An Introduction to Python Programming for Scientists and Engineers

Python is one of the most popular programming languages, widely used for data analysis and modelling, and is fast becoming the leading choice for scientists and engineers. Unlike other textbooks introducing Python, typically organised by language syntax, this book uses many examples from across Biology, Chemistry, Physics, Earth science, and Engineering to teach and motivate students in science and engineering. The text is organised by the tasks and workflows students undertake day-to-day, helping them see the connections between programming tools and their disciplines. The pace of study is carefully developed for complete beginners, and a spiral pedagogy is used so concepts are introduced across multiple chapters, allowing readers to engage with topics more than once. “Try This!” exercises and online Jupyter notebooks encourage students to test their new knowledge, and further develop their programming skills. Online solutions are available for instructors, alongside discipline-specific homework problems across the sciences and engineering.

Johnny Wei-Bing Lin is an Associate Teaching Professor and Director of Undergraduate Computing Education in the Division of Computing and Software Systems at the University of Washington Bothell, and an Affiliate Professor of Physics and Engineering at North Park University. He was the founding Chair of the American Meteorological Society’s annual Python Symposium.

Hannah Aizenman is a Ph.D. candidate in Computer Science at The Graduate Center, City University of New York. She studies visualization and is a core developer of the Python library Matplotlib.

Erin Manette Cartas Espinel graduated with a Ph.D. in physics from the University of California, Irvine. After more than 10 years at the University of Washington Bothell, she is now a software development engineer.

Kim Gunnerson recently retired as an Associate Teaching Professor at the University of Washington Bothell, where she taught chemistry and introductory computer programming.

Joanne Liu received her Ph.D. in Bioinformatics and Systems Biology from the University of California San Diego.

"This book provides an excellent introduction to the Python language especially targeted at those interested in carrying out calculations in the physical sciences. I especially like the strong coverage of graphics and of good coding practice."

Raymond Pierrehumbert, *University of Oxford*

"An excellent introduction to Python for scientists and engineers. Much more than teaching you how to program with Python, it teaches you how to do science with Python."

Eric Shaffer, *University of Illinois at Urbana-Champaign*

"Python has achieved an essential role in many disciplines within science, engineering, and beyond. Students and professionals are expected to be fluent in it, and (as I see in my daily job of helping users of a high-performance computing facility) they often struggle to reach that fluency. The authors have succeeded in the daunting task of writing a single book to help people reach a very advanced level of fluency, starting very gently and assuming no background. Unlike other books on the subject, An Introduction to Python Programming for Scientists and Engineers focuses on teaching for the intended end goal of scientists and engineers – investigating their scientific problems – not writing software for its own sake. I am looking forward to working with the generation who will learn how to program in Python using this book!"

Davide Del Vento, *NCAR Computational & Information Services Laboratory*

"*An Introduction to Python Programming for Scientists and Engineers* introduces programming in Python using evidence-based approaches to active learning. The exercises help both students and instructors identify misconceptions in programming, allowing students to build a strong foundation in Python programming. The book streamlines content such that there is a focus on mastering immediately useful concepts, normalizing errors, and demonstrating recovery."

Kari L. Jordan, *Executive Director, The Carpentries*

An Introduction to Python Programming for Scientists and Engineers

Johnny Wei-Bing Lin
University of Washington Bothell and North Park University

Hannah Aizenman
City College of New York

Erin Manette Cartas Espinel
Envestnet Tamarac

Kim Gunnerson
University of Washington Bothell

Joanne Liu
Novozymes A/S

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314-321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre,
New Delhi – 110025, India

103 Penang Road, #05–06/07, Visioncrest Commercial, Singapore 238467

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of
education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/highereducation/isbn/9781108701129
DOI: 10.1017/9781108571531

First published 2022

Printed in the United Kingdom by TJ Books Limited, Padstow Cornwall

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging-in-Publication Data
Names: Lin, Johnny Wei-Bing, 1972– author. | Aizenman, Hannah, 1987– author. |
Espinel, Erin Manette Cartas, 1965– author. | Gunnerson, Kim Noreen, 1965– author. |
Liu, Joanne (Joanne K.), author.
Title: An introduction to Python programming for scientists and engineers /
Johnny Wei-Bing Lin, University of Washington, Bothell, Hannah Aizenman,
City College of New York, Erin Manette Cartas Espinel, Envestnet Tamarac,
Kim Gunnerson, University of Washington, Bothell, Joanne Liu, Biota Technology Inc.
Description: First edition. | Cambridge, United Kingdom ; New York, NY :
Cambridge University Press, 2022. | Includes bibliographical references and index.
Identifiers: LCCN 2022000136 | ISBN 9781108701129 (paperback)
Subjects: LCSH: Python (Computer program language) | Computer programming. |
Engneering–Data processing. | BISAC: SCIENCE / Earth Sciences / General
Classification: LCC QA76.73.P98 L55 2022 | DDC 005.13/3–dc23/eng/20220304
LC record available at https://lccn.loc.gov/2022000136

ISBN 978-1-108-70112-9 Paperback

Contents

Detailed Contents

Preface

Most introductory programming textbooks are written with the assumption that the student thinks like a computer scientist. That is, writers assume that the student best learns programming by focusing on the structure and syntax of programming languages. The result is an introductory textbook that teaches programming in a way that is accessible to future programmers and developers but not as much to scientists or engineers who mainly want to investigate scientific problems.

This textbook is written to teach programming to scientists and engineers, not to computer scientists. We assume that the reader has no background, formal or informal, in computer programming. Thus, this textbook is distinct from other introductory programming textbooks in the following ways:

- **It is organized around a scientist or engineer's workflow.** What are the tasks of a scientist or engineer that a computer can help with? Doing calculations (e.g., Chapters 2 and 6), making a plot (e.g., Chapters 4 and 5), handling missing data (e.g., Chapter 15), and saving and storing data (e.g., Chapters 9 and 18) are just a few of the tasks we address.
- **It teaches programming, not numerical methods, statistics, data analytics, or image processing.** The level of math that the reader needs is modest so the text is accessible to a first-year college student.
- **It provides examples pertinent to the natural sciences and engineering.** Jupyter notebooks associated with this textbook provide structured practice using examples from physics, chemistry, and biology, and additional notebooks for engineering are planned. For instance, the physics notebooks include problems dealing with electromagnetic fields, optics, and gravitational acceleration.
- **Syntax is secondary.** The primary goal is to teach the student how to use Python to do scientific and engineering work. Thus, we teach as much language syntax and structure as needed to do a task. Later, as we address more complex science and engineering tasks, we teach additional aspects of language syntax and structure. As a result, this textbook is not intended as a Python language reference where all (or most) of the aspects of a given feature of the language are addressed at the same time.
- **It is paced for the beginner.** This text offers many examples, explanations, and opportunities to practice. We take things slowly because learning is a step-by-step process, not a toss-into-the-deep-end process. As a result, this text is not concise, particularly in the beginning. It will seem ponderous to an expert programmer. This is intentional.

Structure of the Textbook

The textbook is divided into four parts. Parts I–III, collectively, cover most of the topics of a CS1 and CS2 sequence:

- Part I shows how to get the basic scientific and engineering workflow tasks done, including visualization, modeling, analysis, input/output, and computer administration.
- Part II shows us how to do more advanced tasks, building off of what we have seen in Part I. Throughout, the programming is taught through learning how to do the science and engineering workflow tasks, so almost every chapter in Parts I–II addresses a different science or engineering task we can use Python programming for.
- The chapters in Part III, because they cover more advanced programming concepts that are better discussed in computer science terms, are organized more traditionally, with chapters on more advanced data structures, classes and inheritance, basic searching and sorting, and recursion. Nonetheless, even in Part III, we connect those concepts to science and engineering workflow tasks.

Typical CS2 topics the textbook does not cover include linked lists, divide-and-conquer sorting algorithms, and trees.

Part IV goes beyond the topics of a sequence of introductory programming or scientific computing courses to describe how to turn our programs into something really robust.

Structure of the Chapters

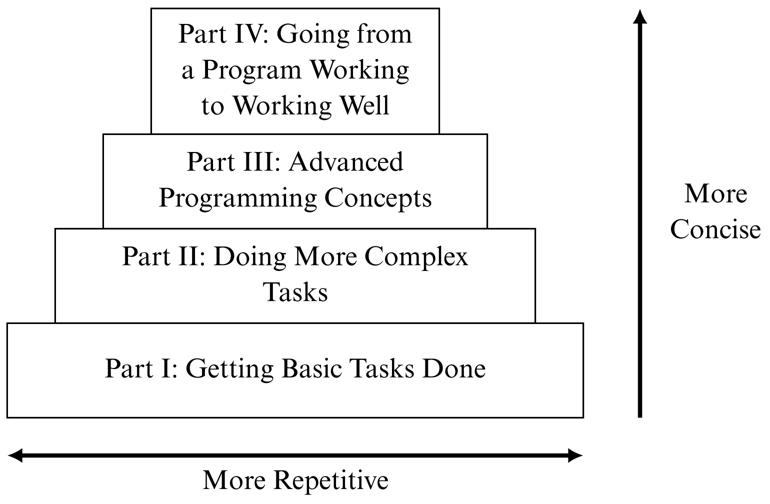
The text takes a very uniform approach to each of the chapters in Parts I–III, as follows:

- The first section describes the task and gives some examples of using Python to accomplish that task. These examples and problems are general enough to be understood by most scientists and engineers.
- The second section describes why the example in the first section works the way it does: the programming concepts and Python syntax are described and demonstrated in this section.
- The third section provides exercises/examples, often similar to the examples in the first section, for the reader to try, along with solutions and discussion of why the solution works. Additional programming and Python concepts are often discussed in these solutions.
- The fourth section briefly describes the online exercises and problems that are discipline-specific (physics, chemistry, and biology), for further explication and practice.
- The final section is a chapter review containing self-test questions (with answers at the end of the section) and a chapter summary.

Chapters 1 and 11 are the exceptions in Parts I–III, having a different organization and lacking the exercises, questions, and chapter review as described above.

Because the chapters in Part IV cover techniques, packages, and utilities (some of which are still evolving) that do not lend themselves to a simple scientific or engineering workflow example nor to exercises in a Jupyter notebook, the chapters in Part IV are not organized using the five-section pattern most of the chapters in Parts I–III use. The aim of Part IV is to introduce software engineering tools and practices that help us to write better and more reliable code, in order for the reader to seek more information in works that specialize in these topics. We hope that our introduction will whet your appetite to learn more on your own about these tools and practices.

When we first start to learn something, we require more explanation and practice to get comfortable with the material, learn the vocabulary, and think in new ways. The textbook is thus structured like a pyramid:


Part IV: Going from a Program Working to Working Well
Part III: Advanced Programming Concepts
Part II: Doing More Complex Tasks
Part I: Getting Basic Tasks Done
More Concise
More Repetitive

The chapters in Part I are the longest, and the chapters in Part IV are the shortest. The earlier parts are more repetitive and less concise while the later parts are less repetitive and more concise. The wider the block in the pyramid above, the more repetitive the part. The higher up the pyramid, the more concise the part.

The Logic Behind Some Decisions on Topic Coverage and Sequence

Some of the decisions about coverage and sequence may be surprising, so the rationale is explained here:

- By basing the structure of most of the chapters in Parts I–II on science and engineering workflow tasks, syntax is addressed in pieces. That is, there is no single chapter on variables, single chapter on branching, etc. Rather, those topics are covered over several chapters, in a kind of spiral approach. Pedagogically, this is better because it allows us to learn things a little at a time with subsequent treatments of a topic reinforcing what we learned earlier. Learning in this way is treated as a spiral rather than as a line. But, there are at least two costs with this method of learning. First, the first time we see a particular structure in a given chapter, because we do not fully describe that structure in that chapter, the code examples may seem awkward and more complicated than needed. Second, by spreading out the description of a single structure over multiple chapters, this textbook does not function well as a reference. What can you do to mitigate these costs? For the first, we ask for your patience as we slowly build up the description of the topic. The code examples will become more concise and Pythonic as the book progresses. For the second, we have written the textbook to provide a substantial amount of cross-referencing between sections, so those cross-references might be enough to lead you where you want to go. Skimming the Detailed Contents, **which details the subtopics included in each chapter,** or Index, may also help you find the occurrences of the topics you are interested in. Appendix C also lists the contents of the book by programming topic.

- The "how to write a program" chapter does not come early on as it does in most programming textbooks. We briefly mention how to write programs in Section 6.2.7 and provide a more detailed treatment in Chapter 11. How come? If the goal of the textbook is to teach programming by doing, rather than reading about how to "do," it should start off with using Python to do science and engineering tasks. Once the reader has some experience writing short programs, they will have more context to understand advice on how to write a longer program.
- The workflow focus influences how looping and branching are introduced in this textbook. In most textbooks, these topics are introduced separately, in separate chapters. The result, however, is to limit what can be done with such knowledge. An `if` statement by itself, without the possibility of being visited multiple times, does not accomplish much. In this textbook, we introduce both concepts together, in Chapter 6 on basic diagnostic data analysis. Neither topic is treated exhaustively in this chapter, but with the introduction of both topics in one chapter, we can vividly show how useful these structures are for using the computer to investigate a dataset.
- Because the textbook is aimed at novices rather than students with programming experience, we had to make difficult choices regarding what concepts to introduce and when. One result of these choices is that, particularly in earlier chapters, our code examples are less than Pythonic in order to make our treatment of the concepts at that point in the book clearer to novice students. We also ignore some packages and tools that can accomplish some of the tasks we address more concisely and efficiently, in order to focus on the learning goal at hand. Reasonable people will disagree regarding our choices, but we wanted instructors to know we recognize the tension, and we heartily support whatever customization will best meet the needs of your students.

Topic Sequence and Flexible Approach for Different Course Lengths

- Part I is material for approximately one quarter-long (10 week) class. The material works well for an introductory programming course where the students have no prior experience with programming.

- Parts I–II are material for approximately one semester-long (15 week) introductory programming course.
- Parts I–III are written so later chapters build on the contents of earlier chapters. They contain material for a two-quarter introductory programming sequence.
- The chapters in Part IV are supplemental and can be read independently and used for self-study.
- Going in order is one way to use the textbook. However, other sequences are possible. For instructors who want to cover the material by programming topic, the table of contents in Appendix C shows where various aspects of the programming topics are covered.

Typesetting and Coloring Conventions

Throughout the textbook, we use different forms of typesetting and coloring to provide additional clarity and functionality. Some of the special typesetting conventions we use include:

- Inline source code. These are instructions interspersed in the paragraphs of the textbook that tell Python what to do. They are typeset in a dark red, monospace font, as in `a = 4`.
- Inline commands to type on your keyboard or printed to the screen. Typeset in a dark red, monospace font, as in `print('hello')`.
- Inline generic arguments or values. These labels are placeholders to be replaced by snippets of code or concrete values. These are typeset in a dark red, italicized, proportional font, in between a less than sign and a greater than sign, as in <*condition*>.
- Blocks or listing items of source code, commands, generic arguments, or values. These are typeset in an indented block quotation structure or listing structure in a black font.
- File contents (that are not source code). These are typeset in an indented block quotation structure in a black, monospace font.
- File, directory, and app names. Typeset in a black, italicized, proportional font, as in */usr/bin*.
- Key terms. On first use, these are typeset in a dark blue, bold, proportional font, as in **program**, and can be found in the Glossary.

General references to application, library, module, and package names are typeset the same as regular text. Thus, references to the Matplotlib package are typeset just as in this sentence. As most packages have unique names, this should not be confusing. In the few cases where the package names are regular English words (e.g., the time module), references to the module will hopefully be clear from the context.

Assessment and Practice for Students

This text and the online Jupyter notebooks provide an abundance of opportunities for students to practice what they are learning:

- Try This! exercises are designed to provide practice for students to take bite-sized, incremental steps in growing their understanding the material. In a classroom setting, they are appropriate for active learning problems, group work, and as components in programming labs. The answers are publicly available, so these exercises are best used for practice and development rather than summative assessment.
- The Homework Problems are provided online as Jupyter notebooks and require more time for students to work on than the Try This!. Thus, generally speaking, they are more appropriate for students to work on outside of class. Solutions for homework notebooks are only provided to verified instructors.

The "To the Student" section following this Preface provides further description of the kinds of exercises and problems that are available.

Supporting Resources and Updates to This Textbook

The textbook's website, www.cambridge.org/core/resources/pythonforscientists, contains updates to and supporting resources for the textbook. This includes:

- Web pages describing some topics in more detail.
- Jupyter notebooks with exercises and problems.
- Copies of the larger datasets and images referenced in this textbook.
- A list of addenda and errata.
- Links to key external sites.

All the above resources (with the exception of the images, and solutions for exercises and problems) are accessible by both students and instructors.

The landscape of Python teaching resources is vast and constantly changing. We provide a web page listing some of these resources at www.cambridge.org/core/resources/pythonforscientists/instructors/.

Please let us know of any corrections by emailing us at ipyses@johnny-lin.com.

To the Student

An Introduction to Python Programming for Scientists and Engineers consists of two components:

- The print/digital book that you are now reading.
- Online resources including web pages with additional content and exercises and problems as Jupyter notebooks.

The latter are not "supplements" because they tightly link to the content and flow of the textbook. The two pieces form an integrated whole.

Applications and Exercises

The textbook provides the following kinds of questions and problems in Parts I–III:

- Try This!. These are exercises/worked examples and are found in the print/digital book. They are similar in complexity to the main chapter examples and contain solutions and discussion of the solution. The Try This! sections also extend the discussion earlier in the chapters, introducing new concepts.
- Chapter Review Self-Test Questions. These are found in the print/digital book. These are relatively short questions that give you a first-cut assessment of your understanding of the material in the chapter. The answers to these questions are found at the end of the chapter.
- Discipline-Specific Try This!. These are found online. They are similar to the exercises discussed in the print/digital book in the Try This! sections. We give you a Jupyter notebook with the Discipline-Specific Try This! and then another Jupyter notebook with answers to the Try This!.
- Discipline-Specific Homework Problems. These are found online. They are designed to be assigned by instructors for homework. We give you a blank Jupyter notebook with the Problems. These Problems are generally more involved than the Discipline-Specific Try This!.

All Discipline-Specific exercises and problems are in the form of Jupyter notebooks that can be downloaded to your own computer and run locally. The Discipline-Specific Jupyter notebooks are at www.cambridge.org/core/resources/pythonforscientists/jupyter-notebooks/. Currently, notebooks for biology, chemistry, and physics are provided. We hope to include notebooks for other disciplines in the future. Details on using Jupyter notebooks begin in Section 2.2.3.

The larger datasets referenced in this textbook and the Jupyter notebooks are found at www.cambridge.org/core/resources/pythonforscientists/datasets/. These are freely available for download.

Using the Textbook

We recommend that you approach the chapters in the following manner.

First, read the main chapter example. If possible, **type in the code** for the example and see what you get. This is more doable for earlier chapters, where the code is shorter. In later chapters, the benefits of typing in the code are probably not worth the time to key it all in. As you read the main chapter example, you might not understand everything in the example. If so, that is okay. The purpose of the main chapter example is not to teach you the concepts in the chapter. That is what the rest of the chapter is for. The main chapter example is there to give you a sense of how we can use Python to solve a particular science or engineering task and to motivate the rest of the chapter's discussion and exercises.

Second, read the Python Programming Essentials section. In most chapters, this section will have small examples you can type in. Please do so. This will help your learning.

Third, do the Try This!. Do not read read the solution that immediately follows until you have done the Try This!. If you just read the Try This! and skip immediately to the solution, you will circumvent the learning process. Doing and working problems really help us learn! Also, remember the discussion of these exercises contains additional material regarding the chapter's topics. Do not skip the Try This! as if they are optional, "mere examples."

Fourth, do as many of the Discipline-Specific Try This! you have the time to do. After you do them, look at their solutions and see if you can explain *the reason* for any differences between your solution and the solution notebook.

Fifth, to increase the sophistication of your understanding of the topics involved, do at least a few of the Discipline-Specific Homework Problems. The solutions are only available to instructors, but the practice itself will help with your learning.

The Chapter Review Self-Test Questions can be used to test your understanding of the material. You might want to use them for studying for exams, but you do not need to wait for an exam to go through them. As you are reading the textbook, these questions might also help give you a sense of your comprehension. Note, though, that these questions are not very difficult, so you should not conclude that if you are able to answer all the questions easily that you will do fine on an exam.

The Chapter Summary provides one additional opportunity for you to see the topics again. It should not be used as a substitute for taking your own notes or creating your own study sheet for an exam.

One final piece of advice. Wherever we suggest you read the text, we are implicitly assuming you will read *slowly.* Really, really slowly. When we read code, we have to go line-by-line and term-by-term. We have to ask what the state of the variables are before that line, what the state of the variables are after that line, and what did the line do to cause (or not cause) any changes. When we encounter code, we should ask how the program would behave differently if we made a change to the values and order of the code. And we should take notes (by hand, if possible, because we learn more that way) on what we have read and write down any questions we have as they come to us. If we do not write down questions immediately, we will forget them and falsely believe we understand the material. We have to *actively engage* the text.

We cannot read a programming textbook (or any math, science, or engineering textbook) as if it were a novel. When we read a novel, we can just read the words and sentences themselves, skimming if we are in a rush. The prose itself tells us about the characters, plot, and setting. But in a programming textbook, this kind of reading will not work. Skimming a programming textbook is a recipe for disaster. We have to unpack and excavate the meaning of the code and the text describing the code. If we do not, our understanding will be limited. So, read s-l-o-w-l-y! ☺

Notices and Disclaimers

Mark and Trademark Acknowledgments

Anaconda, Anaconda Navigator, Conda, and Numba are marks and/or registered trademarks of Anaconda, Inc. Apple, Mac, Mac OS, and OS X are registered trademarks of Apple Inc. Azure, Excel, Microsoft, PowerShell, Windows, and Word are registered trademarks of Microsoft Corporation in the United States and/or other countries. Chrome^{TM} browser is a trademark of Google LLC. Debian is a registered trademark of Software in the Public Interest, Inc. Django is a trademark of the Django Software Foundation. Git is either a registered trademark or trademark of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries. GITHUB® is an exclusive trademark registered in the United States by GitHub, Inc. GitLab is a registered trademark of GitLab, Inc. GNOME® is a trademark of the GNOME Foundation. GNU is an operating system supported by the Free Software Foundation. Jupyter®, JupyterHub, and derivative word marks are trademarks or registered trademarks of NumFOCUS. Kubernetes® is a registered trademark in the United States and/or other countries of The Linux Foundation. LibreOffice is a registered trademark of The Document Foundation. Linux is a trademark owned by Linus Torvalds. Matlab and MathWorks are registered trademarks of The MathWorks, Inc. PyCharm[1] is a trademark of JetBrains s.r.o. Python is a registered trademark of the Python Software Foundation. Stack Overflow is a trademark of Stack Exchange Inc. Ubuntu is a registered trademark of Canonical Ltd. All other trademarks and marks mentioned in this book are the property of their respective owners. Any errors or omissions in trademark and/or other mark attributions are not meant to be assertions or denials of trademark and/or other mark rights.

Copyright Acknowledgments

Screenshots of Chrome browser sessions (e.g., Figures 2.3, 3.7, 3.8, 3.9, 4.6, and 4.7) include elements from Google LLC and are used by permission.

Images from Volkman et al. (2004) in Chapter 13 are copyright © 2004 by Volkman et al. and are used by permission under the conditions of the Creative Commons Attribution License. Volkman et al. do not specify the license version, but the Creative Commons

[1] www.jetbrains.com.

Attribution 4.0 International Public License is available at https://creativecommons.org/licenses/by/4.0/legalcode. The images in the present work have been modified from their original form in Volkman et al.

The screen shots in Figures 2.3, 3.7, 3.8, 3.9, 21.2, 21.3, and 21.4 are reprinted with permission from Apple Inc.

The code in Figure 24.1 is © 2010–2021, Holger Krekel and others. The code is used by permission under the conditions of the MIT License. The license agreement is available at https://github.com/tox-dev/tox/blob/master/LICENSE.[2]

Code portions and ideas referenced throughout the text are footnoted or otherwise referenced in the text. The code portions and ideas in Chapter 14, however, are built off of longer blocks of code as well as a larger variety of code sources, most prominently the cartopy manual (Met Office, 2010–2015), licensed under an Open Government License (www.nationalarchives.gov.uk/doc/open-government-licence/version/2/), and the Matplotlib documentation (matplotlib.org/contents.html; also see Hunter (2007)). Traditional footnoting and referencing does not work as well for this kind of synthetic work. Thus, we cite those references here and at www.cambridge.org/core/resources/pythonforscientists/refs/.

Use in this book of information from copyrighted sources is by permission (and is noted either in this acknowledgments section or in the respective figure captions) or is usage believed to be covered under Fair Use doctrine.

Data and Other Usage Acknowledgments

This work includes data from the Mikulski Archive for Space Telescopes (MAST) (e.g., in Chapter 1). STScI is operated by the Association of Universities for Research in Astronomy, Inc., under National Aeronautics and Space Administration (NASA) contract NAS5-26555. This includes data collected by the Kepler Mission. Funding for the Kepler Mission is provided by the NASA Science Mission directorate.

This work includes output from simulations using the molecular dynamics simulation package NAMD (Phillips et al., 2005). NAMD was developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. The official NAMD web page is at www.ks.uiuc.edu/Research/namd/.

This work includes surface/near-surface air temperature data (e.g., in Section 7.1) that is from National Centers for Environmental Prediction (NCEP) Reanalysis data provided by the NOAA/OAR/ESRL PSD, Boulder, Colorado, USA, from their website at www.esrl.noaa.gov/psd.

This work includes data from the UCI Machine Learning Repository (Dua and Graff, 2019). The repository is at https://archive.ics.uci.edu/ml.

[2] Accessed January 11, 2021.

Section 13.3 includes images from Patel and Dauphin (2019), published by NASA's Earth Observatory, which were created using Black Marble data from Ranjay Shrestha at NASA's Goddard Space Flight Center and Landsat data from the U.S. Geological Survey.

World Time Buddy (www.worldtimebuddy.com) provided help with time zone conversions. (See Sections 7.1 and 12.1.)

Data used for displaying natural features and political boundries (in Chapter 14) are provided by OpenStreetMap (© OpenStreetMap contributors) and Natural Earth (public domain). OpenStreetMap data are available under the Open Database Licence (www.openstreetmap.org/copyright). Free vector and raster map data are available at naturalearthdata.com.

Data in this work are also acknowledged in descriptions in the main text and footnotes. All data and images in this work are used by permission. Many of these materials are in the public domain or are otherwise freely available for republication and reuse. Please see the sources of the data for details.

Data presented in this work that are not explicitly attributed to a source should be considered fictional and created by the authors for the purposes of illustration or teaching. They must not be considered genuine or accurate descriptions of any natural or artificial phenomena or system. Most (though not all) occurrences of such data are accompanied by the term "fictitious" or "pretend."

Disclaimers

Although we have worked hard to make the text, code, and related online resources (together, "Resources") accurate and correct, the Resources are provided "as-is," without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the Resources or the use or other dealings in the Resources.[3]

Permission to use marks, trademarks, copyrighted materials, or any other materials by their owners does not imply an endorsement of that use or of the Resources.

[3] Copied and adapted from the MIT License, as listed on Opensource.org, https://opensource.org/licenses/MIT (accessed July 14, 2021).

Acknowledgments

We are grateful for the editorial help of Charles Howell, Matt Lloyd, Lisa Pinto, and Melissa Shivers at Cambridge University Press. We thank Spencer Cotkin for editorial suggestions, most we have implemented and that have made the book much better. We thank Beverley Lawrence for copyediting the text. A number of anonymous reviewers provided helpful feedback which have been incorporated into the text.

We are thankful for Cynthia Gustafson-Brown's assistance on this project, particularly in finding and providing some of the description in the Section 13.1 example.

We are appreciative of conversations with and assistance by: Bill Erdly, Michael Grossberg, Charity Flener Lovitt, Laurence Molloy, Hansel Ong, Jim Phillips, and Rob Nash. Whether through publications, workshops, conferences, or discussions, the communities we are a part of – personal, workplace, disciplinary, and, of course, the Python community – contributed ideas, encouragement, and support.

Parts of this book are based on the book, *A Hands-On Introduction to Using Python in the Atmospheric and Oceanic Sciences*,[4] slides from the 2020 American Meteorological Society's *Beginner's Course to Using Python in Climate and Meteorology*,[5] and the set of notes, *Lecture Notes on Programming Theory for Management Information Systems.*[6] These resources are by Johnny Lin and the acknowledgments made in those resources also apply to this text. We are grateful for those who gave us permission to use material they created. These are acknowledged in the Notices section, the captions of the included or adapted figures, or in the online resources.

[4] Lin (2012).
[5] Not formally published.
[6] Lin (2019).

I thank my wife Karen, and my children Timothy, James, and Christianne for their encouragement and love. *S.D.G.*

Johnny Wei-Bing Lin
Bellevue, Washington

I thank my advisor, Professor Michael Grossberg, for all his help and the AMS Python community and Matplotlib Development Team for their influence and conversations on all things code.

Hannah Aizenman
New York City, New York

I thank my husband Vicente, my family Duffy, Cheryl, and Sara, and my friends Valerine, Kaitlyn, and Alanna for all of their love and support.

Erin Manette Cartas Espinel
Kenmore, Washington

I thank Samantha Gunnerson for her help reading through notebooks to give me advice, Eric Gunnerson for his support as well as technical skills, and Dr. Paola Rodríguez Hidalgo for listening to me talk about the different Python writing I was doing during this process.

Kim Gunnerson
Bellevue, Washington

I thank my family for their support and Cynthia Gustafson-Brown for providing several of the More Discipline-Specific Practice problems in Chapters 2 and 3.

Joanne Liu
San Diego, California