Discovering Computer Science: Interdisciplinary Problems, Principles, and Python Programming is a problem-oriented introduction to computational problem solving and programming in Python, appropriate for a first course for computer science majors, a more targeted disciplinary computing course or, at a slower pace, any introductory computer science course for a general audience.

Realizing that an organization around language features only resonates with a narrow audience, this textbook instead connects programming to students' prior interests using a range of authentic problems from the natural and social sciences and the digital humanities. The presentation begins with an introduction to the problem-solving process, contextualizing programming as an essential component. Then, as the book progresses, each chapter guides students through solutions to increasingly complex problems, using a spiral approach to introduce Python language features.

The text also places programming in the context of fundamental computer science principles, such as abstraction, efficiency, testing, and algorithmic techniques, offering glimpses of topics that are traditionally put off until later courses.

This book contains 30 well-developed independent projects that encourage students to explore questions across disciplinary boundaries, over 750 homework exercises, and 300 integrated reflection questions engage students in problem solving and active reading.

The accompanying website — https://www.discoveringcs.net — includes more advanced content, solutions to selected exercises, sample code and data files, and pointers for further exploration.

“Havill’s problem-driven approach introduces algorithmic concepts in context and motivates students with a wide range of interests and backgrounds.”
— Janet Davis, Associate Professor and Microsoft Chair of Computer Science, Whitman College

“This book looks really great and takes exactly the approach I think should be used for a CS 1 course. I think it really fills a need in the textbook landscape.”
— Marie desJardins, Dean of the College of Organizational, Computational, and Information Sciences, Simmons University

“Discovering Computer Science is a refreshing departure from introductory programming texts, offering students a much more sincere introduction to the breadth and complexity of this ever-growing field.”
— James Deverick, Senior Lecturer, The College of William and Mary

“This unique introduction to the science of computing guides students through broad and universal approaches to problem solving in a variety of contexts and their ultimate implementation as computer programs.”
— Daniel Kaplan, DeWitt Wallace Professor, Macalester College
Contents

Data visualization 62

2.3 FUNCTIONAL ABSTRACTION 66
  Function parameters 69

2.4 PROGRAMMING IN STYLE 77
  Program structure 78
  Documentation 79
    Tangent 2.2 Global variables 80
  Self-documenting code 83

2.5 A RETURN TO FUNCTIONS 87
  The math module 88
  Writing functions with return values 89
  Return vs. print 92

2.6 SCOPE AND NAMESPACES 97
  Local namespaces 98
  The global namespace 101

2.7 SUMMARY AND FURTHER DISCOVERY 105

Chapter 3 • Inside a Computer 107

3.1 COMPUTERS ARE DUMB 108
  Tangent 3.1 High performance computing 109
  Machine language 111
    Tangent 3.2 Byte code 112

3.2 EVERYTHING IS BITS 112
  Bits are switches 112
  Bits can represent anything 113
    Tangent 3.3 Hexadecimal notation 114
  Computing with bits 114

3.3 COMPUTER ARITHMETIC 118
  Limited precision 118
    Tangent 3.4 Floating point notation 120
  Error propagation 120
  Division 121
  Complex numbers 122

*3.4 BINARY ARITHMETIC 124
  More limited precision

*Sections with *** in lieu of a page number are available on the book website.
Negative integers
Designing an adder
Implementing an adder

3.5 THE UNIVERSAL MACHINE 124
3.6 SUMMARY AND FURTHER DISCOVERY 126

CHAPTER 4 • Growth and Decay 129

4.1 ACCUMULATORS 130
Managing a fishing pond 130
Measuring network value 136
Organizing a concert 139

4.2 DATA VISUALIZATION 150

4.3 CONDITIONAL ITERATION 155
When will the fish disappear? 155
When will your nest egg double? 157

*4.4 CONTINUOUS MODELS 159
Difference equations
Radiocarbon dating
Tradeoffs between accuracy and time
Simulating an epidemic

*4.5 NUMERICAL ANALYSIS 160
The harmonic series
Approximating π
Approximating square roots

4.6 SUMMING UP 161

Tangent 4.1 Triangular numbers 163

4.7 FURTHER DISCOVERY 164

*4.8 PROJECTS 166

4.1 Parasitic relationships
4.2 Financial calculators
4.3 Market penetration
4.4 Wolves and moose

CHAPTER 5 • Forks in the Road 165

5.1 RANDOM WALKS 166

Tangent 5.1 Interval notation 167

One small step 167
### Contents

- Monte Carlo simulation  171

*5.2 PSEUDORANDOM NUMBER GENERATORS  
  Implementation  
  Testing randomness

*5.3 SIMULATING PROBABILITY DISTRIBUTIONS  
  The central limit theorem

5.4 BACK TO BOOLEANS  180
  Predicate functions  182
  Short circuit evaluation  183
  DeMorgan's laws  184
  Thinking inside the box  187
  Many happy returns  192

5.5 DEFENSIVE PROGRAMMING  199
  Checking parameters  199
  Assertions  202
  Unit testing  204
    *Tangent 5.2 Unit testing frameworks  205
  Testing floats  207
  Catching exceptions  207

5.6 GUESS MY NUMBER  210
  Ending the game nicely  212
  Friendly hints  213
    *A proper win/lose message  214

5.7 SUMMARY AND FURTHER DISCOVERY  219

*5.8 PROJECTS  
  5.1 The magic of polling
  5.2 Escape!

---

**CHAPTER 6 • Text, Documents, and DNA  221**

6.1 FIRST STEPS  222
  Normalization  223
    *Tangent 6.1 Natural language processing  224
  Tokenization  228
  Creating your own module  232
  Testing your module  233

6.2 TEXT DOCUMENTS  238
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading from text files</td>
</tr>
<tr>
<td>Writing to text files</td>
</tr>
<tr>
<td>Reading from the web</td>
</tr>
<tr>
<td>6.3 ENCODING STRINGS</td>
</tr>
<tr>
<td>Computing checksums</td>
</tr>
<tr>
<td>Unicode</td>
</tr>
<tr>
<td><em>Tangent 6.2 Compressing text files</em></td>
</tr>
<tr>
<td>Indexing and slicing</td>
</tr>
<tr>
<td>6.4 A CONCORDANCE</td>
</tr>
<tr>
<td>Finding a word</td>
</tr>
<tr>
<td>A concordance entry</td>
</tr>
<tr>
<td>A complete concordance</td>
</tr>
<tr>
<td>6.5 WORD FREQUENCY TRENDS</td>
</tr>
<tr>
<td>Finding the frequency of a word</td>
</tr>
<tr>
<td>Getting the frequencies in slices</td>
</tr>
<tr>
<td>Plotting the frequencies</td>
</tr>
<tr>
<td>6.6 COMPARING TEXTS</td>
</tr>
<tr>
<td>Dot plots</td>
</tr>
<tr>
<td>6.7 TIME COMPLEXITY</td>
</tr>
<tr>
<td>Best case vs. worst case</td>
</tr>
<tr>
<td>Asymptotic time complexity</td>
</tr>
<tr>
<td>6.8 COMPUTATIONAL GENOMICS</td>
</tr>
<tr>
<td>A genomics primer</td>
</tr>
<tr>
<td>Basic DNA analysis</td>
</tr>
<tr>
<td>Transforming sequences</td>
</tr>
<tr>
<td>Comparing sequences</td>
</tr>
<tr>
<td>Reading sequence files</td>
</tr>
<tr>
<td>6.9 SUMMARY AND FURTHER DISCOVERY</td>
</tr>
<tr>
<td>*6.10 PROJECTS</td>
</tr>
<tr>
<td>6.1 Polarized politics</td>
</tr>
<tr>
<td>6.2 Finding genes</td>
</tr>
<tr>
<td>Chapter 7 • Data Analysis</td>
</tr>
<tr>
<td>7.1 SUMMARY STATISTICS</td>
</tr>
<tr>
<td>Mean and variance</td>
</tr>
<tr>
<td>Minimum and maximum</td>
</tr>
</tbody>
</table>
7.2 WRANGLING DATA 293
   Smoothing data 294
   A more efficient algorithm 295
   Modifying lists in place 297
   List operators and methods 302
   *List comprehensions 305
       Tangent 7.1 NumPy arrays 306

7.3 TALLYING FREQUENCIES 310
   Word frequencies 310
   Dictionaries 311
       Tangent 7.2 Hash tables 315
   Finding the most frequent word 315
   Bigram frequencies 317
       Tangent 7.3 Sentiment analysis 318

7.4 READING TABULAR DATA 325
   Earthquakes 326

*7.5 DESIGNING EFFICIENT ALGORITHMS ***
   Removing duplicates
   A first algorithm
   A more elegant algorithm
   A more efficient algorithm

*7.6 LINEAR REGRESSION ***

*7.7 DATA CLUSTERING ***
   Defining similarity
   A simple example
   Implementing k-means clustering
   Locating bicycle safety programs

7.8 SUMMARY AND FURTHER DISCOVERY 333
   Tangent 7.4 Privacy in the age of big data 334

*7.9 PROJECTS ***
   7.1 Climate change
   7.2 Does education influence unemployment?
   7.3 Maximizing profit
   7.4 Admissions
   7.5 Preparing for a 100-year flood
   7.6 Voting methods
7.7 Heuristics for traveling salespeople

CHAPTER 8 • Flatland

8.1 TABULAR DATA
Reading a table of temperatures
Tangent 8.1 Pandas

8.2 THE GAME OF LIFE
Creating a grid
Initial configurations
Surveying the neighborhood
Performing one pass
Tangent 8.2 NumPy arrays in two dimensions
Updating the grid

8.3 DIGITAL IMAGES
Colors
Tangent 8.3 Additive vs. subtractive color models
Image filters
Tangent 8.4 Image storage and compression
Transforming images

8.4 SUMMARY AND FURTHER DISCOVERY

*8.5 PROJECTS
8.1 Modeling segregation
8.2 Modeling ferromagnetism
8.3 Growing dendrites
8.4 Simulating an epidemic

CHAPTER 9 • Self-similarity and Recursion

9.1 FRACTALS
Trees
Snowflakes

9.2 RECURSION AND ITERATION
Solving a problem recursively
Palindromes
Guessing passwords

9.3 THE MYTHICAL TOWER OF HANOI
*Is the end of the world nigh?

9.4 RECURSIVE LINEAR SEARCH
Contents

9.5 DIVIDE AND CONQUER 396
Buy low, sell high 397
Navigating a maze 400

*9.6 LINDENMAYER SYSTEMS 400
Formal grammars
L-systems
Implementing L-systems

9.7 SUMMARY AND FURTHER DISCOVERY 405

*9.8 PROJECTS 407
9.1 Lindenmayer’s beautiful plants
9.2 Gerrymandering
9.3 Percolation

Chapter 10 • Organizing Data 407

10.1 BINARY SEARCH 408

Tangent 10.1 Databases 409
Efficiency of iterative binary search 412
A spelling checker 414
Recursive binary search 415
*Efficiency of recursive binary search 416

10.2 SELECTION SORT 418
Implementing selection sort 419
Efficiency of selection sort 422
Querying data 423

10.3 INSERTION SORT 427
Implementing insertion sort 428
Efficiency of insertion sort 430

10.4 EFFICIENT SORTING 433
Merge sort 433
Internal vs. external sorting 437
Efficiency of merge sort 437

*10.5 TRACTABLE AND INTRACTABLE ALGORITHMS 437
Hard problems

10.6 SUMMARY AND FURTHER DISCOVERY 441

*10.7 PROJECTS 447
10.1 Creating a searchable database
10.2 Binary search trees

**CHAPTER 11 • Networks**

11.1 MODELING WITH GRAPHS 444
Making friends 446

11.2 SHORTEST PATHS 451
Breadth-first search 451
Finding the actual paths 455

11.3 IT’S A SMALL WORLD. . . 458
Small world networks 458
Clustering coefficients 459
Scale-free networks 461

11.4 RANDOM GRAPHS 464

11.5 SUMMARY AND FURTHER DISCOVERY 467

*11.6 PROJECTS ***

11.1 Diffusion of ideas and influence
11.2 Slowing an epidemic
11.3 The Oracle of Bacon

**CHAPTER 12 • Object-oriented Design**

12.1 SIMULATING AN EPIDEMIC 470
Object design 471
Person class 472
Augmenting the Person class 477
World class 479
The simulation 481

12.2 OPERATORS AND POLYMORPHISM 486
Designing a Pair ADT 487
Pair class 488
Arithmetic methods 489
Special methods 491
Comparison operators 493
Indexing 494

*12.3 A FLOCKING SIMULATION ***
The World
Boids
Preface

In my view, an introductory computer science course should strive to accomplish three things. First, it should demonstrate to students how computing has become a powerful mode of inquiry, and a vehicle of discovery, in a wide variety of disciplines. This orientation is also inviting to students of the natural and social sciences, and the humanities, who increasingly benefit from an introduction to computational thinking, beyond the limited “black box” recipes often found in manuals and “Computing for X” books. Second, the course should engage students in computational problem solving, and lead them to discover the power of abstraction, efficiency, and data organization in the design of their solutions. Third, the course should teach students how to implement their solutions as computer programs. In learning how to program, students more deeply learn the core principles, and experience the thrill of seeing their solutions come to life.

Unlike most introductory computer science textbooks, which are organized around programming language constructs, I deliberately lead with interdisciplinary problems and techniques. This orientation is more interesting to a more diverse audience, and more accurately reflects the role of programming in problem solving and discovery. A computational discovery does not, of course, originate in a programming language feature in search of an application. Rather, it starts with a compelling problem which is modeled and solved algorithmically, by leveraging abstraction and prior experience with similar problems. Only then is the solution implemented as a program.

Like most introductory computer science textbooks, I introduce programming skills in an incremental fashion, and include many opportunities for students to practice them. The topics in this book are arranged to ease students into computational thinking, and encourage them to incrementally build on prior knowledge. Each chapter focuses on a general class of problems that is tackled by new algorithmic techniques and programming language features. My hope is that students will leave the course, not only with strong programming skills, but with a set of problem solving strategies and simulation techniques that they can apply in their future work, whether or not they take another computer science course.

I use Python to introduce computer programming for two reasons. First, Python’s intuitive syntax allows students to focus on interesting problems and powerful principles, without unnecessary distractions. Learning how to think algorithmically is hard enough without also having to struggle with a non-intuitive syntax. Second, the expressiveness of Python (in particular, low-overhead lists and dictionaries) expands tremendously the range of accessible problems in the introductory course.
Teaching with Python over the last fifteen years has been a revelation; introductory computer science has become fun again.

Changes in the second edition

In this comprehensive, cover-to-cover update, some sections were entirely rewritten while others saw only minor revisions. Here are the highlights:

Problem solving  The new first chapter, *How to Solve It*, sets the stage by focusing on Polya’s elegant four-step problem solving process, adapted to a computational framework. I introduce informal pseudocode, functional decomposition, hand-execution with informal trace tables, and testing, practices that are now carried on throughout the book. The introduction to Python (formally Chapter 2) is integrated into this framework. Chapter 7, *Designing Programs*, from the first edition has been eliminated, with that material spread out more naturally among Chapters 1, 5, and 6 in the second edition.

Chapter 2, *Visualizing Abstraction* (based on the previous Chapter 3), elaborates on the themes in Chapter 1, and their implementations in Python, introducing turtle graphics, functions, and loops. The new Chapter 3, *Inside a Computer* (based on the previous Sections 1.4 and 2.5), takes students on a brief excursion into the simple principles underlying how computers work.

Online materials  To reduce the size of the printed book, we have moved some sections and all of the projects online. These sections are marked in the table of contents with ***. Online materials are still indexed in the main book for convenience.

Exercises  I’ve added exercises to most sections, bringing the total to about 750. Solutions to exercises marked with an asterisk are available online for both students and self-learners.

Digital humanities  The interdisciplinary problems in the first edition were focused primarily in the natural and social sciences. In this edition, especially in Chapters 1, 6, and 7, we have added new material on text analysis techniques commonly used in the “digital humanities.”

Object-oriented design  Chapter 12 begins with a new section to introduce object-oriented design in a more concrete way through the development of an agent-based simulation of a viral epidemic. The following sections flesh out more details on how to implement polymorphic operators and collection classes.
Book website

Online materials for this book are available at

https://www.discoveringCS.net.

Here you will find

- additional “optional” sections, marked with an asterisk in the main text,
- over thirty interdisciplinary programming projects,
- solutions to selected exercises,
- programs and data files referenced in the text, exercises, and projects, and
- pointers for further exploration and links to additional documentation.

To students

Active learning Learning how to solve computational problems and implement them as computer programs requires daily practice. Like an athlete, you will get out of shape and fall behind quickly if you skip it. There are no shortcuts. Your instructor is there to help, but he or she cannot do the work for you.

With this in mind, it is important that you type in and try the examples throughout the text, and then go beyond them. Be curious! There are numbered “Reflection” questions throughout the book that ask you to stop and think about, or apply, something that you just read. Often, the question is answered in the book immediately thereafter, so that you can check your understanding, but peeking ahead will rob you of an important opportunity.

Further discovery There are many opportunities to delve into topics more deeply. “Tangent” boxes scattered throughout the text briefly introduce related, but more technical or applied, topics. For the most part, these are not strictly required to understand what comes next, but I encourage you to read them anyway. In the “Summary and Further Discovery” section of each chapter, you can find both a high-level summary of the chapter and additional pointers to explore chapter topics in more depth.

Exercises and projects At the end of most sections are several programming exercises that ask you to further apply concepts from that section. Often, the exercises assume that you have already worked through all of the examples in that section. Solutions to the starred exercises are available on the book website. There are also more involved projects available on the book website that challenge you to solve a variety of interdisciplinary problems.

No prerequisites The book assumes no prior knowledge of computer science. However, it does assume a modest comfort with high school algebra. In optional sections,
trigonometry is occasionally mentioned, as is the idea of convergence to a limit, but these are not relevant to understanding the main topics in the book.

**Have fun!** Programming and problem solving should be a fun, creative activity. I hope that this book sparks your curiosity and love of learning, and that you enjoy the journey as much as I have enjoyed writing this book.

**To instructors**

This book is appropriate for a traditional CS1 course for majors, a CS0 course for non-majors (at a slower pace and omitting more material), or a targeted introductory computing course for students in the natural sciences, social sciences, or humanities. The approach is gentle and holistic, introducing programming concepts in the context of interdisciplinary problems. We start with problem-solving, featuring pseudocode and hand-execution with trace tables, and carry these techniques forward, especially in the first half of the book.

**Problem focus** Most chapters begin with an interesting problem, and new concepts and programming techniques are introduced in the context of solving it. As new techniques are introduced, students are frequently challenged to re-solve old problems in different ways. They are also encouraged to reuse their previous functions as components in later programs.

**Reflection questions, exercises, and projects** “Reflection” questions are embedded in every section to encourage active reading. These may also be assigned as “reading questions” before class. The end-of-section exercises are appropriate for regular homework, and some more complex ones may form the basis of longer-term assignments. The book website also hosts a few dozen interdisciplinary projects that students may work on independently or in pairs over a longer time frame. I believe that projects like these are crucial for students to develop both problem solving skills and an appreciation for the many fascinating applications of computer science.

**Additional instructor resources** All of the reflection questions and exercises are available to instructors as Jupyter notebooks. Solutions to all exercises and projects are also available. Please visit the publisher’s website to request access.

**Python coverage** This book is not intended to be a Python manual. Some features of the language were intentionally omitted because they would have muddled the core problem solving focus or are not commonly found in other languages that students may see in future CS courses (e.g., simultaneous swap, chained comparisons, zip, enumerate in for loops).

**Topic coverage** There is more in this book than can be covered in a single semester, giving instructors the opportunity to tailor the content to their particular situation
Figure 1  An overview of chapter dependencies.

and interests. As illustrated in Figure 1, Chapters 1–7 form the core of the book, and should be covered sequentially. The remaining chapters can be covered, partially or entirely, at your discretion, although I would expect that most instructors will cover at least parts of Chapters 8–10, and 12 if the course covers object-oriented design. Chapter 11 introduces social network graphs and small-world and scale-free networks as additional powerful applications of dictionaries, and may come any time after Chapter 7. Sections marked with an asterisk are optional, in the sense that they are not assumed for future sections in that chapter. When exercises and projects depend on optional sections, they are also marked with an asterisk, and the dependency is stated at the beginning of the project.

Chapter outlines  The following tables provide brief overviews of what is available in each chapter. Each table’s three columns, reflecting the three parts of the book’s subtitle, provide three lenses through which to view the chapter.

## 1  How to Solve It

<table>
<thead>
<tr>
<th>Sample problems</th>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>reading level</td>
<td>problems, input/output</td>
<td><code>int, float, str</code> types</td>
</tr>
<tr>
<td>counting syllables, words</td>
<td>functional abstraction</td>
<td>arithmetic</td>
</tr>
<tr>
<td>sphere volume</td>
<td>functional decomposition</td>
<td>assignment</td>
</tr>
<tr>
<td>digital music</td>
<td>top-down design</td>
<td>variable names</td>
</tr>
<tr>
<td>search engines</td>
<td>bottom-up implementation</td>
<td>calling built-in functions</td>
</tr>
<tr>
<td>GPS devices</td>
<td>algorithms and programs</td>
<td>using strings</td>
</tr>
<tr>
<td>phone trees</td>
<td>pseudocode</td>
<td>string operators</td>
</tr>
<tr>
<td>wind chill</td>
<td>names as references</td>
<td>print and input</td>
</tr>
<tr>
<td>compounding interest</td>
<td>trace tables</td>
<td>constant- vs. linear-time</td>
</tr>
<tr>
<td>Mad Libs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Preface

### 2 Visualizing Abstraction

<table>
<thead>
<tr>
<th>Sample problems</th>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>visualizing earthquakes</td>
<td>using abstract data types</td>
<td>using classes and objects</td>
</tr>
<tr>
<td>drawing flowers</td>
<td>creating functional abstractions</td>
<td>turtle module</td>
</tr>
<tr>
<td>random walks</td>
<td>functional decomposition</td>
<td>for loops (range and lists)</td>
</tr>
<tr>
<td>ideal gas</td>
<td>bottom-up implementation</td>
<td>using and writing functions</td>
</tr>
<tr>
<td>groundwater flow</td>
<td>turtle graphics</td>
<td>return vs. print</td>
</tr>
<tr>
<td>demand functions</td>
<td>trace tables with loops</td>
<td>namespaces and scope</td>
</tr>
<tr>
<td>reading level</td>
<td></td>
<td>docstrings and comments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>self-documenting code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program structure</td>
</tr>
</tbody>
</table>

### 3 Inside a Computer

<table>
<thead>
<tr>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer organization</td>
<td>int and float types</td>
</tr>
<tr>
<td>machine language</td>
<td>arithmetic errors</td>
</tr>
<tr>
<td>binary representations</td>
<td>true vs. floor division</td>
</tr>
<tr>
<td>computer arithmetic</td>
<td></td>
</tr>
<tr>
<td>finite precision, error propagation</td>
<td></td>
</tr>
<tr>
<td>Boolean logic, truth tables, logic gates</td>
<td></td>
</tr>
<tr>
<td>Turing machines, finite state machines</td>
<td></td>
</tr>
</tbody>
</table>

### 4 Growth and Decay

<table>
<thead>
<tr>
<th>Sample problems</th>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>population models</td>
<td>accumulators</td>
<td>for loops, range</td>
</tr>
<tr>
<td>network value</td>
<td>list accumulators</td>
<td>format strings</td>
</tr>
<tr>
<td>demand and profit</td>
<td>data visualization</td>
<td>matplotlib.pyplot</td>
</tr>
<tr>
<td>loans and investing</td>
<td>conditional iteration</td>
<td>appending to lists</td>
</tr>
<tr>
<td>bacterial growth</td>
<td>classes of growth</td>
<td>while loops</td>
</tr>
<tr>
<td>radiocarbon dating</td>
<td>continuous models</td>
<td></td>
</tr>
<tr>
<td>epidemics (SIR, SIS)</td>
<td>accuracy vs. time</td>
<td></td>
</tr>
<tr>
<td>diffusion models</td>
<td>numerical approximation</td>
<td></td>
</tr>
</tbody>
</table>

### 5 Forks in the Road

<table>
<thead>
<tr>
<th>Sample problems</th>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>random walks</td>
<td>random number generators</td>
<td>random module</td>
</tr>
<tr>
<td>Monte Carlo simulation</td>
<td>simulating probabilities</td>
<td>if/elif/else</td>
</tr>
<tr>
<td>guessing games</td>
<td>flag variables</td>
<td>comparison operators</td>
</tr>
<tr>
<td>polling and sampling</td>
<td>using distributions</td>
<td>Boolean operators</td>
</tr>
<tr>
<td>particle escape</td>
<td>DeMorgan’s laws</td>
<td>short circuit evaluation</td>
</tr>
<tr>
<td></td>
<td>defensive programming</td>
<td>predicate functions</td>
</tr>
<tr>
<td></td>
<td>pre- and post-conditions</td>
<td>assert, isinstance</td>
</tr>
<tr>
<td></td>
<td>unit testing</td>
<td>catching exceptions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>histograms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>while loops</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 6 Text, Documents, and DNA

<table>
<thead>
<tr>
<th>Sample problems</th>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>• text analysis</td>
<td>• functional decomposition</td>
<td>• <code>str</code> class and methods</td>
</tr>
<tr>
<td>• word frequency trends</td>
<td>• unit testing</td>
<td>• iterating over strings, lists</td>
</tr>
<tr>
<td>• checksums</td>
<td>• ASCII, Unicode</td>
<td>• indexing and slicing</td>
</tr>
<tr>
<td>• concordances</td>
<td>• linear-time algorithms</td>
<td>• iterating over indices</td>
</tr>
<tr>
<td>• dot plots, plagiarism</td>
<td>• time complexity</td>
<td>• creating a module</td>
</tr>
<tr>
<td>• congressional votes</td>
<td>• linear search</td>
<td>• text files and the web</td>
</tr>
<tr>
<td>• genomics</td>
<td>• string accumulators</td>
<td>• <code>break</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• nested loops</td>
</tr>
</tbody>
</table>

## 7 Data Analysis

<table>
<thead>
<tr>
<th>Sample problems</th>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>• word, bigram frequencies</td>
<td>• histograms</td>
<td>• <code>list</code> class</td>
</tr>
<tr>
<td>• smoothing data</td>
<td>• hash tables</td>
<td>• indexing and slicing</td>
</tr>
<tr>
<td>• 100-year floods</td>
<td>• tabular data files</td>
<td>• list operators and methods</td>
</tr>
<tr>
<td>• traveling salesman</td>
<td>• efficient algorithm design</td>
<td>• reading CSV files</td>
</tr>
<tr>
<td>• meteorite sites</td>
<td>• linear regression</td>
<td>• modifying lists in place</td>
</tr>
<tr>
<td>• zebra migration</td>
<td>• <code>k</code>-means clustering</td>
<td>• list parameters</td>
</tr>
<tr>
<td>• tumor diagnosis</td>
<td>• heuristics</td>
<td>• tuples</td>
</tr>
<tr>
<td>• supply and demand</td>
<td></td>
<td>• list comprehensions</td>
</tr>
<tr>
<td>• voting methods</td>
<td></td>
<td>• dictionaries</td>
</tr>
</tbody>
</table>

## 8 Flatland

<table>
<thead>
<tr>
<th>Sample problems</th>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>• earthquake data</td>
<td>• 2-D data</td>
<td>• lists of lists</td>
</tr>
<tr>
<td>• Game of Life</td>
<td>• cellular automata</td>
<td>• nested loops</td>
</tr>
<tr>
<td>• image filters</td>
<td>• digital images</td>
<td>• 2-D data in a dictionary</td>
</tr>
<tr>
<td>• racial segregation</td>
<td>• color models</td>
<td></td>
</tr>
<tr>
<td>• ferromagnetism</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• dendrites</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• epidemics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• tumor growth</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## 9 Self-similarity and Recursion

<table>
<thead>
<tr>
<th>Sample problems</th>
<th>Principles</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>• fractals</td>
<td>• self-similarity</td>
<td>• writing recursive functions</td>
</tr>
<tr>
<td>• cracking passwords</td>
<td>• recursion</td>
<td>• divide and conquer</td>
</tr>
<tr>
<td>• Tower of Hanoi</td>
<td>• linear search</td>
<td>• backtracking</td>
</tr>
<tr>
<td>• maximizing profit</td>
<td>• recurrence relations</td>
<td></td>
</tr>
<tr>
<td>• navigating a maze</td>
<td>• divide and conquer</td>
<td></td>
</tr>
<tr>
<td>• Lindenmayer systems</td>
<td>• depth-first search</td>
<td></td>
</tr>
<tr>
<td>• gerrymandering</td>
<td>• grammars</td>
<td></td>
</tr>
<tr>
<td>• percolation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Software assumptions

To follow along in this book and complete the exercises, you will need to have installed Python 3.6 or later on your computer, and have access to IDLE or another programming environment. The book also assumes that you have installed the `matplotlib.pyplot` and `numpy` modules. The easiest way to get this software is to install the free open source Anaconda distribution from http://www.anaconda.com.

Errata

While I (and my students) have ferreted out many errors, readers will inevitably find more. You can find an up-to-date list of errata on the book website. If you find an error in the text or have another suggestion, please let me know at havill@denison.edu.