acknowledgments

Thank you, everyone at Manning who helped in the production of this book: Cheryl Weisman, Deirdre Hiam, Katie Tennant, Dottie Marsico, Janet Vail, Barbara Mirecki, Aleksandar Dragosavljević, Mary Piergies, and Marija Tudor.

I thank acquisitions editor Brian Sawyer, who wisely steered us toward attacking Python after I finished Swift. Thank you, development editor Jennifer Stout, for always having a positive attitude. Thanks go to technical editor Frances Buontempo, who provided careful consideration of each chapter and gave detailed, useful feedback at every turn. I thank copyeditor Andy Carroll, whose superb attention to detail on both the Swift book and this one caught several of my mistakes, and also my technical proofreader, Juan Rufes.

The following people also reviewed the book: Al Krinker, Al Pezewski, Alan Bogusiewicz, Brian Canada, Craig Henderson, Daniel Kenney-Jung, Edmond Sesay, Ewa Baranowska, Gary Barnhart, Geoff Clark, James Watson, Jeffrey Lim, Jens Christian, Bredahl Madsen, Juan Jimenez, Juan Rufes, Matt Lemke, Mayur Patil, Michael Bright, Roberto Casadei, Sam Zaydel, Thorsten Weber, Tom Jeffries, and Will Lopez. Thanks go to all who provided constructive and specific criticism during the book's development. Your feedback was incorporated.

I thank my family, friends, and colleagues who encouraged me to take on this book project immediately following the publication of *Classic Computer Science Problems in Swift*. I thank my online friends on Twitter and elsewhere who have provided encouraging words and helped promote the book in ways small and large. And I thank my wife, Rebecca Kopec, and my mom, Sylvia Kopec, who are always supportive of my projects.

We developed this book in a fairly short period of time. The vast majority of the manuscript was written over the summer of 2018, based on the earlier Swift version. I appreciate that Manning was willing to compress its (usually much longer) process to enable me to work during a schedule that was convenient to me. I know this put pressure on the entire team as we went through three rounds of reviews at multiple different levels amongst many different people in just a few months. Most readers would be amazed at how many different kinds of reviews a technical book by a traditional publisher goes through and how many people have their part in critiquing and revising it. From the technical proofer to the copy editor, the review editor, all of the official reviewers, and everyone in between, I thank you!

Finally, most importantly, I thank my readers for purchasing this book. In a world full of halfhearted online tutorials, I think it is important to support the development of books that provide the same author's voice throughout an extended volume. Online tutorials can be superb resources, but your purchase enables full-length, vetted, and carefully developed books to still have a place in computer science education.

about this book

Trademarks

Trademarked names appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, the names are only used in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark. “Python” is a registered trademark of the Python Software Foundation. “Connect Four” is a trademark of Hasbro, Inc.

Book forum

Purchase of *Classic Computer Science Problems in Python* includes free access to a private web forum run by Manning Publications where you can make comments about the book, ask technical questions, and receive help from the author and from other users. To access the forum, go to https://www.manning.com/books/classic-computer-science-problems-in-python. You can also learn more about Manning's forums and the rules of conduct at https://forums.manning.com/forums/about.

Manning’s commitment to our readers is to provide a venue where a meaningful dialogue between individual readers and between readers and the author can take place. It is not a commitment to any specific amount of participation on the part of the author, whose contribution to the forum remains voluntary (and unpaid). We suggest you try asking him some challenging questions lest his interest stray! The forum and the archives of previous discussions will be accessible from the publisher’s website as long as the book is in print.

about the author

David Kopec is an assistant professor of Computer Science & Innovation at Champlain College in Burlington, Vermont. He is an experienced software developer and the author of *Classic Computer Science Problems in Swift* (Manning, 2018), and *Dart for Absolute Beginners* (Apress, 2014). David holds a bachelor's degree in economics and a master's in computer science, both from Dartmouth College. You can reach David on Twitter @davekopec.

about the cover illustration

The figure on the cover of *Classic Computer Science Problems in Python* is captioned "Habit of a Bonza or Priest in China." The illustration is taken from Thomas Jefferys' *A Collection of the Dresses of Different Nations, Ancient and Modern* (four volumes), London, published between 1757 and 1772. The title page states that these are hand-colored copperplate engravings, heightened with gum arabic.

Thomas Jefferys (1719–1771) was called "Geographer to King George III." He was an English cartographer who was the leading map supplier of his day. He engraved and printed maps for government and other official bodies and produced a wide range of commercial maps and atlases, especially of North America. His work as a map maker sparked an interest in local dress customs of the lands he surveyed and mapped, which are brilliantly displayed in this collection. Fascination with faraway lands and travel for pleasure were relatively new phenomena in the late eighteenth century, and collections such as this one were popular, introducing both the tourist as well as the armchair traveler to the inhabitants of other countries.

The diversity of the drawings in Jefferys' volumes speaks vividly of the uniqueness and individuality of the world's nations some 200 years ago. Dress codes have changed since then, and the diversity by region and country, so rich at the time, has faded away. It's now often hard to tell the inhabitants of one continent from another. Perhaps, trying to view it optimistically, we've traded a cultural and visual diversity for a more varied personal life—or a more varied and interesting intellectual and technical life.

At a time when it's difficult to tell one computer book from another, Manning celebrates the inventiveness and initiative of the computer business with book covers based on the rich diversity of regional life of two centuries ago, brought back to life by Jefferys' pictures.

Introduction

Thank you for purchasing *Classic Computer Science Problems in Python*. Python is one of the most popular programming languages in the world, and people become Python programmers from a variety of backgrounds. Some have a formal computer science education. Others learn Python as a hobby. Still others use Python in a professional setting, but their primary job is not to be a software developer. The problems in this intermediate book will help seasoned programmers refresh themselves on ideas from their CS education while learning some advanced features of the language. Self-taught programmers will accelerate their CS education by learning classic problems in the language of their choice: Python. This book covers such a diversity of problem-solving techniques that there is truly something for everyone.

This book is not an introduction to Python. There are numerous excellent books from Manning and other publishers in that vein.[1] Instead, this book assumes that you are already an intermediate or advanced Python programmer. Although this book requires Python 3.7, mastery of every facet of the latest version of Python is not assumed. In fact, the book's content was created with the assumption that it would serve as learning material to help readers achieve such mastery. On the other hand, this book is not appropriate for readers completely new to Python.

Why Python?

Python is used in pursuits as diverse as data science, film-making, computer science education, IT management, and much more. There really is no computing field that

[1] If you are just starting your Python journey, you may want to first check out *The Quick Python Book*, 3rd edition, by Naomi Ceder (Manning, 2018) before beginning this book.

Python has not touched (except maybe kernel development). Python is loved for its flexibility, beautiful and succinct syntax, object-oriented purity, and bustling community. The strong community is important because it means Python is welcoming to newcomers and has a large ecosystem of available libraries for developers to build upon.

For the preceding reasons, Python is sometimes thought of as a beginner-friendly language, and that characterization is probably true. Most people would agree that Python is easier to learn than C++, for example, and its community is almost certainly friendlier to newcomers. As a result, many people learn Python because it is approachable, and they start writing the programs they want to write fairly quickly. But they may never have received an education in computer science that teaches them all of the powerful problem-solving techniques available to them. If you are one of those programmers who knows Python but does not know CS, this book is for you.

Other people learn Python as a second, third, fourth, or fifth language after a long time working in software development. For them, seeing old problems they've already seen in another language will help them accelerate their learning of Python. For them, this book may be a good refresher before a job interview, or it might expose them to some problem-solving techniques they had not previously thought of exploiting in their work. I would encourage them to skim the table of contents to see if there are topics in this book that excite them.

What is a classic computer science problem?

Some say that computers are to computer science as telescopes are to astronomy. If that's the case, then perhaps a programming language is like a telescope lens. In any event, the term "classic computer science problems" is used here to mean "programming problems typically taught in an undergraduate computer science curriculum."

There are certain programming problems that are given to new programmers to solve and that have become commonplace enough to be deemed classic, whether in a classroom setting during the pursuit of a bachelor's degree (in computer science, software engineering, and the like) or within the confines of an intermediate programming textbook (for example, a first book on artificial intelligence or algorithms). A selection of such problems is what you will find in this book.

The problems range from the trivial, which can be solved in a few lines of code, to the complex, which require the buildup of systems over multiple chapters. Some problems touch on artificial intelligence, and others simply require common sense. Some problems are practical, and other problems are fanciful.

What kinds of problems are in this book?

Chapter 1 introduces problem-solving techniques that will likely look familiar to most readers. Things like recursion, memoization, and bit manipulation are essential building blocks of other techniques explored in later chapters.

This gentle introduction is followed by chapter 2, which focuses on search problems. Search is such a large topic that you could arguably place most problems in the book under its banner. Chapter 2 introduces the most essential search algorithms, including binary search, depth-first search, breadth-first search, and A*. These algorithms are reused throughout the rest of the book.

In chapter 3, you will build a framework for solving a broad range of problems that can be abstractly defined by variables of limited domains that have constraints between them. This includes such classics as the eight queens problem, the Australian map-coloring problem, and the cryptarithmetic SEND+MORE=MONEY.

Chapter 4 explores the world of graph algorithms, which to the uninitiated are surprisingly broad in their applicability. In this chapter, you will build a graph data structure and then use it to solve several classic optimization problems.

Chapter 5 explores genetic algorithms, a technique that is less deterministic than most covered in the book but that sometimes can solve problems traditional algorithms cannot solve in a reasonable amount of time.

Chapter 6 covers k-means clustering and is perhaps the most algorithmically specific chapter in the book. This clustering technique is simple to implement, easy to understand, and broadly applicable.

Chapter 7 aims to explain what a neural network is and to give the reader a taste of what a very simple neural network looks like. It does not aim to provide comprehensive coverage of this exciting and evolving field. In this chapter, you will build a neural network from first principles, using no external libraries, so you can really see how a neural network works.

Chapter 8 is on adversarial search in two-player perfect information games. You will learn a search algorithm known as minimax, which can be used to develop an artificial opponent that can play games like chess, checkers, and Connect Four well.

Finally, chapter 9 covers interesting (and fun) problems that did not quite fit anywhere else in the book.

Who is this book for?

This book is for both intermediate and experienced programmers. Experienced programmers who want to deepen their knowledge of Python will find comfortably familiar problems from their computer science or programming education. Intermediate programmers will be introduced to these classic problems in the language of their choice: Python. Developers getting ready for coding interviews will likely find this book to be valuable preparation material.

In addition to professional programmers, students enrolled in undergraduate computer science programs who have an interest in Python will likely find this book helpful. It makes no attempt to be a rigorous introduction to data structures and algorithms. *This is not a data structures and algorithms textbook*. You will not find proofs or extensive use of big-O notation within its pages. Instead, it is positioned as an approachable, hands-on tutorial to the problem-solving techniques that should be the end product of taking data structure, algorithm, and artificial intelligence classes.

Once again, knowledge of Python's syntax and semantics is assumed. A reader with zero programming experience will get little out of this book, and a programmer with zero Python experience will almost certainly struggle. In other words, *Classic Computer Science Problems in Python* is a book for working Python programmers and computer science students.

Python versioning, source code repository, and type hints

The source code in this book was written to adhere to version 3.7 of the Python language. It utilizes features of Python that only became available in Python 3.7, so some of the code will not run on earlier versions of Python. Instead of struggling and trying to make the examples run in an earlier version, please just download the latest version of Python before starting the book.

This book only makes use of the Python standard library (with a slight exception in chapter 2, where the `typing_extensions` module is installed), so all of the code in this book should run on any platform where Python is supported (macOS, Windows, GNU/Linux, and so on). The code in this book was only tested against CPython (the main Python interpreter available from python.org), although it is likely that most of it will run in a Python 3.7–compatible version of another Python interpreter.

This book does not explain how to use Python tools like editors, IDEs, debuggers, and the Python REPL. The book's source code is available online from the GitHub repository: https://github.com/davecom/ClassicComputerScienceProblems InPython. The source code is organized into folders by chapter. As you read each chapter, you will see the name of a source file in the header of each code listing. You can find that source file in its respective folder in the repository. You should be able to run the problem by just entering `python3 filename.py` or `python filename.py` depending on your computer's setup with regards to the name of the Python 3 interpreter.

Every code listing in this book makes use of Python type hints, also known as type annotations. These annotations are a relatively new feature for the Python language, and they may look intimidating to Python programmers who have never seen them before. They are used for three reasons:

1. They provide clarity about the types of variables, function parameters, and function returns.
2. They self-document the code in a sense, as a result of reason 1. Instead of having to search through a comment or docstring to find the return type of a function, you can just look at its signature.
3. They allow the code to be type-checked for correctness. One popular Python type checker is mypy.

Not everyone is a fan of type hints, and choosing to use them throughout the book was frankly a gamble. I hope they will be a help instead of a hindrance. It takes a little more time to write Python with type hints, but it provides more clarity when read back. An interesting note is that type hints have no effect on the actual running of the code in the Python interpreter. You can remove the type hints from any of the code in this book, and it should still run. If you have never seen type hints before and feel you need a more comprehensive introduction to them before diving into the book, please see appendix C, which provides a crash course in type hints.

No graphics, no UI code, just the standard library

There are no examples in this book that produce graphical output or that make use of a graphical user interface (GUI). Why? The goal is to solve the posed problems with solutions that are as concise and readable as possible. Often, doing graphics gets in the way or makes solutions significantly more complex than they need to be to illustrate the technique or algorithm in question.

Further, by not making use of any GUI framework, all of the code in the book is eminently portable. It can as easily run on an embedded distribution of Python running on Linux as it can on a desktop running Windows. Also, a conscious decision was made to only use packages from the Python standard library instead of any external libraries, as most advanced Python books do. Why? The goal is to teach problem-solving techniques from first principles, not to "pip install a solution." By having to work through every problem from scratch, you will hopefully gain an understanding about how popular libraries work behind the scenes. At a minimum, only using the standard library makes the code in this book more portable and easier to run.

This is not to say that graphical solutions are not sometimes more illustrative of an algorithm than text-based solutions. It simply was not the focus of this book. It would add another layer of unnecessary complexity.

Part of a series

This is the second book in a series titled *Classic Computer Science Problems* published by Manning. The first book was *Classic Computer Science Problems in Swift*, published in 2018. In each book in the series, we aim to provide language-specific insight while teaching through the lens of the same (mostly) computer science problems.

If you enjoy this book and plan to learn another language covered by the series, you may find going from one book to another an easy way to improve your mastery of that language. For now, the series covers just Swift and Python. I wrote the first two books myself, because I have significant experience in both of those languages, but we are already discussing plans for future books in the series co-authored by people who are experts in other languages. I encourage you to look out for them if you enjoy this book. For more information about the series, visit https://classicproblems.com/.