The authors and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in this book. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Copyright © 2019 Pearson Education, Inc., Hoboken, New Jersey 07030. All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions department, please visit www.pearsonhighed.com/permissions/.

Many of the designations by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Unless otherwise indicated herein, any third-party trademarks that may appear in this work are the property of their respective owners and any references to third-party trademarks, logos or other trade dress are for demonstrative or descriptive purposes only. Such references are not intended to imply any sponsorship, endorsement, authorization, or promotion of Pearson’s products by the owners of such marks, or any relationship between the owner and Pearson Education, Inc. or its affiliates, authors, licensees or distributors.

Library of Congress Cataloging-in-Publication Data
Title: Building Python programs / Stuart Reges, University of Washington, Marty Stepp, Stanford University, Allison Obourn, University of Arizona.
Subjects: LCSH: Python (Computer program language)
Classification: LCC QA76.73.P98 R445 2019 | DDC 005.13/3—dc23
LC record available at https://lccn.loc.gov/2018028848

ISBN 10: 0-13-520598-0
Preface

The Python programming language has become enormously popular in recent years. Many people are impressed with how quickly you can learn Python’s simple and intuitive syntax and that has led many users to create popular libraries. Python was designed by Guido van Rossum who has been affectionately dubbed “Benevolent Dictator For Life (BDFL)” by the Python community. He has said that he chose the name Python because he was “in a slightly irreverent mood” and that he is “a big fan of Monty Python’s Flying Circus” (a British comedy show). Who wouldn’t want to learn a programming language named after a group of comedians?

Our new Building Python Programs text is designed for use in a first course in computer science. We have class-tested it with hundreds of undergraduates at the University of Arizona, most of whom were not computer science majors. This textbook is based on our previous text, Building Java Programs, now in its fourth edition. The Java text has proven effective in our class testing with thousands of students including our own at the University of Washington since 2007.

Introductory computer science courses have a long history at many universities of being “killer” courses with high failure rates. But as Douglas Adams says in The Hitchhiker’s Guide to the Galaxy, “Don’t panic.” Students can master this material if they can learn it gradually.

Python has many attributes that make it an appealing language for a first computer science course. It has a simple and concise yet powerful syntax that makes it pleasant to learn and great for writing many common programs. A student can write their first Python program with only a single line of code, as opposed to several lines in most other languages such as Java or C++. Python includes a built-in interpreter and read-evaluate-print loop (REPL) for quickly running and testing code, encouraging students to test and explore the language. Python also offers a rich set of libraries that students can use for graphics, animation, math, scientific computing, games, and much more. This text has been built from the start for Python 3, the most modern version of the language as of this writing, and it embraces the modern features and idioms of that version of the language.

Our teaching materials are based on a “back to basics” approach that focuses on procedural programming and program decomposition. This is also called the “objects later” approach, as opposed to the “objects early” approach taught in some schools. We know from years of experience that a broad range of scientists, engineers, and others can learn how to program in a procedural manner. Once we have built a solid foundation of procedural techniques, we turn to object-oriented programming. By the end of the text, students will have learned about both styles of programming.
Preface

The following are the key features of our approach and materials:

• **Focus on problem solving.** Many textbooks focus on language details when they introduce new constructs. We focus instead on problem solving. What new problems can be solved with each construct? What pitfalls are novices likely to encounter along the way? What are the most common ways to use a new construct?

• **Emphasis on algorithmic thinking.** Our procedural approach allows us to emphasize algorithmic problem solving: breaking a large problem into smaller problems, using pseudocode to refine an algorithm, and grappling with the challenge of expressing a large program algorithmically.

• **Thorough discussion of topics.** The authors have found that many introductory texts rapidly cover new syntax and concepts and then quickly race on to the next topic. We feel that the students who crack open their textbook are exactly the sort that want more thorough and careful explanation and discussion of tricky topics. In this text we favor longer explanations, with more verbiage, figures, and code examples than in many other texts.

• **Layered approach.** Programming involves many concepts that are difficult to learn all at once. Teaching a novice to code is like trying to build a house of cards; each new card has to be placed carefully. If the process is rushed and you try to place too many cards at once, the entire structure collapses. We teach new concepts gradually, layer by layer, allowing students to expand their understanding at a manageable pace.

• **Emphasis on good coding style.** We show code that uses proper and consistent programming style and design. All complete programs shown in the text are thoroughly commented and properly decomposed. Throughout the text we discuss common idioms, good and bad style choices, and how to choose elegant and appropriate ways to decompose and solve each new category of problem.

• **Carefully chosen language subset.** Rather than a “kitchen sink” approach that tries to show the student every language construct and feature, we instead go out of our way to explain and use a core subset of the Python language that we feel is most well suited to solving introductory level problems.

• **Case studies.** We end most chapters with a significant case study that shows students how to develop a complex program in stages and how to test it as it is being developed. This structure allows us to demonstrate each new programming construct in a rich context that cannot be achieved with short code examples.
Layers and Dependencies

Many introductory computer science texts are language-oriented, but the early chapters of our approach are layered. For example, Python has many control structures (including loops and if/else statements), and many texts include all of these control structures in a single chapter. While that might make sense to someone who already knows how to program, it can be overwhelming for a novice who is learning how to program. We find that it is much more effective to spread these control structures into different chapters so that students learn one structure at a time rather than trying to learn them all at once.

The following table shows how the layered approach works in the first seven chapters:

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Control Flow</th>
<th>Data</th>
<th>Techniques</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>functions</td>
<td>string literals</td>
<td>decomposition</td>
<td>print</td>
</tr>
<tr>
<td>2</td>
<td>definite loops (for loops)</td>
<td>expressions/variables, integers, real numbers</td>
<td>local variables, global constants, pseudocode</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>parameters, return using objects</td>
<td>decomposition with param/return</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>conditionals (if/else)</td>
<td>strings</td>
<td>pre/postconditions, raising exceptions</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>indefinite loops (while loops)</td>
<td>Boolean logic</td>
<td>assertions, robust programs</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>file objects</td>
<td>line-based processing, line-based processing</td>
<td>file I/O</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>lists</td>
<td>traversals</td>
<td>files as lists</td>
<td></td>
</tr>
</tbody>
</table>

Chapters 1–5 are designed to be worked through in order, with greater flexibility of study then beginning in Chapter 6. Chapter 6 (File I/O) may be skipped, although the case study in Chapter 7 (Lists) involves reading from a file, a topic that is covered in Chapter 6.

The following figure represents a dependency chart for the book. A strong dependency is drawn as a solid arrow; we recommend not covering chapters outside of their strong dependency order. A weak dependency is drawn as a dashed arrow. Weak dependencies are ones where the later chapter briefly mentions a topic from the earlier chapter, but the chapter can still be read and explored without having covered the earlier chapter if necessary.
Here are more detailed explanations of the weak dependencies between chapters:

- A few examples from Chapter 7 on lists, and from Chapter 8 on dictionaries and sets, read data from files. File input/output is covered in Chapter 6. But overall file-reading is not required in order to discuss lists or other collections, so Chapter 6 can be skipped if desired.

- A few examples from Chapter 11 on classes and objects mention the concept of reference semantics, which is introduced in Chapter 7 on lists. But the concept of references is re-explained in Chapter 11, so classes can be covered early before lists if desired.

- Some of the recursive functions in Chapter 9 process lists, and one recursive function recursively reverses the lines of a file. So Chapter 9 weakly depends on Chapter 7. But almost every recursive function written in Chapter 9 can be written and understood using only the Chapter 1–5 core material.

As you can see from the diagram, Chapter 7 on Lists is perhaps the most important chapter after the first five, and its material is used by many other chapters. A common chapter order swap would be to cover Chapters 1–5, then do Chapter 7 on Lists, then go back to Chapter 6 on Files with the extra knowledge of lists to help you.

**Supplements**

Answers to all self-check problems appear on our web site and are accessible to anyone: http://www.buildingpythonprograms.com/
Preface

In addition, our web site also has the following additional resources available for students:

- Online-only supplemental content
- Source code and data files for all case studies and other complete program examples.
- The DrawingPanel class used in Chapter 3.
- Links to web-based programming practice tools.

Instructors can access the following resources from our web site:

- PowerPoint slides suitable for lectures.
- Solutions to exercises and programming projects, along with homework specification documents for many projects.
- Sample Exams and solution keys.

To access instructor resources, contact us at authors@buildingpythonprograms.com. For other questions related to resources, contact the authors and/or your Pearson representative.

MyLab Programming

MyLab Programming helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyLab Programming improves the programming competence of beginning students, who often struggle with the basic concepts and paradigms of popular high-level programming languages. A self-study and homework tool, the MyLab Programming course consists of hundreds of small practice exercises organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted by students for review.

MyLab Programming is offered to users of this book in partnership with Turing’s Craft, the makers of the CodeLab interactive programming exercise system. For a full demonstration, to see feedback from instructors and students, or to get started using MyLab Programming in your course, visit: http://www.pearson.com/mylab/programming.
Preface

Acknowledgments

We would also like to thank the staff at Pearson who helped produce the book. Rose Kernan managed the project and was our primary point of contact during book production. Rose did a phenomenal job; she was diligent, responsive, and helpful at every step of the process. Amanda Brands was our content producer, and she also provided excellent support along the way. Thank you to Martha McMaster for proofreading the text, and thanks to Shelly Gerger-Knechtl for copy editing and indexing. We thank Yvonne Vannatta, our marketing manager, and Meghan Jacoby, our editorial assistant. We also want to thank the team of artists and compositors from Pearson’s partner institutions who helped produce the chapters of this text.

We would like to thank our lead editor at Pearson, Matt Goldstein. Over a decade ago Matt believed in our work and partnered with us to create the original Building Java Programs on which this text is based. Matt has been a stalwart supporter and is always a pleasure to work with.

Last but not least, the authors would like to thank the CSC 110 students at the University of Arizona who class-tested our chapters in rough draft form. Students provided helpful suggestions for improving the content and also submitted corrections for typos and errors in drafts of chapters.

Stuart Reges, University of Washington
Marty Stepp, Stanford University
Allison Obourn, University of Arizona
MyLab Programming

Through the power of practice and immediate personalized feedback, MyLab Programming™ helps students master programming fundamentals and build computational thinking skills.

PROGRAMMING PRACTICE
With MyLab Programming, your students will gain first-hand programming experience in an interactive online environment.

IMMEDIATE, PERSONALIZED FEEDBACK
MyLab Programming automatically detects errors in the logic and syntax of their code submission and offers targeted hints that enables students to figure out what went wrong and why.

GRADUATED COMPLEXITY
MyLab Programming breaks down programming concepts into short, understandable sequences of exercises. Within each sequence the level and sophistication of the exercises increase gradually but steadily.

DYNAMIC ROSTER
Students’ submissions are stored in a roster that indicates whether the submission is correct, how many attempts were made, and the actual code submissions from each attempt.

PEARSON eTEXT
The Pearson eText gives students access to their textbook anytime, anywhere.

STEP-BY-STEP VIDEONOTE TUTORIALS
These step-by-step video tutorials enhance the programming concepts presented in select Pearson textbooks.

For more information and titles available with MyLab Programming, please visit www.pearson.com/mylab/programming

Copyright © 2018 Pearson Education, Inc. or its affiliate(s). All rights reserved. HELO88173 · 11/15
## Brief Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>Introduction to Python Programming</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Data and Definite Loops</td>
<td>57</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Parameters and Graphics</td>
<td>132</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Conditional Execution</td>
<td>219</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Program Logic and Indefinite Loops</td>
<td>295</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>File Processing</td>
<td>364</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Lists</td>
<td>418</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>Dictionaries and Sets</td>
<td>517</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Recursion</td>
<td>563</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Searching and Sorting</td>
<td>636</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Classes and Objects</td>
<td>686</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Functional Programming</td>
<td>738</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Python Summary</td>
<td>785</td>
</tr>
</tbody>
</table>
# Contents

## Chapter 1 Introduction to Python Programming

1. **Basic Computing Concepts**
   - Why Programming? 2
   - Hardware and Software 3
   - The Digital Realm 4
   - The Process of Programming 6
   - Why Python? 7
   - The Python Programming Environment 8

2. **And Now: Python**
   - Printing Output 14
   - String Literals (Strings) 15
   - Escape Sequences 16
   - Printing a Complex Figure 18
   - Comments, Whitespace, and Readability 19

3. **Program Errors**
   - Syntax Errors 23
   - Logic Errors (Bugs) 25

4. **Procedural Decomposition**
   - Functions 27
   - Flow of Control 31
   - Identifiers and Keywords 34
   - Functions That Call Other Functions 36
   - An Example Runtime Error 38

5. **Case Study: Drawing Figures**
   - Structured Version 41
   - Final Version without Redundancy 42
   - Analysis of Flow of Execution 44

## Chapter 2 Data and Definite Loops

1. **Basic Data Concepts**
   - Types 58
   - Expressions 59
   - Literals 62

2. **Case Study: Drawing Figures**
   - Structured Version 41
   - Final Version without Redundancy 42
   - Analysis of Flow of Execution 44
## Contents

Arithmetic Operators 62  
Precedence 66  
Mixing and Converting Types 69  

### 2.2 Variables 70
- A Program with Variables 74  
- Increment/Decrement Operators 79  
- Printing Multiple Values 80  

### 2.3 The for Loop 83
- Using a Loop Variable 87  
- Details about Ranges 90  
- String Multiplication and Printing Partial Lines 94  
- Nested for Loops 98  

### 2.4 Managing Complexity 101
- Scope 101  
- Pseudocode 103  
- Constants 108  

### 2.5 Case Study: Hourglass Figure 111
- Problem Decomposition and Pseudocode 112  
- Initial Structured Version 114  
- Adding a Constant 115  

## Chapter 3 Parameters and Graphics 132

### 3.1 Parameters 133
- The Mechanics of Parameters 139  
- Limitations of Parameters 141  
- Multiple Parameters 145  
- Parameters versus Constants 148  
- Optional Parameters 149  

### 3.2 Returning Values 151
- The `math` Module 153  
- The `random` Module 156  
- Defining Functions That Return Values 160  
- Returning Multiple Values 165  

### 3.3 Interactive Programs 167
- Sample Interactive Program 170  

### 3.4 Graphics 172
- Introduction to `DrawingPanel` 173  
- Drawing Lines and Shapes 176  
- Colors 179  
- Drawing with Loops 183  
- Text and Fonts 186  

## Contents

<table>
<thead>
<tr>
<th>Images</th>
<th>188</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural Decomposition with Graphics</td>
<td>189</td>
</tr>
</tbody>
</table>

### 3.5 Case Study: Projectile Trajectory

<table>
<thead>
<tr>
<th>Unstructured Solution</th>
<th>195</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Solution</td>
<td>196</td>
</tr>
<tr>
<td>Graphical Version</td>
<td>199</td>
</tr>
</tbody>
</table>

### Chapter 4 Conditional Execution

**4.1 if/else Statements**

<table>
<thead>
<tr>
<th>Relational Operators</th>
<th>222</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested if/else Statements</td>
<td>225</td>
</tr>
<tr>
<td>Factoring if/else Statements</td>
<td>231</td>
</tr>
<tr>
<td>Testing Multiple Conditions</td>
<td>232</td>
</tr>
</tbody>
</table>

**4.2 Cumulative Algorithms**

<table>
<thead>
<tr>
<th>Cumulative Sum</th>
<th>233</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min/Max Loops</td>
<td>236</td>
</tr>
<tr>
<td>Cumulative Sum with if</td>
<td>239</td>
</tr>
<tr>
<td>Roundoff Errors</td>
<td>242</td>
</tr>
</tbody>
</table>

**4.3 Functions with Conditional Execution**

<table>
<thead>
<tr>
<th>Preconditions and Postconditions</th>
<th>245</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raising Exceptions</td>
<td>246</td>
</tr>
<tr>
<td>Revisiting Return Values</td>
<td>250</td>
</tr>
<tr>
<td>Reasoning about Paths</td>
<td>253</td>
</tr>
</tbody>
</table>

**4.4 Strings**

<table>
<thead>
<tr>
<th>String Methods</th>
<th>257</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing Characters by Index</td>
<td>260</td>
</tr>
<tr>
<td>Converting between Letters and Numbers</td>
<td>264</td>
</tr>
<tr>
<td>Cumulative Text Algorithms</td>
<td>267</td>
</tr>
</tbody>
</table>

**4.5 Case Study: Basal Metabolic Rate**

| One-Person Unstructured Solution | 270 |
| Two-Person Unstructured Solution | 273 |
| Two-Person Structured Solution | 275 |
| Procedural Design Heuristics | 280 |

### Chapter 5 Program Logic and Indefinite Loops

**5.1 The while Loop**

| A Loop to Find the Smallest Divisor | 298 |
| Loop Priming | 300 |
Contents

5.2 Fencepost Algorithms 303
   Fencepost with if 306
   Sentinel Loops 308
   Sentinel with Min/Max 310

5.3 Boolean Logic 312
   Logical Operators 315
   Boolean Variables and Flags 318
   Predicate Functions 320
   Boolean Zen 322
   Short-Circuited Evaluation 325

5.4 Robust Programs 329
   The try/except Statement 330
   Handling User Errors 333

5.5 Assertions and Program Logic 335
   Reasoning about Assertions 337
   A Detailed Assertions Example 339

5.6 Case Study: Number Guessing Game 343
   Initial Version without Hinting 344
   Randomized Version with Hinting 346
   Final Robust Version 348

Chapter 6 File Processing 364

6.1 File-Reading Basics 365
   Data and Files 365
   Reading a File in Python 369
   Line-Based File Processing 372
   Structure of Files and Consuming Input 373
   Prompting for a File 378

6.2 Token-Based Processing 381
   Numeric Input 383
   Handling Invalid Input 385
   Mixing Lines and Tokens 386
   Handling Varying Numbers of Tokens 388
   Complex Input Files 392

6.3 Advanced File Processing 394
   Multi-Line Input Records 395
   File Output 397
   Reading Data from the Web 400

6.4 Case Study: ZIP Code Lookup 403
## Contents

### Chapter 7 Lists

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 List Basics</td>
<td>419</td>
</tr>
<tr>
<td>Creating Lists</td>
<td>420</td>
</tr>
<tr>
<td>Accessing List Elements</td>
<td>423</td>
</tr>
<tr>
<td>Traversing a List</td>
<td>429</td>
</tr>
<tr>
<td>A Complete List Program</td>
<td>430</td>
</tr>
<tr>
<td>Random Access</td>
<td>434</td>
</tr>
<tr>
<td>List Methods</td>
<td>435</td>
</tr>
<tr>
<td>7.2 List-Traversal Algorithms</td>
<td>443</td>
</tr>
<tr>
<td>Lists as Parameters</td>
<td>443</td>
</tr>
<tr>
<td>Searching a List</td>
<td>445</td>
</tr>
<tr>
<td>Replacing and Removing Values</td>
<td>449</td>
</tr>
<tr>
<td>Reversing a List</td>
<td>450</td>
</tr>
<tr>
<td>Shifting Values in a List</td>
<td>456</td>
</tr>
<tr>
<td>Nested Loop Algorithms</td>
<td>462</td>
</tr>
<tr>
<td>List Comprehensions</td>
<td>463</td>
</tr>
<tr>
<td>7.3 Reference Semantics</td>
<td>464</td>
</tr>
<tr>
<td>Values and References</td>
<td>465</td>
</tr>
<tr>
<td>Modifying a List Parameter</td>
<td>468</td>
</tr>
<tr>
<td>The Value None</td>
<td>470</td>
</tr>
<tr>
<td>Mutability</td>
<td>472</td>
</tr>
<tr>
<td>Tuples</td>
<td>476</td>
</tr>
<tr>
<td>7.4 Multidimensional Lists</td>
<td>482</td>
</tr>
<tr>
<td>Rectangular Lists</td>
<td>483</td>
</tr>
<tr>
<td>Jagged Lists</td>
<td>485</td>
</tr>
<tr>
<td>Lists of Pixels</td>
<td>491</td>
</tr>
<tr>
<td>7.5 Case Study: Benford's Law</td>
<td>495</td>
</tr>
<tr>
<td>Tallying Values</td>
<td>497</td>
</tr>
<tr>
<td>Completing the Program</td>
<td>501</td>
</tr>
</tbody>
</table>

### Chapter 8 Dictionaries and Sets

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Dictionary Basics</td>
<td>518</td>
</tr>
<tr>
<td>Creating a Dictionary</td>
<td>521</td>
</tr>
<tr>
<td>Dictionary Operations</td>
<td>524</td>
</tr>
<tr>
<td>Looping Over a Dictionary</td>
<td>527</td>
</tr>
<tr>
<td>Dictionary Ordering</td>
<td>528</td>
</tr>
<tr>
<td>8.2 Advanced Dictionary Usage</td>
<td>531</td>
</tr>
<tr>
<td>Dictionary for Tallying</td>
<td>531</td>
</tr>
</tbody>
</table>
## Contents

Nested Collections 536  
Dictionary Comprehensions 541  

### 8.3 Sets 543  
Set Basics 543  
Set Operations 547  
Set Efficiency 549  
Set Example: Lottery 552  

### Chapter 9 Recursion 563  

#### 9.1 Thinking Recursively 564  
A Nonprogramming Example 564  
Iteration to Recursion 567  
Structure of Recursive Solutions 570  
Reversing a File 573  
The Recursive Call Stack 575  

#### 9.2 Recursive Functions and Data 581  
Integer Exponentiation 581  
Greatest Common Divisor 584  
Directory Crawler 590  

#### 9.3 Recursive Graphics 594  
Cantor Set 594  
Sierpinski Triangle 597  

#### 9.4 Recursive Backtracking 601  
Traveling North/East 601  
Eight Queens Puzzle 607  
Stopping after One Solution 615  

#### 9.5 Case Study: Prefix Evaluator 618  
Infix, Prefix, and Postfix Notation 618  
Evaluating Prefix Expressions 619  
Complete Program 623  

### Chapter 10 Searching and Sorting 636  

#### 10.1 Searching and Sorting Libraries 637  
Binary Search 638  
Sorting 644  
Shuffling 645  

#### 10.2 Program Complexity 646  
Empirical Analysis 649  
Complexity Classes 656
## Contents

### 10.3 Implementing Searching and Sorting Algorithms
- Sequential Search 658
- Binary Search 659
- Recursive Binary Search 662
- Selection Sort 664

### 10.4 Case Study: Implementing Merge Sort
- Splitting and Merging lists 668
- Recursive Merge Sort 671
- Runtime Performance 674
- Hybrid Approach 677

### Chapter 11 Classes and Objects

#### 11.1 Object-Oriented Programming
- Classes and Objects 688
- Date Objects 690

#### 11.2 Object State and Behavior
- Data Attributes 691
- Initializers 694
- Methods 698
- Accessors and Mutators 702
- Making Objects Printable 705
- Object Equality and Ordering 707

#### 11.3 Encapsulation
- Motivation for Encapsulation 711
- Private Attributes and Properties 711
- Class Invariants 717

#### 11.4 Case Study: Designing a Stock Class
- Object-Oriented Design Heuristics 723
- Stock Attributes and Method Headers 725
- Stock Method and Property Implementation 727

### Chapter 12 Functional Programming

#### 12.1 Functional Programming Concepts
- Side Effects 740
- First-Class Functions 742
- Higher-Order Functions 744
- Lambda Expressions 746

#### 12.2 Functional Operations on Collections
- Using Map 751
- Using Filter 752