

CHAPMAN & HALL/CRC  
TEXTBOOKS IN COMPUTING

# DISCOVERING COMPUTER SCIENCE

Interdisciplinary Problems,  
Principles, and Python  
Programming

Jessen Havill

Denison University  
Granville, Ohio, USA



CRC Press

Taylor & Francis Group

Boca Raton London New York

---

CRC Press is an imprint of the  
Taylor & Francis Group, an **informa** business  
A CHAPMAN & HALL BOOK

---

# Contents

---

Preface	xv
Acknowledgments	xxiii
About the author	xxv
<b>CHAPTER 1 ■ What is computation?</b>	<b>1</b>
1.1 PROBLEMS AND ABSTRACTION	2
1.2 ALGORITHMS AND PROGRAMS	4
1.3 EFFICIENT ALGORITHMS	11
Organizing a phone tree	11
A smoothing algorithm	13
A better smoothing algorithm	17
1.4 COMPUTERS ARE DUMB	20
Inside a computer	20
Machine language	21
Everything is bits	22
The universal machine	26
1.5 SUMMARY	29
1.6 FURTHER DISCOVERY	30
<b>CHAPTER 2 ■ Elementary computations</b>	<b>31</b>
2.1 WELCOME TO THE CIRCUS	31
2.2 ARITHMETIC	32
Finite precision	34
Division	34
Order of operations	35
Complex numbers	37
2.3 WHAT'S IN A NAME?	38
2.4 USING FUNCTIONS	45

Built-in functions	45
Strings	47
Modules	51
*2.5 BINARY ARITHMETIC	54
Finite precision	55
Negative integers	56
Designing an adder	57
Implementing an adder	58
2.6 SUMMARY	62
2.7 FURTHER DISCOVERY	62
<b>CHAPTER 3 ■ Visualizing abstraction</b>	<b>65</b>
3.1 DATA ABSTRACTION	66
3.2 VISUALIZATION WITH TURTLES	70
Drawing with iteration	72
3.3 FUNCTIONAL ABSTRACTION	76
Function parameters	78
Let's plant a garden	84
3.4 PROGRAMMING IN STYLE	89
Program structure	89
Documentation	91
Descriptive names and magic numbers	95
3.5 A RETURN TO FUNCTIONS	97
Return vs. print	100
3.6 SCOPE AND NAMESPACES	103
Local namespaces	104
The global namespace	107
3.7 SUMMARY	111
3.8 FURTHER DISCOVERY	112
<b>CHAPTER 4 ■ Growth and decay</b>	<b>113</b>
4.1 DISCRETE MODELS	114
Managing a fishing pond	114
Measuring network value	121
Organizing a concert	124
4.2 VISUALIZING POPULATION CHANGES	136
4.3 CONDITIONAL ITERATION	140

*4.4	CONTINUOUS MODELS	145
	Difference equations	145
	Radiocarbon dating	148
	Tradeoffs between accuracy and time	150
	Propagation of errors	152
	Simulating an epidemic	153
*4.5	NUMERICAL ANALYSIS	159
	The harmonic series	159
	Approximating $\pi$	162
	Approximating square roots	164
4.6	SUMMING UP	167
4.7	FURTHER DISCOVERY	171
4.8	PROJECTS	171
	Project 4.1 Parasitic relationships	171
	Project 4.2 Financial calculators	173
	*Project 4.3 Market penetration	177
	*Project 4.4 Wolves and moose	180
<b>CHAPTER 5 ■ Forks in the road</b>		<b>185</b>
5.1	RANDOM WALKS	185
	A random walk in Monte Carlo	192
	Histograms	195
*5.2	PSEUDORANDOM NUMBER GENERATORS	200
	Implementation	201
	Testing randomness	203
*5.3	SIMULATING PROBABILITY DISTRIBUTIONS	205
	The central limit theorem	206
5.4	BACK TO BOOLEANS	209
	Short circuit evaluation	212
	Complex expressions	214
	*Using truth tables	216
	Many happy returns	218
5.5	A GUESSING GAME	224
5.6	SUMMARY	233
5.7	FURTHER DISCOVERY	234
5.8	PROJECTS	234

Project 5.1 The magic of polling	234
Project 5.2 Escape!	237
<b>CHAPTER 6 ■ Text, documents, and DNA</b>	<b>241</b>
6.1 COUNTING WORDS	242
6.2 TEXT DOCUMENTS	250
Reading from text files	250
Writing to text files	253
Reading from the web	254
6.3 ENCODING STRINGS	259
Indexing and slicing	259
Creating modified strings	261
Encoding characters	263
6.4 LINEAR-TIME ALGORITHMS	270
Asymptotic time complexity	274
6.5 ANALYZING TEXT	279
Counting and searching	279
A concordance	284
6.6 COMPARING TEXTS	289
*6.7 GENOMICS	297
A genomics primer	297
Basic DNA analysis	301
Transforming sequences	302
Comparing sequences	304
Reading sequence files	306
6.8 SUMMARY	312
6.9 FURTHER DISCOVERY	313
6.10 PROJECTS	313
Project 6.1 Polarized politics	313
*Project 6.2 Finding genes	316
<b>CHAPTER 7 ■ Designing programs</b>	<b>321</b>
7.1 HOW TO SOLVE IT	322
Understand the problem	323
Design an algorithm	324
Implement your algorithm as a program	327
Analyze your program for clarity, correctness, and efficiency	330

*7.2	DESIGN BY CONTRACT	331
	Preconditions and postconditions	331
	Checking parameters	332
	Assertions	334
*7.3	TESTING	340
	Unit testing	340
	Regression testing	342
	Designing unit tests	343
	Testing floating point values	347
7.4	SUMMARY	350
7.5	FURTHER DISCOVERY	350
<b>CHAPTER 8 ■ Data analysis</b>		<b>351</b>
<hr/>		
8.1	SUMMARIZING DATA	351
8.2	CREATING AND MODIFYING LISTS	360
	List accumulators, redux	360
	Lists are mutable	361
	Tuples	365
	List operators and methods	366
	*List comprehensions	368
8.3	FREQUENCIES, MODES, AND HISTOGRAMS	373
	Tallying values	373
	Dictionaries	374
8.4	READING TABULAR DATA	384
*8.5	DESIGNING EFFICIENT ALGORITHMS	390
	A first algorithm	391
	A more elegant algorithm	399
	A more efficient algorithm	400
*8.6	LINEAR REGRESSION	403
*8.7	DATA CLUSTERING	409
	Defining similarity	410
	A $k$ -means clustering example	411
	Implementing $k$ -means clustering	414
	Locating bicycle safety programs	416
8.8	SUMMARY	421
8.9	FURTHER DISCOVERY	421

8.10	PROJECTS	422
	Project 8.1 Climate change	422
	Project 8.2 Does education influence unemployment?	425
	Project 8.3 Maximizing profit	427
	Project 8.4 Admissions	428
	*Project 8.5 Preparing for a 100-year flood	430
	Project 8.6 Voting methods	435
	Project 8.7 Heuristics for traveling salespeople	438
<b>CHAPTER 9 ■ Flatland</b>		<b>443</b>
9.1	TWO-DIMENSIONAL DATA	443
9.2	THE GAME OF LIFE	449
	Creating a grid	451
	Initial configurations	452
	Surveying the neighborhood	453
	Performing one pass	454
	Updating the grid	457
9.3	DIGITAL IMAGES	461
	Colors	461
	Image filters	463
	Transforming images	467
9.4	SUMMARY	471
9.5	FURTHER DISCOVERY	471
9.6	PROJECTS	471
	Project 9.1 Modeling segregation	471
	Project 9.2 Modeling ferromagnetism	473
	Project 9.3 Growing dendrites	474
<b>CHAPTER 10 ■ Self-similarity and recursion</b>		<b>477</b>
10.1	FRACTALS	477
	A fractal tree	479
	A fractal snowflake	481
10.2	RECURSION AND ITERATION	488
	Solving a problem recursively	491
	Palindromes	492
	Guessing passwords	495
10.3	THE MYTHICAL TOWER OF HANOI	500

*Is the end of the world nigh?	502
10.4 RECURSIVE LINEAR SEARCH	503
Efficiency of recursive linear search	505
10.5 DIVIDE AND CONQUER	508
Buy low, sell high	508
Navigating a maze	512
*10.6 LINDENMAYER SYSTEMS	518
Formal grammars	518
Implementing L-systems	522
10.7 SUMMARY	525
10.8 FURTHER DISCOVERY	526
10.9 PROJECTS	526
*Project 10.1 Lindenmayer's beautiful plants	526
Project 10.2 Gerrymandering	531
Project 10.3 Percolation	536
<b>CHAPTER 11 ■ Organizing data</b>	<b>541</b>
11.1 BINARY SEARCH	542
Efficiency of iterative binary search	546
A spelling checker	548
Recursive binary search	549
Efficiency of recursive binary search	550
11.2 SELECTION SORT	553
Implementing selection sort	553
Efficiency of selection sort	557
Querying data	558
11.3 INSERTION SORT	563
Implementing insertion sort	564
Efficiency of insertion sort	566
11.4 EFFICIENT SORTING	570
Internal vs. external sorting	574
Efficiency of merge sort	574
*11.5 TRACTABLE AND INTRACTABLE ALGORITHMS	577
Hard problems	579
11.6 SUMMARY	580
11.7 FURTHER DISCOVERY	581



11.8	PROJECTS	581
	Project 11.1 Creating a searchable database	581
	Project 11.2 Binary search trees	581
<b>CHAPTER 12 ■ Networks</b>		<b>587</b>
12.1	MODELING WITH GRAPHS	588
	Making friends	590
12.2	SHORTEST PATHS	594
	Finding the actual paths	598
12.3	IT'S A SMALL WORLD. . .	601
	Clustering coefficients	603
	Scale-free networks	605
12.4	RANDOM GRAPHS	608
12.5	SUMMARY	611
12.6	FURTHER DISCOVERY	611
12.7	PROJECTS	612
	Project 12.1 Diffusion of ideas and influence	612
	Project 12.2 Slowing an epidemic	614
	Project 12.3 The Oracle of Bacon	616
<b>CHAPTER 13 ■ Abstract data types</b>		<b>621</b>
13.1	DESIGNING CLASSES	622
	Implementing a class	625
	Documenting a class	632
13.2	OPERATORS AND SPECIAL METHODS	637
	String representations	637
	Arithmetic	638
	Comparison	640
	Indexing	642
13.3	MODULES	645
	Namespaces, redux	646
13.4	A FLOCKING SIMULATION	648
	The World ADT	649
	The Boid ADT	655
13.5	A STACK ADT	665
13.6	A DICTIONARY ADT	671
	Hash tables	672

Implementing a hash table	673
Implementing indexing	676
ADTs vs. data structures	678
13.7 SUMMARY	682
13.8 FURTHER DISCOVERY	682
13.9 PROJECTS	683
Project 13.1 Tracking GPS coordinates	683
Project 13.2 Economic mobility	687
Project 13.3 Slime mold aggregation	690
Project 13.4 Boids in space	692
<b>APPENDIX A ■ Installing Python</b>	<b>697</b>
A.1 AN INTEGRATED DISTRIBUTION	697
A.2 MANUAL INSTALLATION	697
<b>APPENDIX B ■ Python library reference</b>	<b>701</b>
B.1 MATH MODULE	701
B.2 TURTLE METHODS	702
B.3 SCREEN METHODS	703
B.4 MATPLOTLIB.PYPLOT MODULE	704
B.5 RANDOM MODULE	704
B.6 STRING METHODS	705
B.7 LIST METHODS	706
B.8 IMAGE MODULE	706
B.9 SPECIAL METHODS	707
Bibliography	709
Index	713



---

# 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, who increasingly benefit from an introduction to computational thinking, beyond the limited “black box” recipes often found in manuals. 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 ten years has been a revelation; introductory computer science has become fun again.

## Web resources

The text, exercises, and projects often refer to files on the book’s accompanying web site, which can be found at

`http://discoverCS.denison.edu` .

This web site also includes pointers for further exploration, links to additional documentation, and errata.

## To students

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.

There are many opportunities to delve into topics more deeply. Boxes scattered throughout the text briefly introduce related, but more technical, 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 “Further discovery” section of each chapter, you can find additional pointers to explore chapter topics in more depth.

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. All later chapters conclude with a selection of more involved interdisciplinary projects that you may be asked by your instructor to tackle.

The book assumes no prior knowledge of computer science. However, it does assume a modest comfort with high school algebra and mathematical functions. Occasionally, trigonometry is mentioned, as is the idea of convergence to a limit, but these are not crucial to an understanding of the main topics in this book.

## To instructors

This book may be appropriate for a traditional CS1 course for majors, a CS0 course for non-majors (at a slower pace and omitting more material), or an introductory computing course for students in the natural and/or social sciences.

As suggested above, I emphasize computer science principles and the role of abstraction, both functional and data, throughout the book. I motivate functions as implementations of functional abstractions, and point out that strings, lists, and dictionaries are all abstract data types that allow us to solve more interesting problems than would otherwise be possible. I introduce the idea of time complexity

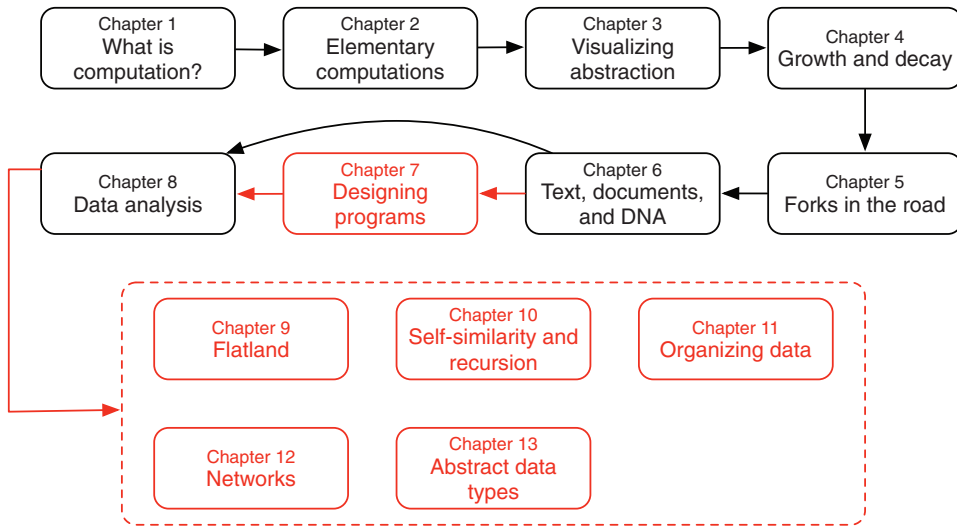


Figure 1 An overview of chapter dependencies.

intuitively, without formal definitions, in the first chapter and return to it several times as more sophisticated algorithms are developed. The book uses a spiral approach for many topics, returning to them repeatedly in increasingly complex contexts. Where appropriate, I also weave into the book topics that are traditionally left for later computer science courses. A few of these are presented in boxes that may be covered at your discretion. None of these topics is introduced rigorously, as they would be in a data structures course. Rather, I introduce them informally and intuitively to give students a sense of the problems and techniques used in computer science. I hope that the tables below will help you navigate the book, and see where particular topics are covered.

This book contains over 600 end-of-section exercises and over 300 in-text reflection questions that may be assigned as homework or discussed in class. At the end of most chapters is a selection of projects (about 30 in all) 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.

Because this book is intended for a student who may take additional courses in computer science and learn other programming languages, I intentionally omit some features of Python that are not commonly found elsewhere (e.g., simultaneous swap, chained comparisons, `enumerate` in `for` loops). You may, of course, supplement with these additional syntactical features.

There is more in this book than can be covered in a single semester, giving an instructor the opportunity to tailor the content to his or her particular situation and interests. Generally speaking, as illustrated in Figure 1, Chapters 1–6 and 8 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 7, 10, 11, and 13. Chapter 7 contains

additional material on program design, including design by contract, assertions and unit testing that may be skipped without consequences for later chapters. Chapters 9–13 are, more or less, independent of each other. Sections marked with an asterisk are optional, in the sense that they are not assumed for future sections in that chapter. When 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 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. The first column lists a selection of representative *problems* that are used to motivate the material. The second column lists computer science *principles* that are introduced in that chapter. Finally, the third column lists Python *programming* topics that are either introduced or reinforced in that chapter to implement the principles and/or solve the problems.

### Chapter 1. What is computation?

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• digital music</li> <li>• search engines</li> <li>• GPS devices</li> <li>• smoothing data</li> <li>• phone trees</li> </ul>	<ul style="list-style-type: none"> <li>• problems, input/output</li> <li>• abstraction</li> <li>• algorithms and programs</li> <li>• computer architecture</li> <li>• binary representations</li> <li>• time complexity</li> <li>• Turing machine</li> </ul>	—

### Chapter 2. Elementary computations

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• wind chill</li> <li>• geometry</li> <li>• compounding interest</li> <li>• Mad Libs</li> </ul>	<ul style="list-style-type: none"> <li>• finite precision</li> <li>• names as references</li> <li>• using functional abstractions</li> <li>• binary addition</li> </ul>	<ul style="list-style-type: none"> <li>• <code>int</code> and <code>float</code> numeric types</li> <li>• arithmetic and the <code>math</code> module</li> <li>• variable names and assignment</li> <li>• calling built-in functions</li> <li>• using strings, <code>+</code> and <code>*</code> operators</li> <li>• <code>print</code> and <code>input</code></li> </ul>

### Chapter 3. Visualizing abstraction

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• visualizing an archaeological dig</li> <li>• random walks</li> <li>• ideal gas</li> <li>• groundwater flow</li> <li>• demand functions</li> </ul>	<ul style="list-style-type: none"> <li>• using abstract data types</li> <li>• creating functional abstractions</li> <li>• basic functional decomposition</li> </ul>	<ul style="list-style-type: none"> <li>• using classes and objects</li> <li>• <code>turtle</code> module</li> <li>• basic <code>for</code> loops</li> <li>• writing functions</li> <li>• namespaces</li> <li>• docstrings and comments</li> </ul>

*Chapter 4. Growth and decay*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• network value</li> <li>• demand and profit</li> <li>• loans and investing</li> <li>• bacterial growth</li> <li>• radiocarbon dating</li> <li>• diffusion models               <ul style="list-style-type: none"> <li>– SIR, SIS, Bass</li> </ul> </li> <li>• competition models               <ul style="list-style-type: none"> <li>– Nicholson-Bailey</li> <li>– Lotka-Volterra</li> <li>– indirect</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• accumulators</li> <li>• list accumulators</li> <li>• difference equations</li> <li>• approximating continuous models</li> <li>• accuracy vs. time</li> <li>• error propagation</li> <li>• numerical approximation</li> <li>• classes of growth</li> </ul>	<ul style="list-style-type: none"> <li>• <b>for</b> loops</li> <li>• format strings</li> <li>• <b>range</b></li> <li>• <b>matplotlib</b></li> <li>• appending to lists</li> <li>• <b>while</b> loops</li> </ul>

*Chapter 5. Forks in the road*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• random walks</li> <li>• guessing games</li> <li>• polling and sampling</li> <li>• particle escape</li> </ul>	<ul style="list-style-type: none"> <li>• Monte Carlo simulation</li> <li>• pseudorandom number generators</li> <li>• simulating probabilities</li> <li>• flag variables</li> <li>• using uniform and normal distributions</li> <li>• DeMorgan's laws</li> </ul>	<ul style="list-style-type: none"> <li>• <b>random</b> module</li> <li>• <b>if/elif/else</b></li> <li>• comparison operators</li> <li>• Boolean operators</li> <li>• <b>matplotlib</b> histograms</li> <li>• <b>while</b> loops</li> </ul>

*Chapter 6. Text, documents, and DNA*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• word count</li> <li>• textual analysis</li> <li>• parsing XML</li> <li>• checksums</li> <li>• concordances</li> <li>• detecting plagiarism</li> <li>• congressional votes</li> <li>• genomics</li> </ul>	<ul style="list-style-type: none"> <li>• ASCII, Unicode</li> <li>• linear-time algorithms</li> <li>• asymptotic time complexity</li> <li>• linear search</li> <li>• dot plots</li> <li>• string accumulators</li> </ul>	<ul style="list-style-type: none"> <li>• <b>str</b> class and methods</li> <li>• iterating over strings</li> <li>• indexing and slices</li> <li>• iterating over indices</li> <li>• reading and writing text files</li> <li>• nested loops</li> </ul>

*Chapter 7. Designing programs*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• word frequency analysis</li> </ul>	<ul style="list-style-type: none"> <li>• problem solving</li> <li>• top-down design</li> <li>• pre and postconditions</li> <li>• assertions</li> <li>• unit testing</li> </ul>	<ul style="list-style-type: none"> <li>• <b>assert</b> statement</li> <li>• conditional execution of <b>main</b></li> <li>• writing modules</li> </ul>



*Chapter 8. Data analysis*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• 100-year floods</li> <li>• traveling salesman</li> <li>• Mohs scale</li> <li>• meteorite sites</li> <li>• zebra migration</li> <li>• tumor diagnosis</li> <li>• education levels</li> <li>• supply and demand</li> <li>• voting methods</li> </ul>	<ul style="list-style-type: none"> <li>• histograms</li> <li>• hash tables</li> <li>• tabular data files</li> <li>• efficient algorithms</li> <li>• linear regression</li> <li>• <math>k</math>-means clustering</li> <li>• heuristics</li> </ul>	<ul style="list-style-type: none"> <li>• <code>list</code> class</li> <li>• iterating over lists</li> <li>• indexing and slicing</li> <li>• list operators and methods</li> <li>• lists in memory; mutability</li> <li>• list parameters</li> <li>• tuples</li> <li>• list comprehensions</li> <li>• dictionaries</li> </ul>

*Chapter 9. Flatland*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• earthquake data</li> <li>• Game of Life</li> <li>• image filters</li> <li>• racial segregation</li> <li>• ferromagnetism</li> <li>• dendrites</li> </ul>	<ul style="list-style-type: none"> <li>• 2-D data</li> <li>• cellular automata</li> <li>• digital images</li> <li>• color models</li> </ul>	<ul style="list-style-type: none"> <li>• 2-D data in list of lists</li> <li>• nested loops</li> <li>• 2-D data in a dictionary</li> </ul>

*Chapter 10. Self-similarity and recursion*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• fractals</li> <li>• cracking passwords</li> <li>• Tower of Hanoi</li> <li>• maximizing profit</li> <li>• path through a maze</li> <li>• Lindenmayer system</li> <li>• electoral districting</li> <li>• percolation</li> </ul>	<ul style="list-style-type: none"> <li>• self-similarity</li> <li>• recursion</li> <li>• linear search</li> <li>• recurrence relations</li> <li>• divide and conquer</li> <li>• depth-first search</li> <li>• grammars</li> </ul>	<ul style="list-style-type: none"> <li>• writing recursive functions</li> </ul>

*Chapter 11. Organizing data*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• spell check</li> <li>• querying data sets</li> </ul>	<ul style="list-style-type: none"> <li>• binary search</li> <li>• recurrence relations</li> <li>• basic sorting algorithms</li> <li>• quadratic-time algorithms</li> <li>• parallel lists</li> <li>• merge sort</li> <li>• intractability</li> <li>• <math>P=NP</math> (intuition)</li> <li>• Moore's law</li> <li>• binary search trees</li> </ul>	<ul style="list-style-type: none"> <li>• nested loops</li> <li>• writing recursive functions</li> </ul>

*Chapter 12. Networks*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• Facebook, Twitter, web graphs</li> <li>• diffusion of ideas</li> <li>• epidemics</li> <li>• Oracle of Bacon</li> </ul>	<ul style="list-style-type: none"> <li>• graphs</li> <li>• adjacency list</li> <li>• adjacency matrix</li> <li>• breadth-first search</li> <li>• distance and shortest paths</li> <li>• depth-first search</li> <li>• small-world networks</li> <li>• scale-free networks</li> <li>• clustering coefficient</li> <li>• uniform random graphs</li> </ul>	<ul style="list-style-type: none"> <li>• dictionaries</li> </ul>

*Chapter 13. Abstract data types*

Sample problems	Principles	Programming
<ul style="list-style-type: none"> <li>• data sets</li> <li>• genomic sequences</li> <li>• rational numbers</li> <li>• flocking behavior</li> <li>• slime mold aggregation</li> </ul>	<ul style="list-style-type: none"> <li>• abstract data types</li> <li>• data structures</li> <li>• stacks</li> <li>• hash tables</li> <li>• agent-based simulation</li> <li>• swarm intelligence</li> </ul>	<ul style="list-style-type: none"> <li>• writing classes</li> <li>• special methods</li> <li>• overriding operators</li> <li>• modules</li> </ul>

*Software assumptions*

To follow along in this book and complete the exercises, you will need to have installed Python 3.4 (or later) on your computer, and have access to IDLE or another programming environment. The book also assumes that you have installed the `matplotlib` and `numpy` modules. Please refer to Appendix A for more information.

*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 web site. If you find an error in the text or have another suggestion, please let me know at [havill@denison.edu](mailto:havill@denison.edu).