First Edition

Building Python Programs

Stuart Reges
University of Washington

Marty Stepp
Stanford University

Allison Obourn
University of Arizona

Pearson 330 Hudson Street, NY, NY 10013

Senior Vice President Courseware Portfolio Management: Marcia Horton
Vice President, Portfolio Management: Engineering, Computer Science & Global Editions: Julian Partridge
Executive Portfolio Manager: Matt Goldstein
Portfolio Management Assistant: Meghan Jacoby
Field Marketing Manager: Demetrius Hall
Product Marketing Manager: Yvonne Vannatta
Marketing Assistant: Jon Bryant
Managing Producer: Scott Disanno
Content Producer: Amanda Brands
Project Manager: Rose Kernan, RPK Editorial Services
Manufacturing Buyer, Higher Ed, Lake Side Communications, Inc. (LSC): Maura Zaldivar-Garcia
Cover Design: Pearson CSC
R&P Manager: Ben Ferrini
Inventory Manager: Bruce Boundy
Cover Art: Matt Walford/Cultura/Getty Images
Full Service Vendor: Pearson CSC
Full-Service Project Management: Pearson CSC/Rose Kernan
Composition: Pearson CSC
Printer/Binder: LSC Communications, Inc

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

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

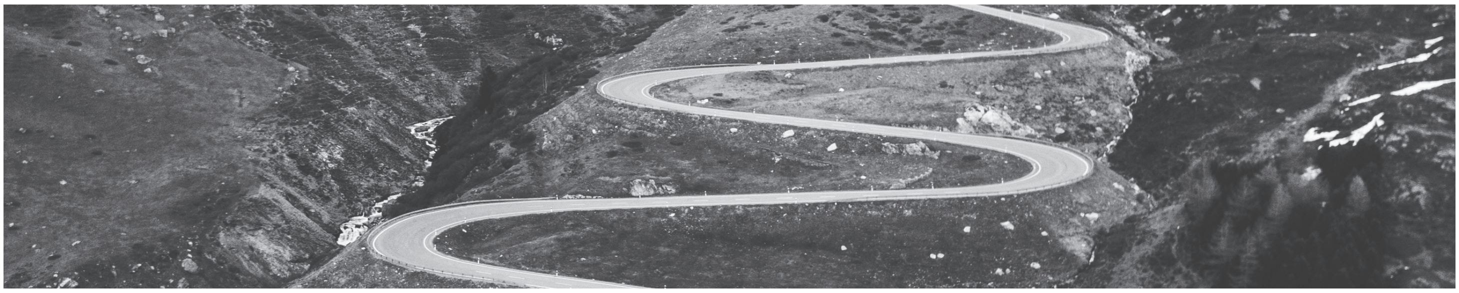
Many of the designations by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

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

Library of Congress Cataloging-in-Publication Data
Names: Reges, Stuart, author. | Stepp, Martin, author. | Obourn, Allison, author.
Title: Building Python programs / Stuart Reges, University of Washington, Marty Stepp, Stanford University, Allison Obourn, University of Arizona.
Description: First edition. | New York, NY : Pearson, [2019] | Includes index.
Identifiers: LCCN 2018028848| ISBN 9780135205983 | ISBN 0135205980
Subjects: LCSH: Python (Computer program language)
Classification: LCC QA76.73.P98 R445 2019 | DDC 005.13/3—dc23
LC record available at https://lccn.loc.gov/2018028848
1 18

Pearson

ISBN 10: 0-13-520598-0
ISBN 13: 978-0-13-520598-3



Preface

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

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

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

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

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

A01_REGE5983_01_SE_FM.indd 3 7/12/18 9:50 PM

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

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

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

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

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

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

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

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

Layers and Dependencies

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

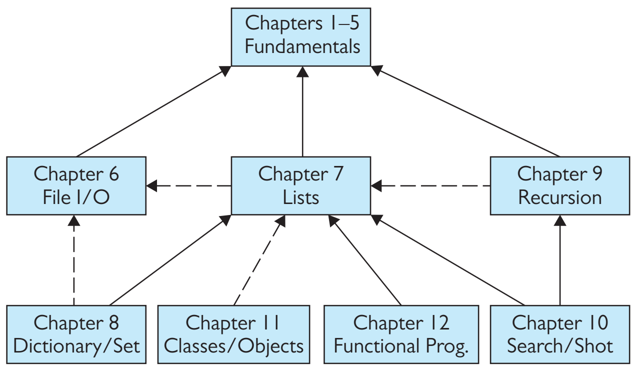
The following table shows how the layered approach works in the first seven chapters:

Layers in Chapters 1–7

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

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

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



Chapter dependency chart

Here are more detailed explanations of the weak dependencies between chapters:

- A few examples from Chapter 7 on lists, and from Chapter 8 on dictionaries and sets, read data from files. File input/output is covered in Chapter 6. But overall file-reading is not required in order to discuss lists or other collections, so Chapter 6 can be skipped if desired.
- A few examples from Chapter 11 on classes and objects mention the concept of reference semantics, which is introduced in Chapter 7 on lists. But the concept of references is re-explained in Chapter 11, so classes can be covered early before lists if desired.
- Some of the recursive functions in Chapter 9 process lists, and one recursive function recursively reverses the lines of a file. So Chapter 9 weakly depends on Chapter 7. But almost every recursive function written in Chapter 9 can be written and understood using only the Chapter 1–5 core material.

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

Supplements

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

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

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

Instructors can access the following resources from our web site:

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

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

MyLab Programming

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

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

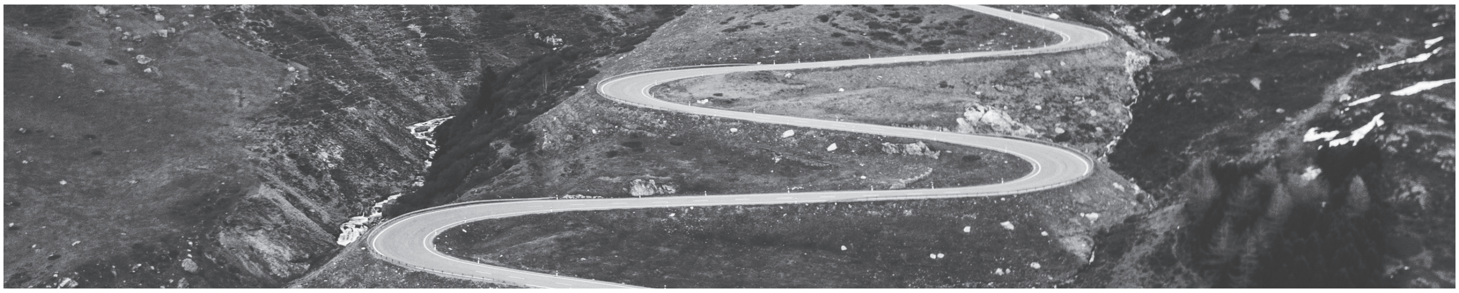
Acknowledgments

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

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

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

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



MyLab Programming

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

PROGRAMMING PRACTICE

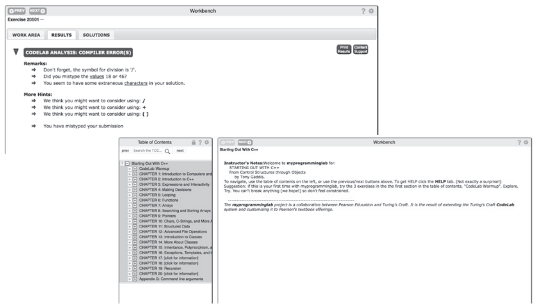
With MyLab Programming, your students will gain first-hand programming experience in an interactive online environment.

IMMEDIATE, PERSONALIZED FEEDBACK

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

GRADUATED COMPLEXITY

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



Exercise 20501 —

CODELAB ANALYSIS: COMPILER ERROR(S)

Remarks:
- Don't forget, the symbol for division is '/'.
- Did you mistype the values 18 or 46?
- You seem to have some extraneous characters in your solution.

More Hints:
- We think you might want to consider using: /
- We think you might want to consider using: +
- We think you might want to consider using: ( )

- You have mistyped your submission

Starting Out With C++
- CodeLab Warmup
- CHAPTER 1: Introduction to Computers and
- CHAPTER 2: Introduction to C++
- CHAPTER 3: Expressions and Interactivity
- CHAPTER 4: Making Decisions
- CHAPTER 5: Looping
- CHAPTER 6: Functions
- CHAPTER 7: Arrays
- CHAPTER 8: Searching and Sorting Arrays
- CHAPTER 9: Pointers
- CHAPTER 10: Chars, C-Strings, and More A
- CHAPTER 11: Structured Data
- CHAPTER 12: Advanced File Operations
- CHAPTER 13: Introduction to Classes
- CHAPTER 14: More About Classes
- CHAPTER 15: Inheritance, Polymorphism, a
- CHAPTER 16: Exceptions, Templates, and t
- CHAPTER 17: [click for information]
- CHAPTER 18: [click for information]
- CHAPTER 19: Recursion
- CHAPTER 20: [click for information]
- Appendix G: Command line arguments

Starting Out With C++

Instructor's Notes: Welcome to **myprogramminglab** for:
STARTING OUT WITH C++
From Control Structures through Objects
by Tony Gaddis.
To navigate, use the table of contents on the left, or use the previous/next buttons above. To get HELP click the **HELP** tab. (Not exactly a surprise!)
Suggestion: if this is your first time with myprogramminglab, try the 3 exercises in the the first section in the table of contents, "CodeLab Warmup". Explore.
Try. You can't break anything (we hope!) so don't feel constrained.

---

*The **myprogramminglab** project is a collaboration between Pearson Education and Turing's Craft. It is the result of extending the Turing's Craft **CodeLab** system and customizing it to Pearson's textbook offerings.*

DYNAMIC ROSTER

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

PEARSON eTEXT

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

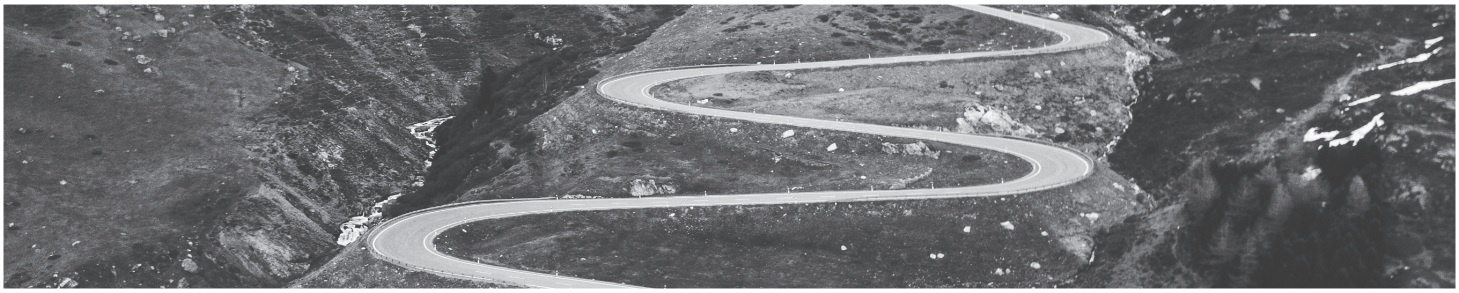
STEP-BY-STEP VIDEONOTE TUTORIALS

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

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

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

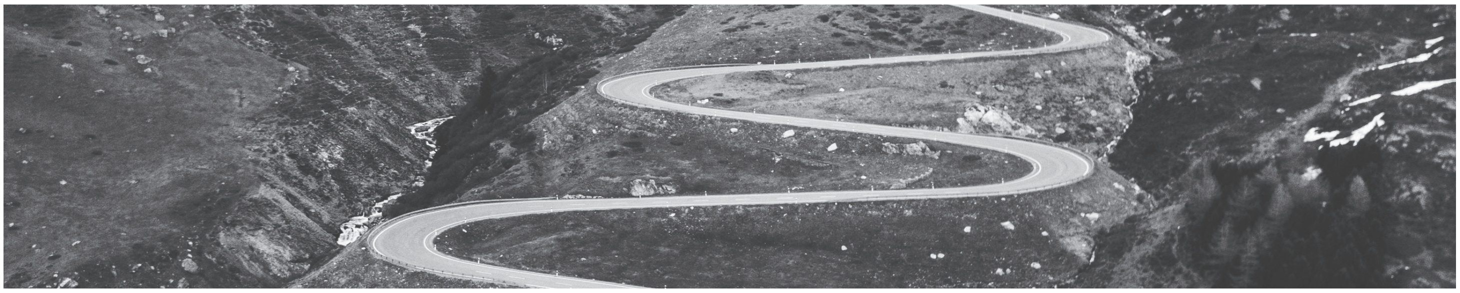
A01_REGE5983_01_SE_FM.indd 10 7/12/18 9:50 PM



Brief Contents

A01_REGE5983_01_SE_FM.indd 11 7/12/18 9:50 PM

A01_REGE5983_01_SE_FM.indd 12 7/12/18 9:50 PM



Contents

A01_REGE5983_01_SE_FM.indd 13 7/12/18 9:50 PM