon. Line 7, indented twice: return self dot double underscore vel X. Line 8. Blank. Line 9, indented once: d e f get Y Vel left parenthesis self right parenthesis colon. Line 10, indented twice: return self dot double underscore vel Y. Line 11. Blank. Line 12, indented once: d e f set X Vel left parenthesis self comma new Vel X right parenthesis colon. Line 13, indented twice: self dot double underscore vel X equals new Vel X. Line 14: Blank. Line 15, indented once: d e f set Y Vel left parenthesis self comma

new Vel Y right parenthesis colon. Line 16, indented twice: self dot double underscore vel Y equals new Vel Y.

Back to Listing

Line 1: d e f move Planets left parenthesis self right parenthesis colon. Line 2, indented once: G equals .1. Line 3, indented once: d t equals .001. Line 4. Blank. Line 5, indented once: for p in self dot double underscore planets colon. Line 6, indented twice: p dot move to left parenthesis p dot get X Pos left parenthesis right parenthesis plus d t asterisk p dot get X Vel left parenthesis right parenthesis comma. Line 7, indented thrice: p dot get Y Pos left parenthesis right parenthesis plus d t asterisk p dot get Y Vel left parenthesis right parenthesis right parenthesis. Line 8. Blank. Line 9, indented twice: r X equals self dot double underscore the Sun dot get X Pos left parenthesis right parenthesis hyphen p dot get X Pos left parenthesis right parenthesis. Line 10, indented twice: r Y equals self dot double underscore the Sun dot get Y Pos left parenthesis right parenthesis hyphen p dot get Y Pos left parenthesis right parenthesis. Line 11, indented twice: r equals math dot sqrt left parenthesis, r X asterisk asterisk 2 plus r Y asterisk asterisk 2, right parenthesis. Line 12. Blank. Line 13, indented twice: acc X equals G asterisk self dot double underscore the Sun dot get Mass left parenthesis right parenthesis asterisk r X forward slash r asterisk asterisk 3. Line 14, indented twice: acc Y equals G asterisk self dot double underscore the Sun dot get Mass left parenthesis right parenthesis asterisk r Y forward slash r asterisk asterisk 3. Line 15. Blank. Line 16, indented twice: p dot set X Vel left parenthesis p dot get X Vel left parenthesis right parenthesis plus dt asterisk acc X right parenthesis. Line 17, indented twice: p dot set Y Vel left

parenthesis p dot get Y Vel left parenthesis right parenthesis plus d t asterisk acc Y right parenthesis.

Back to Listing

Line 1. d e f create S S and Animate left parenthesis right parenthesis colon. Line 2, indented once: s s equals Solar System left parenthesis, 2, comma, 2, right parenthesis. Line 3. Blank. Line 4, indented once: sun equals Sun left parenthesis, single quotes, Sun, single quotes, comma, 5000, comma, 10, comma, 5800, right parenthesis. Line 5, indented once: s s dot add Sun left parenthesis sun, right parenthesis. Line 6. Blank. Line 7, indented once: m equals Planet left parenthesis, single quotes, Mercury, single quotes, comma, 19.5, comma, 1000, comma, 0.25, comma, 0, comma, 2, comma, single quotes, blue, single quotes, right parenthesis. Line 8, indented once: s s dot add Planet left parenthesis m, right parenthesis. Line 9. Blank. Line 10, indented once: m equals Planet left parenthesis, single quotes, Earth, single quotes, comma, 47.5, comma, 5000, comma, 0.3, comma, 0 comma, 2.0, comma, single quotes, green, single quotes, right parenthesis. Line 11, indented once: s s dot add Planet left parenthesis m, right parenthesis. Line 12. Blank. Line 13, indented once: m equals Planet left parenthesis, single quotes, Mars, single quotes, comma, 50, comma, 9000, comma, 0.5, comma, 0, comma, 1.63, comma, single quotes, red, single quotes, right parenthesis. Line 14, indented once: s s dot add Planet left parenthesis m, right parenthesis. Line 15. Blank. Line 16, indented once: m equals Planet left parenthesis, single quotes, Jupiter, single quotes, comma, 100, comma, 49000, comma, 0.7, comma, 0, comma, 1, comma, single quotes, black, single quotes, right parenthesis. Line 17, indented once: s s dot add Planet left parenthesis m, right parenthesis. Line 18. Blank. Line 19,

indented once: Num Time Periods equals 2000. Line 20, indented once: for a Move in range left parenthesis num Time Periods, right parenthesis colon. Line 21, indented twice: s s dot move Planets left parenthesis right parenthesis. Line 22. Blank. Line 23, indented once: s s dot freeze left parenthesis right parenthesis.

Back to Listing

Line 1: from animate S S import asterisk Line 2. Create S S and Animate left parenthesis right parenthesis.

Back to Session

The Python Turtle Graphics window has a yellow dot in the center. Red, Green, Blue, and Black dots revolve around the Sun and the rotational paths are represented as ellipses.

Back to Figure

The diagram has four tables. Each table consists of the name of the class, a list of instance variables, and a list of methods On the top is the table named Class Name with instance variables methods. On the bottom left is a table with the class name world. The instance variables are, double underscore max X, double underscore max Y, double underscore thing List, double underscore grid, and double underscore w Turtle. The methods are draw, freeze World, get Max X, get Max Y, add Thing, del Thing, move Thing, live A Little, empty Location, and look At Location. On the center is a table with the class name fish. The instance variables are, double underscore x Pos, double underscore y Pos, double underscore world, double underscore breed Tick, and double underscore turtle. The methods are get X, get Y, set X, set Y, set World, appear, hide, move, live A Little, try

To Breed, and try To Move. On the bottom right is a table with the class name bear. The instance variables are double underscore x Pos, double underscore y Pos, double underscore world, double underscore breed Tick, double underscore starve Tick, and double underscore turtle. The methods are get X, get Y, set X, set Y, set World, appear, hide, move, live A Little, try To Breed, try To Move, and try To Eat.

Back to Figure

The diagram resembles a grid of 12-by-6 with three life forms. Each position of the grid is addressed by a unique (x, y) pair. The horizontal direction reads 12 positions in the X direction from 0 to 11. The vertical direction reads 6 positions in the Y direction from 0 to 5. Bears are currently living at positions (2, 4) and (11, 5), and a fish is currently at position (5, 0).

Back to Figure

The list reads the following. g, equals, left square bracket, left square bracket, None, comma, None, comma, None, comma, None, comma, None, comma, FISH, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, right square bracket, comma, left square bracket, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, right square bracket, comma, left square bracket, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, right square bracket, comma, left square bracket, None, comma, None, comma, None, comma,

None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, right square bracket, comma, left square bracket, None, comma, None, comma, BEAR, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, right square bracket, comma, left square bracket, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, None, comma, BEAR, right square bracket, right square bracket.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: import turtle Line 2: class World, colon. Line 3, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, m X, comma, m Y, right parenthesis, colon. Line 4, indented twice: self, dot, double underscore, max X, equals, m X, hash, maximum x value Line 5, indented twice: self, dot, double underscore, max Y, equals, m Y, hash, maximum y value Line 6, indented twice: self, dot, double underscore, thing List, equals, left square bracket, right square bracket, hash, list of living things Line 7, indented twice: self, dot, double underscore, grid, equals, left square bracket, right square bracket, hash, grid to hold living things Line 8: Blank. Line 9, indented twice: for a Row in range, left parenthesis, self, dot, double underscore, max Y, right parenthesis, colon, hash, set all locations to empty Line 10, indented thrice: row, equals, left square bracket, right square bracket. Line 11, indented thrice: for a Col in range, left parenthesis, self, dot, double underscore, max X, right parenthesis, colon. Line 12, indented quarce: row dot

append, left parenthesis, None, right parenthesis. Line 13, indented thrice: self, dot, double underscore, grid dot append, left parenthesis, row, right parenthesis. Line 14: Blank. Line 15, indented twice: self, dot, double underscore, w Turtle minus turtle dot Turtle, left parenthesis, right parenthesis. Line 16, indented twice: self, dot, double underscore, w Screen, equals, turtle dot Screen, left parenthesis, right parenthesis. Line 17, indented twice: self, dot, double underscore, w Screen dot set world coordinates, left parenthesis, 0, comma, 0, comma, self, dot, double underscore, max X minus 1, comma, self, dot, double underscore, max Y minus 1, right parenthesis. Line 19, indented twice: self, dot, double underscore, w Screen dot add shap, left parenthesis, left double quotes, Bear dot gif, right double quotes, right parenthesis. Line 20, indented twice: self, dot, double underscore, w Screen dot add a hape, left parenthesis, left double quotes, Fish dot gif, right double quotes, right parenthesis. Line 21, indented twice: self, dot, double underscore, w Turtle dot hide turtle, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 23, indented once: def draw, left parenthesis, self, right parenthesis, colon. Line 24, indented twice: self, dot, double underscore, w Screen dot tracer, left parenthesis, 0, right parenthesis. Line 25, indented twice: self, dot, double underscore, w Turtle dot forward, left parenthesis, self, dot, double underscore, max X minus 1, right parenthesis. Line 26, indented twice: self, dot, double underscore, w Turtle dot left, left parenthesis, 90, right parenthesis. Line 27, indented twice: self, dot, double underscore, w Turtle dot forward, left parenthesis,

self, dot, double underscore, max Y minus 1, right parenthesis. Line 28, indented twice: self, dot, double underscore, w Turtle dot left, left parenthesis, 90, right parenthesis. Line 29, indented twice: self, dot, double underscore, w Turtle dot forward, left parenthesis, self, dot, double underscore, max X minus 1, right parenthesis. Line 30, indented twice`: self, dot, double underscore, w Turtle dot left, left parenthesis, 90, right parenthesis. Line 31, indented twice: self, dot, double underscore, w Turtle dot forward, left parenthesis, self, dot, double underscore, max Y minus 1, right parenthesis. Line 32, indented twice: self, dot, double underscore, w Turtle dot left, left parenthesis, 90, right parenthesis. Line 33: Blank. Line 34, indented twice: for i in range, left parenthesis, self, dot, double underscore, max Y minus 1, right parenthesis, colon. Line 35, indented thrice: self, dot, double underscore, w Turtle dot forward, left parenthesis, self, dot, double underscore, max X minus 1, right parenthesis. Line 36, indented thrice: self, dot, double underscore, w Turtle dot backward, left parenthesis, self, dot, double underscore, max X minus 1, right parenthesis. Line 37, indented thrice: self, dot, double underscore, w Turtle dot left, left parenthesis, 90, right parenthesis. Line 38, indented thrice: self, dot, double underscore, w Turtle dot forward, left parenthesis, 1, right parenthesis. Line 39, indented thrice: self, dot, double underscore, w Turtle dot right, left parenthesis, 90, right parenthesis. Line 40, indented twice: self, dot, double underscore, w Turtle dot forward, left parenthesis, 1, right parenthesis. Line 41, indented twice: self, dot, double underscore, w Turtle dot right, left parenthesis, 90, right parenthesis. Line 42, indented twice: for i in range, left parenthesis, self, dot, double underscore, max X minus 2, right parenthesis, colon. Line 43, indented thrice: self, dot, double underscore, w Turtle dot forward, left parenthesis, self, dot, double underscore, max Y

minus 1, right parenthesis. Line 44, indented thrice: self, dot, double underscore, w Turtle dot backward, left parenthesis, self, dot, double underscore, max Y minus1, right parenthesis. Line 45, indented thrice: self, dot, double underscore, w Turtle dot left, left parenthesis, 90, right parenthesis. Line 46, indented thrice: self, dot, double underscore, w Turtle dot forward, left parenthesis, 1, right parenthesis. Line 47, indented thrice: self, dot, double underscore, w Turtle dot right, left parenthesis, 90, right parenthesis. Line 48, indented twice: self, dot, double underscore, w Screen dot tracer, left parenthesis, 1, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: Line 50, indented once: def add Thing, left parenthesis, self, comma, a Thing, comma, x, comma, y, right parenthesis, colon. Line 51, indented twice: A Thing dot set X, left parenthesis, x, right parenthesis. Line 52, indented twice: A Thing dot set Y, left parenthesis, y, right parenthesis. Line 53, indented twice: self, dot, double underscore, grid, left square bracket, y, right square bracket, left square bracket, x, right square bracket, equals, a Thing, hash, add life-form to grid. Line 54, indented twice: A Thing dot set World, left parenthesis, self, right parenthesis. Line 55, indented twice: self, dot, double underscore, thing List dot append, left parenthesis, a Thing, right parenthesis, hash, add to list of life forms. Line 56, indented twice: A Thing dot appear, left parenthesis, right parenthesis. Line 57: Blank. Line 58, indented once: def del Thing, left parenthesis, self, comma, a Thing, right parenthesis, colon. Line 59, indented twice: A Thing dot hide, left parenthesis, right parenthesis. Line 60, indented twice: self, dot, double underscore,

grid, left square bracket, a Thing dot get Y, left parenthesis, right parenthesis, right square bracket, left square bracket, a Thing dot get X, left parenthesis, right parenthesis, right square bracket, equals, None. Line 61, indented twice: self, dot, double underscore, thing List dot remove, left parenthesis, a Thing, right parenthesis. Line 62: Blank. Line 63, indented once: def move Thing, left parenthesis, self, comma, old X, comma, old Y, comma, new X, comma, new Y, right parenthesis, colon. Line 64, indented twice: self, dot, double underscore, grid, left square bracket, new Y, right square bracket, left square bracket, new X, right square bracket, equals, self, dot, double underscore, grid, left square bracket, old Y, right square bracket, left square bracket, old X, right square bracket. Line 65, indented twice: self, dot, double underscore, grid, left square bracket, old Y, right square bracket, left square bracket, old X, right square bracket, equals, None. Line 66: Blank. Line 67, indented once: def get Max X, left parenthesis, self, right parenthesis, colon. Line 68, indented twice: return self, dot, double underscore, max X. Line 69: Blank. Line 70, indented once: def get Max Y, left parenthesis, self, right parenthesis, colon. Line 71, indented twice: return self, dot, double underscore, max Y. Line 72: Blank. Line 73, indented once: def live A Little, left parenthesis, self, right parenthesis, colon. Line 74, indented twice: if self, dot, double underscore, thing List, not equal to, left square bracket, right square bracket, colon, hash, if any life forms. Line 75, indented thrice: A Thing, equals, random dot rand range, left parenthesis, len, left parenthesis, self, dot, double underscore, thing List, right parenthesis, right parenthesis. Line 76, indented thrice: Random Thing, equals, self, double underscore, thing List, left square bracket, a Thing, right square bracket. Line 77, indented thrice: Random Thing dot live A Little, left parenthesis, right parenthesis. Line 78: Blank. Line 79, indented once: def empty

Location, left parenthesis, self, comma, x, comma, y, right parenthesis, colon. Line 80, indented twice: if self, dot, double underscore, grid, left square bracket, y, right square bracket, left square bracket, x, right square bracket, equals, equals, None, colon. Line 81, indented thrice: return True. Line 82, indented twice: else, colon. Line 83, indented thrice: return False. Line 84: Blank. Line 85, indented once: def look At Location, left parenthesis, self, comma, x, comma, y, right parenthesis, colon. Line 86, indented twice: return self, dot, double underscore, grid, left square bracket, y, right square bracket, left square bracket, x, right square bracket. Line 87: Blank. Line 88, indented once: def freeze World, left parenthesis, self, right parenthesis, colon. Line 89, indented twice: self, dot, double underscore, w Screen dot exit on click, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: class Fish, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, turtle, equals, turtle dot, Turtle, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, double underscore, turtle dot, up, left parenthesis, right parenthesis. Line 5, indented twice: self, dot, double underscore, turtle, dot, hide turtle, left parenthesis, right parenthesis. Line 6, indented twice: self, dot, double underscore, turtle, dot, shape, left parenthesis, left double quotes, Fish, dot, gif, right double quotes, right parenthesis. Line 7: Blank. Line 8, indented twice: self, dot, double underscore, x Pos, equals, 0 Line 9, indented twice: self, dot, double underscore, y Pos, equals, 0 Line 10, indented twice: self, dot, double underscore, world, equals, None Line 11: Blank. Line 12,

indented twice: self, dot, double underscore, breed Tick, equals, 0.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 14, indented once: def set X, left parenthesis, self, new X, right parenthesis, colon. Line 15, indented twice: self, dot, double underscore, x Pos, equals, new X. Line 16: Blank. Line 17, indented once: def set Y, left parenthesis, self, comma, new Y, right parenthesis, colon. Line 18, indented twice: self, dot, double underscore, y Pos, equals, new Y. Line 19: Blank. Line 20, indented once: def get X, left parenthesis, self, right parenthesis, colon. Line 21, indented twice: return self, dot, double underscore, x Pos. Line 22: Blank. Line 23, indented once: def get Y, left parenthesis, self, right parenthesis, colon. Line 24, indented twice: return self, dot, double underscore, y Pos. Line 25: Blank. Line 26, indented once: def set World, left parenthesis, self, a World, right parenthesis, colon. Line 27, indented twice: self, dot, double underscore, world, equals, a World. Line 28: Blank. Line 29, indented once: def appear, left parenthesis, self, right parenthesis, colon. Line 30, indented twice: self, dot, double underscore, turtle, dot, go to, left parenthesis, self, dot, double underscore, x Pos, comma, self, dot, double underscore, y Pos, right parenthesis. Line 31, indented twice: self, dot, double underscore, turtle dot, show turtle, left parenthesis, right parenthesis. Line 32: Blank. Line 33, indented once: def hide, left parenthesis, self, right parenthesis, colon. Line 34, indented twice: self, dot, double underscore, turtle dot, hide turtle, left parenthesis, right parenthesis. Line 35: Blank. Line 36, indented once: def move, left parenthesis, self, comma, new X, comma, new Y, right parenthesis, colon. Line 37,

indented twice: self, dot, double underscore, world dot, move Thing, left parenthesis, self, dot, double underscore, x Pos, comma, self, dot, double underscore, y Pos, new X, new Y, right parenthesis. Line 38, indented twice: self, dot, double underscore, x Pos, equals, new X. Line 39, indented twice: self, dot, double underscore, y Pos, equals, new Y. Line 40, indented twice: self, dot, double underscore, turtle dot, go to, left parenthesis, self, dot, double underscore, x Pos, comma, self, dot, double underscore, y Pos, right parenthesis.

Back to Listing

The size of the grid is 3-by-3. It reads the following. Row 1: left parenthesis, x, minus, 1, comma, y, plus, 1, right parenthesis; left parenthesis, x, plus, 0, comma, y, plus, 1, right parenthesis; left parenthesis, x, plus, 1, comma, y, plus, 1, right parenthesis. Row 2: left parenthesis, x, minus, 1, comma, y, plus, 0, right parenthesis; left parenthesis, x, comma, y, right parenthesis; left parenthesis, x, plus, 1, comma, y, plus, 0, right parenthesis. Row 3: left parenthesis, x, minus, 1, comma, y, minus, 1, right parenthesis; left parenthesis, x, plus, 0, comma, y, minus, 1, right parenthesis; left parenthesis, x, plus, 1, comma, y, minus, 1, right parenthesis.

Back to Figure

The size of the grid is 3-by-3. It reads the following. Row 1: left parenthesis, negative 1, comma, positive 1, right parenthesis; left parenthesis, 0, comma, positive 1, right parenthesis; left parenthesis, positive 1, comma, positive 1, right parenthesis. Row 2: left parenthesis, negative 1, comma, 0, right parenthesis; left parenthesis, x, comma, y, right parenthesis; left parenthesis,

positive 1, comma, 0, right parenthesis. Row 3: left parenthesis, negative 1, comma, negative 1, right parenthesis; left parenthesis, 0, comma, negative 1, right parenthesis; left parenthesis, positive 1, comma, negative 1, right parenthesis.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 43, indented once: def live A Little, left parenthesis, self, right parenthesis, colon. Line 44, indented twice: Offset List, equals, left square bracket, left parenthesis, negative 1, comma, 1, right parenthesis, comma, left parenthesis, 0, comma, 1, right parenthesis, comma, left parenthesis, 1, comma, 1, right parenthesis, comma, left parenthesis, negative 1, comma, 0, right parenthesis, comma, left parenthesis, 1, comma, 0, right parenthesis, comma, left parenthesis, negative 1, comma, negative 1, right parenthesis, comma, left parenthesis, 0, comma, negative 1, right parenthesis, comma, left parenthesis, 1, comma, negative 1, right parenthesis, right square bracket. Line 47, indented twice: a d j Fish, equals, 0, hash, count adjacent Fish. Line 48, indented twice: for offset in offset List, colon. Line 49, indented thrice: New X, equals, self, dot, double underscore, x Pos, plus, offset, left square bracket, 0, right square bracket. Line 50, indented thrice: new Y, equals, self, dot, double underscore, y Pos, plus, offset, left square bracket, 1, right square bracket. Line 51, indented thrice: if 0, less than, equals, new X, less than, self, dot, double underscore, world, dot, get Max X, left parenthesis, right parenthesis, and, backslash, 0, less than, equals, new Y, less than, self, dot, double underscore, world, dot, get Max Y, left parenthesis, right parenthesis, colon. Line 53, indented quarce: if, left parenthesis, not self, dot, double underscore, world, dot, empty Location, left parenthesis, new X, comma, new Y, right

parenthesis, right parenthesis, and, backslash, is instance, left parenthesis, self, dot, double underscore, world, dot, look At Location, left parenthesis, new X, comma, new Y, right parenthesis, comma, Fish, right parenthesis, colon. Line 55, indented quince: a d j Fish, equals, a d j Fish, plus, 1. Line 56: Blank. Line 57, indented twice: if a d j Fish, greater than, equals, 2, colon, hash, if 2 or more adjacent Fish, die. Line 58, indented thrice: self, dot, double underscore, world, dot, del Thing, left parenthesis, self, right parenthesis. Line 59, indented twice: else, colon. Line 60, indented thrice: self, dot, double underscore, breed Tick, equals, self, dot, double underscore, breed Tick, plus, 1. Line 61, indented thrice: if self, dot, double underscore, breed Tick, greater than, equals, 12, colon, hash, if alive 12 or more ticks, breed. Line 62, indented quarce: self, dot, try To Breed, left parenthesis, right parenthesis. Line 63: Blank. Line 64, indented thrice: self, dot, try To Move, left parenthesis, right parenthesis, hash, try to move.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 66, indented once: def try To Breed, left parenthesis, self, right parenthesis, colon. Line 67, indented twice: Offset List, equals, left square bracket, left parenthesis, negative 1, comma, 1, right parenthesis, comma, left parenthesis, 0, comma, 1, right parenthesis, comma, left parenthesis, 1, comma, 1, right parenthesis, comma, left parenthesis, negative 1, comma, 0, right parenthesis, comma, left parenthesis, 1, comma, 0, right parenthesis, comma, left parenthesis, negative 1, comma, negative 1, right parenthesis, comma, left parenthesis, 0, comma, negative 1, right parenthesis, comma, left parenthesis, 1, comma, negative 1, right parenthesis, right square bracket. Line 70,

indented twice: Random Offset Index, equals, random, dot, rand range, left parenthesis, len, left parenthesis, offset List, right parenthesis, right parenthesis. Line 71, indented twice: Random Offset, equals, offset List, left square bracket, random Offset Index, right square bracket. Line 72, indented twice: next X, equals, self, dot, double underscore, x Pos, plus, random Offset, left square bracket, 0, right square bracket. Line 73, indented twice: next Y, equals, self, dot, double underscore, y Pos, plus, random Offset, left square bracket, 1, right square bracket. Line 74, indented twice: while not, left parenthesis, 0, less than, equals, next X, less than, self, dot, double underscore, world, dot, get Max X, left parenthesis, right parenthesis, and, backslash, 0, less than, equals, next Y, less than, self, dot, double underscore, world, dot, get Max Y, left parenthesis, right parenthesis, right parenthesis, colon. Line 76, indented thrice: Random Offset Index, equals, random, dot, rand range, left parenthesis, len, left parenthesis, offset List, right parenthesis, right parenthesis. Line 77, indented thrice: Random Offset, equals, offset List, left square bracket, random Offset Index, right square bracket. Line 78, indented thrice: Next X, equals, self, dot, double underscore, x Pos, plus, random Offset, left square bracket, 0, right square bracket. Line 79, indented thrice: Next Y, equals, self, dot, double underscore, y Pos, plus, random Offset, left square bracket, 1, right square bracket. Line 80: Blank. Line 81, indented twice: if self, dot, double underscore, world, dot, empty Location, left parenthesis, next X, comma, next Y, right parenthesis, colon. Line 82, indented thrice: Child Thing, equals, Fish, left parenthesis, right parenthesis. Line 83, indented thrice: self, dot, double underscore, world, dot, add Thing, left parenthesis, child Thing, comma, next X, comma, next Y, right parenthesis. Line 84, indented thrice: self, dot, double underscore, breed Tick, equals, 0, hash, reset breed Tick.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 86, indented once: def try To Move, left parenthesis, self, right parenthesis, colon. Line 87, indented twice: offset List, equals, left square bracket, left parenthesis, negative 1, comma, 1, right parenthesis, comma, left parenthesis, 0, comma, 1, right parenthesis, comma, left parenthesis, 1, comma, 1, right parenthesis, comma, left parenthesis, negative 1, comma, 0, right parenthesis, comma, blank, left parenthesis, 1, comma, 0, right parenthesis, comma, left parenthesis, negative 1, comma, negative 1, right parenthesis, comma, left parenthesis, 0, comma, negative 1, right parenthesis, comma, left parenthesis, 1, comma, negative 1, right parenthesis, right square bracket. Line 90, indented twice: random Offset Index, equals, random, dot, rand range, left parenthesis, len, left parenthesis, offset List, right parenthesis, right parenthesis. Line 91, indented twice: random Offset, equals, offset List, left square bracket, random Offset Index, right square bracket. Line 92, indented twice: next X, equals, self, dot, double underscore, x Pos, plus, random Offset, left square bracket, 0, right square bracket. Line 93, indented twice: next Y, equals, self, dot, double underscore, y Pos, plus, random Offset, left square bracket, 1, right square bracket. Line 94, indented twice: while not, left parenthesis, 0, less than, equals, next X, less than, self, dot, double underscore, world, dot, get Max X, left parenthesis, right parenthesis, and, backslash, 0, less than, equals, next Y, less than, self, dot, double underscore, world, dot, get Max Y, left parenthesis, right parenthesis, right parenthesis, colon. Line 96, indented thrice: random Offset Index, equals, random, dot, rand range, left parenthesis, len, left parenthesis, offset List, right parenthesis, right parenthesis. Line 97, indented thrice: random Offset, equals,

offset List, left square bracket, random Offset Index, right square bracket. Line 98, indented thrice: next X, equals, self, dot, double underscore, x Pos, plus, random Offset, left square bracket, 0, right square bracket. Line 99, indented thrice: next Y, equals, self, dot, double underscore, y Pos, plus, random Offset, left square bracket, 1, right square bracket. Line 100: Blank. Line 101, indented twice: if self, dot, double underscore, world, dot, empty Location, left parenthesis, next X, comma, next Y, right parenthesis, colon. Line 102, indented thrice: self, dot, move, left parenthesis, next X, comma, next Y, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: class Bear, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 3, indented thrice: self, dot, double underscore, turtle, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 4, indented thrice: self, dot, double underscore, turtle, dot, up, left parenthesis, right parenthesis. Line 5, indented thrice: self, dot, double underscore, turtle, dot, hide turtle, left parenthesis, right parenthesis. Line 6, indented thrice: self, dot, double underscore, turtle, dot, shape, left parenthesis, left double quotes, Bear dot, gif, right double quotes, right parenthesis. Line 7: Blank. Line 8, indented thrice: self, dot, double underscore, x Pos, equals, 0. Line 9, indented thrice: self, dot, double underscore, y Pos, equals, 0. Line 10, indented thrice: self, dot, double underscore, world, equals, None. Line 11: Blank. Line 12, indented thrice: self, dot, double underscore, starve Tick, equals, 0. Line 13, indented thrice: self, dot, double underscore, breed Tick, equals, 0. Line 14: Blank. Line 15, indented once: hash, same method as Fish. Line 16, indented once: hash, set X, set Y,

get X, get Y, set World, appear, hide, move. Line 17, indented once: hash, try To Breed, left parenthesis, create a Bear, right parenthesis, try To Move.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 81, indented once: def live A Little, left parenthesis, self, right parenthesis, colon. Line 82, indented twice: self, dot, double underscore, breed Tick, equals, self, dot, double underscore, breed Tick, plus, 1. Line 83, indented twice: if self, dot, double underscore, breed Tick, greater than, equals, 8, colon, hash, if alive 8 or more ticks, breed. Line 84, indented thrice: self, dot, try To Breed, left parenthesis, right parenthesis. Line 85: Blank. Line 86, indented twice: self, dot, try To Eat, left parenthesis, right parenthesis. Line 87: Blank. Line 88, indented twice: if self, dot, double underscore, starve Tick, equals, equals, 10, colon, hash, if not eaten for 10 ticks, die. Line 89, indented thrice: self, dot, double underscore, world, dot, del Thing, left parenthesis, self, right parenthesis. Line 90, indented twice: else, colon. Line 91, indented thrice: self, dot, try To Move, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 93, indented once: def try To Eat, left parenthesis, self, right parenthesis, colon. Line 94, indented twice: Offset List, equals, left square bracket, left parenthesis, negative 1, comma, 1, right parenthesis, comma, left parenthesis, 0, comma, 1, right parenthesis, comma, left parenthesis, 1, comma, 1, right parenthesis, comma, left parenthesis, negative 1, comma, 0, right

parenthesis, comma, blank, left parenthesis, 1, comma, 0, right parenthesis, comma, left parenthesis, negative 1, comma, negative 1, right parenthesis, comma, left parenthesis, 0, comma, negative 1, right parenthesis, comma, left parenthesis, 1, comma, negative 1, right parenthesis, right square bracket. Line 97, indented twice: adj Prey, equals, left square bracket, right square bracket, hash, create list of adjacent prey. Line 98, indented twice: for offset in offset List, colon. Line 99, indented thrice: new X, equals, self, dot, double underscore, x Pos, plus, offset, left square bracket, 0, right square bracket. Line 100, indented thrice: New Y, equals, self, dot, double underscore, y Pos, plus, offset, left square bracket, 1, right square bracket. Line 101, indented thrice: if 0, less than, equals, new X, less than, self, dot, double underscore, world, dot, get Max X, left parenthesis, right parenthesis, and, backslash, 0, less than, equals, new Y, less than, self, dot, double underscore, world, dot, get Max Y, left parenthesis, right parenthesis, colon. Line 103, indented quarce: if, left parenthesis, not self, dot, double underscore, world, dot, empty Location, left parenthesis, new X, comma, new Y, right parenthesis, right parenthesis, and, backslash, is instance, left parenthesis, self, dot, double underscore, world, dot, look At Location, left parenthesis, new X, comma, new Y, right parenthesis, comma, Fish, right parenthesis, colon. Line 105, indented quince: adj Prey, dot, append, left parenthesis, self, dot, double underscore, world, dot, look At Location, left parenthesis, new X, comma, new Y, right parenthesis, right parenthesis. Line 106: Blank. Line 107, indented twice: if len, left parenthesis, adj Prey, right parenthesis, greater than, 0, colon, hash, if any Fish are adjacent, pick random Fish to eat. Line 108, indented thrice: Random Prey, equals, adj Prey, left square bracket, random, dot, rand range, left parenthesis, len, left parenthesis, adj Prey, right parenthesis, right parenthesis, right square bracket. Line 109,

indented thrice: prey X, equals, random Prey, dot, get X, left parenthesis, right parenthesis. Line 110, indented thrice: prey Y, equals, random Prey, dot, get Y, left parenthesis, right parenthesis. Line 111: Blank. Line 112, indented thrice: self, dot, double underscore, world, dot, del Thing, left parenthesis, random Prey, right parenthesis, hash, delete the Fish. Line 113, indented thrice: self, dot, move, left parenthesis, prey X, comma, prey Y, right parenthesis, hash, move to the Fish's location. Line 114, indented thrice: self, dot, double underscore, starve Tick, equals, 0. Line 115, indented twice: else, colon. Line 116, indented thrice: self, dot, double underscore, starve Tick, equals, self, dot, double underscore, starve Tick, plus, 1.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def main Simulation, left parenthesis, right parenthesis, colon. Line 2, indented once: Number Of Bears, equals, 10. Line 3, indented once: Number Of Fish, equals, 10. Line 4, indented once: World Life Time, equals, 2500. Line 5, indented once: World Width, equals, 50. Line 6, indented once: World Height, equals, 25. Line 7: Blank. Line 8, indented once: My World, equals, World, left parenthesis, world Width, comma, world Height, right parenthesis, hash, create the world. Line 9, indented once: my World, dot, draw, left parenthesis, right parenthesis. Line 10: Blank. Line 11, indented once: for i in range, left parenthesis, number Of Fish, right parenthesis, colon, hash, create fish. Line 12, indented twice: New Fish, equals, Fish, left parenthesis, right parenthesis. Line 13, indented twice: x, equals, random, dot, rand range, left parenthesis, my World, dot, get Max X, left parenthesis, right parenthesis, right parenthesis. Line 14, indented twice: y, equals, random, dot, rand range, left

parenthesis, my World, dot, get Max Y, left parenthesis, right parenthesis, right parenthesis. Line 15, indented twice: while not my World, dot, empty Location, left parenthesis, x, y, right parenthesis, colon. Line 16, indented thrice: x, equals, random, dot, rand range, left parenthesis, my World, dot, get Max X, left parenthesis, right parenthesis, right parenthesis. Line 17, indented thrice: y, equals, random, dot, rand range, left parenthesis, my World, dot, get Max Y, left parenthesis, right parenthesis, right parenthesis. Line 18, indented twice: my World, dot, add Thing, left parenthesis, new Fish, x, comma, y, right parenthesis. Line 19: Blank. Line 20, indented once: for i in range, left parenthesis, number Of Bears, right parenthesis, colon, hash, create bears. Line 21, indented twice: New Bear, equals, Bear, left parenthesis, right parenthesis. Line 22, indented twice: x, equals, random, dot, rand range, left parenthesis, my World, dot, get Max X, left parenthesis, right parenthesis, right parenthesis. Line 23, indented twice: y, equals, random, dot, rand, dot, range, left parenthesis, my World, dot, get Max Y, left parenthesis, right parenthesis, right parenthesis. Line 24, indented twice: while not my World, dot, empty Location, left parenthesis, x, comma, y, right parenthesis, colon. Line 25, indented thrice: x, equals, random, dot, rand range, left parenthesis, my World, dot, get Max X, left parenthesis, right parenthesis, right parenthesis. Line 26, indented thrice: y, equals, random, dot, rand range, left parenthesis, my World, dot, get Max Y, left parenthesis, right parenthesis, right parenthesis. Line 27, indented twice: my World, dot, add Thing, left parenthesis, new Bear, x, comma, y, right parenthesis. Line 28: Blank. Line 29, indented once: for i in range, left parenthesis, world Life Time, right parenthesis, colon, hash, run the simulation. Line 30, indented twice: My World, dot, live A Little, left parenthesis, right parenthesis. Line 31: Blank. Line 32,

indented once: my World, dot, freeze World, left parenthesis, right parenthesis.

Back to Listing

The window is titled Python Turtle Graphics. The screen resembles an n-cross-n grid. The bears and fishes move diagonally from the bottom left to the center, and from the top left to the center.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 56, indented once: def live A Little, left parenthesis, self, right parenthesis, colon. Line 57, indented twice: self, dot, double underscore, breed Tick, equals, self, dot, double underscore, breed Tick, plus, 1. Line 58, indented twice: if self, dot, double underscore, breed Tick, greater than, equals, 5, colon, hash, if alive 5 or more ticks, breed. Line 59, indented thrice: self, dot, try To Breed, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 21, indented twice: self, dot, double underscore, w Screen, dot, add shap, left parenthesis, left double quotes, Bear, dot, gif, right double quotes, right parenthesis. Line 22, indented twice: self, dot, double underscore, w Screen, dot, add a hape, left parenthesis, left double quotes, Fish, dot, gif, right double quotes, right parenthesis. Line 23, indented twice: self, dot, double underscore, w Screen, dot, add a hape, left parenthesis, left double quotes, Fish, dot, gif, right double quotes, right parenthesis.

Back to Listing

The window is titled Python Turtle Graphics. The screen resembles an n-cross-n grid. The bears and fishes move randomly from the left to the right. A few plants are randomly placed on the grids.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def draw House, left parenthesis, right parenthesis, colon. Line 2, indented once: my Canvas, equals, Canvas, left parenthesis, 800, comma, 600, right parenthesis. Line 3, indented once: house, equals, Rectangle, left parenthesis, left parenthesis, Point, left parenthesis, negative 100, comma, 100, right parenthesis, comma, Point, left parenthesis, 100, comma, 100, right parenthesis, right parenthesis, right parenthesis. Line 4, indented once: house, dot, set Fill, left parenthesis, single quote, blue, single quote, right parenthesis. Line 5, indented once: door, equals, Rectangle, left parenthesis, Point, left parenthesis, negative 50, comma, negative 100, right parenthesis, comma, Point, left parenthesis, 0, comma, 200, right parenthesis, right parenthesis. Line 6, indented once: Door, dot, set Fill, left parenthesis, single quote, brown, single quote, right parenthesis. Line 7, indented once: roof 1, equals, Line, left parenthesis, Point, left parenthesis, negative 100, comma, 100, right parenthesis, comma, Point, left parenthesis, 0, comma, 200, right parenthesis, right parenthesis. Line 8, indented once: roof 2, equals, Line, left parenthesis, Point, left parenthesis, 0, comma, 200, right parenthesis, comma, Point, left parenthesis, 100, comma, 100, right parenthesis, right parenthesis. Line 9, indented once: roof 1, dot, set Width, left parenthesis, 3, right parenthesis.

Line 10, indented once: roof 2, dot, set Width, left parenthesis, 3, right parenthesis. Line 11, indented once: my Canvas, dot, draw, left parenthesis, house, right parenthesis. Line 12, indented once: my Canvas, dot, draw, left parenthesis, door, right parenthesis. Line 13, indented once: my Canvas, dot, draw, left parenthesis, roof 1, right parenthesis. Line 14, indented once: my Canvas, dot, draw, left parenthesis, roof 2, right parenthesis. Line 15, indented once: sun, equals, Circle, left parenthesis, Point, left parenthesis, 150, comma, 250, right parenthesis, comma, 20, right parenthesis. Line 16, indented once: sun, dot, set Fill, left parenthesis, single quote, yellow, single quote, right parenthesis. Line 17, indented once: my Canvas, dot, draw, left parenthesis, sun, right parenthesis.

Back to Listing

The arrows indicate inheritance. A box at the center reads shape. On the left, an ellipse inherits from the shape, a circle inherits from an ellipse. On the right, a polygon inherits from the shape, a rectangle inherits from the polygon, a square inherits from the rectangle, and a triangle inherits from the polygon.

Back to Figure

The arrows indicate inheritance. A box at the top reads Geometric Object which is linked to Canvas. Line, Point, and Shape inherit from the Geometric Object. The shape has further inheritances. An ellipse inherits from the shape, a circle inherits from an ellipse. A polygon inherits from the shape, a rectangle inherits from the polygon, a square inherits from the rectangle, and a triangle inherits from the polygon.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: my Canvas, equals, Canvas, left parenthesis, 800, comma, 600, right parenthesis. Line 2: my Line, Line, left parenthesis, Point, left parenthesis, negative 100, comma, negative 100, right parenthesis, comma, Point, left parenthesis, 100, comma, 100, right parenthesis, right parenthesis. Line 3: my Canvas, dot, draw, left parenthesis, my Line, right parenthesis.

Back to Listing

Line and Point inherit from the Geometric object, which is linked to a canvas. The shape inherits from the geometric object and has several branches. An ellipse inherits from the shape, a circle inherits from an ellipse. A polygon inherits from the shape, a rectangle inherits from the polygon, a square inherits from the rectangle, and a triangle inherits from the polygon. Each class has both instance variables and methods. The instance variables of geometric objects are double underscore line color and double underscore line width. The methods of geometric objects are get Width, set Color, set Width, and underscore draw. The instance variables of the canvas are double underscore turtle and double underscore screen. The method of the canvas is draw. The instance variables of the line are double underscore p 1 and double underscore p 2. The methods of the line are get P 1, get P 2, underscore draw. The instance variables of the point are double underscore x and double underscore y. The methods of the line are get x, get y, get coord, and underscore draw.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: import turtle Line 2: Blank. Line 3: class Canvas, colon. Line 4, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, w, coma, h, right parenthesis, colon. Line 5, indented twice: self, dot, double underscore, turtle, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 6, indented twice: self, dot, double underscore, screen, equals, turtle, dot, Screen, left parenthesis, right parenthesis. Line 7: Blank. Line 8, indented twice: self, dot, double underscore, screen, dot, setup, left parenthesis, width, equals, w, comma, height, equals, h, right parenthesis. Line 9, indented twice: self, dot, double underscore, turtle, dot, hide turtle, left parenthesis, right parenthesis. Line 10: Blank. Line 11, indented once: def draw, left parenthesis, self, g Object, right parenthesis, colon. Line 12, indented twice: self, dot, double underscore, turtle, dot, up, left parenthesis, right parenthesis, hash, prepare to move turtle without a trail Line 13, indented twice: self, dot, double underscore, screen, dot, tracer, left parenthesis, 0, right parenthesis, hash, turn off animation Line 14, indented twice: g Object, dot, underscore, draw, left parenthesis, self, dot, double underscore, turtle, right parenthesis. Line 15, indented twice: self, dot, double underscore, screen, dot, tracer, left parenthesis, 1, right parenthesis, hash, turn on animation.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: if is instance, left parenthesis, g Object, comma, Point, right parenthesis, colon. Line 2, indented once: hash, code to draw a Point. Line 3: elif is instance, left parenthesis, g Object, comma, Line, right parenthesis, colon. Line 4, indented once: hash, code to draw a Line. Line 5: elif is instance, left

parenthesis, g OBject, comma, Circle, right parenthesis, colon. Line 6, indented once: hash, code to draw a Circle. Line 7: ellipses. Line 8: else, colon. Line 9, indented once: print, left parenthesis, single quote, unknown geometric object, single quote, right parenthesis.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: from a b c import, asterisk. Line 2: class Geometric Object, left parenthesis, A B C, right parenthesis, colon, hash, inherit from Abstract Base Class. Line 3, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 4, indented twice: self, dot, double underscore, line Color, equals, single quote, black, single quote, hash, default drawing color. Line 5, indented twice: self, dot, double underscore, line Width, equals, 1, hash, default line size. Line 6: Blank. Line 7, indented once: def get Color, left parenthesis, self, right parenthesis, colon. Line 8, indented twice: return self, dot, double underscore, line Color. Line 9: Blank. Line 10, indented once: def get Width, left parenthesis, self, right parenthesis, colon. Line 11, indented twice: return self, dot, double underscore, line Width. Line 12: Blank. Line 13, indented once: def set Color, left parenthesis, self, comma, color, right parenthesis, colon. Line 14, indented twice: self, dot, double underscore, line Color, equals, color. Line 15: Blank. Line 16, indented once: def set Width, left parenthesis, self, comma, width, right parenthesis, colon. Line 17, indented twice: self, dot, double underscore, line Width, equals, width. Line 18: Blank. Line 19, indented once: at the rate of, abstract method, hash, indicate that, underscore, draw is abstract. Line 20, indented once: def, underscore, draw, left parenthesis, self,

comma, some turtle, right parenthesis, colon. Line 21, indented twice: pass, hash, do nothing. Line 22: Blank.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: class Point, left parenthesis, Geometric Object, right parenthesis, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, x, comma, y, right parenthesis, colon. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, double underscore, x, equals, x. Line 5, indented twice: self, dot, double underscore, y, equals, y. Line 6: Blank. Line 7, indented once: def get Coord, left parenthesis, self, right parenthesis, colon. Line 8, indented twice: return, left parenthesis, self, dot, double underscore, x, comma, self, dot, double underscore, y, right parenthesis. Line 9: Blank. Line 10, indented once: def get X, left parenthesis, self, right parenthesis, colon. Line 11, indented twice: return self, dot, double underscore, x. Line 12: Blank. Line 13, indented once: def get Y, left parenthesis, self, right parenthesis, colon. Line 14, indented twice: return self, dot, double underscore, y. Line 15: Blank. Line 16, indented once: def, underscore, draw, left parenthesis, self, comma, turtle, right parenthesis, colon. Line 17, indented twice: turtle, dot, go to, left parenthesis, self, dot, double underscore, x, comma, self, dot, double underscore, y, right parenthesis. Line 18, indented twice: turtle, dot, left parenthesis, self, dot, get Width, left parenthesis, right parenthesis, comma, self, dot, get Color, left parenthesis, right parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: turtle, dot, left parenthesis, self, dot, get Width, left parenthesis, right parenthesis, self, dot, get Color, left parenthesis, right parenthesis, right parenthesis.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: class Line, left parenthesis, Geometric Object, right parenthesis, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, p 1, comma, p 2, right parenthesis, colon. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, double underscore, p 1, equals, p 1. Line 5, indented twice: self, dot, double underscore, p 2, equals, p 2. Line 6: Blank. Line 7, indented once: def get P 1, left parenthesis, self, right parenthesis, colon. Line 8, indented twice: return self, dot, double underscore, p 1. Line 9: Blank. Line 10, indented once: def get P 2, left parenthesis, self, right parenthesis, colon. Line 11, indented twice: return self, dot, double underscore, p 2. Line 12: Blank. Line 13, indented once: def, underscore, draw, left parenthesis, self, comma, turtle, right parenthesis, colon. Line 14, indented twice: turtle, dot, color, left parenthesis, self, dot, get Color, left parenthesis, right parenthesis, right parenthesis. Line 15, indented twice: turtle, dot, width, left parenthesis, self, dot, get Width, left parenthesis, right parenthesis, right parenthesis. Line 16, indented twice: turtle, dot, go to, left parenthesis, self, dot, double underscore, p 1, dot, get Coord, left parenthesis, right parenthesis, right parenthesis. Line

17, indented twice: turtle, dot, down, left parenthesis, right parenthesis. Line 18, indented twice: turtle, dot, go to, left parenthesis, self, dot, double underscore, p 2, dot, get Coord, left parenthesis, right parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: from draw import, asterisk. Line 2: my Canvas, equals, Canvas, left parenthesis, 800, comma, 600, right parenthesis. Line 3: my Line, equals, Line, left parenthesis, Point, left parenthesis, negative 100, comma, negative 100, right parenthesis, comma, Point, left parenthesis, 100, comma, 100, right parenthesis, right parenthesis. Line 4: my Canvas, dot, draw, left parenthesis, my Line, right parenthesis.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from draw import, asterisk. Line 2: my Line, equals, Line, left parenthesis, Point, left parenthesis, negative 100, coma, negative 100, right parenthesis, comma, Point, left parenthesis, 100, comma, 100, right parenthesis, right parenthesis. Input 2: Line 1: is instance, left parenthesis, my Line, comma, Line, right parenthesis. Output 2: True. Input 3: Line 1: is instance, left parenthesis, my Line, comma, Geometric Object, right parenthesis, hash, Line IS-A Geometric Object. Output 3: True. Input 4: Line 1: my Line, dot, get Width, left parenthesis, right parenthesis, hash, call inherited methods. Output 4: 1. Input 5: Line 1: my Line, dot, get Color, left parenthesis, right parenthesis. Output 5: single quote, black, single quote. Input 6: Line 1: p, equals, my Line, dot, get P 1, left

parenthesis, right parenthesis, hash, get Point object from Line. Input 10: Line 1: Is instance, left parenthesis, p, comma, Point, right parenthesis. Output 10: True. Input 11: Line 1: is instance, left parenthesis, p, comma, Geometric Object, right parenthesis, hash, Point IS-A Geometric Object. Output 11: True. Input 12: Line 1: p, dot, get Width, left parenthesis, right parenthesis, hash, call inherited methods. Output 12: 1. Input 13: Line 1: p, dot, get Color, left parenthesis, right parenthesis. Output 13: single quote, black, single quote. Input 14: Line 1: p, dot, get X, left parenthesis, right parenthesis, hash, call Point methods. Output 14: negative 100. Input 15: Line 1: p, dot, get Y, left parenthesis, right parenthesis. Output 15: negative 100. Input 16: Line 1: p, dot, get Coord, left parenthesis, right parenthesis. Output 16: left parenthesis, negative 100, comma, negative 100, right parenthesis.

Back to Session

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from draw import, asterisk. Line 2: my Line, equals, Line, left parenthesis, Point, left parenthesis, negative 100, coma, negative 100, right parenthesis, comma, Point, left parenthesis, 100, comma, 100, right parenthesis, right parenthesis. Input 2: Line 1: Line, dot, double underscore, bases, double underscore. Output 2: left parenthesis, draw, dot, Geometric Object, comma, right parenthesis. Input 3: Line 1: Geometric Object, dot, double underscore, bases, double underscore. Output 3: left parenthesis, a b c, dot, A B C, comma, right parenthesis. Input 4: Line 1: A B C, dot, double underscore, bases, double underscore. Output 4: left parenthesis, object, comma, right parenthesis. Input 5: Line 1: object, dot, double underscore, bases, double underscore. Output

5: left parenthesis, right parenthesis. Input 6: Line 1: Line, dot, double underscore, dict, double underscore, dot, keys, left parenthesis, right parenthesis. Output 6: dict, underscore, keys, left parenthesis, left square bracket, single quote, double underscore, module, double underscore, single quote, comma, single quote, double underscore, init, double underscore, single quote, comma, single quote, get P 1, single quote, comma, single quote, get P 2, single quote, comma, single quote, underscore, draw, single quote, comma, single quote, double underscore, doc, double underscore, single quote, comma, single quote, double underscore, abstract methods, double underscore, single quote, comma, single quote, underscore, a b c, underscore, impl, single quote, right square bracket, right parenthesis. Input 7: Line 1: Geometric Object, dot, double underscore, dict, double underscore, dot, keys, left parenthesis, right parenthesis. Output 7: dict, underscore, keys, left parenthesis, left square bracket, single quote, double underscore, module, double underscore, single quote, comma, single quote, double underscore, init, double underscore, single quote, comma, single quote, get Color, single quote, comma, single quote, get Width, single quote, comma, single quote, set Color, single quote, comma, single quote, set Width, single quote, comma, single quote, underscore, draw, single quote, comma, single quote, double underscore, dict, double underscore, single quote, comma, single quote, double underscore, weak ref, double underscore, single quote, comma, single quote, double underscore, doc, double underscore, single quote, comma, single quote, double underscore, abstract methods, double underscore, single quote, comma, single quote, underscore, a b c, underscore, impl, single quote, right square bracket, right parenthesis. Input 8: Line 1: my Line, dot, double underscore, class, double underscore. Output 8: draw, dot, Line. Input 9: Line 1: my Line, dot, double

underscore, dict, double underscore, dot, keys, left parenthesis, right parenthesis. Output 9: dict, underscore, keys, left parenthesis, left square bracket, single quote, underscore, Geometric Object, double underscore, line Color, single quote, comma, single quote, underscore, Geometric Object, double underscore, line Width, single quote, comma, single quote, underscore, Line, double underscore, p 1, single quote, comma, single quote, underscore, Line, double underscore, p 2, single quote, right square bracket, right parenthesis.

Back to Session

The steps are as follows. 1. Python evaluates the expression, my Line, dot, get Width, left parenthesis, right parenthesis, and dereferences it. 2. The get width key is searched. 3. Follow the class link to the line class, since the lines are not found. 4. The get width key is searched. 5. Follow the base link to the superclass Geometric Object. 6. The get width key is searched. 7. The get width key is found in the Geometric Object class.

Back to Figure

The key get width is searched in the local parameter of the global class that has a formal parameter self. The key is searched in all the classes. My Line and self both reference the same instance of a Line.

Back to Figure

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from draw import, asterisk. Input 2: Line 1: def test 2, left parenthesis, right parenthesis, colon. Line 2, indented once: my Canvas, equals,

Canvas, left parenthesis, 500, comma, 500, right parenthesis. Line 3, indented once: Line 1, equals, Line, left parenthesis, Point, left parenthesis, negative 100, comma, negative 100, right parenthesis, comma, Point, left parenthesis, 100, comma, 100, right parenthesis, right parenthesis. Line 4, indented once: Line 2, equals, Line, left parenthesis, Point, left parenthesis, negative 100, comma, negative 100, right parenthesis, comma, Point, left parenthesis, 100, comma, negative 100, right parenthesis, right parenthesis. Line 5, indented once: line 1, dot, set Width, left parenthesis, 4, right parenthesis. Line 6, indented once: my Canvas, dot, draw, left parenthesis, line 1, right parenthesis. Line 7, indented once: my Canvas, dot, draw, left parenthesis, line 2, right parenthesis. Line 8: Blank. Line 9, indented once: line 1, dot, set Color, left parenthesis, single quote, red, single quote, right parenthesis. Line 10, indented once: line 2, dot, set Width, left parenthesis, 4, right parenthesis. Line 11: blank. Input 3: Line 1: test 2, left parenthesis, right parenthesis.

Back to Session

Line and Point inherit from the Geometric object, which is linked to a canvas. The shape inherits from the geometric object and has several branches. An ellipse inherits from the shape, a circle inherits from an ellipse. A polygon inherits from the shape, a rectangle inherits from the polygon, a square inherits from the rectangle, and a triangle inherits from the polygon. Each class has both instance variables and methods. The instance variables of geometric objects are double underscore line color, double underscore line width, double underscore visible, and double underscore my canvas. The methods of geometric objects are get color, get Width, set Color, set Width, underscore draw, set Visible, get Visible, set Canvas, and get Canvas. The instance

variables of the canvas are double underscore turtle, double underscore screen, and double underscore visible Objects. The methods of the canvas are draw, draw all, and shape. The instance variables of the line are double underscore p 1 and double underscore p 2. The methods of the line are get P 1, get P 2, and underscore draw. The instance variables of the point are double underscore x and double underscore y. The methods of the line are get x, get y, get coord, and underscore draw.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: class Canvas, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, w, comma, h, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, visible Objects, equals, left square bracket, right square bracket, hash, list of shapes to draw. Line 4, indented twice: self, dot, double underscore, turtle, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 5, indented twice: self, dot, double underscore, screen, equals, turtle, dot, Screen, left parenthesis, right parenthesis. Line 6, indented twice: self, dot, double underscore, screen, dot, setup, left parenthesis, width, equals, w, comma, height, equals, h, right parenthesis. Line 7, indented twice: self, dot, double underscore, turtle, dot, hide turtle, left parenthesis, right parenthesis. Line 8: Blank. Line 9, indented once: def draw All, left parenthesis, self, right parenthesis, colon. Line 10, indented twice: self, dot, double underscore, turtle, dot, reset, left parenthesis, right parenthesis. Line 11, indented twice: self, dot, double underscore, turtle, dot, up, left parenthesis, right parenthesis. Line 12, indented twice: self, dot, double underscore, screen, dot, tracer, left parenthesis, 0, right parenthesis. Line 13, indented twice: for shape in self,

dot, double underscore, visible Objects, colon, hash, draw all shapes in order. Line 14, indented thrice: shape, dot, underscore, draw, left parenthesis, self, dot, double underscore, turtle, right parenthesis. Line 15, indented twice: self, dot, double underscore, screen, dot, tracer, left parenthesis, 1, right parenthesis. Line 16, indented twice: self, dot, double underscore, turtle, dot, hide turtle, left parenthesis, right parenthesis. Line 17: Blank. Line 18, indented once: def add Shape, left parenthesis, self, comma, shape, right parenthesis, colon. Line 19, indented twice: self, dot, double underscore, visible Objects, dot, append, left parenthesis, shape, right parenthesis. Line 20: Blank. Line 21, indented once: def draw, left parenthesis, self, comma, g Object, right parenthesis, colon. Line 22, indented twice: g Object, dot, set Canvas, left parenthesis, self, right parenthesis. Line 23, indented twice: g Object, dot, set Visible, left parenthesis, True, right parenthesis. Line 24, indented twice: self, dot, double underscore, turtle, dot, up, left parenthesis, right parenthesis. Line 25, indented twice: self, dot, double underscore, screen, dot, tracer, left parenthesis, 0, right parenthesis. Line 26, indented twice: g Object, dot, underscore, draw, left parenthesis, self, dot, double underscore, turtle, right parenthesis. Line 27, indented twice: self, dot, double underscore, screen, dot, tracer, left parenthesis, 1, right parenthesis. Line 28, indented twice: self, dot, add Shape, left parenthesis, g Object, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: from a b c import, asterisk. Line 2: class Geometric Object, left parenthesis, A B C, right parenthesis, colon. Line 3, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 4,

indented twice: self, dot, double underscore, line Color, equals, single quote, black, single quote. Line 5, indented twice: self, dot, double underscore, line Width, equals, 1. Line 6, indented twice: self, dot, double underscore, visible, equals, False. Line 7, indented twice: self, dot, double underscore, my Canvas, equals, None. Line 8: Blank. Line 9, indented once: def set Color, left parenthesis, self, comma, color, right parenthesis, colon, hash, modified to redraw visible shapes. Line 10, indented twice: self, dot, double underscore, line Color, equals, color. Line 11, indented twice: if self, dot, double underscore, visible, colon. Line 12, indented thrice: self, dot, double underscore, my Canvas, dot, draw All, left parenthesis, right parenthesis. Line 13: Blank. Line 14, indented once: def set Width, left parenthesis, self, comma, width, right parenthesis, colon, hash, modified to redraw visible shapes. Line 15, indented twice: self, dot, double underscore, line Width, equals, width. Line 16, indented twice: if self, dot, double underscore, visible, colon. Line 17, indented thrice: self, dot, double underscore, my Canvas, dot, draw All, left parenthesis, right parenthesis. Line 18. Blank. Line 19, indented once: def get Color, left parenthesis, self, right parenthesis, colon. Line 20, indented twice: return self, dot, double underscore, line Color. Line 21: Blank. Line 22, indented once: def get Width, left parenthesis, self, right parenthesis, colon. Line 23, indented twice: return self, dot, double underscore, line Width. Line 24: Blank. Line 25, indented once: At, abstract method Line 26, indented once: def, underscore, draw, left parenthesis, self, right parenthesis, colon. Line 27, indented twice: pass. Line 28: Blank. Line 29, indented once: def set Visible, left parenthesis, self, comma, v Flag, right parenthesis, colon. Line 30, indented twice: self, dot, double underscore, visible, equals, v Flag. Line 31: Blank. Line 32, indented once: def get Visible, left parenthesis, self, right

parenthesis, colon. Line 33, indented twice: return self, dot, double underscore, visible. Line 34: Blank. Line 35, indented once: def set Canvas, left parenthesis, self, dot, comma, the Canvas, right parenthesis, colon. Line 36, indented twice: self, dot, double underscore, my Canvas, equals, the Canvas. Line 37: Blank. Line 38, indented once: def get Canvas, left parenthesis, self, right parenthesis, colon. Line 39, indented twice: return self, dot, double underscore, my Canvas.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: class Shape, left parenthesis, Geometric Object, right parenthesis, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, double underscore, fill Color, equals, None. Line 5: Blank. Line 6, indented once: def set Fill Cover, left parenthesis, self, comma, a Color, right parenthesis, colon. Line 7, indented twice: self, dot, double underscore, fill Color, equals, a Color. Line 8, indented twice: if self, dot, get Visible, left parenthesis, right parenthesis, colon. Line 9, indented thrice: self, dot, get Canvas, left parenthesis, right parenthesis, dot, draw All, left parenthesis, right parenthesis. Line 10: Blank. Line 11, indented once: def get Fill Color, left parenthesis, self, right parenthesis, colon. Line 12, indented twice: return self, dot, double underscore, fill Color.

Back to Listing

The parallelogram has four vertices labeled p 0, p 1, p 2, and p 3. The lines are drawn from p 0 to p 1, from p 1 to p 2, from p 2 to p 3, and from p 3 to p 0.

Back to Figure

The parallelogram has four vertices labeled p 0, p 1, p 2, and p 3. The lines are drawn from p 0 to p 1, from p 1 to p 2, from p 2 to p 3, and from p 3 to p 0. The points needed to specify a rectangle are the lower-left corner and the upper-right corner

Back to Figure

Line and Point inherit from the Geometric object, which is linked to a canvas. The shape inherits from the geometric object and has several branches. An ellipse inherits from the shape, a circle inherits from an ellipse. A polygon inherits from the shape, a rectangle inherits from the polygon, a square inherits from the rectangle, and a triangle inherits from the polygon. Each class has both instance variables and methods. The instance variables of geometric objects are double underscore line color, double underscore line width, double underscore visible, and double underscore my canvas. The methods of geometric objects are get color, get Width, set Color, set Width, underscore draw, set Visible, get Visible, set Canvas, and get Canvas. The instance variables of the canvas are double underscore turtle, double underscore screen, and double underscore visible Objects. The methods of the canvas are draw, draw all, and shape. The instance variables of the line are double underscore p 1 and double underscore p 2. The methods of the line are get P 1, get P 2, and underscore draw. The instance variables of the point are double underscore x and double underscore y. The methods of

the line are get x, get y, get coord, and underscore draw. The instance variable of the shape is double underscore fill Color. The methods of the shape are set Fill Color and get Fill Color. The instance variable of the polygon is double underscore point List. The methods of the polygon are set add Point, get Points, and underscore draw. The instance variables of the rectangle are double underscore lower Left and double underscore upper Right. The methods of the rectangle are get Lower Left and get Upper Right. The instance variables of the Triangle are double underscore p 1, double underscore p 2, and double underscore p 3. The methods of the Triangle are get P 1, get P 2, and get P 3.

Back to Figure

A box on the left reads Get event from the queue and a box on the right reads Call function to handle the event. An arrow flows from box 1 to box 2 and box 2 to box 1.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1. class Event Handler, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, queue, equals, left square bracket, right square bracket, hash, holds events. Line 4, indented twice: self, dot, double underscore, event Keeper, equals, left brace, right brace, hash, dict of event types and callbacks. Line 5: Blank. Line 6, indented once: def add Event, left parenthesis, self, event Name, right parenthesis, colon. Line 7, indented twice: self, dot, double underscore, queue, dot, append, left parenthesis, event Name, right parenthesis. Line 8: Blank. Line 9, indented once:

hash, specify function to call when event occurs. Line 10, indented once: def register Callback, left parenthesis, self, comma, event, comma, func, right parenthesis, colon. Line 11, indented twice: self, dot, double underscore, event Keeper, left square bracket, event, right square bracket, equals, func. Line 12: Blank. Line 13, indented once: def run, left parenthesis, self, right parenthesis, colon. Line 14, indented twice: while, left parenthesis, True, right parenthesis, colon, hash, endless loop Line 15, indented thrice: if len, left parenthesis, self, dot, double underscore, queue, right parenthesis, greater than, 0, colon. Line 16, indented quarce: Next Event, equals, self, dot, double underscore, queue, dot, pop, left parenthesis, 0, right parenthesis, hash, get next event. Line 17, indented quarce: self, dot, double underscore, event Keeper, left square bracket, next Event, right square bracket, left parenthesis, right parenthesis, hash, run callback function. Line 18, indented thrice: else, colon. Line 19, indented quarce: print, left parenthesis, single quote, queue is empty, single quote, right parenthesis.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from event handler import, asterisk. Input 2: Line 1: def my Mouse, left parenthesis, right parenthesis, colon. Line 2, indented once: print, left parenthesis, single quote, Oh no, the mouse was clicked, exclamation mark, single quote, right parenthesis. Line 3: blank. Input 3: Line 1: def my Key, left parenthesis, right parenthesis, colon. Line 2, indented once: print, left parenthesis, single quote, A key has been pressed, dot, single quote, right parenthesis. Line 3: blank. Input 4: Line 1: def run Event Handler, left parenthesis, right parenthesis, colon. Line 2, indented once:

eh, equals, Event Handler, left parenthesis, right parenthesis. Line 3, indented once: eh, dot, register Callback, left parenthesis, single quote, key, single quote, comma, my Key, right parenthesis. Line 4, indented once: eh, dot, register Callback, left parenthesis, single quote, mouse, single quote, comma, my Mouse, right parenthesis. Line 5, indented once: eh, dot, add Event, left parenthesis, single quote, mouse, single quote, right parenthesis. Line 6, indented once: eh, dot, add Event, left parenthesis, single quote, key, single quote, right parenthesis. Line 7, indented once: eh, dot, add Event, left parenthesis, single quote, mouse, single quote, right parenthesis. Line 8, indented once: eh, dot, run, left parenthesis, right parenthesis. Line 9. Blank. Input asterisk: Line 1: run Event Handler, left parenthesis, right parenthesis. The output displays the following. Line 1: Oh no, the mouse was clicked, exclamation mark. Line 2: A key has been pressed, dot. Line 3: Oh no, the mouse was clicked, exclamation mark. Line 4: queue is empty. Line 5: queue is empty. Line 6: ellipses. Line 7: queue is empty.

Back to Session

In the top sequence, there is a single thread of execution, which has three steps. The input passes through two add Event blocks and calls the run. In the bottom sequence, which illustrates multithreading. The input passes through two add Event blocks, a start block, and two more add Event blocks. The start call launches a new thread and calls the run method inside the new thread. The new thread runs in parallel with the old thread.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: from threading import Thread. Line 2: import time. Line 3: Blank. Line 4: class Event Handler, left parenthesis, Thread, right parenthesis, colon, hash, inherits from Thread class. Line 5, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 6, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis, hash, call Thread constructor. Line 7, indented twice: self, dot, double underscore, queue, equals, left square bracket, right square bracket. Line 8, indented twice: self, dot, double underscore, event Keeper, equals, left brace, right brace. Line 9: Blank. Line 10, indented once: def add Event, left parenthesis, self, comma, event Name, right parenthesis, colon. Line 11, indented twice: self, dot, double underscore, queue, dot, append, left parenthesis, event Name, right parenthesis. Line 12: Blank. Line 13, indented once: def register Callback, left parenthesis, self, comma, event, comma, func, right parenthesis, colon. Line 14, indented twice: self, dot, double underscore, event Keeper, left square bracket, event, right square bracket, equals, func. Line 15: Blank. Line 16, indented once: def run, left parenthesis, self, right parenthesis, colon. Line 17, indented twice: while, left parenthesis, True, right parenthesis, colon. Line 18, indented thrice: if len, left parenthesis, self, dot, double underscore, queue, right parenthesis, greater than, 0, colon. Line 19, indented quarce: Next Event, equals, self, dot, double underscore, queue, dot, pop, left parenthesis, 0, right parenthesis. Line 20, indented quarce: Call Back, equals, self, dot, double underscore, event Keeper, left square bracket, next Event, right square bracket. Line 21, indented quarce: Call Back, left parenthesis, right parenthesis, hash, call registered function for event Line 22, indented thrice:

else, colon. Line 23, indented quarce: time, dot, sleep, left parenthesis, 1, right parenthesis, hash, pause for 1 second.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: from event handler multithread import, asterisk. Input 2: Line 1: eh, equals, Event Handler, left parenthesis, right parenthesis. Input 3: Line 1: eh, dot, add Event, left parenthesis, single quote, mouse, single quote, right parenthesis. Input 4: Line 1: eh, dot, add Event, left parenthesis, single quote, key, single quote, right parenthesis. Input 5: Line 1: eh, dot, add Event, left parenthesis, single quote, mouse, single quote, right parenthesis. Input 6: eh, dot, register Callback, left parenthesis, single quote, key, single quote, comma, my Key, right parenthesis. Input 7: eh, dot, register Callback, left parenthesis, single quote, mouse, single quote, comma, my Mouse, right parenthesis. Input 8: Line 1: eh, dot, start, left parenthesis, right parenthesis. The output displays the following. Line 1: Oh no, the mouse was clicked, exclamation mark. Line 2: A key has been pressed, dot. Line 3: Oh no, the mouse was clicked, exclamation mark. Input 9: Line 1: eh, dot, add Event, left parenthesis, single quote, mouse, single quote, right parenthesis. Line 2: eh, dot, add Event, left parenthesis, single quote, key, single quote, right parenthesis. Line 3: eh, dot, add Event, left parenthesis, single quote, mouse, single quote, right parenthesis. The output displays the following. Line 1: Oh no, the mouse was clicked, exclamation mark. Line 2: A key has been pressed, dot. Line 3: Oh no, the mouse was clicked, exclamation mark.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: import turtle. Line 2: Blank. Line 3: class Etch, colon. Line 4, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 5, indented twice: self, dot, double underscore, my T, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 6, indented twice: self, dot, double underscore, my Screen, equals, turtle, dot, Screen, left parenthesis, right parenthesis. Line 7, indented twice: self, dot, double underscore, my T, dot, color, left parenthesis, single quote, blue, single quote, right parenthesis. Line 8, indented twice: self, dot, double underscore, my T, dot, pen size, left parenthesis, 2, right parenthesis. Line 9, indented twice: self, dot, double underscore, my T, dot, peed, left parenthesis, 0, right parenthesis. Line 10, indented twice: self, dot, double underscore, distance, equals, 5, hash, distance to move. Line 11, indented twice: self, dot, double underscore, turn, equals, 10, hash, degrees to turn. Line 12, indented twice: self, dot, double underscore, my Screen, dot, on key, left parenthesis, self, dot, f w d, comma, double quotes, Up, double quotes, right parenthesis, hash, register callbacks. Line 13, indented twice: self, dot, double underscore, my Screen, dot, on key, left parenthesis, self, dot, b k w d, comma, double quotes, Down, double quotes, right parenthesis. Line 14, indented twice: self, dot, double underscore, my Screen, dot, on key, left parenthesis, self, dot, left, comma, double quotes, Left, double quotes, right parenthesis. Line 15, indented twice: self, dot, double underscore, my Screen, dot, on key, left parenthesis, self, dot, right, comma, double quotes, Right, double quotes, right parenthesis. Line 16, indented twice: self, dot, double underscore, my Screen, dot, on key, left parenthesis, self, dot, quit, comma, double quotes, q, double quotes, right parenthesis. Line 17, indented twice: self, dot, double underscore, my Screen, dot, listen, left parenthesis,

right parenthesis, hash, listen for key presses. Line 18, indented twice: self, dot, main, left parenthesis, right parenthesis, hash, start the Turtle main loop. Line 19: Blank. Line 20, indented once: def f w d, left parenthesis, self, right parenthesis, colon. Line 21, indented twice: self, dot, double underscore, my T, dot, forward, left parenthesis, self, dot, double underscore, distance, right parenthesis. Line 22: Blank. Line 23, indented once: def b k w d, left parenthesis, self, right parenthesis, colon. Line 24, indented twice: self, dot, double underscore, my T, dot, backward, left parenthesis, self, dot, double underscore, distance, right parenthesis. Line 25: Blank. Line 26, indented once: def left, left parenthesis, self, right parenthesis, colon. Line 27, indented twice: self, dot, double underscore, my T, dot, left, left parenthesis, self, dot, double underscore, turn, right parenthesis. Line 28: Blank. Line 29, indented once: def right, left parenthesis, self, right parenthesis, colon. Line 30, indented twice: self, dot, double underscore, my T, dot, right, left parenthesis, self, dot, double underscore, turn, right parenthesis. Line 31: Blank. Line 32, indented once: def quit, left parenthesis, self, right parenthesis, colon. Line 33, indented twice: self, dot, double underscore, my Screen, dot, bye, left parenthesis, right parenthesis. Line 34: Blank. Line 35, indented once: def main, left parenthesis, self, right parenthesis, colon. Line 36, indented twice: turtle, dot, main loop, left parenthesis, right parenthesis. Line 37: Blank.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: left square bracket, x, asterisk, asterisk, for x in range, left parenthesis, 5, right parenthesis, right square bracket, hash, list of squares from

1 to 4. Output 1: left square bracket, 0, comma, 1, comma, 4, comma, 9, comma, 16, right square bracket. Input 2: Line 1: movies, equals, left square bracket, single quote, alien, single quote, comma, single quote, starwars, single quote, comma, single quote, bourne, single quote, right square bracket. Input 3: Line 1: left square bracket, x, dot, capitalize, left parenthesis, right parenthesis, for x in movies, right square bracket, hash, capitalize each movie. Output 3: left square bracket, single quote, Alien, single quote, comma, single quote, Starwars, single quote, comma, single quote, Bourne, single quote, right square bracket. Input 4: Line 1: left square bracket, x, left square bracket, negative 1, right square bracket, for x in movies if len, left parenthesis, x, right parenthesis, greater than, 6, right square bracket, hash, last letter if longer than 6 chars. Output 4: left square bracket, single quote, s, single quote, right square bracket. Input 5: Line 1: left square bracket, x, left square bracket, negative 1, right square bracket, for x in movies if len, left parenthesis, x, right parenthesis, less than or equal to, 6, right square bracket, hash, last letter if 6 or fewer chars. Out 5: left square bracket, single quote, n, single quote, comma, single quote, e, single quote, right square bracket.

Back to Session

Program code. In the code, the words in the variables are merged. Line 1: from turtle import Turtle, main loop. Line 2: Blank. Line 3: class Etch, left parenthesis, Turtle, right parenthesis, colon. Line 4, indented once: def, double underscore, init, double underscore, left parenthesis, self, right parenthesis, colon. Line 5, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis. Line 6, indented

twice: self, dot, double underscore, screen, equals, self, dot, get screen, left parenthesis, right parenthesis. Line 7, indented twice: self, dot, color, left parenthesis, single quote, blue, single quote, right parenthesis. Line 8, indented twice: self, dot, pen size, left parenthesis, 2, right parenthesis. Line 9, indented twice: self, dot, speed, left parenthesis, 0, right parenthesis. Line 10, indented twice: self, dot, double underscore, distance, equals, 5. Line 11, indented twice: self, dot, double underscore, turn, equals, 10. Line 12, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, f w d, comma, double quotes, Up, double quotes, right parenthesis. Line 13, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, b k w d, comma, double quotes, Down, double quotes, right parenthesis. Line 14, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, left 10, comma, double quotes, Left, double quotes, right parenthesis. Line 15, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, right 10, comma, double quotes, Right, double quotes, right parenthesis. Line 16, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, quit, comma, double quotes, q, double quotes, right parenthesis. Line 17, indented twice: self, dot, double underscore, screen, dot, listen, left parenthesis, right parenthesis. Line 18, indented twice: self, dot, main, left parenthesis, right parenthesis. Line 19: Blank. Line 20, indented once: def f w d, left parenthesis, self, right parenthesis, colon. Line 21, indented twice: self, dot, forward, left parenthesis, self, dot, double underscore, distance, right parenthesis. Line 22: Blank. Line 23, indented once: def b k w d, left parenthesis, self, right parenthesis, colon. Line 24, indented twice: self, dot, backward, left parenthesis, self, dot, double underscore, distance, right parenthesis. Line 25: Blank. Line 26, indented once: def left

10, left parenthesis, self, right parenthesis, colon. Line 27, indented twice: self, dot, left, left parenthesis, self, dot, double underscore, turn, right parenthesis. Line 28: Blank. Line 29, indented once: def right 10, left parenthesis, self, right parenthesis, colon. Line 30, indented twice: self, dot, right, left parenthesis, self, dot, double underscore, turn, right parenthesis. Line 31: Blank. Line 32, indented once: def quit, left parenthesis, self, right parenthesis, colon. Line 33, indented twice: self, dot, double underscore, screen, dot, bye, left parenthesis, right parenthesis. Line 34: Blank. Line 35, indented once: def main, left parenthesis, self, right parenthesis, colon. Line 36, indented twice: Main loop, left parenthesis, right parenthesis. Line 37: Blank. Line 38: if, double underscore, name, double underscore, equals, equals, single quote, double underscore, main, double underscore, single quote, colon. Line 39, indented once: etch, equals, Etch, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: from turtle import Turtle, comma, main loop. Line 2: import random. Line 3: Blank. Line 4: class Turtle Place, colon. Line 5, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, max Turtles, comma, h Wall, equals, 200, comma, v Wall, equals, 200, right parenthesis, colon. Line 6, indented twice: self, dot, double underscore, big T, equals, Turtle, left parenthesis, right parenthesis. Line 7, indented twice: self, dot, double underscore, big T screen, equals, self, dot, double underscore, big T, dot, get screen, left parenthesis, right parenthesis. Line 8, indented twice: self, dot, double underscore, big T, dot, color, left parenthesis, single quote, blue, single quote, right parenthesis. Line 9, indented twice: self, dot,

double underscore, big T screen, dot, on click, left parenthesis, self, dot, place Turtle, right parenthesis. Line 10, indented twice: self, dot, double underscore, big T, dot, hide turtle, left parenthesis, right parenthesis. Line 11, indented twice: self, dot, double underscore, num Turtles, equals, 0. Line 12, indented twice: self, dot, double underscore, max Turtles, equals, max Turtles. Line 13, indented twice: self, dot, double underscore, h Wall, equals, h Wall. Line 14, indented twice: self, dot, double underscore, vWall, equals, v Wall. Line 15, indented twice: self, dot, draw Field, left parenthesis, right parenthesis. Line 16, indented twice: Main loop, left parenthesis, right parenthesis. Line 17: Blank. Line 18, indented once: def place Turtle, left parenthesis, self, comma, x, comma, y, right parenthesis, colon. Line 19, indented twice: New T, equals, Turtle, left parenthesis, right parenthesis. Line 20, indented twice: new T, dot, hide turtle, left parenthesis, right parenthesis. Line 21, indented twice: new T screen, equals, new T, dot, get screen, left parenthesis, right parenthesis. Line 22, indented twice: new T screen, dot, tracer, left parenthesis, 0, right parenthesis. Line 23, indented twice: new T, dot, up, left parenthesis, right parenthesis. Line 24, indented twice: new T, dot, go to, left parenthesis, x, comma, y, right parenthesis. Line 25, indented twice: new T, dot, shape, left parenthesis, single quote, turtle, single quote, right parenthesis. Line 26, indented twice: new T, dot, show turtle, left parenthesis, right parenthesis. Line 27, indented twice: new T, dot, set heading, left parenthesis, random, dot, rand int, left parenthesis, 1, comma, 359, right parenthesis, right parenthesis. Line 28, indented twice: new T screen, dot, tracer, left parenthesis, 1, right parenthesis. Line 29, indented twice: self, dot, double underscore, num Turtles, equals, self, dot, double underscore, num Turtles, plus, 1. Line 30, indented twice: if self, dot, double underscore, num Turtles, right angle bracket, equals, self, dot, double

underscore, max Turtles, colon. Line 31, indented thrice: self, dot, double underscore, big T screen, dot, on click, left parenthesis, None, right parenthesis, hash, cancel event handler. Line 32: Blank. Line 33, indented once: def draw Field, left parenthesis, self, right parenthesis, colon. Line 34, indented twice: self, dot, double underscore, big T screen, dot, tracer, left parenthesis, 0, right parenthesis. Line 35, indented twice: self, dot, double underscore, big T, dot, up, left parenthesis, right parenthesis. Line 36, indented twice: self, dot, double underscore, big T, dot, go to, left parenthesis, hyphen, left parenthesis, self, dot, double underscore, h Wall, right parenthesis, comma, hyphen, left parenthesis, self, dot, double underscore, v Wall, right parenthesis, right parenthesis. Line 37, indented twice: self, dot, double underscore, big T, dot, down, left parenthesis, right parenthesis. Line 38, indented twice: for i in range, left parenthesis, 4, right parenthesis, colon. Line 39, indented thrice: self, dot, double underscore, big T, dot, forward, left parenthesis, 2, asterisk, self, dot, double underscore, h Wall, right parenthesis. Line 40, indented thrice: self, dot, double underscore, big T, dot, left, left parenthesis, 90, right parenthesis. Line 41, indented twice: self, dot, double underscore, big T screen, dot, tracer, left parenthesis, 1, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: from turtle import Turtle, comma, main loop. Line 2: import random. Line 3: Blank. Line 4: class Animated Turtle, left parenthesis, Turtle, right parenthesis, colon. Line 5, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, h Wall, comma, v Wall, right parenthesis, colon. Line 6, indented twice: super, left parenthesis,

right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis. Line 7, indented twice: self, dot, double underscore, s c r, equals, self, dot, get screen, left parenthesis, right parenthesis. Line 8, indented twice: self, dot, double underscore, x Min, equals, negative v Wall. Line 9, indented twice: self, dot, double underscore, x Max, equals, v Wall. Line 10, indented twice: self, dot, double underscore, y Min, equals, negative h Wall. Line 11, indented twice: self, dot, double underscore, y Max, equals, h Wall. Line 12, indented twice: self, dot, double underscore, s c r, dot, on timer, left parenthesis, self, dot, double underscore, move One Step, comma, 100, right parenthesis. Line 13: Blank. Line 14, indented once: def, double underscore, move One Step, left parenthesis, self, right parenthesis, colon. Line 15, indented twice: self, dot, double underscore, compute New Heading, left parenthesis, right parenthesis. Line 16, indented twice: self, dot, forward, left parenthesis, 5, right parenthesis. Line 17, indented twice: self, dot, double underscore, s c r, dot, on timer, left parenthesis, self, dot, double underscore, move One Step, comma, 100, right parenthesis. Line 18: Blank. Line 19, indented once: def, double underscore, compute New Heading, left parenthesis, self, right parenthesis, colon. Line 20, indented twice: x Pos, comma, y Pos, equals, self, dot, position, left parenthesis, right parenthesis. Line 21, indented twice: Old Head, equals, self, dot, heading, left parenthesis, right parenthesis. Line 22, indented twice: new Head, equals, old Head. Line 23: Blank. Line 24, indented twice: if x Pos, greater than, self, dot, double underscore, x Max or x Pos, less than, self, dot, double underscore, x Min, colon. Line 25, indented thrice: new Head, equals, 180, minus, old Head. Line 26, indented twice: if y Pos, greater than, self, dot, double underscore, y Max or y Pos, less than, self, dot, double underscore, y Min, colon. Line 27, indented thrice: New Head,

equals, 360, minus, old Head. Line 28, indented twice: if new Head, not equal to, old Head, colon. Line 29, indented thrice: self, dot, set heading, left parenthesis, new Head, right parenthesis. Line 30: Blank.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1, indented once: def place Turtle left parenthesis, self, comma, x, comma, y, right parenthesis, colon. Line 2, indented twice: new T, equals, Animated Turtle, left parenthesis, self, dot, double underscore, h Wall, comma, self, dot, double underscore, v Wall, right parenthesis. Line 3, indented twice: new T, dot, hide turtle, left parenthesis, right parenthesis. Line 4, indented twice: new T screen, equals, new T, dot, get screen, left parenthesis, right parenthesis. Line 5, indented twice: new T screen, dot, tracer, left parenthesis, 0, right parenthesis. Line 6, indented twice: new T, dot, up, left parenthesis, right parenthesis. Line 7, indented twice: new T, dot, go to, left parenthesis, x, comma, y, right parenthesis. Line 8, indented twice: new T, dot, shape, left parenthesis, single quote, turtle, single quote, right parenthesis. Line 9, indented twice: new T, dot, color, left parenthesis, single quote, green, single quote, right parenthesis. Line 10, indented twice: new T, dot, show turtle, left parenthesis, right parenthesis. Line 11, indented twice: new T, dot, set heading, left parenthesis, random, dot, rand int, left parenthesis, 1, comma, 359, right parenthesis, right parenthesis. Line 12, indented twice: new T screen, dot, tracer, left parenthesis, 1, right parenthesis. Line 13, indented twice: self, dot, double underscore, num Turtles, equals, self, dot, double underscore, num Turtles, plus, 1. Line 14, indented twice: if self, dot, double underscore, num Turtles, greater than or equal to, self, dot, double

underscore, max Turtles, colon. Line 15, indented thrice: self, dot, double underscore, big T screen, dot, on click, left parenthesis, None, right parenthesis, hash, remove event handler. Line 16: Blank.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: from turtle import Turtle, comma, main loop. Line 2: import random. Line 3: Blank. Line 4: class Animated Turtle, left parenthesis, Turtle, right parenthesis, colon. Line 5: Blank. Line 6, indented once: double underscore, all Turtles, equals, left square bracket, right square bracket. Line 7: Blank. Line 8, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, h Wall, comma, v Wall, right parenthesis, colon. Line 9, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis. Line 10, indented twice: self, dot, double underscore, s c r, equals, self, dot, get screen, left parenthesis, right parenthesis. Line 11, indented twice: self, dot, double underscore, x Min, equals, negative v Wall, plus, 10. Line 12, indented twice: self, dot, double underscore, x Max, equals, v Wall, minus, 10. Line 13, indented twice: self, dot, double underscore, y Min, equals, negative h Wall, plus, 10. Line 14, indented twice: self, dot, double underscore, y Max, equals, h Wall, minus, 10. Line 15, indented twice: self, dot, double underscore, s c r, dot, on timer, left parenthesis, self, dot, double underscore, move One Step, comma, 100, right parenthesis. Line 16, indented twice: Animated Turtle, dot, double underscore, all Turtles, dot, append, left parenthesis, self, right parenthesis. Line 17: Blank. Line 18, indented once: def, double underscore, move One Step, left

parenthesis, self, right parenthesis, colon. Line 19, indented twice: self, dot, double underscore, compute New Heading, left parenthesis, right parenthesis. Line 20, indented twice: self, dot, forward, left parenthesis, 5, right parenthesis. Line 21, indented twice: self, dot, double underscore, check Collisions, left parenthesis, right parenthesis. Line 22, indented twice: self, dot, double underscore, s c r, dot, on timer, left parenthesis, self, dot, double underscore, move One Step, comma, 100, right parenthesis. Line 23: Blank. Line 24, indented once: def, double underscore, compute New Heading, left parenthesis, self, right parenthesis, colon. Line 25, indented twice: x Pos, comma, y Pos, equals, self, dot, position, left parenthesis, right parenthesis. Line 26, indented twice: old Head, equals, self, dot, heading, left parenthesis, right parenthesis. Line 27, indented twice: new Head, equals, old Head. Line 28: Blank. Line 29, indented twice: if x Pos, greater than, self, dot, double underscore, x Max or x Pos, less than, self, dot, double underscore, x Min, colon. Line 30, indented thrice: new Head, equals, 180, minus, old Head. Line 31, indented twice: if y Pos, greater than, self, dot, double underscore, y Max or y Pos, less than, self, dot, double underscore, y Min, colon. Line 32, indented thrice: new Head, equals, 360, minus, old Head. Line 33, indented twice: if new Head, not equal to, old Head, colon. Line 34, indented thrice: self, dot, set heading, left parenthesis, new Head, right parenthesis. Line 35: Blank. Line 36, indented once: def, double underscore, check Collisions, left parenthesis, self, right parenthesis, colon. Line 37, indented twice: for other T in Animated Turtle, dot, double underscore, all Turtles, colon. Line 38, indented thrice: if self, not equal to, other T, colon. Line 39, indented quarce: if self, dot, distance, left parenthesis, other T, right parenthesis, less than, 20, colon, hash, if close, swap headings Line 40, indented quince: temp Heading, equals, self, dot, heading, left parenthesis, right

parenthesis. Line 41, indented quince: self, dot, set heading, left parenthesis, other T, dot, heading, left parenthesis, right parenthesis, right parenthesis. Line 42, indented quince: other T, dot, set heading, left parenthesis, temp Heading, right parenthesis. Line 43, indented quince: while self, dot, distance, left parenthesis, other T, right parenthesis, less than, 20, colon, hash, move away. Line 44, indented 6 times: self, dot, forward, left parenthesis, 1, right parenthesis. Line 45, indented 6 times: other T, dot, forward, left parenthesis, 1, right parenthesis.

Back to Listing

The screen is shaded green. At the bottom center of the window is the laser cannon, which is the usual turtle triangle. The drones at the top are the enemies, and the small circles are the bombs that have been shot from the laser cannon towards the drones.

Back to Figure

A box at the top reads turtle. The box represents the class. The laser cannon and bounded turtle inherit from the turtle. The bomb and the drone inherit from the bounded turtle. Each class has static variables and static methods. The static variables and static methods of the class turtle reads ellipses. The static variables of the laser cannon class are double underscore x Min, double underscore x Max, double underscore y Min, double underscore y Max, and double underscore screen. The static methods of the laser cannon class are aim, shoot, and quit. The static variables of the Bounded Turtle class are double underscore x Min, double underscore x Max, double underscore y Min, double underscore y Max, and double underscore speed. The static methods of the Bounded Turtle class are out Of Bounds, get

Speed, get X Min, get X Max, get Y Min, get Y Max, remove, and move. The static variables of the Bomb class are empty. The static methods of the Bomb class are distance, move, and remove. The static variables of the Drone class are drone List and double underscore alive. The static methods of the Drone class are get Drones, move, and remove. The static variable drone List and the static method get Drones are underlined.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: class Laser Cannon, left parenthesis, Turtle, right parenthesis, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, x Min, comma, x Max, comma, y Min, comma, y Max, right parenthesis, colon. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, double underscore, x Min, equals, x Min. Line 5, indented twice: self, dot, double underscore, x Max, equals, x Max. Line 6, indented twice: self, dot, double underscore, y Min, equals, y Min. Line 7, indented twice: self, dot, double underscore, y Max, equals, y Max. Line 8, indented twice: self, dot, double underscore, screen self, dot, get screen, left parenthesis, right parenthesis. Line 9, indented twice: self, dot, double underscore, screen, dot, on click, left parenthesis, self, dot, aim, right parenthesis. Line 10, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, shoot, comma, comma, single quote, s, single quote, right parenthesis. Line 11, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, quit, comma, single quote, q, single quote, right parenthesis. Line 12: Blank. Line 13,

indented once: def aim, left parenthesis, self, x, comma, y, right parenthesis, colon. Line 14, indented twice: heading, equals, self, dot, towards, left parenthesis, x, comma, y, right parenthesis. Line 15, indented twice: self, dot, set heading, left parenthesis, heading, right parenthesis. Line 16: Blank. Line 17, indented once: def shoot, left parenthesis, self, right parenthesis, colon. Line 18, indented twice: Bomb, left parenthesis, self, dot, heading, left parenthesis, right parenthesis, comma, 5, self, dot, double underscore, x Min, comma, self, dot, double underscore, x Max, comma, self, dot, double underscore, y Min, comma, self, dot, double underscore, y Max, right parenthesis. Line 20: Blank. Line 21, indented once: def quit, left parenthesis, self, right parenthesis, colon. Line 22, indented twice: self, dot, double underscore, screen, dot, bye, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: from a b c import, asterisk. Line 2: class Bounded Turtle, left parenthesis, Turtle, right parenthesis, colon. Line 3, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, speed, comma, x Min, comma, x Max, comma, y Min, comma, y Max, right parenthesis, colon. Line 4, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis. Line 5, indented twice: self, dot, double underscore, x Min, equals, x Min. Line 6, indented twice: self, dot, double underscore, x Max, equals, x Max. Line 7, indented twice: self, dot, double underscore, y Min, equals, y Min. Line 8, indented twice: self, dot, double underscore, y Max, equals, y Max. Line 9, indented twice: self, dot, double underscore, speed, equals, speed. Line 10: Blank. Line 11,

indented once: def out Of Bounds, left parenthesis, self, right parenthesis, colon. Line 12, indented twice: x Pos, comma, y Pos, equals, self, dot, position, left parenthesis, right parenthesis. Line 13, indented twice: out, equals, False. Line 14, indented twice: if x Pos, left angle bracket, self, dot, double underscore, x Min or x Pos, right angle bracket, self, dot, double underscore, x Max, colon. Line 15, indented thrice: out, equals, True. Line 16, indented twice: if y Pos, left angle bracket, self, dot, double underscore, y Min or y Pos, right angle bracket, self, dot, double underscore, y Max, colon. Line 17, indented thrice: out, equals, True. Line 18, indented twice: return out. Line 19: Blank. Line 20, indented once: def get Speed, left parenthesis, self, right parenthesis, colon. Line 21, indented twice: return self, dot, double underscore, speed. Line 22: Blank. Line 23, indented once: def get X Min, left parenthesis, self, right parenthesis, colon. Line 24, indented twice: return self, dot, double underscore, x Min. Line 25: Blank. Line 26, indented once: def get X Max, left parenthesis, self, right parenthesis, colon. Line 27, indented twice: return self, dot, double underscore, x Max. Line 28: Blank. Line 29, indented once: def get Y Min, left parenthesis, self, right parenthesis, colon. Line 30, indented twice: return self, dot, double underscore, y Min. Line 31: Blank. Line 32, indented once: def get Y Max, left parenthesis, self, right parenthesis, colon. Line 33, indented twice: return self, dot, double underscore, y Max. Line 34: Blank. Line 35, indented once: At, abstract method. Line 36, indented once: def remove, left parenthesis, self, right parenthesis, colon. Line 37, indented twice: Pass. Line 38: Blank. Line 39, indented once: At, abstract method. Line 40, indented once: def move, left parenthesis, self, right parenthesis, colon. Line 41, indented twice: Pass.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: class Drone, left parenthesis, Bounded Turtle, right parenthesis, colon. Line 2: Blank. Line 3, indented once: drone List, equals, left square bracket, right square bracket, hash, static variable. Line 4. Blank. Line 5, indented once: At, static method. Line 6, indented once: def get Drones, left parenthesis, right parenthesis, colon. Line 7, indented twice: return, left square bracket, for x in Drone, dot, drone List if x, dot, double underscore, alive, right square bracket. Line 8: Blank. Line 9, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, speed, comma, x Min, comma, x Max, comma, y Min, comma, y Max, right parenthesis, colon. Line 10, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, speed, comma, x Min, comma, x Max, comma, y Min, comma, y Max, right parenthesis. Line 11, indented twice: self, dot, get screen, left parenthesis, right parenthesis, dot, tracer, left parenthesis, 0, right parenthesis. Line 12, indented twice: self, dot, up, left parenthesis, right parenthesis. Line 13, indented twice: if, single quote, Drone, dot, gif, single quote, not in self, dot, get screen, left parenthesis, right parenthesis, dot, get shapes, left parenthesis, right parenthesis, colon. Line 14, indented thrice: self, dot, get screen, left parenthesis, right parenthesis, dot, add shape, left parenthesis, single quote, Drone, dot, gif, single quote, right parenthesis. Line 15, indented twice: self, dot, shape, left parenthesis, single quote, Drone, dot, gif, single quote, right parenthesis. Line 16, indented twice: self, dot, go to, left parenthesis, random, dot, rand int, left parenthesis, x Min, minus, 1, comma, x Max, minus, 1, right parenthesis, comma, y Max, minus, 20, right parenthesis. Line 17, indented twice: self, dot, set heading, left parenthesis, random, dot, rand int, left parenthesis, 250, comma, 290, right parenthesis, right parenthesis. Line 18,

indented twice: self, dot, get screen, left parenthesis, right parenthesis, dot, tracer, left parenthesis, 1, right parenthesis. Line 19, indented twice: Drone, dot, drone List, equals, Drone, dot, get Drones, left parenthesis, right parenthesis. Line 20, indented twice: Drone, dot, drone List, dot, append, left parenthesis, self, right parenthesis. Line 21, indented twice: self, dot, double underscore, alive, equals, True Line 22, indented twice: self, dot, get screen, left parenthesis, right parenthesis, dot, on timer, left parenthesis, self, dot, move, comma, 200, right parenthesis. Line 23: Blank. Line 24, indented once: def move, left parenthesis, self, right parenthesis, colon. Line 25, indented twice: self, dot, forward, left parenthesis, self, dot, get Speed, left parenthesis, right parenthesis, right parenthesis. Line 26, indented twice: if self, dot, out of Bounds, left parenthesis, right parenthesis, colon. Line 27, indented thrice: self, dot, remove, left parenthesis, right parenthesis. Line 28, indented twice: else, colon. Line 29, indented thrice: self, dot, get screen, left parenthesis, right parenthesis, dot, on timer, left parenthesis, self, dot, move, comma, 200, right parenthesis. Line 30: Blank. Line 31, indented once: def remove, left parenthesis, self, right parenthesis, colon. Line 32, indented twice: self, dot, double underscore, alive, equals, False. Line 33, indented twice: self, dot, hide turtle, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: def get Drones, left parenthesis, right parenthesis, colon. Line 2, indented once: my List, equals, left square bracket, right square bracket. Line 3, indented once: for x in Drone, dot, drone List, colon. Line 4, indented twice: if x, dot, double underscore, alive, colon. Line 5, indented thrice: my List, dot,

append, left parenthesis, x, right parenthesis. Line 6, indented once: return my List.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: class Bomb, left parenthesis, Bounded Turtle, right parenthesis, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, init Heading, comma, speed, comma, x Min, comma, x Max, comma, y Min, comma, y Max, right parenthesis, colon. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, speed, comma, x Min, comma, x Max, comma, y Min, comma, y Max, right parenthesis. Line 4, indented twice: self, dot, resize mode, left parenthesis, single quote, user, single quote, right parenthesis. Line 5, indented twice: self, dot, color, left parenthesis, single quote, red, single quote, comma, single quote, red, single quote, right parenthesis. Line 6, indented twice: self, dot, shape, left parenthesis, single quote, circle, single quote, right parenthesis. Line 7, indented twice: self, dot, turtle size, left parenthesis, 0.25, right parenthesis. Line 8, indented twice: self, dot, set heading, left parenthesis, init Heading, right parenthesis. Line 9, indented twice: self, dot, up, left parenthesis, right parenthesis. Line 10, indented twice: self, dot, get screen, left parenthesis, right parenthesis, dot, on timer, left parenthesis, self, dot, move, comma, 100, right parenthesis. Line 11: Blank. Line 12, indented once: def move, left parenthesis, self, right parenthesis, colon. Line 13, indented twice: exploded, equals, False. Line 14, indented twice: self, dot, forward, left parenthesis, self, dot, get Speed, left parenthesis, right parenthesis, right parenthesis. Line 15, indented twice: for a in Drone, dot, get

Drones, left parenthesis, right parenthesis, colon. Line 16, indented thrice: if self, dot, distance, left parenthesis, a, right parenthesis, less than, 5, colon. Line 17, indented quarce: a, dot, remove, left parenthesis, right parenthesis. Line 18, indented quarce: exploded, equals, True. Line 19, indented twice: if self, dot, out of Bounds, left parenthesis, right parenthesis, or exploded, colon. Line 20, indented thrice: self, dot, remove, left parenthesis, right parenthesis. Line 21, indented twice: else, colon. Line 22, indented thrice: self, dot, get screen, left parenthesis, right parenthesis, dot, on timer, left parenthesis, self, dot, move, comma, 100, right parenthesis. Line 23: Blank. Line 24, indented once: def distance, left parenthesis, self, comma, other, right parenthesis, colon. Line 25, indented twice: p 1, equals, self, dot, position, left parenthesis, right parenthesis. Line 26, indented twice: p 2, equals, other, dot, position, left parenthesis, right parenthesis. Line 27, indented twice: return math, dot, dist, left parenthesis, p 1, comma, p 2, right parenthesis. Line 28: Blank. Line 29, indented once: def remove, left parenthesis, self, right parenthesis, colon. Line 30, indented twice: self, dot, hide turtle, left parenthesis, right parenthesis.

Back to Listing

Program code. In the code, the words in the variables are merged. Line 1: class Drone Invaders, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, comma, x Min, comma, x Max, comma, y Min, comma, y Max, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, x Min, equals, x Min. Line 4, indented twice: self, dot, double underscore, x Max, equals, x Max. Line 5, indented twice: self, dot, double underscore, y Min, equals, y Min. Line 6, indented twice: self, dot, double

underscore, y Max, equals, y Max. Line 7: Blank. Line 8, indented once: def play, left parenthesis, self, right parenthesis, colon. Line 9, indented twice: self, dot, double underscore, main Win, equals, Laser Cannon, left parenthesis, self, dot, double underscore, x Min, comma, self, dot, double underscore, x Max, comma, self, dot, double underscore, y Min, comma, self, dot, double underscore, y Max, right parenthesis, dot, get screen, left parenthesis, right parenthesis. Line 11, indented twice: self, dot, double underscore, main Win, dot, b g color, left parenthesis, single quote, light green, single quote, right parenthesis. Line 12, indented twice: self, dot, double underscore, main Win, dot, set world coordinates, left parenthesis, self, dot, double underscore, x Min, comma, self, dot, double underscore, y Min, comma, self, dot, double underscore, x Max, comma, self, dot, double underscore, y Max, right parenthesis. Line 14, indented twice: self, dot, double underscore, main Win, dot, on timer, left parenthesis, self, dot, add Drone, comma, 1000, right parenthesis. Line 15, indented twice: self, dot, double underscore, main Win, dot, listen, left parenthesis, right parenthesis. Line 16, indented twice: main loop, left parenthesis, right parenthesis. Line 17: Blank. Line 18, indented once: def add Drone, left parenthesis, self, right parenthesis, colon. Line 19, indented twice: if len, left parenthesis, Drone, dot, get Drones, left parenthesis, right parenthesis, right parenthesis, less than, 7, colon. Line 20, indented thrice: Drone, left parenthesis, 1, comma, self, dot, double underscore, x Min, comma, self, dot, double underscore, x Max, comma, self, dot, double underscore, y Min, comma, self, dot, double underscore, y Max, right parenthesis. Line 22, indented twice: self, dot, double underscore, main Win, dot, on timer, left parenthesis, self, dot, add Drone, comma, 1000, right parenthesis.

Back to Listing

The window displays the following program code. In the code, the words in the variables are merged. Input 1: Line 1: game, equals, Drone Invaders, left parenthesis, negative 200, comma, 200, comma, 0, comma, 400, right parenthesis. Line 2: game, dot, play, left parenthesis, right parenthesis.

Back to Session

The code lines are as follows. Line 1: towards left parenthesis x comma y right parenthesis. Hash location as x and y. Line 2: towards left parenthesis left parenthesis x comma y right parenthesis right parenthesis, Hash location as a tuple. Line 3: towards left parenthesis other Turtle right parenthesis, Hash another turtle.

Back to Figure

The code lines are as follows. Line 1: distance left parenthesis x comma y right parenthesis. Hash to a location as x and y. Line 2: distance left parenthesis left parenthesis x comma y right parenthesis right parenthesis, Hash to a location as a tuple. Line 3: distance left parenthesis other Turtle right parenthesis, Hash to a another turtle.

Back to Figure

The code lines are as follows. Line 1: turtle dot begin underscore fill left parenthesis right parenthesis. Line 2: for i in range left parenthesis 4 right parenthesis colon. Line 3: turtle dot forward left parenthesis 100 right parenthesis. Line 4: turtle dot right left

parenthesis 90 right parenthesis. Line 5: turtle dot end underscore fill.

Back to Figure

The code lines are as follows. Line 1: turtle dot begin underscore poly left parenthesis right parenthesis. Line 2: for i in range left parenthesis 4 right parenthesis colon. Line 3: turtle dot forward left parenthesis 10 right parenthesis. Line 4: turtle dot right left parenthesis 90 right parenthesis. Line 5: turtle dot end underscore poly left parenthesis right parenthesis. Line 6: square Shape equals turtle dot get underscore poly left parenthesis right parenthesis. Line 7: turtle dot add shape left parenthesis single quote square pants single quote comma square Shape right parenthesis. Line 8: turtle dot shape left parenthesis single quote square pants single quote right parenthesis.

Back to Figure

The code lines are as follows. Line 1: b g color left parenthesis single quote color name single quote right parenthesis, Hash most reasonable color names are recognized. Line 2: b g color left parenthesis red comma blue comma green right parenthesis, Hash red comma blue comma and green values between 0.0 and 1.0. Line 3: b g color left parenthesis left parenthesis red comma green comma blue right parenthesis right parenthesis, Hash tuple with values between 0.0 and 1.0.

Back to Figure

The code lines are as follows. Line 1: pen color left parenthesis single quote color name single quote right parenthesis, Hash most reasonable color names are recognized. Line 2: pen color left

parenthesis red comma blue comma green right parenthesis, Hash red comma blue comma and green values between 0.0 and 1.0. Line 3: pen color left parenthesis left parenthesis red comma green comma blue right parenthesis right parenthesis, Hash tuple with values between 0.0 and 1.0.

Back to Figure

The code lines are as follows. Line 1: color left parenthesis pen Red comma pen Green comma pen Blue comma fill Red comma fill Green comma fill Blue right parenthesis. Line 2: color left parenthesis left parenthesis pen Red comma pen Green comma pen Blue right parenthesis comma left parenthesis fill Red comma fill Green comma fill Blue right parenthesis right parenthesis. Line 3: color left parenthesis single quote pen color name single quote comma single quote fill color name single quote right parenthesis.

Back to Figure

The code lines are as follows. Line 1: color mode left parenthesis 255 right parenthesis, Hash range is i n t 0 to 255. Line 2: color left parenthesis 200 comma 149 comma 58 right parenthesis. Line 3: color mode left parenthesis 1.0 right parenthesis, Hash range is float 0.0 to 1.0. Line 4: color left parenthesis point 75 comma point 69 comma point 02 right parenthesis.

Back to Figure

Program code. In the code, the words in the variables are merged. Line 1: def get Max So Far, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: max So Far, equals, a List, left square bracket, 0, right square bracket. Line 3, indented

once: for pos in range, left parenthesis, 1, comma, len, left parenthesis, a List, right parenthesis, right parenthesis, colon. Line 4, indented twice: if a List, left square bracket, pos, right square bracket, greater than, max So Far, colon. Line 5, indented thrice: max So Far, equals, a List, left square bracket, pos, right square bracket. Line 6: Blank. Line 7, indented once: return max So Far.

Back to Figure

The Basic control structures and all core data types using objects have chapters 1 to 5 which are as follows. 1. Introduction to Python, 2. Playing by the Rules, 3. Codes and Other Secrets, 4. Introducing the Python Collections, 5. Bigger Data File Input and Output. It splits into two. The left lists More detail and practice on core concepts and data structures in chapters 6 to 9 which are as follows. 6. Image Processing, 7. Data Mining, Cluster Analysis, 8. Cryptanalysis, 9. Fractals, The Geometry of Nature. The right lists the Object-oriented development, Building your own classes in chapters 10 to 13 which are as follows. 10. Planet Objects, 11. Simulation, 12. A Hierarchy of Shapes, 13. Interactive Games.

Back to Figure

The table lists the following text. and, as, assert, async, await, break, class, continue, def, del, elif, else, except, blank, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield.

Back to Table

Program code. In this code, the following variables are merged. Line 1: a, equals, double quotes, Don, single quote, say Hello, dot, double quotes. Line 2: b, equals, single quote, She said, double quotes, Hello World, exclamation mark, double quotes, single quote. Line 3: c, equals, double quotes, double quotes, double quotes, He said, double quotes, You don't say, exclamation mark, double quotes, didn't, he, double quotes, double quotes, double quotes.

Back to Code

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: empty List, equals, left square bracket, right square bracket. Line 2: my List, equals, left square bracket, 1, 2, 3.0, single quote, hello, single quote, 5, 6, right square bracket. Input 2: Line 1: new List, equals, list, left parenthesis, double quotes, hello, double quotes, right parenthesis. Line 2: new List. Output 2: left square bracket, single quote, h, single quote, single quote, e, single quote, single quote, l, single quote, single quote, l, single quote, single quote, o, single quote, right square bracket.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 2: Line 1: list, left parenthesis, range, left parenthesis, 10, right parenthesis, right parenthesis. Output 2: left square bracket, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, right square bracket. Input 3: Line 1: list, left parenthesis, range, left parenthesis, 1, 11, right parenthesis, right parenthesis. Output 3: left square bracket, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, right square bracket. Input 4: Line 1: list, left parenthesis, range, left

parenthesis, 10, 0, negative 1, right parenthesis, right parenthesis. Output 4: left square bracket, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, right square bracket.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: t, equals, left parenthesis, 1, 2, 3, right parenthesis. Line 2: t. Output 1: left parenthesis, 1, 2, 3, right parenthesis. Input 2: Line 1: u, equals, left parenthesis, single quote, a, single quote, single quote, b, single quote, right parenthesis. Line 2: u. Output 2: left parenthesis, single quote, a, single quote, single quote, b, single quote, right parenthesis. Input 3: Line 1: y, equals, tuple, left parenthesis, left square bracket, 1, 2, 3, right square bracket, right parenthesis. Line 2: y. Output 3: left parenthesis, 1, 2, 3, right parenthesis. Input 4: Line 1: z, equals, tuple, left parenthesis, single quote, a b c, single quote, right parenthesis. Line 2: z. Output 4: left parenthesis, single quote, a, single quote, single quote, b, single quote, single quote, c, single quote, right parenthesis.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: list, left parenthesis, zip, left parenthesis, single quote, hello, single quote, range, left parenthesis, len, left parenthesis, single quote, hello, single quote, right parenthesis, right parenthesis, right parenthesis, right parenthesis. Output 1: left square bracket, left parenthesis, single quote, h, single quote, 0, right parenthesis, left parenthesis, single quote, e, single quote, 1, right parenthesis, left

parenthesis, single quote, l, single quote, 2, right parenthesis, left parenthesis, single quote, l, single quote, 3, right parenthesis, left parenthesis, single quote, o, single quote, 4, right parenthesis, right square bracket. Input 2: Line 1: list, left parenthesis, zip, left parenthesis, left square bracket, 1, 2, 3, right square bracket, left square bracket, 4, 5, 6, right square bracket, right parenthesis, right parenthesis. Output 2: left square bracket, left parenthesis, 1, 4, right parenthesis, left parenthesis, 2, 5, right parenthesis, left parenthesis, 3, 6, right parenthesis, right square bracket. Input 3: Line 1: list, left parenthesis, zip, left parenthesis, single quote, a b c, single quote, single quote, d e f, single quote, single quote, g h i, single quote, right parenthesis, right parenthesis. Output 3: left square bracket, left parenthesis, single quote, a, single quote, single quote, d, single quote, single quote, g, single quote, right parenthesis, left parenthesis, single quote, b, single quote, single quote, e, single quote, single quote, h, single quote, right parenthesis, left parenthesis, single quote, c, single quote, single quote, f, single quote, single quote, i, single quote, right parenthesis, right square bracket.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 3: Line 1: empty Dict, equals, left brace, right brace. Line 2: Empty Dict. Output 3: left brace, right brace. Input 4: Line 1: my D, equals, left brace, single quote, a, single quote, colon, 123, single quote, b, single quote, colon, 789, single quote, c, single quote, colon, 246, right brace. Line 2: my D. Output 4: left brace, single quote, a, single quote, colon, 123, single quote, b, single quote, colon, 789, single quote, c, single quote, colon, 246, right brace. Input 5: Line 1: my D, left square bracket, single quote, d, single quote, right square bracket,

equals, 359. Line 2: my D. Output 5: left brace, single quote, a, single quote, colon, 123, single quote, b, single quote, colon, 789, single quote, c, single quote, colon, 246, single quote, d, single quote, colon, 359, right brace.

Back to Figure

Program code. In this code, the following variables are merged. Line 1: with open, left parenthesis, single quote, my File, dot, dat, single quote, single quote, r, single quote, right parenthesis, as f, colon. Line 2, indented once: line, equals, f, dot, read line, left parenthesis, right parenthesis. Line 3, indented once: while line, colon. Line 4, indented twice: print, left parenthesis, line, right parenthesis. Line 5, indented twice: line, equals, f, dot, read line, left parenthesis, right parenthesis. Line 6: hash, f is closed automatically.

Back to Code

Program code. In this code, the following variables are merged. Line 1: with open, left parenthesis, single quote, my File, dot, dat, single quote, right parenthesis, as f, colon. Line 2, indented once: for line in f, colon. Line 3, indented twice: print, left parenthesis, line, right parenthesis.

Back to Code

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: print, left parenthesis, double quotes, left brace, 0, right brace, plus, left brace, 1, right brace, equals, left brace, 2, right brace, double quotes, dot, format, left parenthesis, 5, 6, 11, right parenthesis, right parenthesis. The output displays the following. 5, plus, 6,

equals, 11. Input 2: Line 1: city, equals, double quotes, Orlando, double quotes. Line 2: rainfall, equals, 9.5. Line 3: print, left parenthesis, double quotes, left brace, 0, colon, greater than, 12 s, right brace, left brace, 1, colon, 5.1 f, right brace, inches, backslash, n, double quotes, dot, format, left parenthesis, city, rainfall, right parenthesis, right parenthesis. The output displays the following. Orlando 9.5 inches.

Back to Figure

Program code. In this code, the following keywords are merged. Line 1: for item in sequence, colon. Line 2, indented once: statement 1. Line 3, indented once: statement 2. Line 4, indented once: ellipses.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: for i in range, left parenthesis, n, right parenthesis, colon. Line 2, indented once: print, left parenthesis, i, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: for i in, left square bracket, single quote, a, single quote, 1, single quote, b, single quote, 2, single quote, c, single quote, 3, right square bracket, colon. Line 2, indented once: print, left parenthesis, i, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: for c h in, double quotes, hello world, double quotes,

colon. Line 2, indented once: print, left parenthesis, c h, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: for i in my Dictionary, colon. Line 2, indented once: print, left parenthesis, my Dictionary, left square bracket, i, right square bracket, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: for key, value in my Dictionary, dot, items, left parenthesis, right parenthesis, colon. Line 2, indented once: print, left parenthesis, key, value, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: while, left angle bracket, condition, right angle bracket, colon. Line 2, indented once: Statement 1. Line 3, indented once: statement 2. Line 4, indented once: ellipses. Line 5, indented once: loop Update statement.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: left square bracket, left angle bracket, expression, right angle bracket, for, left angle bracket, item 1, right angle bracket, in, left angle bracket, sequence 1, right angle bracket. Line 2, indented once: for, left angle bracket, item 2, right angle bracket, in, left angle bracket, sequence 2, right angle bracket. Line 3,

indented once: ellipses. Line 4, indented once: if, left angle bracket, condition, right angle bracket, right square bracket.

Back to Code

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: x, equals, left square bracket, 1, 2, 3, right square bracket. Line 2: y, equals, left square bracket, single quote, a, single quote, single quote, b, single quote, single quote, c, single quote, right square bracket. Input 2: Line 1: left square bracket, a, asterisk, a for a in x, right square bracket, hash, square each element in x. Output 2: left square bracket, 1, 4, 9, right square bracket. Input 3: Line 1: left square bracket, for a in x if a, percent, 2, not equal to, 0, right square bracket, hash, odd elements in x. Output 3: left square bracket, 1, 3, right square bracket. Input 4: Line 1: left square bracket, left parenthesis, a, b, right parenthesis, for a in x for b in y, right square bracket, hash, tuples with corresponding elements from x and y. Output 4: left square bracket, left parenthesis, 1, single quote, a, single quote, right parenthesis, left parenthesis, 1, single quote, b, single quote, right parenthesis, left parenthesis, 1, single quote, c, single quote, right parenthesis, left parenthesis, 2, single quote, a, single quote, right parenthesis, left parenthesis, 2, single quote, b, single quote, right parenthesis, left parenthesis, 2, single quote, c, single quote, right parenthesis, left parenthesis, 3, single quote, a, single quote, right parenthesis, left parenthesis, 3, single quote, b, single quote, right parenthesis, left parenthesis, 3, single quote, c, single quote, right parenthesis, right square bracket.

Back to Figure

Program code. In this code, the following keywords are merged. Line 1. if, left angle bracket, condition, right angle bracket, colon. Line 2, indented once: left angle bracket, statements, right angle bracket, hash, execute if condition is True. Line 3. else, colon. Line 4, indented once: left angle bracket, statements, right angle bracket, hash, execute if condition is False.

Back to Code

Program code. In this code, the following keywords are merged. Line 1. if, left angle bracket, condition, right angle bracket, colon. Line 2, indented once: left angle bracket, statements, right angle bracket, hash, execute if condition is True.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: if, left angle bracket, condition1, right angle bracket, colon. Line 2, indented once: left angle bracket, statements, right angle bracket, hash, condition1 is True. Line 3. elif, left angle bracket, condition 2, right angle bracket, colon. Line 4, indented once: left angle bracket, statements, right angle bracket, hash, condition 1 is False, semicolon, condition 2 is True. Line 5: ellipses. Line 6: elif, left angle bracket, condition N, right angle bracket, colon. Line 7, indented once: left angle bracket, statements, right angle bracket, hash, conditions 1, hyphen, N, minus, 1, are False, semicolon, condition N is True. Line 8: else, colon. Line 9, indented once: left angle bracket, statements, right angle bracket, hash, conditions 1, hyphen, N are False.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: match, left angle bracket, subject, right angle bracket, colon. Line 2, indented once: case value 1, colon. Line 3, indented twice: hash, statements to execute if subject, equals, equals, value 1. Line 4: case value 2, colon. Line 5, indented once: hash, statements to execute if subject, equals, equals, value 2. Line 6, indented once: ellipses. Line 7, indented once: case, underscore, colon, hash, optional. Line 8, indented twice: hash, statements to execute if subject does not match any case values.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def function Name, left parenthesis, param 1, param2, ellipses, right parenthesis, colon. Line 2, indented once: statement 1. Line 3, indented once: statement 2. Line 4, indented once: ellipses.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def Class Name, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, ellipses, right parenthesis, colon, hash, constructor. Line 3, indented twice: ellipses. Line 4, indented once: def method 1, left parenthesis, self, ellipses, right parenthesis, colon. Line 5, indented twice: ellipses. Line 6, indented once: def method 2, left parenthesis, self, ellipses, right parenthesis, colon. Line 7, indented twice: ellipses. Line 8, indented once: At, static method, hash, decorator. Line 9, indented once: def static method, left parenthesis, ellipses, right parenthesis, colon, hash, defines a static. Line 10, indented twice: ellipses, hash, method.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: class Class Name, left parenthesis, Super Class, right parenthesis, colon. Line 2, indented once: def, double underscore, init, double underscore, left parenthesis, self, ellipses, right parenthesis, colon, hash, constructor. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, init, double underscore, left parenthesis, right parenthesis, hash, call super class constructor. Line 4, indented twice: ellipses. Line 5, indented once: def method 1, left parenthesis, self, ellipses, right parenthesis, colon. Line 6, indented twice: ellipses. Line 7, indented once: def method 2, left parenthesis, self, ellipses, right parenthesis, colon.

Back to Code

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: def bar, left parenthesis, right parenthesis, colon. Line 2, indented once: return 2, asterisk, 5. Line 3: Blank. Input 2: bar, left parenthesis, right parenthesis. Output 2: 10. Input 3: Line 1: del bar, hash, delete the bar function. Input 4: Line 1: bar, left parenthesis, right parenthesis, hash, attempt to call deleted function. The output displays the following. Name Error, Traceback, left parenthesis, most recent call last, right parenthesis. Input 4 in, left angle bracket, cell line, colon, 1, right angle bracket, left parenthesis, right parenthesis. Line 1: 1 bar, left parenthesis, right parenthesis. Line 2: Blank. Line 3: Name Error, colon, name, single quote, bar, single quote, is not defined.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: import turtle. Input 2: Line 1: t, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 2: t, dot, forward, left parenthesis, 50, right parenthesis. Input 3: Line 1: del t, hash, delete the turtle. Input 4: Line 1: t, dot, forward, left parenthesis, 75, right parenthesis, hash, attempt to use deleted turtle. The output displays the following. Name Error, Traceback, left parenthesis, most recent call last, right parenthesis. Input 4 in, left angle bracket, cell line, colon, 1, right angle bracket, left parenthesis, right parenthesis. Line 1: t, dot, forward, left parenthesis, 75, right parenthesis. Line 2: Blank. Line 3: Name Error, colon, name, single quote, t, single quote, is not defined.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: my List, equals, left square bracket, single quote, a, single quote, single quote, b, single quote, single quote, c, single quote, single quote, d, single quote, right square bracket. Line 2: my List. Output 1: left square bracket, single quote, a, single quote, single quote, b, single quote, single quote, c, single quote, single quote, d, single quote, right square bracket. Input 2: Line 1: del my List, left square bracket, 2, right square bracket, hash, delete the item at index 2. Input 3: Line 1: my List. Output 3: left square bracket, single quote, a, single quote, single quote, b, single quote, single quote, d, single quote, right square bracket.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: my Dict, equals, left brace, single quote, a, single quote, colon, 123, single quote, b, single quote, colon, 456, single quote, c, single quote, colon, 789, right brace. Line 2: my Dict. Output 1: left brace, single quote, a, single quote, colon, 123, single quote, b, single quote, colon, 456, single quote, c, single quote, colon, 789, right brace. Input 2: Line 1: del my Dict, left square bracket, single quote, b, single quote, right square bracket, hash, delete the item with key b. Input 3: Line 1: my Dict. Output 3: left brace, single quote, a, single quote, colon, 123, single quote, c, single quote, colon, 789, right brace.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: def foo, left parenthesis, x, y, right parenthesis. Line 2: Blank. The output displays the following. Input 1. Line 1, indented once: def foo, left parenthesis, x, y, right parenthesis. Line 2: caret. Line 3: Syntax Error, colon, expected, single quote, colon, single quote.

Back to Code

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: 4, plus, spam, asterisk, 3. The output displays the following. Name Error, Traceback, left parenthesis, most recent call last, right parenthesis. Input 1 in, left angle bracket, cell line, colon, 1, right angle bracket, left parenthesis, right parenthesis. Line 1: 4, plus, spam, asterisk, 3. Line 2: Blank. Line 3: Name Error, colon, name, single quote, spam, single quote, is not defined.

Back to Figure

Program code. In this code, the following variables are merged. Line 1: def foo, left parenthesis, right parenthesis, colon. Line 2, indented once: a, equals, 1, plus, 2. Line 3, indented once: b, equals, a, plus, spam. Line 4, indented once: return b. Line 5: Blank. Line 6: def main, left parenthesis, right parenthesis, colon. Line 7, indented once: print, left parenthesis, single quote, hello, single quote, right parenthesis. Line 8, indented once: x, equals, foo, left parenthesis, right parenthesis. Line 9, indented once: print, left parenthesis, x, right parenthesis. Line 10: Blank. Line 11. main, left parenthesis, right parenthesis.

Back to Figure

The output reads the following. Name Error Traceback, left parenthesis, most recent call last, right parenthesis. Input In, left square bracket, 1, right square bracket, in, left angle bracket, cell line, colon, 11, right angle bracket, left parenthesis, right parenthesis. Line 8, indented once: x, equals, foo, left parenthesis, right parenthesis. Line 9, indented once: print, left parenthesis, x, right parenthesis. Line 11: main, left parenthesis, right parenthesis. Input In, left square bracket, 1, right square bracket, in main, left parenthesis, right parenthesis. Line 6: def main, left parenthesis, right parenthesis, colon. Line 7, indented once: print, left parenthesis, single quote, hello, single quote, right parenthesis. Line 8, indented once: x, equals, foo, left parenthesis, right parenthesis. Line 9, indented once: print, left parenthesis, x, right parenthesis. Input In, left square bracket, 1, right square bracket, in foo, left parenthesis, right parenthesis. Line 1: def foo, left parenthesis, right parenthesis, colon. Line 2, indented once: a, equals, 1, plus, 2. Line 3, indented once: b,

equals, a, plus, spam. Line 4, indented once: return b. Name Error, colon, name, single quote, spam, single quote, is not defined. Arrows point to lines 11, 8, and 3, respectively.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: single quote, 2, single quote, plus, 2. The output displays the following. Type Error, Traceback, left parenthesis, most recent call last, right parenthesis. Input 1 in, left angle bracket, cell line, colon, 1, right angle bracket, left parenthesis, right parenthesis. Line 1: single quote, 2, single quote, plus, 2. Line 2: Blank. Line 3: Type Error, colon, can only concatenate s t r, left parenthesis, not, double quotes, i n t, double quotes, right parenthesis, to s t r.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: def bar, left parenthesis, right parenthesis, colon. Line 2, indented once: return 2, asterisk, 5. Input 2: Line 1: a, equals, 3, plus, bar. The output displays the following. Type Error, Traceback, left parenthesis, most recent call last, right parenthesis. Input 2 in, left angle bracket, cell line, colon, 1, right angle bracket, left parenthesis, right parenthesis. Line 1: a, equals, 3, plus, bar. Line 2: Blank. Line 3: Type Error, colon, unsupported operand type, left parenthesis, s, right parenthesis, for, plus, colon, single quote, i n t, single quote, and, single quote, function, single quote.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: b, equals, 1 Line 2: c, equals, 2, plus, b, left parenthesis, right parenthesis. The output displays the following. Type Error, Traceback, left parenthesis, most recent call last, right parenthesis. Input 1 in, left angle bracket, cell line, colon, 2, right angle bracket, left parenthesis, right parenthesis. Line 1: b, equals, 1. Line 2: c, equals, 2, plus, b, left parenthesis, right parenthesis. Line 3: Blank. Line 3: Type Error, colon, single quote, i n t, single quote, object is not callable.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: 10, asterisk, left parenthesis, 1, forward slash, 0, right parenthesis. The output displays the following. Zero Division Error, Traceback, left parenthesis, most recent call last, right parenthesis. Input 1 in, left angle bracket, cell line, colon, 1, right angle bracket, left parenthesis, right parenthesis. Line 1: 10, asterisk, left parenthesis, 1, forward slash, 0, right parenthesis. Line 2: Blank. Line 3: Zero Division Error, colon, division by zero.

Back to Figure

The window displays the following program code. In this code, the following variables are merged. Input 1: Line 1: d, equals, 1 Line 2: for i in d, colon. Line 3, indented once: print, left parenthesis, i, right parenthesis. The output displays the following. Type Error, Traceback, left parenthesis, most recent call last, right parenthesis. Input 1 in, left angle bracket, cell line, colon, 2, right angle bracket, left parenthesis, right parenthesis.

Line 1: d, equals, 1. Line 2: for i in d, colon. Line 3, indented once: print, left parenthesis, i, right parenthesis. Line 4: Blank. Line 5: Type Error, colon, single quote, i n t, single quote, object is not iterable.

Back to Figure

The expression reads the following. Line 1: left parenthesis, 17, asterisk, left parenthesis, 17, plus, 1, right parenthesis, right parenthesis, forward slash, forward slash, 2, hash, assuming that you have 18 classmates. Line 2: hash, 153.

Back to Code

The expression reads the following. Line 1: left parenthesis, 18, plus, 19, plus, 19, plus, 20, plus, 25, right parenthesis, forward slash, 5. Line 2: hash, 20, dot, 2.

Back to Code

Part A. Two boxes labeled and b are on the left and arranged one below the other. A numeric 79 is at the right. Two arrows from a and b points to 79. Part B. Two boxes labeled and b are on the left and arranged one below the other. A numeric 79 is at the top right parallel to a. Another numeric 89 is at the bottom right parallel to b. An arrow from a points to 89. Another arrow from b points to 79.

Back to Figure

Program code. In this code, the following keywords are merged. Line 1: ole, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 2: ole, dot, right, left parenthesis, 45, right

parenthesis. Line 3: ole, dot, forward, left parenthesis, 50, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def draw Rectangle, left parenthesis, my Turtle, width, height, right parenthesis, colon. Line 2, indented once: my Turtle, dot, forward, left parenthesis, width, right parenthesis. Line 3, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 4, indented once: my Turtle, dot, forward, left parenthesis, height, right parenthesis. Line 5, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 6, indented once: my Turtle, dot, forward, left parenthesis, width, right parenthesis. Line 7, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 8, indented once: my Turtle, dot, forward, left parenthesis, height, right parenthesis. Line 9, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis.

Back to Code

The code reads the following. Line 1. range, left parenthesis, negative 10, 11, 1, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def draw Spiral, left parenthesis, turtle 1, turtle 2, max Side, right parenthesis, colon. Line 2, indented once: for side Length in range, left parenthesis, 1, max Side, plus, 1, 5, right parenthesis, colon. Line 3, indented twice: turtle 1, dot, forward, left parenthesis, side Length, right parenthesis. Line 4, indented

twice: turtle 2, dot, forward, left parenthesis, side Length, right parenthesis. Line 5, indented twice: turtle 1, dot, right, left parenthesis, 140, right parenthesis. Line 6, indented twice: turtle 2, dot, right, left parenthesis, 140, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def draw Square, left parenthesis, my Turtle, side Length, right parenthesis, colon. Line 2, indented once: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis. Line 3, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 4, indented once: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis. Line 5, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 6, indented once: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis. Line 7, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 8, indented once: my Turtle, dot, forward, left parenthesis, side Length, right parenthesis. Line 9, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 10: Blank. Line 11: def draw Nested Squares, left parenthesis, my Turtle, right parenthesis, colon. Line 12, indented once: for i in range, left parenthesis, 200, left parenthesis, 200, minus, 5, asterisk, 10, right parenthesis, negative 5, right parenthesis, colon. Line 13, indented twice: draw Square, left parenthesis, my Turtle, i, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def plot Circle, left parenthesis, my Turtle, radius, right

parenthesis, colon. Line 2, indented once: circumference, equals, 2, asterisk, 3.1415, asterisk, radius. Line 3, indented once: side Length, equals, circumference, forward slash, 360. Line 4, indented once: my Turtle, dot, up, left parenthesis, right parenthesis. Line 5, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 6, indented once: my Turtle, dot, forward, left parenthesis, radius, right parenthesis. Line 7, indented once: my Turtle, dot, right, left parenthesis, 90, right parenthesis. Line 8, indented once: my Turtle, dot, down, left parenthesis, right parenthesis. Line 9, indented once: draw Polygon, left parenthesis, my Turtle, side Length, 360, right parenthesis. Line 10, indented once: my Turtle, dot, up, left parenthesis, right parenthesis. Line 11, indented once: my Turtle, dot, right, left parenthesis, 90, right parenthesis. Line 12, indented once: my Turtle, dot, forward, left parenthesis, radius, right parenthesis. Line 13, indented once: my Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 14, indented once: my Turtle, dot, down, left parenthesis, right parenthesis.

Back to Code

Line 1 reads: if result equals equals 100 colon. Line 2 reads: answer equals 1. Line 3 reads: else colon. Line 4 reads: answer equals 2.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: if score, greater than, equals, 90, colon. Line 2, indented once: grade Point, equals, 4. Line 3. elif score, greater than, equals, 80, colon. Line 4, indented once: grade Point, equals, 3 Line 5. elif score, greater than, equals, 70, colon. Line 6, indented

once: grade Point, equals, 2. Line 7: elif score, greater than, equals, 60, colon. Line 8, indented once: grade Point, equals, 1. Line 9: else. Line 10, indented once: grade Point, equals, 0.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def cost Of Oranges, left parenthesis, pounds, right parenthesis, colon. Line 2, indented once: cost Fruit, equals, pounds, asterisk, 0.32. Line 3, indented once: shipping, equals, 7.50. Line 4, indented once: if pounds, greater than, 100, colon. Line 5, indented twice: shipping, equals, shipping, minus, 1.50. Line 6, indented once: total Cost, equals, cost Fruit, plus, shipping. Line 7, indented once: return total Cost.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def paycheck, left parenthesis, rate, hours, right parenthesis, colon. Line 2, indented once: if hours, less than, equals, 40, colon. Line 3, indented twice: total pay, equals, rate, asterisk, hours. Line 4, indented once: else. Line 5, indented twice: total pay, equals, rate, asterisk, 40, plus, left parenthesis, 1.5, asterisk, rate, asterisk, left parenthesis, hours, minus, 40, right parenthesis, right parenthesis. Line 6, indented once: return total pay.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def my chatbot, left parenthesis, question, right parenthesis, colon. Line 2, indented once: question, equals, question, dot, lower, left parenthesis, right parenthesis. Line 3:

Blank. Line 4, indented once: if question, dot, starts with, left parenthesis, double quotes, who, double quotes, right parenthesis, or, double quotes, name, double quotes, in question, colon. Line 5, indented twice: print, left parenthesis, double quotes, The tooth fairy, double quotes, right parenthesis. Line 6, indented once: elif question, dot, starts with, left parenthesis, double quotes, what, double quotes, right parenthesis, colon. Line 7, indented twice: print, left parenthesis, double quotes, To be kind to one another, double quotes, right parenthesis. Line 8, indented once: elif question, dot, starts with, left parenthesis, double quotes, how, double quotes, right parenthesis, and, double quotes, feeling, double quotes, in question, colon. Line 9, indented twice: print, left parenthesis, double quotes, I don't feel anything, dot, I'm a computer, exclamation mark, double quotes, right parenthesis. Line 10, indented once: elif question, dot, starts with, left parenthesis, double quotes, where, double quotes, right parenthesis, and, double quotes, world, double quotes, in question, colon. Line 11, indented twice: print, left parenthesis, double quotes, The south pole, double quotes, right parenthesis. Line 12, indented once: elif question, dot, starts with, left parenthesis, double quotes, why, double quotes, right parenthesis, colon. Line 13, indented twice: print, left parenthesis, double quotes, You'll understand when you're older, dot, double quotes, right parenthesis. Line 14, indented once: elif question, dot, starts with, left parenthesis, double quotes, when, double quotes, right parenthesis, colon. Line 15, indented twice: print, left parenthesis, double quotes, This afternoon, double quotes, right parenthesis. Line 16, indented once: else, colon. Line 17, indented twice: print, left parenthesis, double quotes, You stumped me on that, exclamation mark, double quotes, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f ask my chatbot, left parenthesis, right parenthesis, colon. Line 2, indented once: question, equals, input, left parenthesis, double quotes, what is your question, question mark, double quotes, right parenthesis. Line 3, indented once: question, equals, question, dot, lower, left parenthesis, right parenthesis. Line 4: Blank. Line 5, indented once: if question, dot, start with, left parenthesis, double quotes, who, double quotes, right parenthesis, or, double quotes, name, double quotes, in question, colon. Line 6, indented twice: print, left parenthesis, double quotes, The tooth fairy, double quotes, right parenthesis. Line 7, indented once: elif question, dot, starts with, left parenthesis, double quotes, what, double quotes, right parenthesis, colon. Line 8, indented twice: print, left parenthesis, double quotes, To be kind to one another, double quotes, right parenthesis. Line 9, indented once: elif question, dot, starts with, left parenthesis, double quotes, how, double quotes, right parenthesis, and, double quotes, feeling, double quotes, in question, colon. Line 10, indented twice: print, left parenthesis, double quotes, I don't feel anything, dot, I'm a computer, exclamation mark, double quotes, right parenthesis. Line 11, indented once: elif question, dot, starts with, left parenthesis, double quotes, where, double quotes, right parenthesis, colon. Line 12, indented twice: print, left parenthesis, double quotes, At the dentist, double quotes, right parenthesis. Line 13, indented once: elif question, dot, starts with, left parenthesis, double quotes, why, double quotes, right parenthesis, colon. Line 14, indented twice: print, left parenthesis, double quotes, You'll understand when you're older, dot, double quotes, right parenthesis. Line 15, indented once: elif question, dot, starts with,

left parenthesis, double quotes, when, double quotes, right parenthesis, colon. Line 16, indented twice: print, left parenthesis, double quotes, This afternoon, double quotes, right parenthesis. Line 17, indented once: else, colon. Line 18, indented twice: print, left parenthesis, double quotes, You stumped me on that, exclamation mark, double quotes, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f validate Hashtag, left parenthesis, input Text, right parenthesis, colon. Line 2, indented once: if input Text, dot, starts with, left parenthesis, double quotes, hash, double quotes, right parenthesis, and, double quotes, double quotes, not in input Text, colon. Line 3, indented twice: is Hashtag, equals, True. Line 4, indented once: else, colon. Line 5, indented twice: is Hashtag, equals, False. Line 6, indented once: return is Hashtag.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f sum Of Even Numbers, left parenthesis, right parenthesis, colon. Line 2, indented once: total Sum, equals, 0 Line 3, indented once: count, equals, 0. Line 4, indented once: n u m, equals, 0, hash, mathematically, is considered an even number. Line 5, indented once: while count, left angle bracket, 100, colon. Line 6, indented twice: if n u m, percent, 2, equals, equals, 0, colon. Line 7, indented thrice: total Sum, equals, total Sum, plus, num. Line 8, indented thrice: count, equals, count, plus, 1. Line 9, indented twice: n u m, equals, n u m, plus, 1. Line 10, indented once: return total Sum.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f a v g Of Odd Numbers, left parenthesis, right parenthesis, colon. Line 2, indented once: total Sum, equals, 0. Line 3, indented once: count, equals, 0. Line 4, indented once: n u m, equals, 1. Line 5, indented once: while count, left angle bracket, 100, colon. Line 6, indented twice: if n u m, percent, 2, exclamation mark, equals, 0, colon. Line 7, indented thrice: total Sum, equals, total Sum, plus, num. Line 8, indented thrice: count, equals, count, plus, 1. Line 9, indented twice: n u m, equals, n u m, plus, 1. Line 10, indented once: return total Sum count.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: round, left parenthesis, math, dot, log, left parenthesis, 50, right parenthesis, right parenthesis. Line 2: round, left parenthesis, math, dot, log, left parenthesis, 100, right parenthesis, right parenthesis. Line 3: round, left parenthesis, math, dot, log, left parenthesis, 300, right parenthesis, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: import random. Line 2: def flip Coin, left parenthesis, right parenthesis, colon. Line 3, indented once: side, equals, random, dot, rand int, left parenthesis, 0, 1, right parenthesis. Line 4, indented once: if side, equals, equals, 0, colon. Line 5, indented twice: print, left parenthesis, double quotes, Heads, double quotes, right parenthesis. Line 6, indented once: else,

colon. Line 7, indented twice: print, left parenthesis, double quotes, Tails, double quotes, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f guess A Number, left parenthesis, start, end, right parenthesis, colon. Line 2: Blank. Line 3, indented once: print, left parenthesis, double quotes, Pick a number between, double quotes, plus, s t r, left parenthesis, start, right parenthesis, plus, double quotes, and, double quotes, plus, s t r, left parenthesis, end, right parenthesis, plus, double quotes, dot, double quotes, right parenthesis. Line 4, indented once: print, left parenthesis, double quotes, I'll try to guess it in as few guesses as possible, dot, double quotes, right parenthesis. Line 5. Blank. Line 6, indented once: guess, equals, left parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 7, indented once: n u m Attempts, equals, 1. Line 8, indented once: response, equals, input, left parenthesis, double quotes, Is, double quotes, plus, s t r, left parenthesis, guess, right parenthesis, plus, double quotes, too high, too low, or correct, question mark, double quotes, right parenthesis. Line 9, indented once: response, equals, response, dot, lower, left parenthesis, right parenthesis. Line 10: Blank. Line 11, indented once: while response, not equal to, single quote, correct, single quote, colon. Line 12, indented twice: if response, equals, equals, single quote, too high, single quote, colon. Line 13, indented thrice: end, equals, guess, minus, 1. Line 14, indented twice: elif response, equals, equals, single quote, too low, single quote, colon. Line 15, indented thrice: start, equals, guess, plus, 1. Line 16, indented twice: else, colon, hash, invalid response. Line 17, indented thrice: response, equals, input, left parenthesis, double quotes,

Please enter, single quote, too high, single quote, single quote, too low, single quote, or, single quote, correct, single quote, double quotes, right parenthesis. Line 18, indented thrice: response, equals, response, dot, lower, left parenthesis, right parenthesis. Line 19, indented thrice: continue, hash, skip the remainder of the loop. Line 20: Blank. Line 21, indented twice: guess, equals, left parenthesis, end, plus, start, right parenthesis, forward slash, forward slash, 2. Line 22, indented twice: n u m Attempts, equals, n u m Attempts, plus, 1. Line 23, indented twice: response, equals, input, left parenthesis, double quotes, Is, double quotes, plus, s t r, left parenthesis, guess, right parenthesis, plus, double quotes, too high, too low, or correct, question mark, double quotes, right parenthesis. Line 24, indented twice: response, equals, response, dot, lower, left parenthesis, right parenthesis. Line 25: Blank. Line 26, indented once: print, left parenthesis, double quotes, Your number is, double quotes, plus, s t r, left parenthesis, guess, right parenthesis, right parenthesis. Line 27, indented once: print, left parenthesis, double quotes, guessed it in, double quotes, plus, s t r, left parenthesis, n u m Attempts, right parenthesis, plus, double quotes, trys, dot, double quotes, right parenthesis

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f n i m 2, left parenthesis, pile, right parenthesis, colon. Line 2, indented once: if pile, less than, 5, colon. Line 3, indented twice: print, left parenthesis, double quotes, The pile must contain at least 5 stones, dot, double quotes, right parenthesis. Line 4, indented once: else, colon. Line 5, indented twice: print, left parenthesis, double quotes, The pile has, double quotes, plus, s t r, left parenthesis, pile, right

parenthesis, plus, double quotes, stones, dot, double quotes, right parenthesis. Line 6, indented twice: winner, equals, double quotes, double quotes. Line 7, indented twice: while pile, not equal to, 0, colon. Line 8, indented thrice: c 1 Move, equals, random Move, left parenthesis, pile, right parenthesis. Line 9, indented thrice: pile, equals, pile, minus, c 1 Move. Line 10, indented thrice: print, left parenthesis, double quotes, Random Randy takes, double quotes, plus, s t r, left parenthesis, c1Move, right parenthesis, plus, double quotes, stone, left parenthesis, s, right parenthesis, dot, double quotes, right parenthesis. Line 11, indented thrice: print, left parenthesis, double quotes, Pile is now, double quotes, plus, s t r, left parenthesis, pile, right parenthesis, right parenthesis. Line 12, indented thrice: if pile, equals, equals, 0, colon. Line 13, indented quarce: winner, equals, double quotes, Random Randy, double quotes. Line 14, indented thrice: else, colon. Line 15, indented quarce: c 2 Move, equals, nimble Move, left parenthesis, pile, right parenthesis. Line 16, indented quarce: pile, equals, pile, minus, c 2 Move. Line 17, indented quarce: print, left parenthesis, double quotes, Nimble Nancy takes, double quotes, plus, s t r, left parenthesis, c 2 Move, right parenthesis, plus, double quotes, stone, left parenthesis, s, right parenthesis, dot, double quotes, right parenthesis. Line 18, indented quarce: print, left parenthesis, double quotes, Pile is now, double quotes, plus, s t r, left parenthesis, pile, right parenthesis, right parenthesis. Line 19, indented quarce: if pile, equals, equals, 0, colon. Line 20, indented quince: winner, equals, double quotes, Nimble Nancy, double quotes. Line 21, indented twice: print, left parenthesis, double quotes, The winner is, double quotes, plus, winner, plus, double quotes, exclamation mark, double quotes, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: for i in range, left parenthesis, l e n, left parenthesis, name, right parenthesis, plus, 1, right parenthesis, colon. Line 2, indented once: print, left parenthesis, name, left square bracket, colon, i, right square bracket, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: river, equals, double quotes, mississippi, double quotes. Line 2: river, dot, find, left parenthesis, double quotes, p, double quotes, right parenthesis. Line 3: river, dot, index, left parenthesis, double quotes, p, double quotes, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f to I n t, left parenthesis, c h, right parenthesis, colon. Line 2, indented once: if ord, left parenthesis, c h, right parenthesis, greater than, ord, left parenthesis, single quote, 9, single quote, right parenthesis, or ord, left parenthesis, c h, right parenthesis, less than, ord, left parenthesis, single quote, 0, single quote, right parenthesis, colon. Line 3, indented twice: print, left parenthesis, single quote, error, colon, single quote, c h, single quote, is not a digit, single quote, right parenthesis. Line 4, indented twice: return negative 1. Line 5, indented once: else, colon. Line 6, indented twice: return ord, left parenthesis, c h, right parenthesis, minus, ord, left parenthesis, single quote, 0, single quote, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f letter Grade, left parenthesis, score, right parenthesis, colon. Line 2, indented once: if score, greater than, equals, 90, colon. Line 3, indented twice: letter, equals, single quote, A, single quote. Line 4, indented once: elif score, greater than, equals, 80, colon. Line 5, indented twice: letter, equals, single quote, B, single quote. Line 6, indented once: elif score, greater than, equals, 70, colon. Line 7, indented twice: letter, equals, single quote, C, single quote. Line 8, indented once: elif score, greater than, equals, 60, colon. Line 9, indented twice: letter, equals, single quote, D, single quote. Line 10, indented once: else, colon. Line 11, indented twice: letter, equals, single quote, F, single quote. Line 12, indented once: return letter.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: def remove Matches, left parenthesis, my String, remove String, right parenthesis, colon. Line 2, indented once: new S t r, equals, double quotes, double quotes. Line 3, indented once: for c h in my String, colon. Line 4, indented twice: if c h not in remove String, colon. Line 5, indented thrice: new S t r, equals, new S t r, plus, c h. Line 6, indented once: return new S t r.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f substitution Encrypt No Spaces, left parenthesis, plain Text, key, right parenthesis, colon. Line 2, indented once: alphabet, equals, double quotes, a b c d e f g h i j k l m n o p q r s t u v w x y z, double quotes. Line 3, indented

once: plain Text, equals, plain Text, dot, lower, left parenthesis, right parenthesis. Line 4, indented once: plain Text, equals, plain text, dot, replace, left parenthesis, double quotes, double quotes, double quotes, double quotes, right parenthesis. Line 5, indented once: cipher Text, equals, double quotes, double quotes. Line 6, indented once: for c h in plain Text, colon. Line 7, indented twice: i d x, equals, alphabet, dot, find, left parenthesis, c h, right parenthesis. Line 8, indented twice: cipher Text, equals, cipher Text, plus, key, left square bracket, i d x, right square bracket. Line 9, indented once: return cipher Text.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f substitution Encrypt, left parenthesis, plain Text, password, right parenthesis, colon. Line 2, indented once: key, equals, gen Key From Pass, left parenthesis, password, right parenthesis. Line 3, indented once: alphabet, equals, double quotes, a b c d e f g h i j k l m n o p q r s t u v w x y z, double quotes. Line 4, indented once: plain Text, equals, plain Text, dot, lower, left parenthesis, right parenthesis. Line 5, indented once: cipher Text, equals, double quotes, double quotes. Line 6, indented once: for c h in plain Text, colon. Line 7, indented twice: i d x, equals, letter To Index, left parenthesis, c h, right parenthesis. Line 8, indented twice: cipher Text, equals, cipher Text, plus, key, left square bracket, i d x, right square bracket. Line 9, indented once: return cipher Text.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f rot Encrypt, left parenthesis, message, rot N

u m, right parenthesis, colon. Line 2, indented once: alphabet, equals, double quotes, a b c d e f g h i j k l m n o p q r s t u v w x y z, double quotes. Line 3, indented once: result, equals, double quotes, double quotes. Line 4, indented once: for c h in message, colon. Line 5, indented twice: i d x, equals, alphabet, dot, find, left parenthesis, c h, right parenthesis. Line 6, indented twice: i d x, equals, left parenthesis, i d x, plus, rot N u m, right parenthesis, percent, l e n, left parenthesis, alphabet, right parenthesis. Line 7, indented once: result, equals, result, plus, alphabet, left square bracket, i d x, right square bracket. Line 8: return result. Line 9: Blank. Line 10: d e f rot Decrypt, left parenthesis, cyber Text, rot N u m, right parenthesis, colon. Line 11, indented once: return rot Encrypt, left parenthesis, cyber Text, negative rot N u m, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f undo V i g, left parenthesis, key Letter, c t Letter, right parenthesis, colon. Line 2, indented once: key Index, equals, letter To Index, left parenthesis, key Letter, right parenthesis. Line 3, indented once: c t Index, equals, letter To Index, left parenthesis, c t Letter, right parenthesis. Line 4, indented once: new i d x, equals, left parenthesis, c t Index, minus, key Index, right parenthesis, percent, 26. Line 5, indented once: return index To Letter, left parenthesis, new i d x, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: my List, equals, left square bracket, 7, 9, double quotes, a,

double quotes, double quotes, cat, double quotes, False, right square bracket. Line 2: Blank. Line 3: my List, dot, append, left parenthesis, 3.14, right parenthesis, hash, left parenthesis, a, right parenthesis. Line 4: my List, dot, append, left parenthesis, 7, right parenthesis. Line 5: my List. Line 6: Blank. Line 7: my List, dot, insert, left parenthesis, 3, double quotes, dog, double quotes, right parenthesis, hash, left parenthesis, b, right parenthesis. Line 8: my List. Line 9: Blank. Line 10. my List, dot, index, left parenthesis, double quotes, cat, double quotes, right parenthesis, hash, left parenthesis, c, right parenthesis. Line 11: Blank. Line 12: my List, dot, count, left parenthesis, 7, right parenthesis, hash, left parenthesis, d, right parenthesis. Line 13: Blank. Line 14: my List, dot, remove, left parenthesis, 7, right parenthesis, hash, left parenthesis, e, right parenthesis. Line 15: my List. Line 16: Blank. Line 17: my List, dot, pop, left parenthesis, my List, dot, index, left parenthesis, double quotes, dog, double quotes, right parenthesis, right parenthesis, hash, left parenthesis, f, right parenthesis. Line 18: my List.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: def word Count, left parenthesis, sentence, right parenthesis, colon. Line 2, indented once: word List, equals, sentence, dot, split, left parenthesis, right parenthesis. Line 3, indented once: return len, left parenthesis, word List, right parenthesis.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: import random. Line 2: def shuffle, left parenthesis, a

List, right parenthesis, colon. Line 3, indented once: copy List, equals, a List, left square bracket, colon, right square bracket. Line 4, indented once: new List, equals, left square bracket, right square bracket. Line 5, indented once: for i in range, left parenthesis, len, left parenthesis, copy List, right parenthesis, right parenthesis, colon. Line 6, indented twice: item, equals, copy List, dot, pop, left parenthesis, random, dot, rand range, left parenthesis, len, left parenthesis, copy List, right parenthesis, right parenthesis, right parenthesis. Line 7, indented twice: new List, dot, append, left parenthesis, item, right parenthesis. Line 8, indented once: return new List.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: my List, equals, left square bracket, left square bracket, right square bracket, right square bracket, asterisk. Line 2: my List. Line 3. The statement creates a list containing three copies of an empty list.

Back to Code

Program code. In this code, the following keywords are merged. Line 1: my List, equals, left square bracket, right square bracket. Line 2: for i in range, left parenthesis, 3, right parenthesis, colon. Line 3, indented once: my List, dot, append, left parenthesis, left square bracket, right square bracket, right parenthesis. Line 4: Blank. Line 5: my List. Line 6: Blank. Line 7: my List, left square bracket, 1, right square bracket, dot, append, left parenthesis, 2, right parenthesis. Line 8: my List.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f get Min, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: min So Far, equals, a List, left square bracket, 0, right square bracket. Line 3, indented once: for item in a List, left square bracket, 1, colon, right square bracket, colon. Line 4, indented twice: if item, left angle bracket, min So Far, colon. Line 5, indented thrice: min So Far, equals, item. Line 6, indented once: return min So Far.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f mean, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: sum Of Items, equals, 0. Line 3, indented once: for item in a List, colon. Line 4, indented twice: sum Of Items, equals, sum Of Items, plus, item. Line 5, indented once: mean, equals, sum Of Items, forward slash, l e n, left parenthesis, a List, right parenthesis. Line 6, indented once: return mean.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f make Dictionary, left parenthesis, list 1, list 2, right parenthesis, colon. Line 2, indented once: new Dict, equals, left brace, right brace. Line 3, indented once: for i in range, left parenthesis, l e n, left parenthesis, list 1, right parenthesis, right parenthesis, colon. Line 4, indented twice: new Dict, left square bracket, list 1, left square bracket, i, right square bracket, right square bracket, equals, list 2, left square bracket, i, right square bracket. Line 5, indented once: return new Dict. Line

6: score Dict, equals, make Dictionary, left parenthesis, names, scores, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: scores List, equals, list, left parenthesis, score Dict, dot, values, left parenthesis, right parenthesis, right parenthesis. Line 2: scores List, dot, sort, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: score Dict, equals, left brace, single quote, joe, single quote, colon, 10, single quote, barb, single quote, colon, 13, single quote, sue, single quote, colon, 18, single quote, sally, single quote, colon, 13, single quote, john, single quote, colon, 19, right brace. Line 2: keys, equals, score Dict, dot, keys, left parenthesis, right parenthesis. Line 3: keys, equals, list, left parenthesis, keys, right parenthesis. Line 4: keys. Line 5: hash, left square bracket, single quote, joe, single quote, single quote, barb, single quote, single quote, sue, single quote, single quote, sally, single quote, single quote, john, single quote, right square bracket. Line 6: keys, dot, sort, left parenthesis, right parenthesis. Line 7: keys. Line 8: hash, left square bracket, single quote, barb, single quote, single quote, joe, single quote, single quote, john, single quote, single quote, sally, single quote, single quote, sue, single quote, right square bracket. Line 9: for a Name in keys, colon. Line 10, indented once: print, left parenthesis, a Name, score Dict, left square bracket, a Name, right square bracket, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f mode, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: max Value, equals, max, left parenthesis, a List, right parenthesis. Line 3, indented once: count List, equals, left square bracket, 0, right square bracket, asterisk, left parenthesis, max Value, plus, 1, right parenthesis. Line 4: Blank. Line 5, indented once: for item in a List, colon. Line 6, indented twice: count List, left square bracket, item, right square bracket, equals, count List, left square bracket, item, right square bracket, plus, 1. Line 7, indented once: max Count, equals, max, left parenthesis, count List, right parenthesis. Line 8, indented once: mode List, equals, left square bracket, right square bracket. Line 9, indented once: for i in range, left parenthesis, l e n, left parenthesis, count List, right parenthesis, right parenthesis, colon. Line 10, indented twice: if count List, left square bracket, i, right square bracket, equals, equals, max Count, colon. Line 11, indented thrice: mode List, dot, append, left parenthesis, i, right parenthesis. Line 12, indented once: return mode List.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f averages, left parenthesis, score List, right parenthesis, colon. Line 2, indented once: data Dict, equals, left brace, right brace. Line 3, indented once: for item in score List, colon. Line 4, indented twice: name, equals, item, left square bracket, 0, right square bracket. Line 5, indented twice: score, equals, item, left square bracket, 1, right square bracket. Line 6, indented twice: if name in data Dict, colon. Line 7, indented

thrice: data Dict, left square bracket, name, right square bracket, dot, append, left parenthesis, score, right parenthesis. Line 8, indented twice: else, colon. Line 9, indented thrice: data Dict, left square bracket, name, right square bracket, equals, left square bracket, score, right square bracket. Line 10, indented once: for a Key in data Dict, colon. Line 11, indented twice: print, left parenthesis, a Key, sum, left parenthesis, data Dict, left square bracket, a Key, right square bracket, right parenthesis, forward slash, l e n, left parenthesis, data Dict, left square bracket, a Key, right square bracket, right parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f frequency Table, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: count Dict, equals, left brace, right brace. Line 3: Blank. Line 4, indented once: for item in a List, colon. Line 5, indented twice: if item in count Dict, colon. Line 6, indented thrice: count Dict, left square bracket, item, right square bracket, equals, count Dict, left square bracket, item, right square bracket, plus, 1. Line 7, indented twice: else, colon. Line 8, indented thrice: count Dict, left square bracket, item, right square bracket, equals, 1. Line 9: Blank. Line 10, indented once: items, equals, count Dict, dot, items, left parenthesis, right parenthesis. Line 11, indented once: items, equals, list, left parenthesis, items, right parenthesis. Line 12, indented once: items, dot, sort, left parenthesis, right parenthesis. Line 13: Blank. Line 14, indented once: print, left parenthesis, double quotes, ITEM, double quotes, double quotes, FREQUENCY, double quotes, right parenthesis. Line 15, indented once: for item in items, colon. Line 16, indented twice: print, left parenthesis, item, left square bracket, 0, right square

bracket, double quotes, double quotes, item, left square bracket, 1, right square bracket, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: import mat plot lib, dot, p y plot as p l t. Line 2: import statistics. Line 3: Blank. Line 4: d e f frequency Chart, left parenthesis, a List, right parenthesis, colon. Line 5: Blank. Line 6, indented once: count Dict, equals, left brace, right brace. Line 7: Blank. Line 8, indented once: for item in a List, colon. Line 9, indented twice: if item in count Dict, colon. Line 10, indented thrice: count Dict, left square bracket, item, right square bracket, equals, count Dict, left square bracket, item, right square bracket, plus, 1. Line 11, indented twice: else, colon. Line 12, indented thrice: count Dict, left square bracket, item, right square bracket, equals, 1. Line 13, indented once: keys, equals, list, left parenthesis, count Dict, dot, keys, left parenthesis, right parenthesis, right parenthesis, hash, the x axis. Line 14, indented once: values, equals, list, left parenthesis, count Dict, dot, values, left parenthesis, right parenthesis, right parenthesis, hash, the y axis. Line 15, indented once: mean, equals, statistics, dot, mean, left parenthesis, values, right parenthesis. Line 16, indented once: p l t, dot, bar, left parenthesis, x, equals, keys, height, equals, values, right parenthesis. Line 17, indented once: p l t, dot, a x v line, left parenthesis, x, equals, mean, color, equals, single quote, red, single quote, right parenthesis. Line 18, indented once: p l t, dot, x label, left parenthesis, double quotes, Value, double quotes, right parenthesis. Line 19, indented once: p l t, dot, y label, left parenthesis, double quotes, Frequency, double quotes, right parenthesis. Line 20, indented once: p l t, dot, show, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: with open, left parenthesis, double quotes, rainfall, dot, txt, double quotes, double quotes, r, double quotes, right parenthesis, as rain File, colon. Line 2, indented once: with open, left parenthesis, double quotes, rainfall F m t, dot, txt, double quotes, double quotes, w, double quotes, right parenthesis, as out File, colon. Line 3, indented twice: for a Line in rain File, colon. Line 4, indented thrice: values, equals, a Line, dot, split, left parenthesis, right parenthesis. Line 5, indented thrice: city, equals, values, left square bracket, 0, right square bracket. Line 6, indented thrice: inches, equals, float, left parenthesis, values, left square bracket, 1, right square bracket, right parenthesis. Line 7, indented thrice: n Chars, equals, out File, dot, write, left parenthesis, double quotes, left brace, 0, colon, greater than, 25 s, right brace, left brace, 1, colon, 5.1 f, right brace, backslash, n, double quotes, dot, format, left parenthesis, city, inches, right parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: with open, left parenthesis, double quotes, rainfall, dot, txt, double quotes, right parenthesis, as in File, colon. Line 2, indented once: line 1, equals, in File, dot, read line, left parenthesis, right parenthesis. Line 3, indented once: line 2, equals, in File, dot, read line, left parenthesis, right parenthesis. Line 4, indented once: rest Of File, equals, in File, dot, read lines, left parenthesis, right parenthesis. Line 5: Blank. Line 6: rest Of File. Line 7: hash, left square bracket, single quote, Algona 30.69, backslash, n, single quote, single quote, Allison 33.64, backslash,

n, single quote, single quote, Alton 27.43, backslash, n, single quote, single quote, Ames W 34.07, backslash, n, single quote, single quote, Ames S E 33.95, backslash, n, single quote, single quote, Anamosa 35.33, backslash, n, single quote, single quote, Ankeny 33.38, backslash, n, single quote, single quote, Atlantic 34.77, backslash, n, single quote, single quote, Audubon 33.41, backslash, n, single quote, single quote, Beaconsfield 35.27, backslash, n, single quote, single quote, Bedford 36.35, backslash, n, single quote, single quote, Belle Plaine 35.81, backslash, n, single quote, single quote, Bellevue 34.35, backslash, n, single quote, single quote, Blockton 36.28, backslash, n, single quote, single quote, Bloomfield 38.02, backslash, n, single quote, single quote, Boone 36.30, backslash, n, single quote, single quote, Brighton 33.59, backslash, n, single quote, single quote, Britt 31.54, backslash, n, single quote, single quote, Buckeye 33.66, backslash, n, single quote, single quote, Burlington K B U R 37.94, backslash, n, single quote, single quote, Burlington 36.94, backslash, n, single quote, single quote, Carroll 33.33, backslash, n, single quote, single quote, Cascade 33.48, backslash, n, single quote, right square bracket. Note: The read lines function returns lines 3 through the end of the file.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: with open, left parenthesis, double quotes, rainfall, dot, txt, double quotes, double quotes, r, double quotes, right parenthesis, as in File, colon. Line 2, indented once: line Count, equals, 0. Line 3, indented once: word Count, equals, 0. Line 4, indented once: char Count, equals, 0. Line 5, indented once: for a Line in in File, colon. Line 6, indented twice: line Count, equals, line Count, plus, 1. Line 7, indented twice: word

List, equals, a Line, dot, split, left parenthesis, right parenthesis. Line 8, indented twice: word Count, equals, word Count, plus, len, left parenthesis, word List, right parenthesis. Line 9, indented twice: char Count, equals, char Count, plus, len, left parenthesis, a Line, left square bracket, 0, colon, negative 1, right square bracket, right parenthesis. Line 10, indented once: print, left parenthesis, double quotes. Lines Words Characters, double quotes, right parenthesis. Line 11, indented once: print, left parenthesis, double quotes, left brace, 0, colon, greater than, 5, right brace, left brace, 1, colon, greater than, 5, right brace, left brace, 2, colon, greater than, 10, right brace, double quotes, dot, format, left parenthesis, line Count, word Count, char Count, right parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: def make Mag List By Date, left parenthesis, right parenthesis, colon. Line 2, indented once: mag Dict, equals, left brace, right brace. Line 3, indented once: with open, left parenthesis, double quotes, earthquakes, dot, c s v, double quotes, double quotes, r, double quotes, right parenthesis, as in File, colon. Line 4, indented twice: c s v Reader, equals, csv, dot, reader, left parenthesis, in File, right parenthesis. Line 5, indented twice: titles, equals, next, left parenthesis, c s v Reader, right parenthesis, hash, skip titles line. Line 6, indented twice: for line in c s v Reader, colon. Line 7, indented thrice: mag, equals, line, left square bracket, 4, right square bracket. Line 8, indented thrice: date Time, equals, line, left square bracket, 0, right square bracket. Line 9, indented thrice: date Time Values, equals, date Time, dot, split, left parenthesis, double quotes, T, double quotes, right parenthesis. Line 10, indented thrice: date, equals, date Time

Values, left square bracket, 0, right square bracket. Line 11, indented thrice: if date in mag Dict, colon. Line 12, indented quarce: mag Dict, left square bracket, date, right square bracket, dot, append, left parenthesis, mag, right parenthesis. Line 13, indented thrice: else, colon. Line 14, indented quarce: mag Dict, left square bracket, date, right square bracket, equals, left square bracket, mag, right square bracket. Line 15, indented twice: data List, equals, left square bracket, right square bracket. Line 16, indented twice: for a Date in mag Dict, colon. Line 17, indented thrice: data List, dot, append, left parenthesis, left square bracket, a Date, right square bracket, plus, mag Dict, left square bracket, a Date, right square bracket, right parenthesis. Line 18, indented twice: return data List.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: import pandas as p d. Line 2: d f, equals, p d, dot, read, underscore, c s v, left parenthesis, double quotes, earthquakes, dot, c s v, double quotes, right parenthesis. Line 3: filtered D f, equals, d f, left square bracket, left parenthesis, d f, left square bracket, double quotes, latitude, double quotes, right square bracket, left angle bracket, 0, right parenthesis, ampersand, left parenthesis, d f, left square bracket, double quotes, depth, double quotes, right square bracket, less than, 10, right parenthesis, right square bracket. Line 4: print, left parenthesis, filtered D f, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: import pandas as p d. Line 2: d f, equals, p d,

dot, read, underscore, c s v, left parenthesis, double quotes, earthquakes, dot, c s v, double quotes, right parenthesis. Line 3: print, left parenthesis, double quotes, mean, double quotes, d f, left square bracket, double quotes, depth, double quotes, right square bracket, dot, mean, left parenthesis, right parenthesis, right parenthesis. Line 4: print, left parenthesis, double quotes, median, double quotes, d f, left square bracket, double quotes, depth, double quotes, right square bracket, dot, median, left parenthesis, right parenthesis, right parenthesis. Line 5: print, left parenthesis, double quotes, s t d, double quotes, d f, left square bracket, double quotes, depth, double quotes, right square bracket, dot, s t d, left parenthesis, right parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f get Property, left parenthesis, data, key, right parenthesis, colon. Line 2, indented once: value List, equals, left square bracket, right square bracket. Line 3, indented once: earthquakes, equals, data, dot, get, left parenthesis, double quotes, features, double quotes, right parenthesis. Line 4, indented once: for i in range, left parenthesis, len, left parenthesis, earthquakes, right parenthesis, right parenthesis, colon. Line 5, indented twice: features, equals, earthquakes, left square bracket, i, right square bracket. Line 6, indented twice: props, equals, features, dot, get, left parenthesis, double quotes, properties, double quotes, right parenthesis. Line 7, indented twice: value, equals, props, dot, get, left parenthesis, key, right parenthesis. Line 8, indented twice: value List, dot, append, left parenthesis, value, right parenthesis. Line 9, indented once: return value List.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: felt Short List, equals, left square bracket, right square bracket. Line 2: earthquakes, equals, e Data, dot, get, left parenthesis, double quotes, features, double quotes, right parenthesis. Line 3: for i in range, left parenthesis, l e n, left parenthesis, earthquakes, right parenthesis, right parenthesis, colon. Line 4, indented once: features, equals, earthquakes, left square bracket, i, right square bracket. Line 5, indented once: props, equals, features, dot, get, left parenthesis, double quotes, properties, double quotes, right parenthesis. Line 6, indented once: felt, equals, props, dot, get, left parenthesis, double quotes, felt, double quotes, right parenthesis. Line 7, indented once: if, left parenthesis, felt is not None, right parenthesis, colon. Line 8, indented twice: felt Short List, dot, append, left parenthesis, felt, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f draw Rectangle, left parenthesis, right parenthesis, colon. Line 2, indented once: rect X, equals, 100. Line 3, indented once: rect Y, equals, 100. Line 4, indented once: rect Width, equals, 100. Line 5, indented once: rect Length, equals, 125. Line 6, indented once: white Pixel, equals, Pixel, left parenthesis, 255, 255, 255, right parenthesis. Line 7, indented once: i m, equals, Empty Image, left parenthesis, 300, 300, right parenthesis. Line 8, indented once: hash, draw top and bottom lines Line 9, indented once: for i in range, left parenthesis, rect X, rect X, plus, rect Width, right parenthesis, colon. Line 10, indented twice: i m, dot, set Pixel, left parenthesis, i, rect Y, white

Pixel, right parenthesis. Line 11, indented twice: i m, dot, set Pixel, left parenthesis, i, rect Y, plus, rect Length, white Pixel, right parenthesis. Line 12, indented once: hash, draw left and right lines. Line 13, indented once: for i in range, left parenthesis, rect Y, rect Y, plus, rect Length, right parenthesis, colon. Line 14, indented twice: i m, dot, set Pixel, left parenthesis, rect X, i, white Pixel, right parenthesis. Line 15, indented twice: i m, dot, set Pixel, left parenthesis, rect X, plus, rect Width, i, white Pixel, right parenthesis. Line 16, indented once: return i m. Line 17: Blank. Line 18: i m Win, equals, Image Win, left parenthesis, double quotes, Drawing a Rectangle, double quotes, 300, 300, right parenthesis. Line 19: rect Win, equals, draw Rectangle, left parenthesis, right parenthesis. Line 20: rect Win, dot, draw, left parenthesis, i m Win, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f clear Red, left parenthesis, old Pixel, right parenthesis, colon. Line 2, indented once: new Pixel, equals, Pixel, left parenthesis, 0, old Pixel, dot, get Green, left parenthesis, right parenthesis, old Pixel, dot, get Blue, left parenthesis, right parenthesis, right parenthesis. Line 3, indented once: return new Pixel.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f black White, left parenthesis, old Pixel, right parenthesis, colon. Line 2, indented once: red, equals, old Pixel, dot, get Red, left parenthesis, right parenthesis. Line 3, indented once: green, equals, old Pixel, dot, get Green, left

parenthesis, right parenthesis. Line 4, indented once: blue, equals, old Pixel, dot, get Blue, left parenthesis, right parenthesis. Line 5, indented once: gray, equals, left parenthesis, red, plus, green, plus, blue, right parenthesis, forward slash, forward slash, 3. Line 6, indented once: if gray, less than, equals, 128, colon. Line 7, indented twice: new Pixel, equals, Pixel, left parenthesis, 0, 0, 0, right parenthesis. Line 8, indented once: else, colon. Line 9, indented twice: new Pixel, equals, Pixel, left parenthesis, 255, 255, 255, right parenthesis. Line 10, indented once: return new Pixel.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f sepia, left parenthesis, old Pixel, right parenthesis, colon. Line 2, indented once: red, equals, old Pixel, dot, get Red, left parenthesis, right parenthesis. Line 3, indented once: green, equals, old Pixel, dot, get Green, left parenthesis, right parenthesis. Line 4, indented once: blue, equals, old Pixel, dot, get Blue, left parenthesis, right parenthesis. Line 5, indented once: new Red, equals, i n t, left parenthesis, red, asterisk, 0.393, plus, green, asterisk, 0.769, plus, blue, asterisk, 0.189, right parenthesis. Line 6, indented once: new Green, equals, i n t, left parenthesis, red, asterisk, 0.349, plus, green, asterisk, 0.686, plus, blue, asterisk, 0.168, right parenthesis. Line 7, indented once: new Blue, equals, i n t, left parenthesis, red, asterisk, 0.272, plus, green, asterisk, 0.534, plus, blue, asterisk, 0.131, right parenthesis. Line 8, indented once: if new Red, greater than, 255, colon. Line 9, indented twice: new Red, equals, 255. Line 10, indented once: if new Green, greater than, 255, colon. Line 11, indented twice: new Green, equals, 255. Line 12, indented once: if new Blue, greater than, 255, colon. Line 13, indented twice:

new Blue, equals, 255. Line 14, indented once: new Pixel, equals, Pixel, left parenthesis, new Red, new Green, new Blue, right parenthesis. Line 15, indented once: return new Pixel.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f four Times, left parenthesis, old Image, right parenthesis, colon. Line 2, indented once: old W, equals, old Image, dot, get Width, left parenthesis, right parenthesis. Line 3, indented once: old H, equals, old Image, dot, get Height, left parenthesis, right parenthesis. Line 4, indented once: new Im, equals, Empty Image, left parenthesis, old W, asterisk, 4, old H, asterisk, 4, right parenthesis. Line 5, indented once: for row in range, left parenthesis, new Im, dot, get Height, left parenthesis, right parenthesis, right parenthesis, colon. Line 6, indented twice: for col in range, left parenthesis, new Im, dot, get Width, left parenthesis, right parenthesis, right parenthesis, colon. Line 7, indented thrice: orig Col, equals, col, forward slash, forward slash, 4. Line 8, indented thrice: orig Row, equals, row, forward slash, forward slash, 4. Line 9, indented thrice: old Pixel, equals, old Image, dot, get Pixel, left parenthesis, orig Col, orig Row, right parenthesis. Line 10, indented thrice: new Im, dot, set Pixel, left parenthesis, col, row, old Pixel, right parenthesis. Line 11, indented once: return new Im.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f half Size, left parenthesis, old Image, right parenthesis, colon. Line 2, indented once: new Width, equals, old Image, dot, get Width, left parenthesis, right parenthesis, forward

slash, forward slash, 2. Line 3, indented once: new Height, equals, old Image, dot, get Height, left parenthesis, right parenthesis, forward slash, forward slash, 2. Line 4, indented once: new Im, equals, Empty Image, left parenthesis, new Width, new Height, right parenthesis. Line 5, indented once: for row in range, left parenthesis, new Height, right parenthesis, colon. Line 6, indented twice: for col in range, left parenthesis, new Width, right parenthesis, colon. Line 7, indented thrice: hash, get average of neighbors Line 8, indented thrice: n 1Pixel, equals, old Image, dot, get Pixel, left parenthesis, 2, asterisk, col, 2, asterisk, row, right parenthesis. Line 9, indented thrice: n 2 Pixel, equals, old Image, dot, get Pixel, left parenthesis, 2, asterisk, col, plus, 1, 2, asterisk, row, right parenthesis. Line 10, indented thrice: n 3 Pixel, equals, old Image, dot, get Pixel, left parenthesis, 2, asterisk, col, 2, asterisk, row, plus, 1, right parenthesis. Line 11, indented thrice: n 4 Pixel, equals, old Image, dot, get Pixel, left parenthesis, 2, asterisk, col, plus, 1, 2, asterisk, row, plus, 1, right parenthesis. Line 12, indented thrice: red, equals, left parenthesis, n 1 Pixel, dot, get Red, left parenthesis, right parenthesis, plus, n 2 Pixel, dot, get Red, left parenthesis, right parenthesis, backslash, plus, n 3 Pixel, dot, get Red, left parenthesis, right parenthesis, plus, n 4 Pixel, dot, get Red, left parenthesis, right parenthesis, right parenthesis, forward slash, forward slash, 4. Line 13, indented thrice: green, equals, left parenthesis, n 1 Pixel, dot, get Green, left parenthesis, right parenthesis, plus, n 2 Pixel, dot, get Green, left parenthesis, right parenthesis, backslash, plus, n 3 Pixel, dot, get Green, left parenthesis, right parenthesis, plus, n 4 Pixel, dot, get Green, left parenthesis, right parenthesis, right parenthesis, forward slash, forward slash, 4. Line 14, indented thrice: blue, equals, left parenthesis, n 1 Pixel, dot, get Blue, left parenthesis, right parenthesis, plus, n 2 Pixel, dot, get Blue, left parenthesis, right parenthesis, backslash, plus, n 3 Pixel, dot, get

Blue, left parenthesis, right parenthesis, plus, n 4 Pixel, dot, get Blue, left parenthesis, right parenthesis, right parenthesis, forward slash, forward slash, 4. Line 15, indented thrice: new Pixel, equals, Pixel, left parenthesis, red, green, blue, right parenthesis. Line 16, indented thrice: new Im, dot, set Pixel, left parenthesis, col, row, new Pixel, right parenthesis. Line 17, indented once: return new Im.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f reduce Noise, left parenthesis, old Image, right parenthesis, colon. Line 2, indented once: import statistics. Line 3, indented once: height, equals, old Image, dot, get Height, left parenthesis, right parenthesis. Line 4, indented once: width, equals, old Image, dot, get Width, left parenthesis, right parenthesis. Line 5, indented once: new Im, equals, Empty Image, left parenthesis, width, height, right parenthesis. Line 6, indented once: for row in range, left parenthesis, height, minus, 1, right parenthesis, colon. Line 7, indented twice: for col in range, left parenthesis, width, minus, 1, right parenthesis, colon. Line 8, indented thrice: n 1, equals, old Image, dot, get Pixel, left parenthesis, col, row, right parenthesis. Line 9, indented thrice: n 2, equals, old Image, dot, get Pixel, left parenthesis, col, row, plus, 1, right parenthesis. Line 10, indented thrice: n 3, equals, old Image, dot, get Pixel, left parenthesis, col, plus, 1, row, right parenthesis. Line 11, indented thrice: n 4, equals, old Image, dot, get Pixel, left parenthesis, col, plus, 1, row, plus, 1, right parenthesis. Line 12, indented thrice: red, equals, left square bracket, n 1, dot, get Red, left parenthesis, right parenthesis, n 2, dot, get Red, left parenthesis, right parenthesis, n 3, dot, get Red, left parenthesis, right parenthesis, backslash, n 4, dot, get Red,

left parenthesis, right parenthesis, right square bracket. Line 13, indented thrice: green, equals, left square bracket, n 1, dot, get Green, left parenthesis, right parenthesis, n 2, dot, get Green, left parenthesis, right parenthesis, backslash, n 3, dot, get Green, left parenthesis, right parenthesis, n 4, dot, get Green, left parenthesis, right parenthesis, right square bracket. Line 14, indented thrice: blue, equals, left square bracket, n 1, dot, get Blue, left parenthesis, right parenthesis, n 2, dot, get Blue, left parenthesis, right parenthesis, n 3, dot, get Blue, left parenthesis, right parenthesis, backslash, n 4, dot, get Blue, left parenthesis, right parenthesis, right square bracket. Line 15, indented thrice: med Red, equals, i n t, left parenthesis, statistics, dot, median, left parenthesis, red, right parenthesis, right parenthesis. Line 16, indented thrice: med Green, equals, i n t, left parenthesis, statistics, dot, median, left parenthesis, green, right parenthesis, right parenthesis. Line 17, indented thrice: med Blue, equals, i n t, left parenthesis, statistics, dot, median, left parenthesis, blue, right parenthesis, right parenthesis. Line 18, indented thrice: new Pixel, equals, Pixel, left parenthesis, med Red, med Green, med Blue, right parenthesis. Line 19, indented thrice: new Im, dot, set Pixel, left parenthesis, col, row, new Pixel, right parenthesis. Line 20, indented once: hash, bring over end pixels unchanged. Line 21, indented once: for row in range, left parenthesis, height, minus, 1, right parenthesis, colon. Line 22, indented twice: old Pixel, equals, old Image, dot, get Pixel, left parenthesis, width, minus, 1, row, right parenthesis. Line 23, indented twice: new Im, dot, set Pixel, left parenthesis, width, minus, 1, row, old Pixel, right parenthesis. Line 24, indented once: for col in range, left parenthesis, width, minus, 1, right parenthesis, colon. Line 25, indented twice: old Pixel, equals, old Image, dot, get Pixel, left parenthesis, col, height, minus, 1, right parenthesis. Line 26, indented twice: new Im, dot, set Pixel, left parenthesis, col,

height, minus, 1, old Pixel, right parenthesis. Line 27: Blank. Line 28, indented once: return new Im.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f complete Flip, left parenthesis, old Image, right parenthesis, colon. Line 2, indented once: old W, equals, old Image, dot, get Width, left parenthesis, right parenthesis. Line 3, indented once: old H, equals, old Image, dot, get Height, left parenthesis, right parenthesis. Line 4: Blank. Line 5, indented once: new Im, equals, Empty Image, left parenthesis, old W, old H, right parenthesis. Line 6: Blank. Line 7, indented once: for row in range, left parenthesis, old H, right parenthesis, colon. Line 8, indented twice: for col in range, left parenthesis, old W, right parenthesis, colon. Line 9, indented thrice: old Pixel, equals, old Image, dot, get Pixel, left parenthesis, old W, minus, 1, minus, col, old H, minus, 1, minus, row, right parenthesis. Line 10, indented thrice: new Im, dot, set Pixel, left parenthesis, col, row, old Pixel, right parenthesis. Line 11: Blank. Line 12, indented once: return new Im.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f vertical Mirror, left parenthesis, old Image, right parenthesis, colon. Line 2, indented once: old W, equals, old Image, dot, get Width, left parenthesis, right parenthesis. Line 3, indented once: old H, equals, old Image, dot, get Height, left parenthesis, right parenthesis. Line 4, indented once: new Im, equals, Empty Image, left parenthesis, old W, old H, right parenthesis. Line 5, indented once: if old W, percent, 2, equals,

equals, 0, colon. Line 6, indented twice: halfway, equals, old W, forward slash, forward slash, 2. Line 7, indented once: else, colon. Line 8, indented twice: halfway, equals, old W, forward slash, forward slash, 2, plus, 1. Line 9, indented once: max P, equals, old W, minus, 1. Line 10, indented once: for row in range, left parenthesis, old H, right parenthesis, colon. Line 11, indented twice: for col in range, left parenthesis, halfway, right parenthesis, colon. Line 12, indented thrice: old Pixel, equals, old Image, dot, get Pixel, left parenthesis, max P, minus, col, row, right parenthesis. Line 13, indented thrice: new Im, dot, set Pixel, left parenthesis, max P, minus, col, row, old Pixel, right parenthesis. Line 14, indented thrice: new Im, dot, set Pixel, left parenthesis, col, row, old Pixel, right parenthesis. Line 15, indented once: return new Im.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f rotate image 90, left parenthesis, old Image, right parenthesis, colon. Line 2, indented once: old W, equals, old Image, dot, get Width, left parenthesis, right parenthesis. Line 3, indented once: old H, equals, old Image, dot, get Height, left parenthesis, right parenthesis. Line 4, indented once: new I m, equals, Empty Image, left parenthesis, old H, old W, right parenthesis. Line 5, indented once: right Col, equals, old H, hyphen, 1. Line 6, indented once: for row in range, left parenthesis, old H, right parenthesis, colon. Line 7, indented twice: for col in range, left parenthesis, old W, right parenthesis, colon. Line 8, indented thrice: old Pixel, equals, old Image, dot, get Pixel, left parenthesis, col, row, right parenthesis. Line 9, indented thrice: new I m, dot, set Pixel, left parenthesis, right Col, col, old Pixel, right parenthesis. Line 10, indented twice: right

Col, equals, right Col, hyphen, 1 Line 11, indented once: return new I m.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: with open, left parenthesis, left double quotes, c s 150 exams, dot, t x t, right double quotes, right parenthesis, as data file, colon. Line 2, indented once: scores, equals, left square bracket, right square bracket. Line 3, indented once: line, equals, data file, dot, read line, left parenthesis, right parenthesis. Line 4, indented once: while line, not equal to, left double quotes, right double quotes, colon. Line 5, indented twice: score, equals, line, dot, replace, left parenthesis, left double quotes, backslash, n, right double quotes, comma, left double quotes, right double quotes, right parenthesis. Line 6, indented twice: scores, dot, append, left parenthesis, score, right parenthesis. Line 7, indented twice: line, equals, data file, dot, read line, left parenthesis, right parenthesis. Line 8: print, left parenthesis, scores, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f, space, score Average, left parenthesis, right parenthesis, colon. Line 2, indented once: sum, equals, 0. Line 3, indented once: how Many, equals, 0. Line 4, indented once: score, equals, input, left parenthesis, left double quotes, Please enter a score, left parenthesis, left single quote, stop, right single quote, to exit, right parenthesis, right double quotes, right parenthesis. Line 5, indented once: while score, not equal to, left single quote, stop, right single quote, colon. Line 6, indented

twice: sum, equals, sum, plus, i n t, left parenthesis, score, right parenthesis. Line 7, indented twice: how Many, equals, how Many, plus, 1. Line 8, indented twice: score, equals, input, left parenthesis, left double quotes, Please enter a score, left parenthesis, left single quote, stop, right single quote, to exit, right parenthesis, right double quotes, right parenthesis. Line 9, indented once: average, equals, sum, forward slash, how Many. Line 10, indented once: return average.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f create Clusters, left parenthesis, k, comma, centroids, comma, data List, comma, repeats, right parenthesis, colon. Line 2, indented once: prev Clusters, equals, left square bracket, right square bracket. Line 3, indented once: for i in range, left parenthesis, k, right parenthesis, colon. Line 4, indented twice: prev Clusters, dot, append, left parenthesis, left square bracket, right square bracket, right parenthesis. Line 5, indented once: a Pass, equals, 1. Line 6, indented once: do Again, equals, True. Line 7: Blank. Line 8, indented once: while do Again, colon. Line 9, indented twice: print, left parenthesis, left double quotes, asterisk, asterisk, asterisk, asterisk, PASS, right double quotes, comma, a Pass, comma, left double quotes, asterisk, asterisk, asterisk, asterisk, right double quotes, right parenthesis. Line 10, indented twice: clusters, equals, left square bracket, right square bracket. Line 11, indented twice: for i in range, left parenthesis, k, right parenthesis, colon. Line 12, indented thrice: clusters, dot, append, left parenthesis, left square bracket, right square bracket, right parenthesis. Line 13, indented twice: for data Index in range, left parenthesis, l e n, left parenthesis, data List, right parenthesis, right parenthesis, colon,

hash, calculate the distance to centroid. Line 14, indented thrice: distances, equals, left square bracket, right square bracket. Line 15, indented thrice: for cluster Index in range, left parenthesis, k, right parenthesis, colon. Line 16, indented quadrice: d t o C, equals, Euclid D, left parenthesis, data List, left square bracket, data Index, right square bracket, comma, centroids, left square bracket, cluster Index, right square bracket, right parenthesis. Line 17, indented quadrice: distances, dot, append, left parenthesis, d t o C, right parenthesis. Line 18, indented quadrice: min Dist, equals, min, left parenthesis, distances, right parenthesis, space, hash, minimum distance. Line 19, indented quadrice: closest Cluster, equals, distances, dot, index, left parenthesis, min Dist, right parenthesis, space, hash, index of min distance. Line 20, indented thrice: clusters, left square bracket, closest Cluster, right square bracket, dot, append, left parenthesis, data Index, right parenthesis, space, hash, add to cluster. Line 21: blank. Line 22, indented twice: hash, check previous cluster against new cluster. Line 23, indented twice: if clusters, not equal to, prev Clusters, colon. Line 24, indented thrice: Prev Clusters, equals, clusters. Line 25, indented twice: else, colon. Line 26, indented thrice: do Again, equals, False. Line 27, indented twice: dimensions, equals, l e n, left parenthesis, data List, left square bracket, 0, right square bracket, right parenthesis, space, hash, recompute the clusters. Line 28, indented twice: for cluster Index in range, left parenthesis, k, right parenthesis, colon. Line 29, indented thrice: sums, equals, left square bracket, 0, right square bracket, asterisk, dimensions, hash, i n i t sum for each dimension. Line 28, indented thrice: for data Index in clusters, left square bracket, cluster Index, right square bracket, colon. Line 29, indented quarce: dataPoints, equals, data List, left square bracket, data Index, right square bracket. Line 30, indented quarce: for i n d in range, left parenthesis, l e n, left parenthesis,

data Points, right parenthesis, right parenthesis, colon, hash, calculate sums. Line 31, indented quince: sums, left square bracket, i n d, right square bracket, equals, sums, left square bracket, i n d, right square bracket, plus, data Points, left square bracket, i n d, right square bracket. Line 32, indented thrice: for i n d in range, left parenthesis, l e n, left parenthesis, sums, right parenthesis, right parenthesis, colon, hash, calculate average. Line 33, indented quarce: cluster Len, equals, l e n, left parenthesis, clusters, left square bracket, cluster Index, right square bracket, right parenthesis. Line 34, indented quarce: if cluster Len, not equal to, 0, colon, hash, do not divide by 0. Line 35, indented quince: sums, left square bracket, i n d, right square bracket, equals, sums, left square bracket, i n d, right square bracket, forward slash, cluster Len. Line 36, indented thrice: centroids, left square bracket, cluster Index, right square bracket, equals, sums, hash, assign a v g to centroids. Line 37, indented twice: for c in range, left parenthesis, k, right parenthesis, colon, hash, output the clusters. Line 38, indented thrice: print, left parenthesis, left double quotes, CLUSTER, hash, right double quotes, plus, s t r, left parenthesis, c, plus, 1, right parenthesis, right parenthesis. Line 39, indented thrice: for data Index in clusters, left square bracket, c, right square bracket, colon. Line 40, indented quarce: print, left parenthesis, data List, left square bracket, data Index, right square bracket, end, equals, left double quotes, right double quotes, right parenthesis. Line 41, indented thrice: print, left parenthesis, right parenthesis. Line 42, indented twice: a Pass, equals, a Pass, plus, 1. Line 43: return clusters.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f create Word List, left parenthesis, filename,

right parenthesis, colon. Line 2, indented once: my List, equals, left square bracket, right square bracket. Line 3, indented once: with open, left parenthesis, filename, comma, left double quotes, r, right double quotes, right parenthesis, as my File, colon. Line 4, indented twice: for a Line in my File, colon. Line 5, indented thrice: my List, dot, append, left parenthesis, a Line, left square bracket, colon, negative 1, right square bracket, right parenthesis. Line 6, indented once: return my List.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: import time. Line 2: d e f time Functions, left parenthesis, right parenthesis, colon. Line 3, indented once: start, equals, time, dot, time, left parenthesis, right parenthesis. Line 4, indented once: create Word Dict, left parenthesis, left double quotes, wordlist, dot, t x t, right double quotes, right parenthesis. Line 5, indented once: end, equals, time, dot, time, left parenthesis, right parenthesis. Line 6, indented once: print, left parenthesis, left double quotes, Dict, colon, right double quotes, comma, end, hyphen, start, right parenthesis. Line 7. start, equals, time, dot, time, left parenthesis, right parenthesis. Line 8, indented once: create Word List, left parenthesis, left double quotes, wordlist, dot, txt, right double quotes, right parenthesis. Line 9, indented once: end, equals, time, dot, time, left parenthesis, right parenthesis. Line 10, indented once: print, left parenthesis, left double quotes, List, colon, right double quotes, comma, end, hyphen, start, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: W List, equals, left square bracket, left single quote, the, right single quote, comma, left single quote, quick, right single quote, comma, left single quote, brown, right single quote, comma, single quote, fox, single quote, right square bracket. Line 2: Left single quote, space, right single quote, dot, join, left parenthesis, w List, right parenthesis. Line 3: hash, left single quote, the quick brown fox, right single quote. Line 4: left single quote, colon, right single quote, dot, join, left parenthesis, w List, right parenthesis. Line 5: hash, left single quote, the, colon, quick, colon, brown, colon, fox, right single quote. Line 6: left single quote, comma, right single quote, dot, join, left parenthesis, w List, right parenthesis. Line 7: hash, left single quote, the, comma, quick, comma, brown, comma, fox, right single quote. Line 8: Left single quote, hyphen hyphen, right single quote, dot, join, left parenthesis, w List, right parenthesis. Line 9: hash, left single quote, the, hyphen hyphen, quick, hyphen hyphen, brown, hyphen hyphen, fox, right single quote.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f rail Break, left parenthesis, cipher Text, right parenthesis, colon. Line 2, indented once: word Dict, equals, create Word Dict, left parenthesis, left double quotes, wordlist, dot, txt, right double quotes, right parenthesis. Line 3, indented once: cipher L e n, equals, l e n, left parenthesis, cipher Text, right parenthesis. Line 4, indented once: max Good, equals, 0. Line 5, indented once: for i in range, left parenthesis, 1, comma, l e n, left parenthesis, cipher Text, right parenthesis, forward slash, forward slash, 2, right parenthesis, colon, hash, part, left parenthesis, a, right parenthesis. Line 6, indented twice:

if cipher L e n, percent, i, equals, equals, 0, colon, hash, part, left parenthesis, b, right parenthesis. Line 7, indented thrice: print, left parenthesis, left double quotes, trying rails, colon, right double quotes, comma, i, right parenthesis. Line 8, indented thrice: words, equals, rail Decrypt, left parenthesis, cipher Text, comma, i, right parenthesis. Line 9, indented thrice: good Count, equals, 0. Line 10, indented thrice: for w in words, colon. Line 11, indented quarce: if w in word Dict, colon. Line 12, indented quince: Good Count, equals, good Count, plus, 1. Line 13, indented thrice: if good Count, right angle bracket, max Good, colon. Line 14, indented quarce: max Good, equals, good Count. Line 15, indented quarce: best Guess, equals, left double quotes, right double quotes, dot, join, left parenthesis, words, right parenthesis. Line 16, indented once: return best Guess.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f word Pop, left parenthesis, text, comma, word Length, right parenthesis, colon. Line 2, indented once: word List, equals, left square bracket, right square bracket. Line 3, indented once: words, equals, text, dot, split, left parenthesis, right parenthesis. Line 4, indented once: for w in words, colon. Line 5, indented twice: if l e n, left parenthesis, w, right parenthesis, equals, equals, word Length and w not in word List, colon. Line 6, indented thrice: word List, dot, append, left parenthesis, w, right parenthesis. Line 7, indented once: word List, dot, sort, left parenthesis, right parenthesis. Line 8, indented once: return wordlist.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f sort Key, left parenthesis, x, right parenthesis, colon. Line 2, indented once: return x, left square bracket, 1, right square bracket. Line 3: space. Line 4: def end Pop, left parenthesis, my String, right parenthesis, colon. Line 5, indented once: word List, equals, my String, dot, split, left parenthesis, right parenthesis. Line 6, indented once: pop Dict, equals, left brace, right brace. Line 7, indented once: for word in word List, colon. Line 8, indented twice: pop Dict, left square bracket, word, left square bracket, negative 1, right square bracket, right square bracket, equals, pop Dict, dot, get, left parenthesis, word, left square bracket, negative 1, right square bracket, comma, 0, right parenthesis, plus, 1. Line 9, indented once: pop List, equals, list, left parenthesis, pop Dict, dot, items, left parenthesis, right parenthesis, right parenthesis. Line 10, indented once: pop List, dot, sort, left parenthesis, key, equal, sort Key, comma, reverse, equals, True, right parenthesis. Line 11, indented once: return pop List, left square bracket, 0, right square bracket.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: r e, dot, match, left parenthesis, left single quote, dot, asterisk, i n g, right single quote, comma, my Word, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: r e, dot, match, left parenthesis, left single quote, left square bracket, a, minus, z, right square bracket, left square

bracket, a e i o u, right square bracket, left square bracket, a e i o u, right square bracket, left square bracket, a, minus, z, right square bracket, dollar sign, right single quote, comma, my Word, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f get Name Suffix, left parenthesis, file name, right parenthesis, colon. Line 2, indented once: g, equals, r e, dot, match, left parenthesis, left single quote, left parenthesis, dot, asterisk, right parenthesis, backslash, dot, left parenthesis, dot, asterisk, right parenthesis, right single quote, comma, filename, right parenthesis. Line 3, indented once: if g, colon. Line 4, indented twice: return g, dot, group, left parenthesis, 1, right parenthesis, comma, g, dot, group, left parenthesis, 2, right parenthesis. Line 5, indented once: else, colon. Line 6, indented thrice: return None.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f draw Centered Square, left parenthesis, a Turtle, x, comma, y, comma, side, right parenthesis, colon. Line 2, indented once: a Turtle, dot, up, left parenthesis, right parenthesis. Line 3, indented once: a Turtle, dot, go to, left parenthesis, x, comma, y, right parenthesis. Line 4, indented once: a Turtle, dot, dot, left parenthesis, right parenthesis. Line 5, indented once: a Turtle, dot, left, left parenthesis, 90, right parenthesis. Line 6, indented once: a Turtle, dot, forward, left parenthesis, side, forward slash, 2, right parenthesis. Line 7, indented once: a Turtle, dot, left, left parenthesis, 90, right

parenthesis. Line 8, indented once: a Turtle, dot, forward, left parenthesis, side, forward slash, 2, right parenthesis. Line 9, indented once: a Turtle, dot, right, left parenthesis, 180, right parenthesis. Line 10, indented once: a Turtle, dot, down, left parenthesis, right parenthesis. Line 11, indented once: for i in range, left parenthesis, 4, right parenthesis, colon. Line 12, indented twice: a Turtle, dot, forward, left parenthesis, side, right parenthesis. Line 13, indented twice: a Turtle, dot, right, left parenthesis, 90, right parenthesis. Line 14: def concentric Box, left parenthesis, a Turtle, comma, x, comma, y, side, right parenthesis, colon. Line 15, indented once: if side, right angle bracket, equals, 1, colon. Line 16, indented twice: draw Centered Square, left parenthesis, a Turtle, x, y, side, right parenthesis. Line 17, indented twice: concentric Box, left parenthesis, a Turtle, x, comma, y, comma, side, hyphen, 10, right parenthesis. Line 18: Blank. Line 19: import turtle. Line 20: t, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 21: concentric Box, left parenthesis, t, comma, 0, comma, 0, comma, 200, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f sum List, left parenthesis, a List, right parenthesis, colon. Line 2, indented once: if a List, equals, equals, left square bracket, right square bracket, colon. Line 3, indented twice: return 0. Line 4, indented once: else, colon. Line 5, indented twice: return a List, left square bracket, 0, right square bracket, plus, sum List, left parenthesis, a List, left square bracket, 1, colon, right square bracket, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f reverse String, left parenthesis, a String, right parenthesis, colon. Line 2, indented once: if a String, equals, equals, left single quote, right single quote, colon. Line 3, indented twice: return, left single quote, right single quote. Line 4, indented once: else, colon. Line 5, indented twice: return reverse String, left parenthesis, a String, left square bracket, 1, colon, right square bracket, right parenthesis, plus, a String, left square bracket, 0, right square bracket.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f tree Left, left parenthesis, t, comma, trunk Length, right parenthesis, colon. Line 2, indented once: if trunk Length, left angle bracket, 5, colon. Line 3, indented twice: return. Line 4, indented once: else, colon. Line 5, indented twice: t, dot, forward, left parenthesis, trunk Length, right parenthesis. Line 6, indented twice: t, dot, left, left parenthesis, 30, right parenthesis. Line 7, indented twice: tree Left, left parenthesis, t, trunk Length, hyphen, 15, right parenthesis. Line 8, indented twice: t, dot, right, left parenthesis, 60, right parenthesis. Line 9, indented twice: tree Left, left parenthesis, t, comma, trunk Length, hyphen, 15, right parenthesis. Line 10, indented twice: t, dot, left, left parenthesis, 30, right parenthesis. Line 11, indented twice: t, dot, backward, left parenthesis, trunk Length, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: import random. Line 2: d e f tree, left

parenthesis, t, comma, trunk Length, right parenthesis, colon. Line 3, indented once: if trunk Length, left angle bracket, 5, colon. Line 4, indented twice: return. Line 5, indented once, else, colon. Line 6, indented twice: t, dot, forward, left parenthesis, trunk Length, right parenthesis. Line 7, indented twice: t, dot, left, left parenthesis, 30, right parenthesis. Line 8, indented twice: tree, left parenthesis, t, comma, trunk Length, hyphen, random, dot, r and i n t, left parenthesis, 5, comma, 25, right parenthesis, right parenthesis. Line 9, indented twice: t, dot, right, left parenthesis, 60, right parenthesis. Line 10, indented twice: tree, left parenthesis, t, comma, trunk Length, hyphen, random, dot, r and i n t, left parenthesis, 5, comma, 25, right parenthesis, right parenthesis. Line 11, indented twice: t, dot, left, left parenthesis, 30, right parenthesis. Line 12, indented twice: t, dot, backward, left parenthesis, trunk Length, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f draw Triangle, left parenthesis, t, comma, p 1, comma, p 2, comma, p 3, comma, a Color, right parenthesis, colon. Line 2, indented once: t, dot, color, left parenthesis, a Color, right parenthesis. Line 3, indented once: t, dot, up, left parenthesis, right parenthesis. Line 4, indented once: t, dot, go to, left parenthesis, p 1, right parenthesis. Line 5, indented once: t, dot, down, left parenthesis, right parenthesis. Line 6, indented once: t, dot, go to, left parenthesis, p 2, right parenthesis. Line 7, indented once: t, dot, go to, left parenthesis, p 3, right parenthesis. Line 8, indented once: t, dot, go to, left parenthesis, p 1, right parenthesis. Line 9: Blank. Line 10: d e f sierpinski, left parenthesis, my Turtle, comma, p 1, comma, p 2, comma, p 3, comma, depth, comma, a Color, right parenthesis, colon. Line 11,

indented once: if depth, right angle bracket, 0, colon. Line 12, indented twice: sierpinski, left parenthesis, my Turtle, comma, p 1, comma, mid Point, left parenthesis, p 1, comma, p 2, right parenthesis, midpoint, left parenthesis, p 1, comma, p 3, right parenthesis, comma. Line 13, indented quarce: depth, hyphen, 1, left single quote, blue, right single quote, right parenthesis. Line 14, indented twice: sierpinski, left parenthesis, my Turtle, comma, p 2, comma, mid Point, left parenthesis, p 2, comma, p 3, right parenthesis, comma, midpoint, left parenthesis, p 2, comma, p 1, right parenthesis, comma. Line 15, indented quarce: depth, hyphen, 1, left single quote, green, right single quote, right parenthesis. Line 16, indented twice: sierpinski, left parenthesis, my Turtle, comma, p 3, comma, midPoint, left parenthesis, p 3, comma, p 1, right parenthesis, comma, midpoint, left parenthesis, p 3, comma, p 2, right parenthesis, comma. Line 17, indented quarce: depth, hyphen, 1, left single quote, red, right single quote, right parenthesis. Line 18, indented once: else, colon. Line 19, indented twice: draw Triangle, left parenthesis, my Turtle, comma, p 1, comma, p 2, comma, p 3, comma, a Color, right parenthesis. Line 20: To execute these functions, colon. Line 21, indented once: import turtle. Line 22, indented once: t, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 23. sierpinski, left parenthesis, t, comma, left parenthesis, 0, comma, 0, right parenthesis, comma, left parenthesis, 200, comma, 200, right parenthesis, comma, left parenthesis, 200, comma, 0, right parenthesis, comma, 4, left single quote, black, right single quote, right parenthesis. Line 24, indented once: t, dot, hide turtle, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: def r e c Koch, left parenthesis, a T, comma, distance, comma, level, right parenthesis, colon. Line 2, indented once: if level, equals, equals, 0, colon. Line 3, indented twice: a T, dot, forward, left parenthesis, distance, right parenthesis. Line 4, indented once: else, colon. Line 5, indented twice: r e c Koch, left parenthesis, a T, distance, forward slash, 3, comma, level, hyphen, 1, right parenthesis. Line 6, indented twice: a T, dot, left, left parenthesis, 60, right parenthesis. Line 7, indented twice: r e c Koch, left parenthesis, a T, distance, forward slash, 3, level, hyphen, 1, right parenthesis. Line 8, indented twice: a T, dot, right, left parenthesis, 120, right parenthesis. Line 9, indented twice: r e c Koch, left parenthesis, a T, distance, forward slash, 3, level, hyphen, 1, right parenthesis. Line 10, indented twice: a T, dot, left, left parenthesis, 60, right parenthesis. Line 11, indented twice: r e c Koch, left parenthesis, a T, distance, forward slash, 3, level, hyphen, 1, right parenthesis. Line 12: Blank. Line 13: import turtle. Line 14: t, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 15: t, dot, up, left parenthesis, right parenthesis. Line 16: t, dot, go to, left parenthesis, negative 100, 100, right parenthesis. Line 17: t, dot, down, left parenthesis, right parenthesis. Line 18: Blank. Line 19: r e c Koch, left parenthesis, t, comma, 200, comma, 4, right parenthesis. Line 20: t, dot, right, left parenthesis, 120, right parenthesis. Line 21: r e c Koch, left parenthesis, t, comma, 200, comma, 4, right parenthesis. Line 22: t, dot, right, left parenthesis, 120, right parenthesis. Line 23: r e c Koch, left parenthesis, t, comma, 200, comma, 4, right parenthesis. Line 24: t, dot, hide turtle, left parenthesis, right parenthesis.

Back to Code

It read the following. Line 1: A B. Line 2: A B A. Line 3: A B A A B. Line 4: A B A A B A B A. Line 5: A B A A B A B A A B A A B.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: import turtle. Line 2: t, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 3: axiom, equals, left single quote, F, plus, plus, F, plus, plus, F, right single quote. Line 4: my Rules, equals, left brace, left single quote, F, right single quote, colon, left single quote, F, minus, F, plus, plus, F, minus F, right single quote, right brace. Line 5: instructions, equals, apply Production, left parenthesis, axiom, comma, my Rules, comma, 3, right parenthesis. Line 6: t, dot, up, left parenthesis, right parenthesis. Line 7: t, dot, go to, left parenthesis, negative 100, 100, right parenthesis. Line 8: t, dot, down, left parenthesis, right parenthesis. Line 9: draw L S, left parenthesis, t, instructions, comma, 60, comma, 10, right parenthesis. Line 10: t, dot, hide turtle, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: case, colon, left single quote, G, right single quote, colon. Line 2, indented once: a Turtle, dot, up, left parenthesis, right parenthesis. Line 3, indented once: a Turtle, dot, forward, left parenthesis, distance, right parenthesis. Line 4, indented once: a Turtle, dot, down, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Planet, colon. Line 2, indented once: def, double underscore, in it, double underscore, left parenthesis, self, comma, i Name, comma, i Rad, comma, i M, comma, i Dist, comma, i Moons, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, name, equals, I Name. Line 4, indented twice: self, dot, double underscore, radius, equals, I Rad. Line 5, indented twice: self, dot, double underscore, mass, equals, I M. Line 6, indented twice: self, dot, double underscore, distance, equals, I Dist. Line 7, indented twice: self, dot, double underscore, n u m Moons, equals, I Moons.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Sentence, colon. Line 2, indented once: d e f, double underscore, in it, double underscore, left parenthesis, self, comma, initial String, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, the String, equals, initial String. Line 4: Blank. Line 5, indented once: d e f get Sentence, left parenthesis, self, right parenthesis, colon. Line 6, indented twice: return self, dot, double underscore, the String. Line 7: Blank. Line 8, indented once: d e f get Words, left parenthesis, self, right parenthesis, colon. Line 9, indented twice: return self, dot, double underscore, the String, dot, split, left parenthesis, right parenthesis. Line 10: Blank. Line 11, indented once: d e f get Length, left parenthesis, self, right parenthesis, colon. Line 12, indented twice: return l e n, left parenthesis, self, dot, double underscore, the String, right parenthesis. Line 13: Blank. Line 14, indented once: d e f get N u m Words, left

parenthesis, self, right parenthesis, colon. Line 15, indented twice: return l e n, left parenthesis, self, dot, double underscore, the String, dot, split, left parenthesis, right parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f set Moons, left parenthesis, self, comma, n u m moons, right parenthesis, colon. Line 2, indented once: self, dot, double underscore, n u m Moons, equals, n u m moons, semicolon.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f add Pun c, left parenthesis, self, comma, pun c, right parenthesis, colon. Line 2, indented once: self, dot, double underscore, the String, equals, self, dot, double underscore, the String, plus, p u n c.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f, double underscore, get item, double underscore, left parenthesis, self, comma, index, right parenthesis, colon. Line 2, indented once: return self, dot, double underscore, the String, dot, split, left parenthesis, right parenthesis, left square bracket, index, right square bracket.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f, double underscore, add, double underscore, left parenthesis, self, other Sentence, right parenthesis, colon. Line 2, indented once: new String, equals, self, dot, double underscore, the String, plus, left single quote, right single quote, backslash. Line 3, indented thrice: plus, other Sentence, dot, get Sentence, left parenthesis, right parenthesis. Line 4, indented once: return Sentence, left parenthesis, new String, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Die, colon. Line 2, indented once: d e f, double underscore, in it, double underscore, left parenthesis, self, comma, I n u m sides, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, n u m Sides, equals, I n u m sides. Line 4, indented twice: self, dot, double underscore, c u r r Roll, equals, self, dot, roll, left parenthesis, right parenthesis. Line 5, indented once: d e f roll, left parenthesis, self, right parenthesis, colon. Line 6, indented twice: import random. Line 7, indented twice: self, dot, double underscore, c u r r Roll, equals, random, dot, rand i n t, left parenthesis, 1, comma, self, dot, double underscore, n u m Sides, right parenthesis. Line 8, indented twice: return self, dot, double underscore, c u r r Roll. Line 9, indented once: d e f get C u r r Roll, left parenthesis, self, right parenthesis, colon. Line 10, indented twice: return self, dot, double underscore, c u r r Roll. Line 11, indented once: d e f, double underscore, s t r, double underscore, left parenthesis, self, right parenthesis, colon. Line 12, indented twice: return s t r, left parenthesis, self, dot, double underscore, c u r r Roll, right parenthesis. Line 13, indented once: d e f, double underscore, e

q, double underscore, left parenthesis, self, comma, other Die, right parenthesis, colon. Line 14, indented twice: return self, dot, get C u r r Roll, left parenthesis, right parenthesis, equals, equals, other Die, dot, get C u r r Roll, left parenthesis, right parenthesis. Line 15, indented once: d e f, double underscore, n e, double underscore, left parenthesis, self, comma, other Die, right parenthesis, colon. Line 16, indented twice: return self, dot, get C u r r Roll, left parenthesis, right parenthesis, not equal to, other Die, dot, get C u r r Roll, left parenthesis, right parenthesis. Line 17, indented once: d e f, double underscore, l t, double underscore, left parenthesis, self, comma, other Die, right parenthesis, colon. Line 18, indented twice: return self, dot, get C u r r Roll, left parenthesis, right parenthesis, left angle bracket, other Die, dot, get C u r r Roll, left parenthesis, right parenthesis. Line 19, indented once: d e f, double underscore, l e, double underscore, left parenthesis, self, other Die, right parenthesis, colon. Line 20, indented twice: return self, dot, get C u r r Roll, left parenthesis, right parenthesis, left angle bracket, equals, other Die, dot, get C u r r Roll, left parenthesis, right parenthesis. Line 21, indented once: d e f, double underscore, g t, double underscore, left parenthesis, self, comma, other Die, right parenthesis, colon. Line 22, indented twice: return self, dot, get C u r r Roll, left parenthesis, right parenthesis, right angle bracket, other Die, dot, get C u r r Roll, left parenthesis, right parenthesis. Line 23, indented once: d e f, double underscore, g e, double underscore, left parenthesis, self, comma, other Die, right parenthesis, colon. Line 24, indented twice: return self, dot, get C u r r Roll, left parenthesis, right parenthesis, right angle bracket, equals, other Die, dot, get C u r r Roll, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d 1, equals, Die, left parenthesis, 6, right parenthesis. Line 2: d 2, equals, Die, left parenthesis, 6, right parenthesis. Line 3: d 1, dot, roll, left parenthesis, right parenthesis. Line 4: print, left parenthesis, d 1, right parenthesis. Line 5: hash, 6. Line 6: d 2, dot, roll, left parenthesis, right parenthesis. Line 7: print, left parenthesis, d 2, right parenthesis. Line 8: hash, 3 Line 9: d 1, equals, equals, d 2. Line 10: hash, False. Line 11: d 1, not equal to, d 2. Line 12: hash, True. Line 13: d 1, left angle bracket, d 2. Line 14: hash, False. Line 15: d 1, left angle bracket, equals, d 2. Line 16: hash, False. Line 17: d 1, right angle bracket, d 2. Line 18: hash, True. Line 19: d 1, right angle bracket, equals, d 2. Line 20: hash, True.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Bank Account, colon. Line 2, indented once: d e f, double underscore, in it, double underscore, left parenthesis, self, comma, i i d, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, i d N u m, equals, i i d. Line 4, indented twice: self, dot, double underscore, balance, equals, 0. Line 5, indented once: d e f deposit, left parenthesis, self, comma, amount, right parenthesis, colon. Line 6, indented twice: self, dot, double underscore, balance, equals, self, dot, double underscore, balance, plus, amount. Line 7, indented once: d e f withdraw, left parenthesis, self, amount, right parenthesis, colon. Line 8, indented twice: self, dot, double underscore, balance, equals, self, dot, double underscore, balance, minus, amount. Line 9, indented once: d e f get Balance, left parenthesis, self, right parenthesis, colon. Line 10, indented twice: return self, dot, double underscore, balance. Line 11, indented once: d e f

transfer To, left parenthesis, self, comma, other Acct, comma, amount, right parenthesis, colon. Line 12, indented twice: self, dot, withdraw, left parenthesis, amount, right parenthesis. Line 13, indented twice: other Acct, dot, deposit, left parenthesis, amount, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Sun, colon. Line 2, indented once: d e f, double underscore, in it, double underscore, left parenthesis, self, comma, i name, comma, i rad, comma, i m, comma, i temp, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, name, equals, i name. Line 4, indented twice: self, dot, double underscore, radius, equals, i rad. Line 5, indented twice: self, dot, double underscore, mass, equals, i m. Line 6, indented twice: self, dot, double underscore, temp, equals, i temp. Line 7, indented once: d e f get Mass, left parenthesis, self, right parenthesis, colon. Line 8, indented twice: return self, dot, double underscore, mass. Line 9, indented once: d e f get Radius, left parenthesis, self, right parenthesis, colon. Line 10, indented twice: return self, dot, double underscore, radius. Line 11, indented once: d e f get Temperature, left parenthesis, self, right parenthesis, colon. Line 12, indented twice: return self, dot, double underscore, temp. Line 13, indented once: d e f get Volume, left parenthesis, self, right parenthesis, colon. Line 14, indented twice: import math. Line 15, indented twice: v, equals, 4, forward slash, 3, asterisk, math, dot, pi, asterisk, self, dot, double underscore, radius, asterisk, asterisk, 3. Line 16, indented twice: return v. Line 17, indented once: d e f get Surface Area, left parenthesis, self, right parenthesis, colon. Line 18, indented twice: import math. Line 19, indented twice: s a, equals, 4,

asterisk, math, dot, p i, asterisk, self, dot, double underscore, radius, asterisk, asterisk, 2. Line 20, indented twice: return s a. Line 21, indented once: d e f get Density, left parenthesis, self, right parenthesis, colon. Line 22, indented twice: d, equals, self, dot, double underscore, mass, forward slash, self, dot, get Volume, left parenthesis, right parenthesis. Line 23, indented twice: return d. Line 24, indented once: d e f, double underscore, s t r, double underscore, left parenthesis, self, right parenthesis, colon. Line 25, indented twice: return self, dot, double underscore, name. Line 26, indented once: d e f set Name, left parenthesis, self, comma, new Name, right parenthesis, colon. Line 27, indented twice: self, dot, double underscore, name, equals, new Name. Line 28, indented once: d e f set Radius, left parenthesis, self, comma, new Rad, right parenthesis, colon. Line 29, indented twice: self, dot, double underscore, radius, equals, new Rad.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f get Total Mass, left parenthesis, self, right parenthesis, colon. Line 2, indented once: total Mass, equals, self, dot, the Sun, dot, get Mass, left parenthesis, right parenthesis. Line 3, indented once: for a Planet in self, dot, planets, colon. Line 4, indented twice: total Mass, equals, total Mass, plus, a Planet, dot, get Mass, left parenthesis, right parenthesis. Line 5, indented once: return total Mass.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f get Nearest, left parenthesis, self, right

parenthesis, colon. Line 2, indented once: nearest, equals, self, dot, planets, left square bracket, 0, right square bracket. Line 3, indented once: for a Planet in self, dot, planets, colon. Line 4, indented twice: if a Planet, dot, get Distance, left parenthesis, right parenthesis, left angle bracket, nearest, dot, get Distance, left parenthesis, right parenthesis, colon. Line 5, indented thrice: nearest, equals, a Planet. Line 6, indented once: return nearest. Line 7: Blank. Line 8: d e f get Farthest, left parenthesis, self, right parenthesis, colon. Line 9, indented once: farthest, equals, self, dot, planets, left square bracket, 0, right square bracket. Line 10, indented once: for a Planet in self, dot, planets, colon. Line 11, indented twice: if a Planet, dot, get Distance, left parenthesis, right parenthesis, right angle bracket, nearest, dot, get Distance, left parenthesis, right parenthesis, colon. Line 12, indented thrice: farthest, equals, a Planet. Line 13, indented once: return farthest. Line 14: Blank. Line 15: To call these methods with a Solar System named s s, colon. Line 16: Blank. Line 17, indented once: print, left parenthesis, s s, dot, get Nearest, left parenthesis, right parenthesis, right parenthesis. Line 18, indented once: print, left parenthesis, s s, dot, get Farthest, left parenthesis, right parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: for a C o l in range, left parenthesis, self, dot, double underscore, max X, right parenthesis, colon. Line 2, indented once: col, equals, left square bracket, right square bracket. Line 3, indented once: for a Row in range, left parenthesis, self, dot, double underscore, max Y, right parenthesis, colon. Line 4, indented twice: c o l, dot, append, left parenthesis, None, right parenthesis. Line 5, indented once: self,

dot, double underscore, grid, dot, append, left parenthesis, c o l, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f try To Move, left parenthesis, self, right parenthesis, colon. Line 2, indented once: offset List, equals, left square bracket, left parenthesis, negative 1, comma, 1, right parenthesis, comma, left parenthesis, 0, comma, 1, right parenthesis, comma, left parenthesis, 1, comma, 1, right parenthesis, comma. Line 3, indented thrice: left parenthesis, negative 1, comma, 0, right parenthesis, comma, space, left parenthesis, 1, comma, 0, right parenthesis, comma. Line 4, indented thrice: left parenthesis, negative 1, comma, negative 1, right parenthesis, comma, left parenthesis, 0, comma negative 1, right parenthesis, comma, left parenthesis, 1, negative 1, right parenthesis, right square bracket. Line 5, indented once: try Count, equals, 0. Line 6, indented once: good Move, equals, False. Line 7, indented once: while try Count, left angle bracket, 4 and not good Move, colon. Line 8, indented twice: random Offset Index, equals, random, dot, rand range, left parenthesis, l e n, left parenthesis, offset List, right parenthesis, right parenthesis. Line 9, indented twice: random Offset, equals, offset List, left square bracket, random Offset Index, right square bracket. Line 10, indented twice: next X, equals, self, dot, double underscore, x P o s, plus, random Offset, left square bracket, 0, right square bracket. Line 11, indented twice: next Y, equals, self, dot, double underscore, y P o s, plus, random Offset, left square bracket, 1, right square bracket. Line 12, indented twice: while not, left parenthesis, 0, left angle bracket, equals, next X, left angle bracket, self, dot, double underscore, world, dot, get Max

X, left parenthesis, right parenthesis, and, backslash. Line 13, indented thrice: 0, left angle bracket, equals, next Y, left angle bracket, self, dot, double underscore, world, dot, get Max Y, left parenthesis, right parenthesis, right parenthesis, colon. Line 14, indented thrice: random Offset Index, equals, backslash. Line 15, indented quarce: random, dot, rand range, left parenthesis, l e n, left parenthesis, offset List, right parenthesis, right parenthesis. Line 16, indented thrice: random Offset, equals, offset List, left square bracket, random Offset Index, right square bracket. Line 17, indented thrice: next X, equals, self, dot, double underscore, x P o s, plus, random Offset, left square bracket, 0, right square bracket. Line 18, indented thrice: next Y, equals, self, dot, double underscore, y P o s, plus, random Offset, left square bracket, 1, right square bracket. Line 19: Blank. Line 20, indented thrice: if self, dot, double underscore, world, dot, empty Location, left parenthesis, next X, comma, next Y, right parenthesis, colon. Line 21, indented quarce: self, dot, move, left parenthesis, next X, comma, next Y, right parenthesis. Line 22, indented quarce: good Move, equals, True. Line 23, indented thrice: else, colon. Line 24, indented quarce: try Count, equals, try Count, plus, 1.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Bear, colon. Line 2, indented once: d e f, double underscore, i n i t, double underscore, left parenthesis, self, right parenthesis, colon. Line 3, indented twice: self, dot, double underscore, turtle, equals, turtle, dot, Turtle, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, double underscore, turtle, dot, up, left parenthesis, right parenthesis. Line 5, indented twice: self, dot, double underscore, turtle, dot, hide turtle, left parenthesis, right parenthesis. Line 6,

indented twice: self, dot, double underscore, turtle, dot, shape, left parenthesis, left double quotes, Bear, dot, g i f, right double quotes, right parenthesis. Line 7: Blank. Line 8, indented twice: self, dot, double underscore, x P o s, equals, 0. Line 9, indented twice: self, dot, double underscore, y P o s, equals, 0. Line 10, indented twice: self, dot, double underscore, world, equals, None. Line 11: Blank. Line 12, indented twice: self, dot, double underscore, starve Tick, equals, 0 Line 13, indented twice: self, dot, double underscore, breed Tick, equals, 0. Line 14, indented twice: self, dot, double underscore, starve Threshold, equals, 10. Line 15, indented twice: self, dot, double underscore, breed Threshold, equals, 8. Line 16: Blank. Line 17: hash, other methods. Line 18: Blank. Line 19, indented once: d e f live A Little, left parenthesis, self, right parenthesis, colon. Line 20, indented twice: self, dot, double underscore, breed Tick, equals, self, dot, double underscore, breed Tick, plus, 1. Line 21, indented twice: if self, dot, double underscore, breed Tick, right angle bracket, equals, breed Threshold, colon. Line 22, indented thrice: self, dot, try To Breed, left parenthesis, right parenthesis. Line 23: Blank. Line 24, indented twice: self, dot, try To Eat, left parenthesis, right parenthesis. Line 25: Blank. Line 26, indented twice: if self, dot, double underscore, starve Tick, equals, equals, starve Threshold, colon. Line 27, indented thrice: self, dot, double underscore, world, dot, del Thing, left parenthesis, self, right parenthesis. Line 28, indented twice: else, colon. Line 29, indented thrice: self, dot, try To Move, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Point, left parenthesis, Geometric Object,

right parenthesis, colon. Line 2, indented once: d e f, double underscore, i n i t, double underscore, left parenthesis, self, comma, x, comma, y, right parenthesis, colon. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, i n i t, double underscore, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, p o s, equals, left parenthesis, x, comma, y, right parenthesis. Line 5, indented once: def get Coord, left parenthesis, self, right parenthesis, colon. Line 6, indented twice: return self, dot, pos. Line 7, indented once: d e f get X, left parenthesis, self, right parenthesis, colon. Line 8, indented twice: return self, dot, p o s, left square bracket, 0, right square bracket. Line 9, indented once: d e f get Y, left parenthesis, self, right parenthesis, colon. Line 10, indented twice: return self, dot, p o s, left square bracket, 1, right square bracket. Line 11, indented once: d e f, underscore, draw, left parenthesis, self, comma, turtle, right parenthesis, colon. Line 12, indented twice: turtle, dot, go to, left parenthesis, self, dot, p o s, left square bracket, 0, right square bracket, comma, self, dot, p o s, left square bracket, 1, right square bracket, right parenthesis. Line 13, indented twice: turtle, dot, dot, left parenthesis, self, dot, line Width, comma, self, dot, line Color, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: from draw import, asterisk. Line 2: my Canvas, equals, Canvas, left parenthesis, 800, comma, 600, right parenthesis. Line 3: x, equals, 0. Line 4: for i in range, left parenthesis, 360, right parenthesis, colon. Line 5, indented once: y, equals, 100, asterisk, math, dot, sin, left parenthesis, math, dot, radians, left parenthesis, i, right parenthesis, right parenthesis.

Line 6, indented once: my Point, equals, Point, left parenthesis, x, comma, y, right parenthesis. Line 7, indented once: my Point, dot, set Width, left parenthesis, 3, right parenthesis. Line 8, indented once: my Point, dot, set Color, left parenthesis, comma, left single quote, comma, blue, comma, right single quote, comma, right parenthesis. Line 9, indented once: my Canvas, dot, draw, left parenthesis, my Point, right parenthesis. Line 10, indented once: x, equals, x, plus, 1.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f undraw, left parenthesis, self, comma, g Object, right parenthesis, colon. Line 2, indented once: self, dot, double underscore, visible Objects, dot, remove, left parenthesis, g Object, right parenthesis. Line 3, indented once: g Object, dot, set Visible, left parenthesis, False, right parenthesis. Line 4, indented once: self, dot, draw All, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Polygon, left parenthesis, Shape, right parenthesis, colon. Line 2, indented once: d e f, double underscore, i n i t, double underscore, left parenthesis, self, comma, p list, right parenthesis, colon. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, i n i t, double underscore, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, double underscore, corners, equals, p list. Line 5: Blank. Line 6, indented once: d e f, underscore, draw, left parenthesis, self, comma, a turtle, right parenthesis, colon.

Line 7, indented twice: a Turtle, dot, color, left parenthesis, self, dot, get Color, left parenthesis, right parenthesis, right parenthesis. Line 8, indented twice: a Turtle, dot, width, left parenthesis, self, dot, get Width, left parenthesis, right parenthesis, right parenthesis. Line 9, indented twice: a Turtle, dot, go to, left parenthesis, self, dot, double underscore, corners, left square bracket, 0, right square bracket, dot, get Coord, left parenthesis, right parenthesis, right parenthesis. Line 10, indented twice: a Turtle, dot, down, left parenthesis, right parenthesis. Line 11, indented twice: if self, dot, get Fill Color, left parenthesis, right parenthesis, not equal to, None, colon. Line 12, indented thrice: a Turtle, dot, fill color, left parenthesis, self, dot, get Fill Color, left parenthesis, right parenthesis, right parenthesis. Line 13, indented thrice: a Turtle, dot, begin, underscore, fill, left parenthesis, right parenthesis. Line 14, indented twice: for c Index in range, left parenthesis, 1, comma, l e n, left parenthesis, self, dot, double underscore, corners, right parenthesis, right parenthesis, colon. Line 15, indented thrice: a Turtle, dot, go to, left parenthesis, self, dot, double underscore, corners, left square bracket, c Index, right square bracket, dot, get Coord, left parenthesis, right parenthesis, right parenthesis. Line 16, indented twice: a Turtle, dot, go to, left parenthesis, self, dot, double underscore, corners, left square bracket, 0, right square bracket, dot, get Coord, left parenthesis, right parenthesis, right parenthesis. Line 17, indented twice: if self, dot, get Fill Color, left parenthesis, right parenthesis, not equal to, None, colon. Line 18, indented thrice: a Turtle, dot, end, underscore, fill, left parenthesis, right parenthesis. Line 19: Blank. Line 20, indented once: d e f add Point, left parenthesis, self, comma, a Point, right parenthesis, colon. Line 21, indented twice: self, dot, double underscore, corners, dot, append, left parenthesis, a Point, right parenthesis. Line 22: Blank. Line 23, indented once: d e f get

Points, left parenthesis, self, right parenthesis, colon. Line 24, indented twice: return self, dot, double underscore, corners.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Rectangle, left parenthesis, Polygon, right parenthesis, colon. Line 2, indented once: d e f, double underscore, i n i t, double underscore, left parenthesis, self, comma, lower Left, comma, upper Right, right parenthesis, colon. Line 3, indented twice: corner List, equals, left square bracket, right square bracket. Line 4, indented twice: corner List, dot, append, left parenthesis, lower Left, right parenthesis. Line 5, indented twice: corner List, dot, append, left parenthesis, Point, left parenthesis, lower Left, dot, get X, left parenthesis, right parenthesis, comma, upper Right, dot, get Y, left parenthesis, right parenthesis, right parenthesis, right parenthesis. Line 6, indented twice: corner List, dot, append, left parenthesis, upper Right, right parenthesis. Line 7, indented twice: corner List, dot, append, left parenthesis, Point, left parenthesis, upper Right, dot, get X, left parenthesis, right parenthesis, comma, lower Left, dot, get Y, left parenthesis, right parenthesis, right parenthesis, right parenthesis. Line 8, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, i n i t, double underscore, left parenthesis, corner List, right parenthesis. Line 9: Blank. Line 10, indented once: d e f get Lower Left, left parenthesis, self, right parenthesis, colon. Line 11, indented twice: return lower Left. Line 12: Blank. Line 13, indented once: d e f get Upper Right, left parenthesis, self, right parenthesis, colon. Line 14, indented twice: return to upper Right.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: from threading import Thread. Line 2: import time Line 3: class Event Handler, left parenthesis, Thread, right parenthesis, colon. Line 4, indented once: d e f, double underscore, i n i t, double underscore, left parenthesis, self, right parenthesis, colon. Line 5, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, i n i t, double underscore, left parenthesis, right parenthesis. Line 6, indented twice: self, dot, double underscore, queue, equals, left square bracket, right square bracket. Line 7, indented twice: self, dot, double underscore, event Keeper, equals, left brace, right brace. Line 8: Blank. Line 9, indented once: d e f add Event, left parenthesis, self, comma, event Name, comma, event Data, right parenthesis, colon. Line 10, indented twice: self, dot, double underscore, queue, dot, append, left parenthesis, left parenthesis, event Name, comma, event Data, right parenthesis, right parenthesis, hash, function, and param. Line 11: Blank. Line 12, indented once: d e f register Callback, left parenthesis, self, comma, event, comma, func, right parenthesis, colon. Line 13, indented twice: self, dot, double underscore, event Keeper, left square bracket, event, right square bracket, equals, func. Line 14, indented once: d e f run, left parenthesis, self, right parenthesis, colon. Line 15, indented twice: while, left parenthesis, True, right parenthesis, colon. Line 16, indented thrice: if l e n, left parenthesis, self, dot, double underscore, queue, right parenthesis, right angle bracket, 0, colon. Line 17, indented thrice: next Event, comma, event Data, equals, self, dot, double underscore, queue, dot, pop, left parenthesis, 0, right parenthesis. Line 18, indented thrice: call Back, equals, self, dot, double underscore, event Keeper, left square bracket, next Event, right square bracket. Line 19, indented thrice: call Back, left parenthesis, event Data, right parenthesis, hash, pass param to

function. Line 20, indented twice: else, colon. Line 21, indented thrice: time, dot, sleep, left parenthesis, 1, right parenthesis, hash, pause for 1 second. Line 22: Blank. Line 23, indented once: def my Key, left parenthesis, key, right parenthesis, colon. Line 24, indented twice: print, left parenthesis, comma, left single quote, comma, The, left brace, 0, right brace, key was pressed, comma, right single quote, comma, dot, format, left parenthesis, key, right parenthesis, right parenthesis. Line 25: Blank. Line 26, indented once: e h, equals, Event Handler, left parenthesis, right parenthesis. Line 27, indented once: e h, dot, add Event, left parenthesis, left single quote, key, right single quote, comma, left single quote, j, right single quote, comma, right parenthesis. Line 28, indented once: e h, dot, register Callback, left parenthesis, left single quote, key, right single quote, comma, my Key, right parenthesis. Line 29, indented once: e h, dot, start, left parenthesis, right parenthesis. Line 30, indented once: hash, The j key was pressed.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: class Etch, left parenthesis, Turtle, right parenthesis, colon. Line 2, indented once: d e f, double underscore, i n i t, double underscore, left parenthesis, self, right parenthesis, colon. Line 3, indented twice: super, left parenthesis, right parenthesis, dot, double underscore, i n i t, double underscore, left parenthesis, right parenthesis. Line 4, indented twice: self, dot, double underscore, screen, equals, self, dot, get screen, left parenthesis, right parenthesis. Line 5, indented twice: self, dot, color, left parenthesis, left single quote, blue, right single quote, right parenthesis. Line 6, indented twice: self, dot, pen size, left parenthesis, 2, right parenthesis. Line 7, indented

twice: self, dot, speed, left parenthesis, 0, right parenthesis. Line 8, indented twice: self, dot, double underscore, distance, equals, 5. Line 9, indented twice: self, dot, double underscore, turn, equals, 10. Line 10. self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, f w d, comma, left double quotes, Up, right double quotes, right parenthesis. Line 11, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, b k w d, comma, left double quotes, Down, right double quotes, right parenthesis. Line 12, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, left 5, comma, left double quotes, Left, right double quotes, right parenthesis. Line 13, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, right 5, comma, left double quotes, Right, right double quotes, right parenthesis. Line 14, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, quit comma, left double quotes, q, right double quotes, right parenthesis. Line 15, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, tail up, comma, left double quotes, u, right double quotes, right parenthesis. Line 16, indented twice: self, dot, double underscore, screen, dot, on key, left parenthesis, self, dot, tail down, comma, left double quotes, d, right double quotes, right parenthesis. Line 17, indented twice: self, dot, double underscore, screen, dot, listen, left parenthesis, right parenthesis. Line 18, indented twice: self, dot, main, left parenthesis, right parenthesis. Line 19: Blank. Line 20, indented once: def f w d, left parenthesis, self, right parenthesis, colon. Line 21, indented twice: self, dot, forward, left parenthesis, self, dot, double underscore, distance, right parenthesis. Line 22: Blank. Line 23, indented once: def b k w d, left parenthesis, self, right parenthesis, colon. Line 24, indented twice: self, dot, backward, left parenthesis, self, dot, double underscore, distance,

right parenthesis. Line 25: Blank. Line 26, indented once: d e f left 5, left parenthesis, self, right parenthesis, colon. Line 27, indented twice: self, dot, left, left parenthesis, self, dot, double underscore, turn, right parenthesis. Line 28: Blank. Line 29, indented once: d e f right 5, left parenthesis, self, right parenthesis, colon. Line 30, indented twice: self, dot, right, left parenthesis, self, dot, double underscore, turn, right parenthesis. Line 31: Blank. Line 32, indented once: d e f quit, left parenthesis, self, right parenthesis, colon. Line 33, indented twice: self, dot, sc3reen, dot, bye, left parenthesis, right parenthesis. Line 34: Blank. Line 35, indented once: d e f tail up, left parenthesis, self, right parenthesis, colon. Line 36, indented twice: self, dot, up, left parenthesis, right parenthesis. Line 37: Blank. Line 38, indented once: d e f tail down, left parenthesis, self, right parenthesis, colon. Line 39, indented twice: self, dot, down, left parenthesis, right parenthesis. Line 40: Blank. Line 41, indented once: d e f main, left parenthesis, self, right parenthesis, colon. Line 42, indented twice: main loop, left parenthesis, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: if self, dot, double underscore, n u m Turtles, right angle bracket, equals, self, dot, double underscore, max Turtles, colon. Line 2, indented once: self, dot, double underscore, big T screen, dot, on click, left parenthesis, self, dot, move To, right parenthesis. Line 3: Blank. Line 4: and then add this method to the class, colon. Line 5: d e f move To, left parenthesis, self, comma, x, comma, y, right parenthesis, colon. Line 6, indented once: for t in self, dot, double underscore, turtle List, colon. Line 7, indented twice: t, dot, set heading, left

parenthesis, t, dot, towards, left parenthesis, x, comma, y, right parenthesis, right parenthesis. Line 8, indented twice: t, dot, forward, left parenthesis, 10, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: 1, dot, In the underscore, underscore, i n i t, underscore, underscore, method, colon. Line 2, indented once: Add a, underscore, underscore, moving instance variable initialized to True, colon. Line 3, indented once: self, dot, double underscore, moving, equals, True. Line 4, indented once: Add a callback for on click, colon. Line 5, indented once: self, dot, on click, left parenthesis, self, dot, double underscore, stop Go, right parenthesis. Line 6: 2 dot, Add this method, colon. Line 7, indented once: d e f, double underscore, stop Go, left parenthesis, self, comma, x, comma, y, right parenthesis, colon. Line 8, indented twice: for t in Animated Turtle, dot, double underscore, all Turtles, colon. Line 9, indented thrice: if t, dot, distance, left parenthesis, x, comma, y, right parenthesis, left angle bracket, 10, colon. Line 10, indented quarce: if t, dot, double underscore, moving, colon. Line 11, indented quince: t, dot, double underscore, moving, equals, False. Line 12, indented quarce: else, colon. Line 13, indented quince: t, dot, double underscore, moving, equals, True. Line 14. 3, dot, Change the move One Step method to, colon. Line 15, indented once: d e f, double underscore, move One Step, left parenthesis, self, right parenthesis, colon. Line 16, indented twice: if self, dot, double underscore, moving, colon. Line 17, indented thrice: self, dot, double underscore, compute New Heading, left parenthesis, right parenthesis. Line 18, indented thrice: self, dot, forward, left parenthesis, 5, right parenthesis. Line 19, indented thrice: self,

dot, double underscore, check Collisions, left parenthesis, right parenthesis. Line 20, indented twice: self, dot, double underscore, s c r, dot, on a timer, left parenthesis, self, dot, double underscore, move One Step, comma, 100, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: a, dot, Add this Score Keeper class, colon. Line 2, indented once: class Score Keeper, left parenthesis, Turtle, right parenthesis, colon. Line 3, indented twice: game Score, equals, 0, hash, static variable. Line 4: Blank. Line 5, indented twice: at, static method. Line 6, indented twice: def alter Score, left parenthesis, amount, right parenthesis, colon. Line 7, indented thrice: Score Keeper, dot, game Score, equals, Score Keeper, dot, game Score, plus, amount. Line 8: Blank. Line 9, indented twice: d e f, double underscore, i n i t, double underscore, left parenthesis, self, comma, score X, comma, score Y, right parenthesis, colon. Line 10, indented thrice: super, left parenthesis, right parenthesis, dot, double underscore, i n i t, double underscore, left parenthesis, right parenthesis. Line 11, indented thrice: self, dot, hide turtle, left parenthesis, right parenthesis. Line 12, indented thrice: self, dot, up, left parenthesis, right parenthesis. Line 13, indented thrice: self, dot, go to, left parenthesis, score X, comma, score Y, right parenthesis. Line 14, indented thrice: self, dot, down, left parenthesis, right parenthesis. Line 15, indented thrice: self, dot, double underscore, old Score, equals, negative 1, hash, force display first time. Line 16, indented thrice: self, dot, write Score, left parenthesis, right parenthesis. Line 17: Blank. Line 18, indented twice: d e f write Score, left parenthesis, self, right parenthesis, colon. Line 19, indented thrice: if Score Keeper, dot,

game Score, not equal to, self, dot, double underscore, old Score, colon. Line 20, indented quarce: hash, update if changed. Line 21, indented quarce: self, dot, clear, left parenthesis, right parenthesis. Line 22, indented quarce: self, dot, write, left parenthesis, left single quote, Score, colon, right single quote, plus, s t r, left parenthesis, Score Keeper, dot, game Score, right parenthesis, comma. Line 23, indented quince: move, equals, False, right parenthesis. Line 24, indented thrice: self, dot, double underscore, old Score, equals, Score Keeper, dot, game Score.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: b, dot, Change the Drone Invaders class, colon. Line 2, indented once: class Drone Invaders, colon. Line 3, indented twice: def, double underscore, i n i t, double underscore, left parenthesis, self, comma, x Min, comma, x Max, comma, y Min, comma, y Max, right parenthesis, colon. Line 4, indented thrice: self, dot, double underscore, x Min, equals, x Min. Line 5, indented thrice: self, dot, double underscore, x Max, equals, x Max. Line 6, indented thrice: self, dot, double underscore, y Min, equals, y Min. Line 7, indented thrice: self, dot, double underscore, y Max, equals, y Max. Line 8: Blank. Line 9, indented twice: d e f play, left parenthesis, self, right parenthesis, colon. Line 10, indented thrice: self, dot, main Win, equals, Laser Cannon, left parenthesis, self, dot, double underscore, x Min, comma, self, dot, double underscore, x Max, comma. Line 11, indented quarce: self, dot, double underscore, y Min, comma, self, dot, double underscore, y Max, right parenthesis, dot, get screen, left parenthesis, right parenthesis. Line 12, indented thrice: self, dot, main Win, dot, b g color, left parenthesis, left single quote, light green, right single quote, right

parenthesis. Line 13, indented thrice: self, dot, main Win, dot, set world coordinates, left parenthesis, self, dot, double underscore, x Min, comma, self, dot, double underscore, y Min, comma. Line 14, indented quince: self, dot, double underscore, x Max, comma, self, dot, double underscore, y Max, right parenthesis. Line 15, indented thrice: hash, start Score Keeper. Line 16, indented thrice: self, dot, double underscore, score Keeper, equals, Score Keeper, left parenthesis, self, dot, double underscore, x Min, plus, 50, comma. Line 17, indented quince: self, dot, double underscore, y Max, negative 50, right parenthesis. Line 18: Blank. Line 19, indented thrice: self, dot, main Win, dot, on timer, left parenthesis, self, dot, add Drone, comma, 200, right parenthesis. Line 20, indented thrice: self, dot, main Win, dot, listen, left parenthesis, right parenthesis. Line 21, indented thrice: main loop, left parenthesis, right parenthesis. Line 22, indented twice: d e f add Drone, left parenthesis, self, right parenthesis, colon. Line 23, indented thrice: if l e n, left parenthesis, Drone, dot, get Drones, left parenthesis, right parenthesis, right parenthesis, left angle bracket, 7, colon. Line 24, indented quarce: Drone, left parenthesis, 1, comma, self, dot, double underscore, x Min, comma, self, dot, double underscore, x Max, comma. Line 25, indented quince: self, dot, double underscore, y Min, comma, self, dot, double underscore, y Max, right parenthesis. Line 26, indented thrice: self, dot, double underscore, score Keeper, dot, write Score, left parenthesis, right parenthesis, hash, write score every 200 m s. Line 27, indented thrice: self, dot, main Win, dot, on timer, left parenthesis, self, dot, add Drone, comma, 200, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f move, left parenthesis, self, right parenthesis, colon. Line 2, indented once: exploded, equals, False. Line 3, indented once: self, dot, forward, left parenthesis, self, dot, get Speed, left parenthesis, right parenthesis, right parenthesis. Line 4, indented once: for a in Drone, dot, get Drones, left parenthesis, right parenthesis, colon. Line 5, indented twice: if self, dot, distance, left parenthesis, a, right parenthesis, left angle bracket, 5, colon. Line 6, indented thrice: a, dot, remove, left parenthesis, right parenthesis. Line 7, indented thrice: Score Keeper, dot, alter Score, left parenthesis, 10, right parenthesis. Line 8, indented thrice: exploded, equals, True. Line 9, indented once: if self, dot, out Of Bounds, left parenthesis, right parenthesis, or exploded, colon. Line 10, indented twice: self, dot, remove, left parenthesis, right parenthesis. Line 11, indented once: else, colon. Line 12, indented twice: self, dot, get screen, left parenthesis, right parenthesis, dot, on timer, left parenthesis, self, dot, move, comma, 100, right parenthesis.

Back to Code

Program code. In this program code, the following keywords are merged. Line 1: d e f move, left parenthesis, self, right parenthesis, colon. Line 2, indented once: self, dot, forward, left parenthesis, self, dot, get Speed, left parenthesis, right parenthesis, right parenthesis. Line 3, indented once: if self, dot, out Of Bounds, left parenthesis, right parenthesis, colon. Line 4, indented twice: self, dot, remove, left parenthesis, right parenthesis. Line 5, indented twice: Score Keeper, dot, alter Score, left parenthesis, negative 10, right parenthesis. Line 6, indented once: else, colon. Line 7, indented twice: self, dot, get

screen, left parenthesis, right parenthesis, dot, on timer, left parenthesis, self, dot, move, comma, 200, right parenthesis.

Back to Code