**2ND EDITION**

**AUTOMATE**

**AUTOMATE**

**THE BORING STUFF**

**THE BORING STUFF WITH PYTHON**

**WITH PYTHON**

**PRACTICAL PROGRAMMING FOR TOTAL BEGINNERS**

**AL SWEIGART**

**AUTOMATE THE BORING STUFF WITH PYTHON**

**A U T O M A T E T H E B O R I N G S T U F F WITH PYTHON 2ND EDITION**

**P r a c t i c a l P r o g r a m m i n g for Total Beginners**

by Al Sweigart

San Francisco

**AUTOMATE THE BORING STUFF WITH PYTHON, 2ND EDITION.** Copyright © 2020 by Al Sweigart.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-10: 1-59327-992-2

ISBN-13: 978-1-59327-992-9

Publisher: William Pollock

Production Editor: Laurel Chun

Cover Illustration: Josh Ellingson

Interior Design: Octopod Studios

Developmental Editors: Frances Saux and Jan Cash

Technical Reviewers: Ari Lacenski and Philip James

Copyeditors: Kim Wimpsett, Britt Bogan, and Paula L. Fleming

Compositors: Susan Glinert Stevens and Danielle Foster

Proofreaders: Lisa Devoto Farrell and Emelie Burnette

Indexer: BIM Indexing and Proofreading Services

For information on distribution, translations, or bulk sales,

please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103

phone: 1.415.863.9900; info@nostarch.com

www.nostarch.com

The Library of Congress Control Number for the first edition is: 2014953114

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 United States License. To view a copy of this license, visit *http://creativecommons.org/licenses/by-nc-sa/3.0/us/* or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

For my nephew Jack

**About the Author**

Al Sweigart is a software developer and tech book author. Python is his favorite programming language, and he is the developer of several open source modules for it. His other books are freely available under a Creative Commons license on his website *https://inventwithpython.com/*. His cat now weighs 11 pounds.

**About the Tech Reviewer**

Philip James has been working in Python for over a decade and is a frequent speaker in the Python community. He speaks on topics ranging from Unix fundamentals to open source social networks. Philip is a Core Contributor to the BeeWare project and lives in the San Francisco Bay Area with his partner Nic and her cat River.

**BRIEF CONTENTS**

Acknowledgments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxv Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxvii

PART I: PYTHON PROGRAMMING BASICS

Chapter 1: Python Basics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3 Chapter 2: Flow Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21 Chapter 3: Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57 Chapter 4: Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77 Chapter 5: Dictionaries and Structuring Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111

Chapter 6: Manipulating Strings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 129

PART II: AUTOMATING TASKS

Chapter 7: Pattern Matching with Regular Expressions . . . . . . . . . . . . . . . . . . . . . . . . . . 161 Chapter 8: Input Validation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 187 Chapter 9: Reading and Writing Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 201 Chapter 10: Organizing Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 231 Chapter 11: Debugging . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 249 Chapter 12: Web Scraping . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 267 Chapter 13: Working with Excel Spreadsheets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 301 Chapter 14: Working with Google Sheets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 329 Chapter 15: Working with PDF and Word Documents . . . . . . . . . . . . . . . . . . . . . . . . . . 347 Chapter 16: Working with CSV Files and JSON Data . . . . . . . . . . . . . . . . . . . . . . . . . . 371 Chapter 17: Keeping Time, Scheduling Tasks, and Launching Programs . . . . . . . . . . . . . . 389 Chapter 18: Sending Email and Text Messages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 415

Chapter 19: Manipulating Images . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 447 Chapter 20: Controlling the Keyboard and Mouse with GUI Automation . . . . . . . . . . . . . 473 Appendix A: Installing Third-Party Modules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 507 Appendix B: Running Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 511 Appendix C: Answers to the Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 517 Index . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 531

**viii** Brief Contents

**CONTENTS IN DETAIL**

**ACKNOWLEDGMENTS xxv**

**INTRODUCTION xxvii**

Whom Is This Book For? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxviii Conventions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxviii What Is Programming? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxix

What Is Python? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxx Programmers Don’t Need to Know Much Math . . . . . . . . . . . . . . . . . . . . . . . xxx You Are Not Too Old to Learn Programming . . . . . . . . . . . . . . . . . . . . . . . xxxi Programming Is a Creative Activity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxi

About This Book . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxi Downloading and Installing Python . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxiii Downloading and Installing Mu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxiv Starting Mu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxiv Starting IDLE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxv The Interactive Shell . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxv Installing Third-Party Modules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxvi How to Find Help . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxvi Asking Smart Programming Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxviii Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxxix

**PART I: PYTHON PROGRAMMING BASICS**

**1**

**PYTHON BASICS 3**

Entering Expressions into the Interactive Shell . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 The Integer, Floating-Point, and String Data Types . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7 String Concatenation and Replication . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7 Storing Values in Variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

Assignment Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9 Variable Names . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10 Your First Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11 Dissecting Your Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13 Comments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13 The print() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13 The input() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14 Printing the User’s Name . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14 The len() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14 The str(), int(), and float() Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

**2**

**FLOW CONTROL 21**

Boolean Values . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22 Comparison Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23 Boolean Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

Binary Boolean Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25 The not Operator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26 Mixing Boolean and Comparison Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26 Elements of Flow Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27 Conditions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27 Blocks of Code . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27 Program Execution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28 Flow Control Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28 if Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28 else Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29 elif Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30 while Loop Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35 break Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39 continue Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40 for Loops and the range() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44 Importing Modules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47 from import Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48 Ending a Program Early with the sys.exit() Function . . . . . . . . . . . . . . . . . . . . . . . . . . 49 A Short Program: Guess the Number . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49 A Short Program: Rock, Paper, Scissors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

**3**

**FUNCTIONS 57**

def Statements with Parameters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59 Define, Call, Pass, Argument, Parameter . . . . . . . . . . . . . . . . . . . . . . . . . . . 59 Return Values and return Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60 The None Value . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61 Keyword Arguments and the print() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62 The Call Stack . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63 Local and Global Scope . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65 Local Variables Cannot Be Used in the Global Scope . . . . . . . . . . . . . . . . . . 66 Local Scopes Cannot Use Variables in Other Local Scopes . . . . . . . . . . . . . . . 67 Global Variables Can Be Read from a Local Scope . . . . . . . . . . . . . . . . . . . . 67 Local and Global Variables with the Same Name . . . . . . . . . . . . . . . . . . . . . 68 The global Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68 Exception Handling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71 A Short Program: Zigzag . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75

**x** Contents in Detail

Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76 The Collatz Sequence . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76 Input Validation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

**4**

**LISTS 77**

The List Data Type . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78 Getting Individual Values in a List with Indexes . . . . . . . . . . . . . . . . . . . . . . . 78 Negative Indexes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80 Getting a List from Another List with Slices . . . . . . . . . . . . . . . . . . . . . . . . . . 80 Getting a List’s Length with the len() Function . . . . . . . . . . . . . . . . . . . . . . . . 81 Changing Values in a List with Indexes . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81 List Concatenation and List Replication . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81 Removing Values from Lists with del Statements . . . . . . . . . . . . . . . . . . . . . . . 82

Working with Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82 Using for Loops with Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84 The in and not in Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84 The Multiple Assignment Trick . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85 Using the enumerate() Function with Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . 86 Using the random.choice() and random.shuffle() Functions with Lists . . . . . . . . 86

Augmented Assignment Operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87 Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88 Finding a Value in a List with the index() Method . . . . . . . . . . . . . . . . . . . . . 88 Adding Values to Lists with the append() and insert() Methods . . . . . . . . . . . . 89 Removing Values from Lists with the remove() Method . . . . . . . . . . . . . . . . . . 90 Sorting the Values in a List with the sort() Method . . . . . . . . . . . . . . . . . . . . . 90 Reversing the Values in a List with the reverse() Method . . . . . . . . . . . . . . . . . 91 Example Program: Magic 8 Ball with a List . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92 Sequence Data Types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93 Mutable and Immutable Data Types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94 The Tuple Data Type . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96 Converting Types with the list() and tuple() Functions . . . . . . . . . . . . . . . . . . . 97 References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97 Identity and the id() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 99 Passing References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100 The copy Module’s copy() and deepcopy() Functions . . . . . . . . . . . . . . . . . . 101 A Short Program: Conway’s Game of Life . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 106 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 106 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107 Comma Code . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107 Coin Flip Streaks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 107 Character Picture Grid . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108

**5**

**DICTIONARIES AND STRUCTURING DATA 111**

The Dictionary Data Type . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111 Dictionaries vs. Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112 The keys(), values(), and items() Methods.......................... 114

Contents in Detail **xi**

Checking Whether a Key or Value Exists in a Dictionary . . . . . . . . . . . . . . . 115 The get() Method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116 The setdefault() Method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116

Pretty Printing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118 Using Data Structures to Model Real-World Things . . . . . . . . . . . . . . . . . . . . . . . . . . 119 A Tic-Tac-Toe Board . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120 Nested Dictionaries and Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 126 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127 Chess Dictionary Validator . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127 Fantasy Game Inventory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127 List to Dictionary Function for Fantasy Game Inventory . . . . . . . . . . . . . . . . . 128

**6**

**MANIPULATING STRINGS 129**

Working with Strings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 130 String Literals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 130 Indexing and Slicing Strings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132 The in and not in Operators with Strings . . . . . . . . . . . . . . . . . . . . . . . . . . 133

Putting Strings Inside Other Strings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 134 Useful String Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 134 The upper(), lower(), isupper(), and islower() Methods . . . . . . . . . . . . . . . . . 135 The isX() Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136 The startswith() and endswith() Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . 138 The join() and split() Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 138 Splitting Strings with the partition() Method . . . . . . . . . . . . . . . . . . . . . . . . . 140 Justifying Text with the rjust(), ljust(), and center() Methods . . . . . . . . . . . . . . 140 Removing Whitespace with the strip(), rstrip(), and lstrip() Methods . . . . . . . . 142 Numeric Values of Characters with the ord() and chr() Functions . . . . . . . . . . . . . . . . 143 Copying and Pasting Strings with the pyperclip Module . . . . . . . . . . . . . . . . . . . . . . 143 Project: Multi-Clipboard Automatic Messages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 144 Step 1: Program Design and Data Structures . . . . . . . . . . . . . . . . . . . . . . . 144 Step 2: Handle Command Line Arguments . . . . . . . . . . . . . . . . . . . . . . . . . 145 Step 3: Copy the Right Phrase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146 Project: Adding Bullets to Wiki Markup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147 Step 1: Copy and Paste from the Clipboard . . . . . . . . . . . . . . . . . . . . . . . . 147 Step 2: Separate the Lines of Text and Add the Star . . . . . . . . . . . . . . . . . . 148 Step 3: Join the Modified Lines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 148 A Short Progam: Pig Latin . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 149 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154 Table Printer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154 Zombie Dice Bots . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155

**xii** Contents in Detail

**PART II: AUTOMATING TASKS**

**7**

**PATTERN MATCHING WITH REGULAR EXPRESSIONS 161**

Finding Patterns of Text Without Regular Expressions . . . . . . . . . . . . . . . . . . . . . . . . 162 Finding Patterns of Text with Regular Expressions . . . . . . . . . . . . . . . . . . . . . . . . . . . 164 Creating Regex Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 165 Matching Regex Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 165 Review of Regular Expression Matching . . . . . . . . . . . . . . . . . . . . . . . . . . . 166 More Pattern Matching with Regular Expressions . . . . . . . . . . . . . . . . . . . . . . . . . . . 166 Grouping with Parentheses . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 166 Matching Multiple Groups with the Pipe . . . . . . . . . . . . . . . . . . . . . . . . . . . 168 Optional Matching with the Question Mark . . . . . . . . . . . . . . . . . . . . . . . . 168 Matching Zero or More with the Star . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 169 Matching One or More with the Plus . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 170 Matching Specific Repetitions with Braces . . . . . . . . . . . . . . . . . . . . . . . . . 170 Greedy and Non-greedy Matching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171 The findall() Method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171 Character Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 172 Making Your Own Character Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 173 The Caret and Dollar Sign Characters . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 174 The Wildcard Character . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 175 Matching Everything with Dot-Star . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 175 Matching Newlines with the Dot Character . . . . . . . . . . . . . . . . . . . . . . . . 176 Review of Regex Symbols . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 177 Case-Insensitive Matching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 177 Substituting Strings with the sub() Method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178 Managing Complex Regexes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 178 Combining re.IGNORECASE, re.DOTALL, and re.VERBOSE . . . . . . . . . . . . . . . . . . . 179 Project: Phone Number and Email Address Extractor . . . . . . . . . . . . . . . . . . . . . . . . 179 Step 1: Create a Regex for Phone Numbers . . . . . . . . . . . . . . . . . . . . . . . . 180 Step 2: Create a Regex for Email Addresses . . . . . . . . . . . . . . . . . . . . . . . . 181 Step 3: Find All Matches in the Clipboard Text . . . . . . . . . . . . . . . . . . . . . . 182 Step 4: Join the Matches into a String for the Clipboard . . . . . . . . . . . . . . . . 183 Running the Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 183 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 183 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 184 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 184 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 186 Date Detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 186 Strong Password Detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 186 Regex Version of the strip() Method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 186

**8**

**INPUT VALIDATION 187**

The PyInputPlus Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 188 The min, max, greaterThan, and lessThan Keyword Arguments . . . . . . . . . . . 190 The blank Keyword Argument . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 191

Contents in Detail **xiii**

The limit, timeout, and default Keyword Arguments . . . . . . . . . . . . . . . . . . . 191 The allowRegexes and blockRegexes Keyword Arguments . . . . . . . . . . . . . . 192 Passing a Custom Validation Function to inputCustom() . . . . . . . . . . . . . . . . 193

Project: How to Keep an Idiot Busy for Hours . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 194 Project: Multiplication Quiz . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 196 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 197 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 198 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 198

Sandwich Maker . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 198 Write Your Own Multiplication Quiz . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 199

**9**

**READING AND WRITING FILES 201**

Files and File Paths . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 201 Backslash on Windows and Forward Slash on macOS and Linux . . . . . . . . . 202 Using the / Operator to Join Paths . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 204 The Current Working Directory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 205 The Home Directory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206 Absolute vs. Relative Paths . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206 Creating New Folders Using the os.makedirs() Function . . . . . . . . . . . . . . . . 207 Handling Absolute and Relative Paths . . . . . . . . . . . . . . . . . . . . . . . . . . . . 208 Getting the Parts of a File Path . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209 Finding File Sizes and Folder Contents . . . . . . . . . . . . . . . . . . . . . . . . . . . . 212 Modifying a List of Files Using Glob Patterns . . . . . . . . . . . . . . . . . . . . . . . 213 Checking Path Validity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 214

The File Reading/Writing Process . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 215 Opening Files with the open() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . 217 Reading the Contents of Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 217 Writing to Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 218

Saving Variables with the shelve Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 219 Saving Variables with the pprint.pformat() Function . . . . . . . . . . . . . . . . . . . . . . . . . 220 Project: Generating Random Quiz Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 221

Step 1: Store the Quiz Data in a Dictionary........................ 222 Step 2: Create the Quiz File and Shuffle the Question Order . . . . . . . . . . . . 223 Step 3: Create the Answer Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 224 Step 4: Write Content to the Quiz and Answer Key Files . . . . . . . . . . . . . . . 225

Project: Updatable Multi-Clipboard . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 226 Step 1: Comments and Shelf Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 227 Step 2: Save Clipboard Content with a Keyword . . . . . . . . . . . . . . . . . . . . 227 Step 3: List Keywords and Load a Keyword’s Content . . . . . . . . . . . . . . . . . 228

Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 229 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 229 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 229

Extending the Multi-Clipboard . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 229 Mad Libs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 230 Regex Search . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 230

**xiv** Contents in Detail

**10**

**ORGANIZING FILES 231**

The shutil Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 232 Copying Files and Folders . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 232 Moving and Renaming Files and Folders . . . . . . . . . . . . . . . . . . . . . . . . . . 233 Permanently Deleting Files and Folders . . . . . . . . . . . . . . . . . . . . . . . . . . . 234 Safe Deletes with the send2trash Module . . . . . . . . . . . . . . . . . . . . . . . . . . 235

Walking a Directory Tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 235 Compressing Files with the zipfile Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237 Reading ZIP Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 238 Extracting from ZIP Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 238 Creating and Adding to ZIP Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 239 Project: Renaming Files with American-Style Dates to European-Style Dates . . . . . . . . . 240 Step 1: Create a Regex for American-Style Dates . . . . . . . . . . . . . . . . . . . . 240 Step 2: Identify the Date Parts from the Filenames . . . . . . . . . . . . . . . . . . . . 241 Step 3: Form the New Filename and Rename the Files . . . . . . . . . . . . . . . . . 242 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 243 Project: Backing Up a Folder into a ZIP File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 243 Step 1: Figure Out the ZIP File’s Name . . . . . . . . . . . . . . . . . . . . . . . . . . . 243 Step 2: Create the New ZIP File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 245 Step 3: Walk the Directory Tree and Add to the ZIP File . . . . . . . . . . . . . . . 245 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 246 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 246 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 247 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 247 Selective Copy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 247 Deleting Unneeded Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 247 Filling in the Gaps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 248

**11**

**DEBUGGING 249**

Raising Exceptions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 250 Getting the Traceback as a String . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 251 Assertions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 253

Using an Assertion in a Traffic Light Simulation . . . . . . . . . . . . . . . . . . . . . . 254 Logging . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 255 Using the logging Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 256 Don’t Debug with the print() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 257 Logging Levels . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 258 Disabling Logging . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 259 Logging to a File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 259 Mu’s Debugger . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 260 Continue . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 261 Step In . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 261 Step Over . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 261 Step Out . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 261 Stop . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 261

Contents in Detail **xv**

Debugging a Number Adding Program . . . . . . . . . . . . . . . . . . . . . . . . . . . 261 Breakpoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 263 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 264 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 265 Practice Project . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 266 Debugging Coin Toss . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 266

**12**

**WEB SCRAPING 267**

Project: mapIt.py with the webbrowser Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . 268 Step 1: Figure Out the URL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 268 Step 2: Handle the Command Line Arguments . . . . . . . . . . . . . . . . . . . . . . 269 Step 3: Handle the Clipboard Content and Launch the Browser . . . . . . . . . . 270 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 270

Downloading Files from the Web with the requests Module . . . . . . . . . . . . . . . . . . . . 271 Downloading a Web Page with the requests.get() Function . . . . . . . . . . . . . 271 Checking for Errors . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 272

Saving Downloaded Files to the Hard Drive . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 273 HTML . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 274 Resources for Learning HTML . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 274 A Quick Refresher . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 274 Viewing the Source HTML of a Web Page . . . . . . . . . . . . . . . . . . . . . . . . . 275 Opening Your Browser’s Developer Tools . . . . . . . . . . . . . . . . . . . . . . . . . . 276 Using the Developer Tools to Find HTML Elements . . . . . . . . . . . . . . . . . . . . 278 Parsing HTML with the bs4 Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 279 Creating a BeautifulSoup Object from HTML . . . . . . . . . . . . . . . . . . . . . . . . 280 Finding an Element with the select() Method . . . . . . . . . . . . . . . . . . . . . . . . 280 Getting Data from an Element’s Attributes . . . . . . . . . . . . . . . . . . . . . . . . . 282 Project: Opening All Search Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 283 Step 1: Get the Command Line Arguments and Request the Search Page . . . . 283 Step 2: Find All the Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 284 Step 3: Open Web Browsers for Each Result . . . . . . . . . . . . . . . . . . . . . . . 285 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 285 Project: Downloading All XKCD Comics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 286 Step 1: Design the Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 287 Step 2: Download the Web Page............................... 288 Step 3: Find and Download the Comic Image . . . . . . . . . . . . . . . . . . . . . . 288 Step 4: Save the Image and Find the Previous Comic . . . . . . . . . . . . . . . . . 289 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 290 Controlling the Browser with the selenium Module . . . . . . . . . . . . . . . . . . . . . . . . . . 291 Starting a selenium-Controlled Browser . . . . . . . . . . . . . . . . . . . . . . . . . . . 291 Finding Elements on the Page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 293 Clicking the Page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 295 Filling Out and Submitting Forms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 295 Sending Special Keys . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 296 Clicking Browser Buttons . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 297 More Information on Selenium . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 297

**xvi** Contents in Detail

Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 297 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 297 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 298

Command Line Emailer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 298 Image Site Downloader . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 298 2048 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 299 Link Verification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 299

**13**

**WORKING WITH EXCEL SPREADSHEETS 301**

Excel Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 302 Installing the openpyxl Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 302 Reading Excel Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 302

Opening Excel Documents with OpenPyXL . . . . . . . . . . . . . . . . . . . . . . . . . 303 Getting Sheets from the Workbook . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 304 Getting Cells from the Sheets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 304 Converting Between Column Letters and Numbers . . . . . . . . . . . . . . . . . . . . 306 Getting Rows and Columns from the Sheets . . . . . . . . . . . . . . . . . . . . . . . . 306 Workbooks, Sheets, Cells . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 308

Project: Reading Data from a Spreadsheet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 308 Step 1: Read the Spreadsheet Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 309 Step 2: Populate the Data Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 310 Step 3: Write the Results to a File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 311 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 312

Writing Excel Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 313 Creating and Saving Excel Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . 313 Creating and Removing Sheets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 314 Writing Values to Cells . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 314

Project: Updating a Spreadsheet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 315 Step 1: Set Up a Data Structure with the Update Information . . . . . . . . . . . . 316 Step 2: Check All Rows and Update Incorrect Prices . . . . . . . . . . . . . . . . . . 317 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 317

Setting the Font Style of Cells . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 318 Font Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 318 Formulas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 319 Adjusting Rows and Columns . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 320

Setting Row Height and Column Width . . . . . . . . . . . . . . . . . . . . . . . . . . . 321 Merging and Unmerging Cells . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 322 Freezing Panes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 322

Charts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 324 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 325 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 326 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 326

Multiplication Table Maker . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 326 Blank Row Inserter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 327 Spreadsheet Cell Inverter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 327 Text Files to Spreadsheet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 328 Spreadsheet to Text Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 328

Contents in Detail **xvii**

**14**

**WORKING WITH GOOGLE SHEETS 329**

Installing and Setting Up EZSheets..................................... 329 Obtaining Credentials and Token Files . . . . . . . . . . . . . . . . . . . . . . . . . . . 330 Revoking the Credentials File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 332

Spreadsheet Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 332 Creating, Uploading, and Listing Spreadsheets . . . . . . . . . . . . . . . . . . . . . . 333 Spreadsheet Attributes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 334 Downloading and Uploading Spreadsheets . . . . . . . . . . . . . . . . . . . . . . . . 335 Deleting Spreadsheets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 336

Sheet Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 336 Reading and Writing Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 337 Creating and Deleting Sheets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 341 Copying Sheets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 343

Working with Google Sheets Quotas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 343 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 344 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 344 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 345

Downloading Google Forms Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 345 Converting Spreadsheets to Other Formats . . . . . . . . . . . . . . . . . . . . . . . . . 345 Finding Mistakes in a Spreadsheet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 345

**15**

**WORKING WITH PDF AND WORD DOCUMENTS 347**

PDF Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 347 Extracting Text from PDFs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 348 Decrypting PDFs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 349 Creating PDFs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 350

Project: Combining Select Pages from Many PDFs . . . . . . . . . . . . . . . . . . . . . . . . . . 355 Step 1: Find All PDF Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 355 Step 2: Open Each PDF . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 356 Step 3: Add Each Page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 357 Step 4: Save the Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 357 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 358

Word Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 358 Reading Word Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 359 Getting the Full Text from a .docx File . . . . . . . . . . . . . . . . . . . . . . . . . . . . 360 Styling Paragraph and Run Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 361 Creating Word Documents with Nondefault Styles . . . . . . . . . . . . . . . . . . . 362 Run Attributes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 362 Writing Word Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 364 Adding Headings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 366 Adding Line and Page Breaks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 366 Adding Pictures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 367

Creating PDFs from Word Documents . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 367 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 368 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 368

**xviii** Contents in Detail

Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 369 PDF Paranoia . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 369 Custom Invitations as Word Documents........................... 369 Brute-Force PDF Password Breaker . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 370

**16**

**WORKING WITH CSV FILES AND JSON DATA 371**

The csv Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 372 reader Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 373 Reading Data from reader Objects in a for Loop . . . . . . . . . . . . . . . . . . . . . 374 writer Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 374 The delimiter and lineterminator Keyword Arguments . . . . . . . . . . . . . . . . . . 375 DictReader and DictWriter CSV Objects . . . . . . . . . . . . . . . . . . . . . . . . . . 376

Project: Removing the Header from CSV Files . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 378 Step 1: Loop Through Each CSV File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 378 Step 2: Read in the CSV File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 379 Step 3: Write Out the CSV File Without the First Row . . . . . . . . . . . . . . . . . 380 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 381

JSON and APIs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 381 The json Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 382 Reading JSON with the loads() Function . . . . . . . . . . . . . . . . . . . . . . . . . . 382 Writing JSON with the dumps() Function . . . . . . . . . . . . . . . . . . . . . . . . . . 383 Project: Fetching Current Weather Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 383 Step 1: Get Location from the Command Line Argument . . . . . . . . . . . . . . . . 384 Step 2: Download the JSON Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 385 Step 3: Load JSON Data and Print Weather . . . . . . . . . . . . . . . . . . . . . . . . 385 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 387 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 387 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 387 Practice Project . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 388 Excel-to-CSV Converter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 388

**17**

**KEEPING TIME, SCHEDULING TASKS,**

**AND LAUNCHING PROGRAMS 389**

The time Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 390 The time.time() Function...................................... 390 The time.sleep() Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 391

Rounding Numbers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 392 Project: Super Stopwatch . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 392 Step 1: Set Up the Program to Track Times . . . . . . . . . . . . . . . . . . . . . . . . . 393 Step 2: Track and Print Lap Times . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 393 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 394 The datetime Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 395 The timedelta Data Type . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 396 Pausing Until a Specific Date . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 397 Converting datetime Objects into Strings . . . . . . . . . . . . . . . . . . . . . . . . . . 398 Converting Strings into datetime Objects . . . . . . . . . . . . . . . . . . . . . . . . . . 399

Contents in Detail **xix**

Review of Python’s Time Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 399 Multithreading . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 400 Passing Arguments to the Thread’s Target Function . . . . . . . . . . . . . . . . . . . 402 Concurrency Issues . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 403 Project: Multithreaded XKCD Downloader . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 403 Step 1: Modify the Program to Use a Function . . . . . . . . . . . . . . . . . . . . . . 403 Step 2: Create and Start Threads . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 405 Step 3: Wait for All Threads to End . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 405 Launching Other Programs from Python . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 406 Passing Command Line Arguments to the Popen() Function . . . . . . . . . . . . . . 408 Task Scheduler, launchd, and cron . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 408 Opening Websites with Python . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 409 Running Other Python Scripts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 409 Opening Files with Default Applications . . . . . . . . . . . . . . . . . . . . . . . . . . . 409 Project: Simple Countdown Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 410 Step 1: Count Down . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 410 Step 2: Play the Sound File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 411 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 412 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 412 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 412 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 413 Prettified Stopwatch . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 413 Scheduled Web Comic Downloader............................. 413

**18**

**SENDING EMAIL AND TEXT MESSAGES 415**

Sending and Receiving Email with the Gmail API . . . . . . . . . . . . . . . . . . . . . . . . . . . 416 Enabling the Gmail API . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 416 Sending Mail from a Gmail Account............................. 417 Reading Mail from a Gmail Account . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 418 Searching Mail from a Gmail Account . . . . . . . . . . . . . . . . . . . . . . . . . . . . 419 Downloading Attachments from a Gmail Account.................... 419

SMTP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 420 Sending Email . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 420 Connecting to an SMTP Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 421 Sending the SMTP “Hello” Message . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 422 Starting TLS Encryption . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 422 Logging In to the SMTP Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 423 Sending an Email . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 423 Disconnecting from the SMTP Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 424 IMAP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 424 Retrieving and Deleting Emails with IMAP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 424 Connecting to an IMAP Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 425 Logging In to the IMAP Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 426 Searching for Email . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 426 Fetching an Email and Marking It as Read . . . . . . . . . . . . . . . . . . . . . . . . . 430 Getting Email Addresses from a Raw Message . . . . . . . . . . . . . . . . . . . . . . 431

**xx** Contents in Detail

Getting the Body from a Raw Message . . . . . . . . . . . . . . . . . . . . . . . . . . . 431 Deleting Emails . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 432 Disconnecting from the IMAP Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 433

Project: Sending Member Dues Reminder Emails . . . . . . . . . . . . . . . . . . . . . . . . . . . 433 Step 1: Open the Excel File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 434 Step 2: Find All Unpaid Members . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 435 Step 3: Send Customized Email Reminders . . . . . . . . . . . . . . . . . . . . . . . . . 436

Sending Text Messages with SMS Email Gateways . . . . . . . . . . . . . . . . . . . . . . . . . 437 Sending Text Messages with Twilio . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 438 Signing Up for a Twilio Account . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 439 Sending Text Messages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 439 Project: “Just Text Me” Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 441 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 442 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 443 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 443 Random Chore Assignment Emailer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 443 Umbrella Reminder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 444 Auto Unsubscriber . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 444 Controlling Your Computer Through Email . . . . . . . . . . . . . . . . . . . . . . . . . 444

**19**

**MANIPULATING IMAGES 447**

Computer Image Fundamentals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 448 Colors and RGBA Values . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 448 Coordinates and Box Tuples . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 449

Manipulating Images with Pillow . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 450 Working with the Image Data Type . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 451 Cropping Images . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 453 Copying and Pasting Images onto Other Images . . . . . . . . . . . . . . . . . . . . . 454 Resizing an Image . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 456 Rotating and Flipping Images . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 457 Changing Individual Pixels . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 459

Project: Adding a Logo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 460 Step 1: Open the Logo Image . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 461 Step 2: Loop Over All Files and Open Images . . . . . . . . . . . . . . . . . . . . . . 462 Step 3: Resize the Images . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 463 Step 4: Add the Logo and Save the Changes . . . . . . . . . . . . . . . . . . . . . . . 463 Ideas for Similar Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 465

Drawing on Images . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 465 Drawing Shapes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 466 Drawing Text . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 468

Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 469 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 470 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 470

Extending and Fixing the Chapter Project Programs . . . . . . . . . . . . . . . . . . . 470 Identifying Photo Folders on the Hard Drive . . . . . . . . . . . . . . . . . . . . . . . . 471 Custom Seating Cards . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 472

Contents in Detail **xxi**

**20**

**CONTROLLING THE KEYBOARD AND MOUSE**

**WITH GUI AUTOMATION 473**

Installing the pyautogui Module . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 474 Setting Up Accessibility Apps on macOS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 474 Staying on Track . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 475

Pauses and Fail-Safes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 475 Shutting Down Everything by Logging Out . . . . . . . . . . . . . . . . . . . . . . . . . 475 Controlling Mouse Movement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 475 Moving the Mouse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 477 Getting the Mouse Position . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 477 Controlling Mouse Interaction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 478 Clicking the Mouse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 478 Dragging the Mouse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 479 Scrolling the Mouse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 480 Planning Your Mouse Movements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 481 Working with the Screen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 482 Getting a Screenshot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 482 Analyzing the Screenshot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 483 Image Recognition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 484 Getting Window Information . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 485 Obtaining the Active Window . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 486 Other Ways of Obtaining Windows . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 487 Manipulating Windows . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 487 Controlling the Keyboard . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 489 Sending a String from the Keyboard . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 489 Key Names . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 490 Pressing and Releasing the Keyboard . . . . . . . . . . . . . . . . . . . . . . . . . . . . 491 Hotkey Combinations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 492 Setting Up Your GUI Automation Scripts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 492 Review of the PyAutoGUI Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 493 Project: Automatic Form Filler . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 495 Step 1: Figure Out the Steps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 496 Step 2: Set Up Coordinates . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 496 Step 3: Start Typing Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 498 Step 4: Handle Select Lists and Radio Buttons . . . . . . . . . . . . . . . . . . . . . . . 499 Step 5: Submit the Form and Wait . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 500 Displaying Message Boxes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 501 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 502 Practice Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 503 Practice Projects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 503 Looking Busy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 503 Using the Clipboard to Read a Text Field . . . . . . . . . . . . . . . . . . . . . . . . . . 503 Instant Messenger Bot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 504 Game-Playing Bot Tutorial . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 505

**xxii** Contents in Detail

**A**

**INSTALLING THIRD-PARTY MODULES 507**

The pip Tool . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 507 Installing Third-Party Modules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 508 Installing Modules for the Mu Editor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 510

**B**

**RUNNING PROGRAMS 511**

Running Programs from the Terminal Window . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 511 Running Python Programs on Windows . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 513 Running Python Programs on macOS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 514 Running Python Programs on Ubuntu Linux . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 514 Running Python Programs with Assertions Disabled . . . . . . . . . . . . . . . . . . . . . . . . . 515

**C**

**ANSWERS TO THE PRACTICE QUESTIONS 517**

Chapter 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 518 Chapter 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 518 Chapter 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 520 Chapter 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 520 Chapter 5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 521 Chapter 6 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 521 Chapter 7 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 522 Chapter 8 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 523 Chapter 9 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 523 Chapter 10 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 524 Chapter 11 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 524 Chapter 12 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 525 Chapter 13 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 526 Chapter 14 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 527 Chapter 15 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 527 Chapter 16 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 528 Chapter 17 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 528 Chapter 18 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 529 Chapter 19 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 529 Chapter 20 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 529

**INDEX 531**

Contents in Detail **xxiii**

**ACKNOWLEDGMENTS**

It’s misleading to have just my name on the cover. I couldn’t have written a book like this without the help of a lot of people. I’d like to thank my publisher, Bill Pollock; my editors, Laurel Chun, Leslie Shen, Greg Poulos, Jennifer Griffith-Delgado, and Frances Saux; and the rest of the staff at No Starch Press for their invaluable help. Thanks to my tech reviewers, Ari Lacenski and Philip James, for great suggestions, edits, and support.

Many thanks to everyone at the Python Software Foundation for their great work. The Python community is the best one I’ve found in the tech industry.

Finally, I would like to thank my family, friends, and the gang at Shotwell’s for not minding the busy life I’ve had while writing this book. Cheers!

**INTRODUCTION**

“You’ve just done in two hours what it takes the three of us two days to do.” My college roommate was working at a retail electronics store in the early 2000s. Occasionally, the store would receive a spreadsheet of thousands of product prices from other stores. A team of three employees would print the spreadsheet onto a thick stack of paper and split it among themselves. For each product price, they would look up their store’s price and note all the products that their competitors sold for less. It usually took a couple of days.

“You know, I could write a program to do that if you have the original file for the printouts,” my roommate told them, when he saw them sitting on the floor with papers scattered and stacked all around.

After a couple of hours, he had a short program that read a competi tor’s price from a file, found the product in the store’s database, and noted whether the competitor was cheaper. He was still new to programming, so he spent most of his time looking up documentation in a programming

book. The actual program took only a few seconds to run. My roommate and his co-workers took an extra-long lunch that day.

This is the power of computer programming. A computer is like a

Swiss Army knife that you can configure for countless tasks. Many people spend hours clicking and typing to perform repetitive tasks, unaware that the machine they’re using could do their job in seconds if they gave it the right instructions.

**Whom Is This Book For?**

Software is at the core of so many of the tools we use today: nearly everyone uses social networks to communicate, many people have internet-connected computers in their phones, and most office jobs involve interacting with a computer to get work done. As a result, the demand for people who can code has skyrocketed. Countless books, interactive web tutorials, and devel oper boot camps promise to turn ambitious beginners into software engi neers with six-figure salaries.

This book is not for those people. It’s for everyone else.

On its own, this book won’t turn you into a professional software devel oper any more than a few guitar lessons will turn you into a rock star. But if you’re an office worker, administrator, academic, or anyone else who uses a computer for work or fun, you will learn the basics of programming so that you can automate simple tasks such as these:

• Moving and renaming thousands of files and sorting them into folders • Filling out online forms—no typing required

• Downloading files or copying text from a website whenever it updates • Having your computer text you custom notifications

• Updating or formatting Excel spreadsheets

• Checking your email and sending out prewritten responses

These tasks are simple but time-consuming for humans, and they’re often so trivial or specific that there’s no ready-made software to perform them. Armed with a little bit of programming knowledge, however, you can have your computer do these tasks for you.

**Conventions**

This book is not designed as a reference manual; it’s a guide for begin ners. The coding style sometimes goes against best practices (for example, some programs use global variables), but that’s a trade-off to make the code simpler to learn. This book is made for people to write throwaway code, so there’s not much time spent on style and elegance. Sophisticated programming concepts—like object-oriented programming, list compre hensions, and generators—aren’t covered because of the complexity they

**xxviii** Introduction

add. Veteran programmers may point out ways the code in this book could be changed to improve efficiency, but this book is mostly concerned with getting programs to work with the least amount of effort on your part.

**What Is Programming?**

Television shows and films often show programmers furiously typing cryptic streams of 1s and 0s on glowing screens, but modern programming isn’t that mysterious. *Programming* is simply the act of entering instructions for the computer to perform. These instructions might crunch some numbers, modify text, look up information in files, or communicate with other com puters over the internet.

All programs use basic instructions as building blocks. Here are a few of the most common ones, in English:

• “Do this; then do that.”

• “If this condition is true, perform this action; otherwise, do that action.” • “Do this action exactly 27 times.”

• “Keep doing that until this condition is true.”

You can combine these building blocks to implement more intricate decisions, too. For example, here are the programming instructions, called the *source code*, for a simple program written in the Python programming language. Starting at the top, the Python software runs each line of code (some lines are run only *if* a certain condition is true or *else* Python runs some other line) until it reaches the bottom.

❶ passwordFile = open('SecretPasswordFile.txt')

❷ secretPassword = passwordFile.read()

❸ print('Enter your password.')

typedPassword = input()

❹ if typedPassword == secretPassword:

❺ print('Access granted')

❻ if typedPassword == '12345':

❼ print('That password is one that an idiot puts on their luggage.') else:

❽ print('Access denied')

You might not know anything about programming, but you could prob ably make a reasonable guess at what the previous code does just by reading it. First, the file *SecretPasswordFile.txt* is opened ❶, and the secret password in it is read ❷. Then, the user is prompted to input a password (from the key board) ❸. These two passwords are compared ❹, and if they’re the same, the program prints *Access granted* to the screen ❺. Next, the program checks to see whether the password is *12345* ❻ and hints that this choice might not be the best for a password ❼. If the passwords are not the same, the pro gram prints *Access denied* to the screen ❽.

Introduction **xxix**

***What Is Python?***

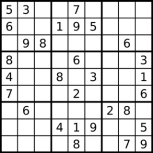
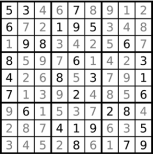
*Python* is a programming language (with syntax rules for writing what is considered valid Python code) and the Python interpreter software that reads source code (written in the Python language) and performs its instructions. You can download the Python interpreter for free at *https:// python.org/*, and there are versions for Linux, macOS, and Windows.

The name Python comes from the surreal British comedy group Monty Python, not from the snake. Python programmers are affectionately called Pythonistas, and both Monty Python and serpentine references usually pep per Python tutorials and documentation.

***Programmers Don’t Need to Know Much Math***

The most common anxiety I hear about learning to program is the notion that it requires a lot of math. Actually, most programming doesn’t require math beyond basic arithmetic. In fact, being good at programming isn’t that different from being good at solving Sudoku puzzles.

To solve a Sudoku puzzle, the numbers 1 through 9 must be filled in for each row, each column, and each 3×3 interior square of the full 9×9 board. Some numbers are provided to give you a start, and you find a solution by making deductions based on these numbers. In the puzzle shown in Figure 0-1, since 5 appears in the first and second rows, it cannot show up in these rows again. Therefore, in the upper-right grid, it must be in the third row. Since the last column also already has a 5 in it, the 5 cannot go to the right of the 6, so it must go to the left of the 6. Solving one row, column, or square will provide more clues to the rest of the puzzle, and as you fill in one group of numbers 1 to 9 and then another, you’ll soon solve the entire grid.

*Figure 0-1: A new Sudoku puzzle (left) and its solution (right). Despite using numbers, Sudoku doesn’t involve much math. (Images © Wikimedia Commons)*

Just because Sudoku involves numbers doesn’t mean you have to be good at math to figure out the solution. The same is true of program ming. Like solving a Sudoku puzzle, writing programs involves breaking

**xxx** Introduction

down a problem into individual, detailed steps. Similarly, when *debugging* programs (that is, finding and fixing errors), you’ll patiently observe what the program is doing and find the cause of the bugs. And like all skills, the more you program, the better you’ll become.

***You Are Not Too Old to Learn Programming***

The second most common anxiety I hear about learning to program is that people think they’re too old to learn it. I read many internet comments from folks who think it’s too late for them because they are already (gasp!) 23 years old. This is clearly not “too old” to learn to program: many people learn much later in life.

You don’t need to have started as a child to become a capable pro

grammer. But the image of programmers as whiz kids is a persistent one. Unfortunately, I contribute to this myth when I tell others that I was in grade school when I started programming.

However, programming is much easier to learn today than it was in the 1990s. Today, there are more books, better search engines, and many more online question-and-answer websites. On top of that, the programming lan guages themselves are far more user-friendly. For these reasons, **everything I learned about programming in the years between grade school and high school graduation could be learned today in about a dozen weekends**. My

head start wasn’t really much of a head start.

It’s important to have a “growth mindset” about programming—in other words, understand that people develop programming skills through practice. They aren’t just born as programmers, and being unskilled at pro gramming now is not an indication that you can never become an expert.

***Programming Is a Creative Activity***

Programming is a creative task, like painting, writing, knitting, or con structing LEGO castles. Like painting a blank canvas, making software has many constraints but endless possibilities.

The difference between programming and other creative activities is that when programming, you have all the raw materials you need in your computer; you don’t need to buy any additional canvas, paint, film, yarn, LEGO bricks, or electronic components. A decade-old computer is more than powerful enough to write programs. Once your program is written, it

can be copied perfectly an infinite number of times. A knit sweater can only be worn by one person at a time, but a useful program can easily be shared online with the entire world.

**About This Book**

The first part of this book covers basic Python programming concepts, and the second part covers various tasks you can have your computer automate. Each chapter in the second part has project programs for you to study. Here’s a brief rundown of what you’ll find in each chapter.

Introduction **xxxi**

**Part I: Python Programming Basics**

**Chapter 1: Python Basics** Covers expressions, the most basic type of Python instruction, and how to use the Python interactive shell soft ware to experiment with code.

**Chapter 2: Flow Control** Explains how to make programs decide which instructions to execute so your code can intelligently respond to different conditions.

**Chapter 3: Functions** Instructs you on how to define your own func tions so that you can organize your code into more manageable chunks. **Chapter 4: Lists** Introduces the list data type and explains how to organize data.

**Chapter 5: Dictionaries and Structuring Data** Introduces the diction ary data type and shows you more powerful ways to organize data. **Chapter 6: Manipulating Strings** Covers working with text data (called *strings* in Python).

**Part II: Automating Tasks**

**Chapter 7: Pattern Matching with Regular Expressions** Covers how Python can manipulate strings and search for text patterns with regular expressions.

**Chapter 8: Input Validation** Explains how your program can verify the information a user gives it, ensuring that the user’s data arrives in a format that won’t cause errors in the rest of the program. **Chapter 9: Reading and Writing Files** Explains how your program can read the contents of text files and save information to files on your hard drive.

**Chapter 10: Organizing Files** Shows how Python can copy, move, rename, and delete large numbers of files much faster than a human user can. Also explains compressing and decompressing files. **Chapter 11: Debugging** Shows how to use Python’s various bug finding and bug-fixing tools.

**Chapter 12: Web Scraping** Shows how to write programs that can automatically download web pages and parse them for information. This is called *web scraping*.

**Chapter 13: Working with Excel Spreadsheets** Covers programmati cally manipulating Excel spreadsheets so that you don’t have to read them. This is helpful when the number of documents you have to ana lyze is in the hundreds or thousands.

**Chapter 14: Working with Google Sheets** Covers how to read and update Google Sheets, a popular web-based spreadsheet application, using Python.

**Chapter 15: Working with PDF and Word Documents** Covers pro grammatically reading Word and PDF documents.

**xxxii** Introduction

**Chapter 16: Working with CSV Files and JSON Data** Continues to explain how to programmatically manipulate documents, now discuss ing CSV and JSON files.

**Chapter 17: Keeping Time, Scheduling Tasks, and Launching Programs** Explains how Python programs handle time and dates and how to

schedule your computer to perform tasks at certain times. Also shows how your Python programs can launch non-Python programs.

**Chapter 18: Sending Email and Text Messages** Explains how to write programs that can send emails and text messages on your behalf.

**Chapter 19: Manipulating Images** Explains how to programmatically manipulate images such as JPEG or PNG files.

**Chapter 20: Controlling the Keyboard and Mouse with GUI Automation** Explains how to programmatically control the mouse and keyboard to automate clicks and keypresses.

**Appendix A: Installing Third-Party Modules** Shows you how to

extend Python with useful additional modules.

**Appendix B: Running Programs** Shows you how to run your Python programs on Windows, macOS, and Linux from outside of the code editor.

**Appendix C: Answers to the Practice Questions** Provides answers and some additional context to the practice questions at the end of

each chapter.

**Downloading and Installing Python**

You can download Python for Windows, macOS, and Ubuntu for free at *https://python.org/downloads/*. If you download the latest version from the website’s download page, all of the programs in this book should work.

**WARNING** *Be sure to download a version of Python 3 (such as 3.8.0). The programs in this book are written to run on Python 3 and may not run correctly, if at all, on Python 2.*

On the download page, you’ll find Python installers for 64-bit and 32-bit computers for each operating system, so first figure out which installer you need. If you bought your computer in 2007 or later, it is most likely a 64-bit system. Otherwise, you have a 32-bit version, but here’s how to find out for sure:

• On Windows, select **Start****Control Panel**4**System** and check whether System Type says 64-bit or 32-bit.

• On macOS, go the Apple menu, select **About This Mac****More**

**Info**4**System Report****Hardware**, and then look at the Processor

Name field. If it says Intel Core Solo or Intel Core Duo, you have a

32-bit machine. If it says anything else (including Intel Core 2 Duo), you have a 64-bit machine.

Introduction **xxxiii**

• On Ubuntu Linux, open a Terminal and run the command **uname -m**. A response of i686 means 32-bit, and x86\_64 means 64-bit.

On Windows, download the Python installer (the filename will end with *.msi*) and double-click it. Follow the instructions the installer displays on the screen to install Python, as listed here:

1. Select **Install for All Users** and click **Next**.

2. Accept the default options for the next several windows by clicking **Next**.

On macOS, download the *.dmg* file that’s right for your version of

macOS and double-click it. Follow the instructions the installer displays on the screen to install Python, as listed here:

1. When the DMG package opens in a new window, double-click the *Python.mpkg* file. You may have to enter the administrator password. 2. Accept the default options for the next several windows by clicking **Continue** and click **Agree** to accept the license.

3. On the final window, click **Install**.

If you’re running Ubuntu, you can install Python from the Terminal by following these steps:

1. Open the Terminal window.

2. Enter **sudo apt-get install python3**.

3. Enter **sudo apt-get install idle3**.

4. Enter **sudo apt-get install python3-pip**.

**Downloading and Installing Mu**

While the *Python interpreter* is the software that runs your Python programs, the *Mu editor software* is where you’ll enter your programs, much the way you type in a word processor. You can download Mu from *https://codewith.mu/*.

On Windows and macOS, download the installer for your operating system and then run it by double-clicking the installer file. If you are on macOS, running the installer opens a window where you must drag the Mu icon to the Applications folder icon to continue the installation. If you are on Ubuntu, you’ll need to install Mu as a Python package. In that case, click the Instructions button in the Python Package section of the download page.

**Starting Mu**

Once it’s installed, let’s start Mu.

• On Windows 7 or later, click the Start icon in the lower-left corner of your screen, enter **Mu** in the search box, and select it.

**xxxiv** Introduction

• On macOS, open the Finder window, click **Applications**, and then click **mu-editor**.

• On Ubuntu, select **Applications****Accessories**4**Terminal** and then enter **python3 –m mu**.

The first time Mu runs, a Select Mode window will appear with options Adafruit CircuitPython, BBC micro:bit, Pygame Zero, and Python 3. Select **Python 3**. You can always change the mode later by clicking the Mode button at the top of the editor window.

**NOTE** *You’ll need to download Mu version 1.10.0 or later in order to install the third-party modules featured in this book. As of this writing, 1.10.0 is an alpha release and is listed on the download page as a separate link from the main download links.*

**Starting IDLE**

This book uses Mu as an editor and interactive shell. However, you can use any number of editors for writing Python code. The *Integrated Development and Learning Environment (IDLE)* software installs along with Python, and it can serve as a second editor if for some reason you can’t get Mu installed or working. Let’s start IDLE now.

• On Windows 7 or later, click the Start icon in the lower-left cor ner of your screen, enter **IDLE** in the search box, and select **IDLE**

**(Python GUI)**.

• On macOS, open the Finder window, click **Applications**, click

**Python 3.8**, and then click the IDLE icon.

• On Ubuntu, select **Applications****Accessories**4**Terminal** and then enter **idle3**. (You may also be able to click **Applications** at the top of the screen, select **Programming**, and then click **IDLE 3**.)

**The Interactive Shell**

When you run Mu, the window that appears is called the *file editor* window. You can open the *interactive shell* by clicking the REPL button. A shell is a program that lets you type instructions into the computer, much like the Terminal or Command Prompt on macOS and Windows, respectively. Python’s interactive shell lets you enter instructions for the Python interpreter software to run. The computer reads the instructions you enter and runs them immediately.

In Mu, the interactive shell is a pane in the lower half of the window with the following text:

Jupyter QtConsole 4.3.1

Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

Introduction **xxxv**

IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:

If you run IDLE, the interactive shell is the window that first appears. It should be mostly blank except for text that looks something like this:

Python 3.8.0b1 (tags/v3.8.0b1:3b5deb0116, Jun 4 2019, 19:52:55) [MSC v.1916 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information. >>>

In [1]: and >>> are called *prompts*. The examples in this book will

use the >>> prompt for the interactive shell since it’s more common. If you run Python from the Terminal or Command Prompt, they’ll use the >>> prompt, as well. The In [1]: prompt was invented by Jupyter Notebook, another popular Python editor.

For example, enter the following into the interactive shell next to the prompt:

>>> **print('Hello, world!')**

After you type that line and press enter, the interactive shell should display this in response:

>>> **print('Hello, world!')**

Hello, world!

You’ve just given the computer an instruction, and it did what you told it to do!

**Installing Third-Party Modules**

Some Python code requires your program to import modules. Some of these modules come with Python, but others are third-party modules cre ated by developers outside of the Python core dev team. Appendix A has detailed instructions on how to use the pip program (on Windows) or pip3 program (on macOS and Linux) to install third-party modules. Consult Appendix A when this book instructs you to install a particular third-party module.

**How to Find Help**

Programmers tend to learn by searching the internet for answers to their questions. This is quite different from the way many people are accustomed to learning—through an in-person teacher who lectures and can answer questions. What’s great about using the internet as a schoolroom is that there are whole communities of folks who can answer your questions.

**xxxvi** Introduction

Indeed, your questions have probably already been answered, and the answers are waiting online for you to find them. If you encounter an error message or have trouble making your code work, you won’t be the first per son to have your problem, and finding a solution is easier than you might think.

For example, let’s cause an error on purpose: enter **'42' + 3** into the interactive shell. You don’t need to know what this instruction means right now, but the result should look like this:

>>> **'42' + 3**

❶ Traceback (most recent call last):

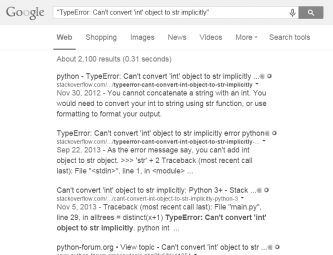
File "<pyshell#0>", line 1, in <module>

'42' + 3

❷ TypeError: Can't convert 'int' object to str implicitly

>>>

The error message ❷ appears because Python couldn’t understand your instruction. The traceback part ❶ of the error message shows the spe cific instruction and line number that Python had trouble with. If you’re not sure what to make of a particular error message, search for it online. Enter **“TypeError: Can’t convert ‘int’ object to str implicitly”** (including the quotes) into your favorite search engine, and you should see tons of links explaining what the error message means and what causes it, as shown in Figure 0-2.

*Figure 0-2: The Google results for an error message can be very helpful.*

Introduction **xxxvii**

You’ll often find that someone else had the same question as you and that some other helpful person has already answered it. No one person can know everything about programming, so an everyday part of any software developer’s job is looking up answers to technical questions.

**Asking Smart Programming Questions**

If you can’t find the answer by searching online, try asking people in a web forum such as Stack Overflow (*https://stackoverflow.com/*) or the “learn programming” subreddit at *https://reddit.com/r/learnprogramming/*. But keep in mind there are smart ways to ask programming questions that help oth ers help you. To begin with, be sure to read the FAQ sections at these web sites about the proper way to post questions.

When asking programming questions, remember to do the following:

• Explain what you are trying to do, not just what you did. This lets your helper know if you are on the wrong track.

• Specify the point at which the error happens. Does it occur at the very start of the program or only after you do a certain action?

• Copy and paste the *entire* error message and your code to *https://pastebin .com/* or *https://gist.github.com/*.

These websites make it easy to share large amounts of code with

people online, without losing any text formatting. You can then put the URL of the posted code in your email or forum post. For example, here some pieces of code I’ve posted: *https://pastebin.com/SzP2DbFx/* and *https://gist.github.com/asweigart/6912168/*.

• Explain what you’ve already tried to do to solve your problem. This tells people you’ve already put in some work to figure things out on your own. • List the version of Python you’re using. (There are some key differences between version 2 Python interpreters and version 3 Python interpret ers.) Also, say which operating system and version you’re running.

• If the error came up after you made a change to your code, explain exactly what you changed.

• Say whether you’re able to reproduce the error every time you run the program or whether it happens only after you perform certain actions. If the latter, then explain what those actions are.

Always follow good online etiquette as well. For example, don’t post your questions in all caps or make unreasonable demands of the people trying to help you.

You can find more information on how to ask for programming

help in the blog post at *https://autbor.com/help/*. You can find a list of frequently asked questions about programming at *https://www.reddit.com /r/learnprogramming/wiki/faq/* and a similar list about getting a job in software development at *https://www.reddit.com/r/cscareerquestions/wiki/ index/*.

**xxxviii** Introduction

I love helping people discover Python. I write programming tutorials on my blog at *https://inventwithpython.com/blog/*, and you can contact me with questions at *al@inventwithpython.com*. Although, you may get a faster response by posting your questions to *https://reddit.com/r/inventwithpython/*.

**Summary**

For most people, their computer is just an appliance instead of a tool. But by learning how to program, you’ll gain access to one of the most powerful tools of the modern world, and you’ll have fun along the way. Programming isn’t brain surgery—it’s fine for amateurs to experiment and make mistakes.

This book assumes you have zero programming knowledge and will teach you quite a bit, but you may have questions beyond its scope. Remember that asking effective questions and knowing how to find answers are invalu able tools on your programming journey.

Let’s begin!

Introduction **xxxix**

**PART I**

**PYTHON PROGRAMMING BASICS**

**1**

**PYTHON BASICS**

The Python programming language has

a wide range of syntactical constructions,

standard library functions, and interactive development environment features. Fortunately, you can ignore most of that; you just need to learn enough to write some handy little programs. You will, however, have to learn some basic programming concepts before you can do anything. Like a wizard in training, you might think these concepts seem arcane and tedious, but with some knowledge and practice, you’ll be able to command your computer like a magic wand and perform incredible feats.

This chapter has a few examples that encourage you to type into the *interactive shell*, also called the *REPL* (Read-Evaluate-Print Loop), which lets you run (or *execute*) Python instructions one at a time and instantly shows you the results. Using the interactive shell is great for learning what basic Python instructions do, so give it a try as you follow along. You’ll remember the things you do much better than the things you only read.

**Entering Expressions into the Interactive Shell**

You can run the interactive shell by launching the Mu editor, which you should have downloaded when going through the setup instructions in the Preface. On Windows, open the Start menu, type “Mu,” and open the Mu app. On macOS, open your Applications folder and double-click **Mu**. Click the **New** button and save an empty file as *blank.py*. When you run this blank file by clicking the **Run** button or pressing F5, it will open the interactive shell, which will open as a new pane that opens at the bottom of the Mu edi tor’s window. You should see a >>> prompt in the interactive shell.

Enter **2 + 2** at the prompt to have Python do some simple math. The Mu window should now look like this:

>>> **2 + 2**

4

>>>

In Python, 2 + 2 is called an *expression*, which is the most basic kind of programming instruction in the language. Expressions consist of *values* (such as 2) and *operators* (such as +), and they can always *evaluate* (that is, reduce) down to a single value. That means you can use expressions any where in Python code that you could also use a value.

In the previous example, 2 + 2 is evaluated down to a single value, 4. A single value with no operators is also considered an expression, though it evaluates only to itself, as shown here:

>>> **2**

2

**ERRORS ARE OK AY!**

Programs will crash if they contain code the computer can’t understand, which will cause Python to show an error message. An error message won’t break

your computer, though, so don’t be afraid to make mistakes. A *crash* just means the program stopped running unexpectedly.

If you want to know more about an error, you can search for the exact

error message text online for more information. You can also check out the

resources at *https://nostarch.com/automatestuff2/* to see a list of common

Python error messages and their meanings.

**4** Chapter 1

You can use plenty of other operators in Python expressions, too. For example, Table 1-1 lists all the math operators in Python.

Table 1-1: Math Operators from Highest to Lowest Precedence

Operator Operation Example Evaluates to . . . \*\* Exponent 2 \*\* 3 8 % Modulus/remainder 22 % 8 6 // Integer division/floored quotient 22 // 8 2 / Division 22 / 8 2.75 \* Multiplication 3 \* 5 15 - Subtraction 5 - 2 3 + Addition 2 + 2 4

The *order of operations* (also called *precedence*) of Python math operators is similar to that of mathematics. The \*\* operator is evaluated first; the \*, /, //, and % operators are evaluated next, from left to right; and the + and - operators are evaluated last (also from left to right). You can use parenthe ses to override the usual precedence if you need to. Whitespace in between the operators and values doesn’t matter for Python (except for the indenta tion at the beginning of the line), but a single space is convention. Enter the following expressions into the interactive shell:

>>> **2 + 3 \* 6**

20

>>> **(2 + 3) \* 6**

30

>>> **48565878 \* 578453**

28093077826734

>>> **2 \*\* 8**

256

>>> **23 / 7**

3.2857142857142856

>>> **23 // 7**

3

>>> **23 % 7**

2

>>> **2 + 2**

4

>>> **(5 - 1) \* ((7 + 1) / (3 - 1))**

16.0

Python Basics **5**

In each case, you as the programmer must enter the expression, but Python does the hard part of evaluating it down to a single value. Python will keep evaluating parts of the expression until it becomes a single value, as shown here:

(5 - 1) \* ((7 + 1) / (3 - 1))

4 \* ((7 + 1) / (3 - 1))

4 \* ( ) 8 / (3 - 1)

4 \* ( ) / ( ) 8 2

4 \* 4.0

16.0

These rules for putting operators and values together to form expres sions are a fundamental part of Python as a programming language, just like the grammar rules that help us communicate. Here’s an example:

**This is a grammatically correct English sentence.**

**This grammatically is sentence not English correct a.**

The second line is difficult to parse because it doesn’t follow the rules of English. Similarly, if you enter a bad Python instruction, Python won’t be able to understand it and will display a SyntaxError error message, as shown here:

>>> **5 +**

File "<stdin>", line 1

5 +

^

SyntaxError: invalid syntax

>>> **42 + 5 + \* 2**

File "<stdin>", line 1

42 + 5 + \* 2

^

SyntaxError: invalid syntax

You can always test to see whether an instruction works by entering it into the interactive shell. Don’t worry about breaking the computer: the worst that could happen is that Python responds with an error message. Professional software developers get error messages while writing code all the time.

**6** Chapter 1

**The Integer, Floating-Point, and String Data Types**

Remember that expressions are just values combined with operators, and they always evaluate down to a single value. A *data type* is a category for values, and every value belongs to exactly one data type. The most common data types in Python are listed in Table 1-2. The values -2 and 30, for example, are said to be *integer* values. The integer (or *int*) data type indicates values that are whole numbers. Numbers with a decimal point, such as 3.14, are called *floating-point numbers* (or *floats*). Note that even though the value 42 is an integer, the value 42.0 would be a floating-point number.

Table 1-2: Common Data Types

Data type Examples

Integers -2, -1, 0, 1, 2, 3, 4, 5

Floating-point numbers -1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25

Strings 'a', 'aa', 'aaa', 'Hello!', '11 cats'

Python programs can also have text values called *strings*, or *strs* (pro nounced “stirs”). Always surround your string in single quote (') characters (as in 'Hello' or 'Goodbye cruel world!') so Python knows where the string begins and ends. You can even have a string with no characters in it, '', called a *blank string* or an *empty string*. Strings are explained in greater detail in Chapter 4.

If you ever see the error message SyntaxError: EOL while scanning string literal, you probably forgot the final single quote character at the end of the string, such as in this example:

>>> **'Hello, world!**

SyntaxError: EOL while scanning string literal

**String Concatenation and Replication**

The meaning of an operator may change based on the data types of the values next to it. For example, + is the addition operator when it operates on two integers or floating-point values. However, when + is used on two string values, it joins the strings as the *string concatenation* operator. Enter the fol lowing into the interactive shell:

>>> **'Alice' + 'Bob'**

'AliceBob'

Python Basics **7**

The expression evaluates down to a single, new string value that com bines the text of the two strings. However, if you try to use the + operator on a string and an integer value, Python will not know how to handle this, and it will display an error message.

>>> **'Alice' + 42**

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

'Alice' + 42

TypeError: can only concatenate str (not "int") to str

The error message can only concatenate str (not "int") to str means that Python thought you were trying to concatenate an integer to the string 'Alice'. Your code will have to explicitly convert the integer to a string because Python cannot do this automatically. (Converting data types will be explained in “Dissecting Your Program” on page 13 when we talk about the str(), int(), and float() functions.)

The \* operator multiplies two integer or floating-point values. But when the \* operator is used on one string value and one integer value, it becomes the *string replication* operator. Enter a string multiplied by a number into the interactive shell to see this in action.

>>> **'Alice' \* 5**

'AliceAliceAliceAliceAlice'

The expression evaluates down to a single string value that repeats the original string a number of times equal to the integer value. String replica tion is a useful trick, but it’s not used as often as string concatenation.

The \* operator can be used with only two numeric values (for multipli cation), or one string value and one integer value (for string replication). Otherwise, Python will just display an error message, like the following:

>>> **'Alice' \* 'Bob'**

Traceback (most recent call last):

File "<pyshell#32>", line 1, in <module>

'Alice' \* 'Bob'

TypeError: can't multiply sequence by non-int of type 'str'

>>> **'Alice' \* 5.0**

Traceback (most recent call last):

File "<pyshell#33>", line 1, in <module>

'Alice' \* 5.0

TypeError: can't multiply sequence by non-int of type 'float'

It makes sense that Python wouldn’t understand these expressions: you can’t multiply two words, and it’s hard to replicate an arbitrary string a fractional number of times.

**8** Chapter 1

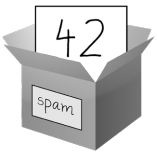
**Storing Values in Variables**

A *variable* is like a box in the computer’s memory where you can store a single value. If you want to use the result of an evaluated expression later in your program, you can save it inside a variable.

***Assignment Statements***

You’ll store values in variables with an *assignment statement*. An assignment statement consists of a variable name, an equal sign (called the *assignment operator*), and the value to be stored. If you enter the assignment state ment spam = 42, then a variable named spam will have the integer value 42 stored in it.

Think of a variable as a labeled box that a value is placed in, as in Figure 1-1.



*Figure 1-1: spam = 42 is like telling the program,*

*“The variable spam now has the integer value 42 in it.”*

For example, enter the following into the interactive shell:

❶ >>> **spam = 40**

>>> **spam**

40

>>> **eggs = 2**

❷ >>> **spam + eggs**

42

>>> **spam + eggs + spam**

82

❸ >>> **spam = spam + 2**

>>> **spam**

42

Python Basics **9**

A variable is *initialized* (or created) the first time a value is stored in it ❶. After that, you can use it in expressions with other variables and values ❷. When a variable is assigned a new value ❸, the old value is forgotten, which is why spam evaluated to 42 instead of 40 at the end of the example. This is called *overwriting* the variable. Enter the following code into the interactive shell to try overwriting a string:

>>> **spam = 'Hello'**

>>> **spam**

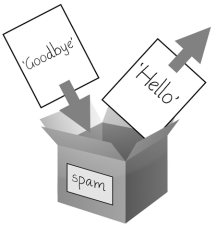
'Hello'

>>> **spam = 'Goodbye'**

>>> **spam**

'Goodbye'

Just like the box in Figure 1-2, the spam variable in this example stores 'Hello' until you replace the string with 'Goodbye'.



*Figure 1-2: When a new value is assigned to a variable,*

*the old one is forgotten.*

***Variable Names***

A good variable name describes the data it contains. Imagine that you moved to a new house and labeled all of your moving boxes as *Stuff*. You’d never find anything! Most of this book’s examples (and Python’s documen tation) use generic variable names like spam, eggs, and bacon, which come from the Monty Python “Spam” sketch. But in your programs, a descriptive name will help make your code more readable.

**10** Chapter 1

Though you can name your variables almost anything, Python does have some naming restrictions. Table 1-3 has examples of legal variable names. You can name a variable anything as long as it obeys the following three rules:

• It can be only one word with no spaces.

• It can use only letters, numbers, and the underscore (\_) character. • It can’t begin with a number.

Table 1-3: Valid and Invalid Variable Names

Valid variable names Invalid variable names

current\_balance current-balance (hyphens are not allowed)

currentBalance current balance (spaces are not allowed)

account4 4account (can’t begin with a number)

\_42 42 (can’t begin with a number)

TOTAL\_SUM TOTAL\_$UM (special characters like $ are not allowed) hello 'hello' (special characters like ' are not allowed)

Variable names are case-sensitive, meaning that spam, SPAM, Spam, and sPaM are four different variables. Though Spam is a valid variable you can use in a pro gram, it is a Python convention to start your variables with a lowercase letter.

This book uses *camelcase* for variable names instead of underscores; that is, variables lookLikeThis instead of looking\_like\_this. Some experienced pro grammers may point out that the official Python code style, PEP 8, says that underscores should be used. I unapologetically prefer camelcase and point

to the “A Foolish Consistency Is the Hobgoblin of Little Minds” section in PEP 8 itself:

Consistency with the style guide is important. But most impor

tantly: know when to be inconsistent—sometimes the style guide

just doesn’t apply. When in doubt, use your best judgment.

**Your First Program**

While the interactive shell is good for running Python instructions one at a time, to write entire Python programs, you’ll type the instructions into the file editor. The *file editor* is similar to text editors such as Notepad or TextMate, but it has some features specifically for entering source code. To open a new file in Mu, click the **New** button on the top row.

The window that appears should contain a cursor awaiting your input, but it’s different from the interactive shell, which runs Python instructions

Python Basics **11**

as soon as you press enter. The file editor lets you type in many instruc tions, save the file, and run the program. Here’s how you can tell the differ ence between the two:

• The interactive shell window will always be the one with the >>> prompt. • The file editor window will not have the >>> prompt.

Now it’s time to create your first program! When the file editor window opens, enter the following into it:

❶ # This program says hello and asks for my name.

❷ print('Hello, world!')

print('What is your name?') # ask for their name

❸ myName = input()

❹ print('It is good to meet you, ' + myName)

❺ print('The length of your name is:')

print(len(myName))

❻ print('What is your age?') # ask for their age

myAge = input()

print('You will be ' + str(int(myAge) + 1) + ' in a year.')

Once you’ve entered your source code, save it so that you won’t have to retype it each time you start Mu. Click the **Save** button, enter *hello.py* in the File Name field, and then click **Save**.

You should save your programs every once in a while as you type them. That way, if the computer crashes or you accidentally exit Mu, you won’t lose the code. As a shortcut, you can press ctrl-S on Windows and Linux or ⌘-S on macOS to save your file.

Once you’ve saved, let’s run our program. Press the **F5** key. Your pro gram should run in the interactive shell window. Remember, you have to press **F5** from the file editor window, not the interactive shell window. Enter your name when your program asks for it. The program’s output in the interactive shell should look something like this:

Python 3.7.0b4 (v3.7.0b4:eb96c37699, May 2 2018, 19:02:22) [MSC v.1913 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information. >>> ================================ RESTART ================================ >>>

Hello, world!

What is your name?

**Al**

It is good to meet you, Al

The length of your name is:

2

What is your age?

**4**

You will be 5 in a year.

>>>

**12** Chapter 1

When there are no more lines of code to execute, the Python program *terminates*; that is, it stops running. (You can also say that the Python pro gram *exits*.)

You can close the file editor by clicking the X at the top of the window. To reload a saved program, select **File**4**Open...** from the menu. Do that now, and in the window that appears, choose ***hello.py*** and click the **Open** button. Your previously saved *hello.py* program should open in the file editor window.

You can view the execution of a program using the Python Tutor visu alization tool at *http://pythontutor.com/*. You can see the execution of this particular program at *https://autbor.com/hellopy/*. Click the forward button to move through each step of the program’s execution. You’ll be able to see how the variables’ values and the output change.

**Dissecting Your Program**

With your new program open in the file editor, let’s take a quick tour of the Python instructions it uses by looking at what each line of code does.

***Comments***

The following line is called a *comment*.

❶ # This program says hello and asks for my name.

Python ignores comments, and you can use them to write notes or remind yourself what the code is trying to do. Any text for the rest of the line following a hash mark (#) is part of a comment.

Sometimes, programmers will put a # in front of a line of code to tem porarily remove it while testing a program. This is called *commenting out* code, and it can be useful when you’re trying to figure out why a program isn’t working. You can remove the # later when you are ready to put the line back in.

Python also ignores the blank line after the comment. You can add as many blank lines to your program as you want. This can make your code easier to read, like paragraphs in a book.

***The print() Function***

The print() function displays the string value inside its parentheses on the screen.

❷ print('Hello, world!')

print('What is your name?') # ask for their name

The line print('Hello, world!') means “Print out the text in the string 'Hello, world!'.” When Python executes this line, you say that Python is *calling* the print() function and the string value is being *passed* to the func tion. A value that is passed to a function call is an *argument*. Notice that

Python Basics **13**

the quotes are not printed to the screen. They just mark where the string begins and ends; they are not part of the string value.

**NOTE** *You can also use this function to put a blank line on the screen; just call print() with nothing in between the parentheses.*

When you write a function name, the opening and closing parentheses at the end identify it as the name of a function. This is why in this book, you’ll see print() rather than print. Chapter 3 describes functions in more detail.

***The input() Function***

The input() function waits for the user to type some text on the keyboard and press enter.

❸ myName = input()

This function call evaluates to a string equal to the user’s text, and the line of code assigns the myName variable to this string value.

You can think of the input() function call as an expression that evalu ates to whatever string the user typed in. If the user entered 'Al', then the expression would evaluate to myName = 'Al'.

If you call input() and see an error message, like NameError: name 'Al' is not defined, the problem is that you’re running the code with Python 2 instead of Python 3.

***Printing the User’s Name***

The following call to print() actually contains the expression 'It is good to meet you, ' + myName between the parentheses.

❹ print('It is good to meet you, ' + myName)

Remember that expressions can always evaluate to a single value. If 'Al' is the value stored in myName on line ❸, then this expression evaluates to 'It is good to meet you, Al'. This single string value is then passed to print(), which prints it on the screen.

***The len() Function***

You can pass the len() function a string value (or a variable containing a string), and the function evaluates to the integer value of the number of characters in that string.

❺ print('The length of your name is:')

print(len(myName))

**14** Chapter 1

Enter the following into the interactive shell to try this:

>>> **len('hello')**

5

>>> **len('My very energetic monster just scarfed nachos.')**

46

>>> **len('')**

0

Just like those examples, len(myName) evaluates to an integer. It is then passed to print() to be displayed on the screen. The print() function allows you to pass it either integer values or string values, but notice the error that shows up when you type the following into the interactive shell:

>>> **print('I am ' + 29 + ' years old.')**

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

print('I am ' + 29 + ' years old.')

TypeError: can only concatenate str (not "int") to str

The print() function isn’t causing that error, but rather it’s the expres sion you tried to pass to print(). You get the same error message if you type the expression into the interactive shell on its own.

>>> **'I am ' + 29 + ' years old.'**

Traceback (most recent call last):

File "<pyshell#7>", line 1, in <module>

'I am ' + 29 + ' years old.'

TypeError: can only concatenate str (not "int") to str

Python gives an error because the + operator can only be used to add two integers together or concatenate two strings. You can’t add an integer to a string, because this is ungrammatical in Python. You can fix this by using a string version of the integer instead, as explained in the next section.

***The str(), int(), and float() Functions***

If you want to concatenate an integer such as 29 with a string to pass to print(), you’ll need to get the value '29', which is the string form of 29. The str() function can be passed an integer value and will evaluate to a string value version of the integer, as follows:

>>> **str(29)**

'29'

>>> **print('I am ' + str(29) + ' years old.')**

I am 29 years old.

Python Basics **15**

Because str(29) evaluates to '29', the expression 'I am ' + str(29) + ' years old.' evaluates to 'I am ' + '29' + ' years old.', which in turn evaluates to 'I am 29 years old.'. This is the value that is passed to the print() function.

The str(), int(), and float() functions will evaluate to the string, inte ger, and floating-point forms of the value you pass, respectively. Try con verting some values in the interactive shell with these functions and watch what happens.

>>> **str(0)**

'0'

>>> **str(-3.14)**

'-3.14'

>>> **int('42')**

42

>>> **int('-99')**

-99

>>> **int(1.25)**

1

>>> **int(1.99)**

1

>>> **float('3.14')**

3.14

>>> **float(10)**

10.0

The previous examples call the str(), int(), and float() functions and pass them values of the other data types to obtain a string, integer, or floating-point form of those values.

The str() function is handy when you have an integer or float that you want to concatenate to a string. The int() function is also helpful if you have a number as a string value that you want to use in some mathematics. For example, the input() function always returns a string, even if the user enters a number. Enter **spam = input()** into the interactive shell and enter **101** when it waits for your text.

>>> **spam = input()**

**101**

>>> **spam**

'101'

The value stored inside spam isn’t the integer 101 but the string '101'. If you want to do math using the value in spam, use the int() function to get the integer form of spam and then store this as the new value in spam.

>>> **spam = int(spam)**

>>> **spam**

101

Now you should be able to treat the spam variable as an integer instead of a string.

**16** Chapter 1

>>> **spam \* 10 / 5**

202.0

Note that if you pass a value to int() that it cannot evaluate as an inte ger, Python will display an error message.

>>> **int('99.99')**

Traceback (most recent call last):

File "<pyshell#18>", line 1, in <module>

int('99.99')

ValueError: invalid literal for int() with base 10: '99.99'

>>> **int('twelve')**

Traceback (most recent call last):

File "<pyshell#19>", line 1, in <module>

int('twelve')

ValueError: invalid literal for int() with base 10: 'twelve'

The int() function is also useful if you need to round a floating-point number down.

>>> **int(7.7)**

7

>>> **int(7.7) + 1**

8

You used the int() and str() functions in the last three lines of your program to get a value of the appropriate data type for the code.

❻ print('What is your age?') # ask for their age

myAge = input()

print('You will be ' + str(int(myAge) + 1) + ' in a year.')

**TEXT AND NUMBER EQUIVALENCE**

Although the string value of a number is considered a completely different value from the integer or floating-point version, an integer can be equal to a floating point.

>>> **42 == '42'**

False

>>> **42 == 42.0**

True

>>> **42.0 == 0042.000**

True

Python makes this distinction because strings are text, while integers and floats are both numbers.

Python Basics **17**

The myAge variable contains the value returned from input(). Because the input() function always returns a string (even if the user typed in a number), you can use the int(myAge) code to return an integer value of the string in myAge. This integer value is then added to 1 in the expression int(myAge) + 1.

The result of this addition is passed to the str() function: str(int(myAge) + 1). The string value returned is then concatenated with the strings 'You will be ' and ' in a year.' to evaluate to one large string value. This large string is finally passed to print() to be displayed on the screen.

Let’s say the user enters the string '4' for myAge. The string '4' is con verted to an integer, so you can add one to it. The result is 5. The str() func tion converts the result back to a string, so you can concatenate it with the second string, 'in a year.', to create the final message. These evaluation steps would look something like the following:

print('You will be ' + str(int(myAge) + 1) + ' in a year.')

print('You will be ' + str(int( ) + 1) + ' in a year.')

'4'

4 + 1

print('You will be ' + str( ) + ' in a year.')

print('You will be ' + str( ) + ' in a year.')

5

print('You will be ' + + ' in a year.')

'5'

print('You will be 5' + ' in a year.')

print('You will be 5 in a year.')

**Summary**

You can compute expressions with a calculator or enter string concatena tions with a word processor. You can even do string replication easily by copying and pasting text. But expressions, and their component values—

operators, variables, and function calls—are the basic building blocks that make programs. Once you know how to handle these elements, you will be able to instruct Python to operate on large amounts of data for you.

It is good to remember the different types of operators (+, -, \*, /, //, %, and \*\* for math operations, and + and \* for string operations) and the three data types (integers, floating-point numbers, and strings) introduced in this chapter.

I introduced a few different functions as well. The print() and input() functions handle simple text output (to the screen) and input (from the keyboard). The len() function takes a string and evaluates to an int of the number of characters in the string. The str(), int(), and float() functions will evaluate to the string, integer, or floating-point number form of the value they are passed.

**18** Chapter 1

In the next chapter, you’ll learn how to tell Python to make intelli gent decisions about what code to run, what code to skip, and what code to repeat based on the values it has. This is known as *flow control*, and it allows you to write programs that make intelligent decisions.

**Practice Questions**

1. Which of the following are operators, and which are values?

\*

'hello'

-88.8

-

/

+

5

2. Which of the following is a variable, and which is a string?

spam

'spam'

3. Name three data types.

4. What is an expression made up of? What do all expressions do? 5. This chapter introduced assignment statements, like spam = 10. What is the difference between an expression and a statement?

6. What does the variable bacon contain after the following code runs?

bacon = 20

bacon + 1

7. What should the following two expressions evaluate to?

'spam' + 'spamspam'

'spam' \* 3

8. Why is eggs a valid variable name while 100 is invalid?

9. What three functions can be used to get the integer, floating-point number, or string version of a value?

Python Basics **19**

10. Why does this expression cause an error? How can you fix it? 'I have eaten ' + 99 + ' burritos.'

**Extra credit:** Search online for the Python documentation for the len() function. It will be on a web page titled “Built-in Functions.” Skim the list of other functions Python has, look up what the round() function does, and experiment with it in the interactive shell.

**20** Chapter 1

**2**

**FLOW CONTROL**

So, you know the basics of individual

instructions and that a program is just a

series of instructions. But programming’s real strength isn’t just running one instruction after another like a weekend errand list. Based on how expressions evaluate, a program can decide to skip instructions, repeat them, or choose one of several instructions to run. In fact, you almost never want your programs to start from the first line of code and simply execute every line, straight to the end. *Flow control statements* can decide which Python instructions to execute under which conditions.

These flow control statements directly correspond to the symbols in a flowchart, so I’ll provide flowchart versions of the code discussed in this chapter. Figure 2-1 shows a flowchart for what to do if it’s raining. Follow the path made by the arrows from Start to End.

Start

Is raining? Have umbrella? Yes

No

Wait a while.

No

Yes

Go outside. No

End

*Figure 2-1: A flowchart to tell you what to do if it is raining*

Is raining? Yes

In a flowchart, there is usually more than one way to go from the start to the end. The same is true for lines of code in a computer program. Flowcharts represent these branching points with diamonds, while the other steps are represented with rectangles. The starting and ending steps are represented with rounded rectangles.

But before you learn about flow control statements, you first need to learn how to represent those *yes* and *no* options, and you need to under stand how to write those branching points as Python code. To that end, let’s explore Boolean values, comparison operators, and Boolean operators.

**Boolean Values**

While the integer, floating-point, and string data types have an unlimited number of possible values, the *Boolean* data type has only two values: True and False. (Boolean is capitalized because the data type is named after mathematician George Boole.) When entered as Python code, the Boolean values True and False lack the quotes you place around strings, and they always start with a capital *T* or *F*, with the rest of the word in lowercase. Enter the following into the interactive shell. (Some of these instructions are intentionally incorrect, and they’ll cause error messages to appear.)

**22** Chapter 2

❶ >>> **spam = True**

>>> **spam**

True

❷ >>> **true**

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

true

NameError: name 'true' is not defined

❸ >>> **True = 2 + 2**

SyntaxError: can't assign to keyword

Like any other value, Boolean values are used in expressions and can be stored in variables ❶. If you don’t use the proper case ❷ or you try to use True and False for variable names ❸, Python will give you an error message.

**Comparison Operators**

*Comparison operators*, also called *relational operators*, compare two values and eval uate down to a single Boolean value. Table 2-1 lists the comparison operators.

Table 2-1: Comparison Operators

Operator Meaning

== Equal to

!= Not equal to

< Less than

> Greater than

<= Less than or equal to

>= Greater than or equal to

These operators evaluate to True or False depending on the values you give them. Let’s try some operators now, starting with == and !=.

>>> **42 == 42**

True

>>> **42 == 99**

False

>>> **2 != 3**

True

>>> **2 != 2**

False

As you might expect, == (equal to) evaluates to True when the values on both sides are the same, and != (not equal to) evaluates to True when the two values are different. The == and != operators can actually work with values of any data type.

Flow Control **23**

>>> **'hello' == 'hello'**

True

>>> **'hello' == 'Hello'**

False

>>> **'dog' != 'cat'**

True

>>> **True == True**

True

>>> **True != False**

True

>>> **42 == 42.0**

True

❶ >>> **42 == '42'**

False

Note that an integer or floating-point value will always be unequal to a string value. The expression 42 == '42' ❶ evaluates to False because Python considers the integer 42 to be different from the string '42'.

The <, >, <=, and >= operators, on the other hand, work properly only with integer and floating-point values.

>>> **42 < 100**

True

>>> **42 > 100**

False

>>> **42 < 42**

False

>>> **eggCount = 42**

❶ >>> **eggCount <= 42**

True

>>> **myAge = 29**

❷ >>> **myAge >= 10**

True

**THE DIFFERENCE BET WEEN THE == AND = OPER ATORS**

You might have noticed that the == operator (equal to) has two equal signs, while the = operator (assignment) has just one equal sign. It’s easy to confuse these two operators with each other. Just remember these points:

• The == operator (equal to) asks whether two values are the same as each other.

• The = operator (assignment) puts the value on the right into the variable on the left.

To help remember which is which, notice that the == operator (equal to) consists of two characters, just like the != operator (not equal to) consists of two characters.

**24** Chapter 2

You’ll often use comparison operators to compare a variable’s value to some other value, like in the eggCount <= 42 ❶ and myAge >= 10 ❷ examples. (After all, instead of entering 'dog' != 'cat' in your code, you could have

just entered True.) You’ll see more examples of this later when you learn about flow control statements.

**Boolean Operators**

The three Boolean operators (and, or, and not) are used to compare Boolean values. Like comparison operators, they evaluate these expressions down to a Boolean value. Let’s explore these operators in detail, starting with the and operator.

***Binary Boolean Operators***

The and and or operators always take two Boolean values (or expressions), so they’re considered *binary* operators. The and operator evaluates an expression to True if *both* Boolean values are True; otherwise, it evaluates to False. Enter some expressions using and into the interactive shell to see it in action.

>>> **True and True**

True

>>> **True and False**

False

A *truth table* shows every possible result of a Boolean operator. Table 2-2 is the truth table for the and operator.

Table 2-2: The and Operator’s Truth Table

Expression Evaluates to . . .

True and True True

True and False False

False and True False

False and False False

On the other hand, the or operator evaluates an expression to True if *either* of the two Boolean values is True. If both are False, it evaluates to False.

>>> **False or True**

True

>>> **False or False**

False

You can see every possible outcome of the or operator in its truth table, shown in Table 2-3.

Flow Control **25**

Table 2-3: The or Operator’s Truth Table

Expression Evaluates to . . .

True or True True

True or False True

False or True True

False or False False

***The not Operator***

Unlike and and or, the not operator operates on only one Boolean value (or expression). This makes it a *unary* operator. The not operator simply evalu ates to the opposite Boolean value.

>>> **not True**

False

❶ >>> **not not not not True**

True

Much like using double negatives in speech and writing, you can nest not operators ❶, though there’s never not no reason to do this in real pro grams. Table 2-4 shows the truth table for not.

Table 2-4: The not Operator’s Truth Table

Expression Evaluates to . . .

not True False

not False True

**Mixing Boolean and Comparison Operators**

Since the comparison operators evaluate to Boolean values, you can use them in expressions with the Boolean operators.

Recall that the and, or, and not operators are called Boolean operators because they always operate on the Boolean values True and False. While expressions like 4 < 5 aren’t Boolean values, they are expressions that evalu ate down to Boolean values. Try entering some Boolean expressions that use comparison operators into the interactive shell.

>>> **(4 < 5) and (5 < 6)**

True

>>> **(4 < 5) and (9 < 6)**

False

>>> **(1 == 2) or (2 == 2)**

True

The computer will evaluate the left expression first, and then it will evaluate the right expression. When it knows the Boolean value for each,

**26** Chapter 2

it will then evaluate the whole expression down to one Boolean value. You can think of the computer’s evaluation process for (4 < 5) and (5 < 6) as the following:

(4 < 5) and (5 < 6)

True and (5 < 6)

True and True

True

You can also use multiple Boolean operators in an expression, along with the comparison operators:

>>> **2 + 2 == 4 and not 2 + 2 == 5 and 2 \* 2 == 2 + 2**

True

The Boolean operators have an order of operations just like the

math operators do. After any math and comparison operators evaluate, Python evaluates the not operators first, then the and operators, and then the or operators.

**Elements of Flow Control**

Flow control statements often start with a part called the *condition* and are always followed by a block of code called the *clause*. Before you learn about Python’s specific flow control statements, I’ll cover what a condition and a block are.

***Conditions***

The Boolean expressions you’ve seen so far could all be considered con ditions, which are the same thing as expressions; *condition* is just a more specific name in the context of flow control statements. Conditions always evaluate down to a Boolean value, True or False. A flow control statement decides what to do based on whether its condition is True or False, and almost every flow control statement uses a condition.

***Blocks of Code***

Lines of Python code can be grouped together in *blocks*. You can tell when a block begins and ends from the indentation of the lines of code. There are three rules for blocks.

• Blocks begin when the indentation increases.

• Blocks can contain other blocks.

• Blocks end when the indentation decreases to zero or to a containing block’s indentation.

Flow Control **27**

Blocks are easier to understand by looking at some indented code, so let’s find the blocks in part of a small game program, shown here:

name = 'Mary'

password = 'swordfish'

if name == 'Mary':

❶ print('Hello, Mary')

if password == 'swordfish':

❷ print('Access granted.')

else:

❸ print('Wrong password.')

You can view the execution of this program at *https://autbor.com/blocks/*. The first block of code ❶ starts at the line print('Hello, Mary') and contains all the lines after it. Inside this block is another block ❷, which has only a single line in it: print('Access Granted.'). The third block ❸ is also one line long: print('Wrong password.').

**Program Execution**

In the previous chapter’s *hello.py* program, Python started executing instruc tions at the top of the program going down, one after another. The *program execution* (or simply, *execution*) is a term for the current instruction being exe

cuted. If you print the source code on paper and put your finger on each line as it is executed, you can think of your finger as the program execution. Not all programs execute by simply going straight down, however. If you use your finger to trace through a program with flow control statements, you’ll likely find yourself jumping around the source code based on condi tions, and you’ll probably skip entire clauses.

**Flow Control Statements**

Now, let’s explore the most important piece of flow control: the statements themselves. The statements represent the diamonds you saw in the flowchart in Figure 2-1, and they are the actual decisions your programs will make.

***if Statements***

The most common type of flow control statement is the if statement. An if statement’s clause (that is, the block following the if statement) will execute if the statement’s condition is True. The clause is skipped if the condition is False.

In plain English, an if statement could be read as, “If this condition is true, execute the code in the clause.” In Python, an if statement consists of the following:

• The if keyword

• A condition (that is, an expression that evaluates to True or False)

**28** Chapter 2

• A colon

• Starting on the next line, an indented block of code (called the if clause)

For example, let’s say you have some code that checks to see whether someone’s name is Alice. (Pretend name was assigned some value earlier.)

if name == 'Alice':

print('Hi, Alice.')

All flow control statements end with a colon and are followed by a new block of code (the clause). This if statement’s clause is the block with print('Hi, Alice.'). Figure 2-2 shows what a flowchart of this code would look like.

Start

name == 'Alice' True

False

End

print('Hi, Alice.')

*Figure 2-2: The flowchart for an if statement*

***else Statements***

An if clause can optionally be followed by an else statement. The else clause is executed only when the if statement’s condition is False. In plain English, an else statement could be read as, “If this condition is true, exe cute this code. Or else, execute that code.” An else statement doesn’t have a condition, and in code, an else statement always consists of the following:

• The else keyword

• A colon

• Starting on the next line, an indented block of code (called the else clause)

Flow Control **29**

Returning to the Alice example, let’s look at some code that uses an else statement to offer a different greeting if the person’s name isn’t Alice.

if name == 'Alice':

print('Hi, Alice.')

else:

print('Hello, stranger.')

Figure 2-3 shows what a flowchart of this code would look like. Start

name == 'Alice' False

End

True

print('Hi, Alice.') print('Hello, stranger.')

*Figure 2-3: The flowchart for an else statement*

***elif Statements***

While only one of the if or else clauses will execute, you may have a case where you want one of *many* possible clauses to execute. The elif statement is an “else if” statement that always follows an if or another elif statement. It provides another condition that is checked only if all of the previous condi tions were False. In code, an elif statement always consists of the following:

• The elif keyword

• A condition (that is, an expression that evaluates to True or False) • A colon

• Starting on the next line, an indented block of code (called the elif clause)

Let’s add an elif to the name checker to see this statement in action.

if name == 'Alice':

print('Hi, Alice.')

**30** Chapter 2

elif age < 12:

print('You are not Alice, kiddo.')

This time, you check the person’s age, and the program will tell them something different if they’re younger than 12. You can see the flowchart for this in Figure 2-4.

Start

name == 'Alice' False

True

print('Hi, Alice.')

age < 12 print('You are not Alice, kiddo.')

True

False

End

*Figure 2-4: The flowchart for an elif statement*

The elif clause executes if age < 12 is True and name == 'Alice' is False. However, if both of the conditions are False, then both of the clauses are skipped. It is *not* guaranteed that at least one of the clauses will be executed. When there is a chain of elif statements, only one or none of the clauses will be executed. Once one of the statements’ conditions is found to be True, the rest of the elif clauses are automatically skipped. For example, open a new file editor window and enter the following code, saving it as *vampire.py*:

name = 'Carol'

age = 3000

if name == 'Alice':

print('Hi, Alice.')

elif age < 12:

print('You are not Alice, kiddo.')

elif age > 2000:

Flow Control **31**

print('Unlike you, Alice is not an undead, immortal vampire.') elif age > 100:

print('You are not Alice, grannie.')

You can view the execution of this program at *https://autbor.com/vampire/*. Here, I’ve added two more elif statements to make the name checker greet a person with different answers based on age. Figure 2-5 shows the flowchart for this.

Start

name == 'Alice' False

True

print('Hi, Alice.')

age < 12 print('You are not Alice, kiddo.') True

False

age > 2000 False

age > 100

False

End

True True

print('Unlike you, Alice is not an undead, immortal vampire.')

print('You are not Alice, grannie.')

*Figure 2-5: The flowchart for multiple elif statements in the* vampire.py *program*

**32** Chapter 2

The order of the elif statements does matter, however. Let’s rearrange them to introduce a bug. Remember that the rest of the elif clauses are automatically skipped once a True condition has been found, so if you swap around some of the clauses in *vampire.py*, you run into a problem. Change the code to look like the following, and save it as *vampire2.py*:

name = 'Carol'

age = 3000

if name == 'Alice':

print('Hi, Alice.')

elif age < 12:

print('You are not Alice, kiddo.')

❶ elif age > 100:

print('You are not Alice, grannie.')

elif age > 2000:

print('Unlike you, Alice is not an undead, immortal vampire.')

You can view the execution of this program at *https://autbor.com/vampire2/*. Say the age variable contains the value 3000 before this code is executed. You might expect the code to print the string 'Unlike you, Alice is not an undead, immortal vampire.'. However, because the age > 100 condition is True (after all, 3,000 *is* greater than 100) ❶, the string 'You are not Alice, grannie.' is printed, and the rest of the elif statements are automatically skipped. Remember that at most only one of the clauses will be executed, and for elif statements, the order matters!

Figure 2-6 shows the flowchart for the previous code. Notice how the diamonds for age > 100 and age > 2000 are swapped.

Optionally, you can have an else statement after the last elif statement. In that case, it *is* guaranteed that at least one (and only one) of the clauses will be executed. If the conditions in every if and elif statement are False, then the else clause is executed. For example, let’s re-create the Alice pro gram to use if, elif, and else clauses.

name = 'Carol'

age = 3000

if name == 'Alice':

print('Hi, Alice.')

elif age < 12:

print('You are not Alice, kiddo.')

else:

print('You are neither Alice nor a little kid.')

You can view the execution of this program at *https://autbor.com /littlekid/*. Figure 2-7 shows the flowchart for this new code, which we’ll save as *littleKid.py*.

In plain English, this type of flow control structure would be “If the first condition is true, do this. Else, if the second condition is true, do that. Otherwise, do something else.” When you use if, elif, and else statements together, remember these rules about how to order them to avoid bugs like the one in Figure 2-6. First, there is always exactly one if statement. Any

Flow Control **33**

elif statements you need should follow the if statement. Second, if you want to be sure that at least one clause is executed, close the structure with an else statement.

Start

name == 'Alice' False

age < 12

False

age > 100

False

age > 2000

False

End

True

True

True

~~X~~ True

print('Hi, Alice.')

print('You are not Alice, kiddo.')

print('You are not Alice, grannie.')

print('Unlike you, Alice is not an undead, immortal vampire.')

*Figure 2-6: The flowchart for the* vampire2.py *program. The X path will logi cally never happen, because if age were greater than 2000, it would have already been greater than 100.*

**34** Chapter 2

Start

name == 'Alice' False

True

print('Hi, Alice.')

age < 12 print('You are not Alice, kiddo.')

True

False

print('You are neither Alice

nor a little kid.')

End

*Figure 2-7: Flowchart for the previous* littleKid.py *program*

***while Loop Statements***

You can make a block of code execute over and over again using a while statement. The code in a while clause will be executed as long as the while statement’s condition is True. In code, a while statement always consists of the following:

• The while keyword

• A condition (that is, an expression that evaluates to True or False) • A colon

• Starting on the next line, an indented block of code (called the while clause)

Flow Control **35**

You can see that a while statement looks similar to an if statement. The difference is in how they behave. At the end of an if clause, the program execution continues after the if statement. But at the end of a while clause, the program execution jumps back to the start of the while statement. The while clause is often called the *while loop* or just the *loop*.

Let’s look at an if statement and a while loop that use the same condi tion and take the same actions based on that condition. Here is the code with an if statement:

spam = 0

if spam < 5:

print('Hello, world.')

spam = spam + 1

Here is the code with a while statement:

spam = 0

while spam < 5:

print('Hello, world.')

spam = spam + 1

These statements are similar—both if and while check the value of spam, and if it’s less than 5, they print a message. But when you run these two code snippets, something very different happens for each one. For the if statement, the output is simply "Hello, world.". But for the while statement, it’s "Hello, world." repeated five times! Take a look at the flowcharts for these two pieces of code, Figures 2-8 and 2-9, to see why this happens.

Start

True

spam < 5 False

End

print('Hello, world.') spam = spam + 1

*Figure 2-8: The flowchart for the if statement code*

**36** Chapter 2

Start

spam < 5

False

End

True

print('Hello, world.')

spam = spam + 1

*Figure 2-9: The flowchart for the while statement code*

The code with the if statement checks the condition, and it prints Hello, world. only once if that condition is true. The code with the while loop, on the other hand, will print it five times. The loop stops after five prints because the integer in spam increases by one at the end of each loop iteration, which means that the loop will execute five times before spam < 5 is False.

In the while loop, the condition is always checked at the start of each *iteration* (that is, each time the loop is executed). If the condition is True, then the clause is executed, and afterward, the condition is checked again. The first time the condition is found to be False, the while clause is skipped.

**An Annoying while Loop**

Here’s a small example program that will keep asking you to type, literally, your name. Select **File**4**New** to open a new file editor window, enter the fol lowing code, and save the file as *yourName.py*:

❶ name = ''

❷ while name != 'your name':

print('Please type your name.')

❸ name = input()

❹ print('Thank you!')

You can view the execution of this program at *https://autbor.com/yourname/*. First, the program sets the name variable ❶ to an empty string. This is so

Flow Control **37**

that the name != 'your name' condition will evaluate to True and the program execution will enter the while loop’s clause ❷.

The code inside this clause asks the user to type their name, which is assigned to the name variable ❸. Since this is the last line of the block, the execution moves back to the start of the while loop and reevaluates the condition. If the value in name is *not equal* to the string 'your name', then the condition is True, and the execution enters the while clause again.

But once the user types your name, the condition of the while loop will be 'your name' != 'your name', which evaluates to False. The condition is now False, and instead of the program execution reentering the while loop’s clause, Python skips past it and continues running the rest of the program ❹. Figure 2-10 shows a flowchart for the *yourName.py* program.

Start

True

name != 'your name' False

print('Thank you!') End

print('Please type your name.') name = input()

*Figure 2-10: A flowchart of the* yourName.py *program*

Now, let’s see *yourName.py* in action. Press **F5** to run it, and enter something other than your name a few times before you give the program what it wants.

Please type your name.

**Al**

Please type your name.

**Albert**

Please type your name.

**%#@#%\*(^&!!!**

**38** Chapter 2