PREFACE

Dear Reader,

what is programming?

This book assumes that you are a new programmer with no prior knowledge of programming. So, what is programming? Programming solves problems by creating solutions—writing programs—in a programming language. The fundamentals of problem solving and programming are the same regardless of which programming language you use. You can learn programming using any high-level programming language such as Python, Java, C++, or C#. Once you know how to program in one language, it is easy to pick up other languages, because the basic techniques for writing programs are the same.

why Python?

So what are the benefits of learning programming using Python? Python is easy to learn and fun to program. Python code is simple, short, readable, intuitive, and powerful, and thus it is effective for introducing computing and problem solving to beginners.

graphics

Beginners are motivated to learn programming so they can create graphics. A big reason for learning programming using Python is that you can start programming using graphics on day one. We use Python's built-in Turtle graphics module in Chapters 1–6 because it is a good pedagogical tool for introducing fundamental concepts and techniques of programming. We introduce Python's built-in Tkinter in Chapter 9, because it is a great tool for developing comprehensive graphical user interfaces and for learning object-oriented programming. Both Turtle and Tkinter are remarkably simple and easy to use. More importantly, they are valuable pedagogical tools for teaching the fundamentals of programming and object-oriented programming.

optional Turtle

To give instructors flexibility to use this book, we cover Turtle at the end of Chapters 1–6 so they can be skipped as optional material.

problem-driven

The book teaches problem solving in a problem-driven way that focuses on problem solving rather than syntax. We stimulate student interests in programming by using interesting examples in a broad context. While the central thread of the book is on problem solving, appropriate Python syntax and library are introduced in order to solve the problems. To support the teaching of programming in a problem-driven way, the book provides a wide variety of problems at various levels of difficulty to motivate students. In order to appeal to students in all majors, the problems cover many application areas in math, science, business, financial management, gaming, animation, and multimedia.

fundamentals first

All data in Python are objects. We introduce and use objects from Chapter 3, but defining custom classes are covered in the middle of the book starting from Chapter 7. The book focuses on fundamentals first: it introduces basic programming concepts and techniques on selections, loops, and functions before writing custom classes.

examples and exercises

The best way to teach programming is *by example*, and the only way to learn programming is *by doing*. Basic concepts are explained by example and a large number of exercises with various levels of difficulty are provided for students to practice. Our goal is to produce a text that teaches problem solving and programming in a broad context using a wide variety of interesting examples and exercises.

Sincerely,

Y. Daniel Liang
y.daniel.liang@gmail.com
www.cs.armstrong.edu/liang
www.pearsonhighered.com/liang

Pedagogical Features

The book uses the following elements to get the most from the material:

- **Objectives** list what students should learn in each chapter. This will help them determine whether they have met the objectives after completing the chapter.
- The **Introduction** opens the discussion with representative problems to give the reader an overview of what to expect from the chapter.
- **Key Points** highlight the important concepts covered in each section.
- **Check Points** provide review questions to help students track their progress and evaluate their learning.
- **Problems**, carefully chosen and presented in an easy-to-follow style, teach problem solving and programming concepts. The book uses many small, simple, and stimulating examples to demonstrate important ideas.
- **Key Terms** are listed with a page number to give students a quick reference to the important terms introducd in the chapter.
- The **Chapter Summary** reviews the important subjects that students should understand and remember. It helps them reinforce the key concepts they have learned in the chapter.
- **Test Questions** are available online, grouped by sections for students to do self-test on programming concepts and techniques.
- **Programming Exercises** are grouped by sections to provide students with opportunities to apply on their own the new skills they have learned. The level of difficulty is rated as easy (no asterisk), moderate (*), hard (**), or challenging (***). The trick of learning programming is practice, practice, and practice. To that end, the book provides a great many exercises.
- **Notes**, **Tips**, and **Cautions** are inserted throughout the text to offer valuable advice and insight on important aspects of program development.

Note

Provides additional information on the subject and reinforces important concepts.

Tip

Teaches good programming style and practice.

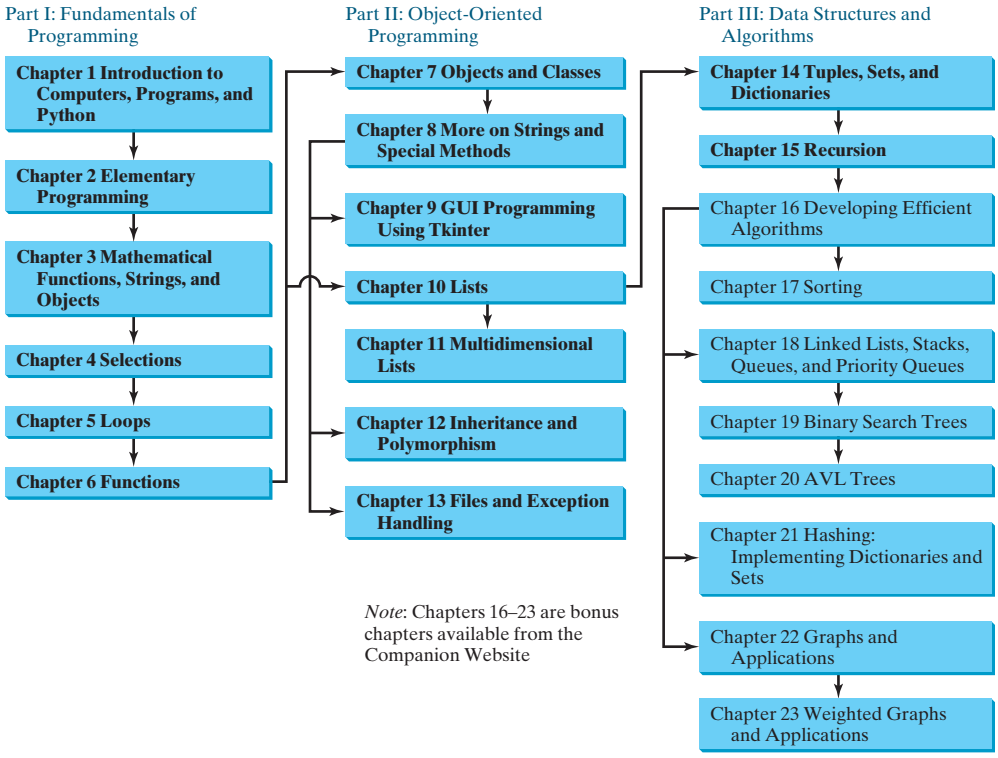
Caution

Helps students steer away from the pitfalls of programming errors.

Flexible Chapter Orderings

Graphics is a valuable pedagogical tool for learning programming. The book uses Turtle graphics in Chapters 1–6 and Tkinter in the rest of the book. However, the book is designed to give the instructors the flexibility to skip the sections on graphics or to cover them later. The following diagram shows the chapter dependencies.

Chapter 10, Lists can be covered right after Chapter 6, Functions. Chapter 14, Tuples, Sets, and Dictionaries can be covered after Chapter 10.



Organization of the Book

The chapters can be grouped into three parts that, taken together, form a comprehensive introduction to Python programming. Because knowledge is cumulative, the early chapters provide the conceptual basis for understanding programming and guide students through simple examples and exercises; subsequent chapters progressively present Python programming in detail, culminating with the development of comprehensive applications.

Part I: Fundamentals of Programming (Chapters 1–6)

The first part of the book is a stepping stone, preparing you to embark on the journey of learning programming. You will begin to know Python (Chapter 1) and will learn fundamental programming techniques with data types, variables, constants, assignments, expressions, operators, objects, and simple functions and string operations (Chapters 2–3), selection statements (Chapter 4), loops (Chapter 5), and functions (Chapter 6).

Part II: Object-Oriented Programming (Chapters 7–13)

This part introduces object-oriented programming. Python is an object-oriented programming language that uses abstraction, encapsulation, inheritance, and polymorphism to provide great flexibility, modularity, and reusability in developing software. You will learn object-oriented programming (Chapters 7–8), GUI programming using Tkinter (Chapter 9), lists (Chapter 10), multidimensional lists (Chapter 11), inheritance, polymorphism, and class design (Chapter 12), and files and exception handling (Chapter 13).

Part III: Data Structures and Algorithms (Chapters 14–15 and Bonus Chapters 16–23)

This part introduces the main subjects in a typical data structures course. Chapter 14 introduces Python built-in data structures: tuples, sets, and dictionaries. Chapter 15 introduces

recursion to write functions for solving inherently recursive problems. Chapters 16–23 are bonus chapters on the Companion Website. Chapter 16 introduces measurement of algorithm efficiency and common techniques for developing efficient algorithms. Chapter 17 discusses classic sorting algorithms. You will learn how to implement linked lists, queues, and priority queues in Chapter 18. Chapter 19 presents binary search trees, and you will learn about AVL trees in Chapter 20. Chapter 21 introduces hashing, and Chapters 22 and 23 cover graph algorithms and applications.

Student Resource Website

The Student Resource Website (www.cs.armstrong.edu/liang/py) contains the following resources:

- Answers to review questions
- Solutions to even-numbered programming exercises
- Source code for the examples in the book
- Interactive self-test questions (organized by sections for each chapter)
- Supplements on using Python IDEs, advanced topics, etc.
- Resource links
- Errata

Additional Supplements

The text covers the essential subjects. The supplements extend the text to introduce additional topics that might be of interest to readers. The supplements listed in this table are available from the Companion Website.

Part I. General Supplements
- A. Glossary
- B. Installing and Using Python
- C. Python IDLE
- D. Python on Eclipse
- E. Python on Eclipse Debugging
- F. Python Coding Style Guidelines

Part II. Advanced Python Topics
- A. Regular Expressions
- B. Obtaining Date and Time
- C. The `str` Class's `format` Method
- D. Pass Arguments from Command Line
- E. Database Programming

Instructor Resource Website

The Instructor Resource Website, accessible from www.cs.armstrong.edu/liang/py, contains the following resources:

- Microsoft PowerPoint slides with interactive buttons to view full-color, syntax-highlighted source code and to run programs without leaving the slides.
- Solutions to all the review questions and exercises. Students will have access to the solutions of even-numbered programming exercises.
- Web-based quiz generator. (Instructors can choose chapters to generate quizzes from a large database of more than 800 questions.)

- Sample exams. In general, each exam has four parts:
  - Multiple-choice questions or short-answer questions
  - Correct programming errors
  - Trace programs
  - Write programs
- Projects. In general, each project gives a description and asks students to analyze, design, and implement the project.

Some readers have requested the materials from the Instructor Resource Website. Please understand that these are for instructors only. Such requests will not be answered.

MyProgrammingLab™

Online Practice and Assessment with MyProgrammingLab

MyProgrammingLab helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyProgrammingLab 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, a MyProgrammingLab course consists of hundreds of small practice problems 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.

MyProgrammingLab 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 MyProgrammingLab in your course, visit www.myprogramminglab.com.

VideoNote

VideoNotes

VideoNotes are Pearson's new visual tool designed for teaching students key programming concepts and techniques. These short step-by-step videos demonstrate how to solve problems from design through coding. VideoNotes allow for self-placed instruction with easy navigation including the ability to select, play, rewind, fast-forward, and stop within each VideoNote exercise.

Margin icons in your textbook let you know when a VideoNote video is available for a particular concept or homework problem.

LiveLab

This book is accompanied by a complementary Web-based course assessment and management system for instructors. The system has four main components:

- The **Automatic Grading System** can automatically grade programs.
- The **Quiz Creation/Submission/Grading System** enables instructors to create and modify quizzes that students can take and be graded upon automatically.
- The **Peer Evaluation System** enables peer evaluations.
- **Checking plagiarisms, tracking grades, attendance, etc.,** lets students track their grades, and enables instructors to view the grades of all students, to check plagiarisms, and to track students' attendance.

The main features of the Automatic Grading System include:

- Students can run and submit exercises. (The system checks whether their program runs correctly—students can continue to run and resubmit the program before the due date.)
- Instructors can review submissions, run programs with instructor test cases, correct them, and provide feedback to students.
- Instructors can create/modify their own exercises, create public and secret test cases, assign exercises, and set due dates for the whole class or for individuals.
- All the exercises in the text can be assigned to students. Additionally, LiveLab provides extra exercises that are not printed in the text.
- Instructors can sort and filter all exercises and check grades (by time frame, student, and/or exercise).
- Instructors can check plagiarisms for a programming exercise.
- Instructors can delete students from the system.
- Students and instructors can track grades on exercises.

The main features of the Quiz System are:

- Instructors can create/modify quizzes from the test bank or a text file or create completely new tests online.
- Instructors can assign the quizzes to students and set a due date and test time limit for the whole class or for individuals.
- Students and instructors can review submitted quizzes.
- Instructors can analyze quizzes and identify students' weaknesses.
- Students and instructors can track grades on quizzes.

The main features of the Peer Evaluation System include:

- Instructors can assign/unassign exercises for peer evaluation.
- Instructors can view peer evaluation reports.

Acknowledgments

I would like to thank Armstrong Atlantic State University for enabling me to teach what I write and for supporting me in writing what I teach. Teaching is the source of inspiration for the book. I am grateful to the instructors and students who have offered comments, suggestions, bug reports, and praise.

This book has been greatly enhanced thanks to the outstanding reviewers. They are:

Claude Anderson – Rose-Hulman Institute of Technology
Lee Cornell – Minnesota State University – Mankato
John Magee – Boston University
Shyamal Mitra – University of Texas – Austin
Yenumula Reddy – Grambling State University
David Sullivan – Boston University
Hong Wang – University of Toledo

It is a great pleasure, honor, and privilege to work with Pearson. I would like to thank Tracy Dunkelberger, Marcia Horton, Michael Hirsch, Matt Goldstein, Carole Snyder, Tim Huddleston, Yez Alayan, Jeff Holcomb, Gillian Hall, Rebecca Greenberg, and their colleagues for organizing, producing, and promoting this project.

As always, I am indebted to my wife, Samantha, for her love, support, and encouragement.