Fundamentals of Python:

From First Programs Through Data Structures

Kenneth A. Lambert

Martin Osborne, Contributing Author

COURSE TECHNOLOGY
CENGAGE Learning™

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

Fundamentals of Python: From First Programs Through Data Structures
Kenneth A. Lambert

Executive Editor: Marie Lee
Acquisitions Editor: Amy Jollymore
Senior Product Manager: Alyssa Pratt
Development Editor: Ann Shaffer
Editorial Assistant: Julia Leroux-Lindsey
Marketing Manager: Bryant Chrzan
Content Project Manager: Matt Hutchinson
Art Director: Marissa Falco
Compositor: Gex Publishing Services

ISBN-13: 978-1-4239-0218-8

ISBN-10: 1-4239-0218-1

Course Technology
25 Thomson Place
Boston, Massachusetts 02210
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at:
international.cengage.com/region

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

For your lifelong learning solutions, visit **course.cengage.com**.

Purchase any of our products at your local college store or at our preferred online store **www.ichapters.com**.

Printed in Canada
1 2 3 4 5 6 7 12 11 10 09 08

PREFACE

Welcome to *Fundamentals of Python*. This text is intended for a complete, first-year study of programming and problem-solving. It covers the material taught in typical Computer Science 1 and Computer Science 2 courses (CS1 and CS2) at the undergraduate level.

This book covers five major aspects of computing:

1. **Programming Basics**—Data types, control structures, algorithm development, and program design with functions are basic ideas that you need to master in order to solve problems with computers. This book examines these core topics in detail and gives you practice employing your understanding of them to solve a wide range of problems.
2. **Object-Oriented Programming (OOP)**—Object-Oriented Programming is the dominant programming paradigm used to develop large software systems. This book introduces you to the fundamental principles of OOP and enables you to apply them successfully.
3. **Data and Information Processing**—Most useful programs rely on data structures to solve problems. These data structures include strings, arrays, files, lists, stacks, queues, trees, sets, dictionaries, and graphs. This book gives you experience using, building, and assessing the performance of data structures. The general concept of an abstract data type is introduced, as is the difference between abstraction and implementation. You'll learn to use complexity analysis to evaluate space/time tradeoffs of different implementations of ADTs.
4. **Software Development Life Cycle**—Rather than isolate software development techniques in one or two chapters, this book deals with them throughout in the context of numerous case studies. Among other things, you'll learn that coding a program is often not the most difficult or challenging aspect of problem solving and software development.
5. **Contemporary Applications of Computing**—The best way to learn about programming and problem solving is to create interesting programs with real-world applications. In this book, you'll begin by creating applications that involve numerical problems and text processing. For example, you'll learn the basics of encryption techniques such as those that are used to make your credit card number and other information secure on the Internet. But unlike many other introductory texts, this one does not restrict itself to problems involving numbers and text. Most contemporary applications involve graphical user interfaces, event-driven programming, graphics, and network communications. These topics are presented in optional, standalone chapters.

Why Python?

Computer technology and applications have become increasingly more sophisticated over the past two decades, and so has the computer science curriculum, especially at the introductory level. Today's students learn a bit of programming and problem–solving, and are then expected to move quickly into topics like software development, complexity analysis, and data structures that, twenty years ago, were relegated to advanced courses. In addition, the ascent of object-oriented programming as the dominant paradigm of problem solving has led instructors and textbook authors to bring powerful, industrial-strength programming languages such as C++ and Java into the introductory curriculum. As a result, instead of experiencing the rewards and excitement of solving problems with computers, beginning computer science students often become overwhelmed by the combined tasks of mastering advanced concepts as well as the syntax of a programming language.

This book uses the Python programming language as a way of making the first year of computer science more manageable and attractive for students and instructors alike. Python has the following pedagogical benefits:

- Python has simple, conventional syntax. Python statements are very close to those of pseudocode algorithms, and Python expressions use the conventional notation found in algebra. Thus, students can spend less time learning the syntax of a programming language and more time learning to solve interesting problems.
- Python has safe semantics. Any expression or statement whose meaning violates the definition of the language produces an error message.
- Python scales well. It is very easy for beginners to write simple programs in Python. Python also includes all of the advanced features of a modern programming language, such as support for data structures and object-oriented software development, for use when they become necessary.
- Python is highly interactive. Expressions and statements can be entered at an interpreter's prompts to allow the programmer to try out experimental code and receive immediate feedback. Longer code segments can then be composed and saved in script files to be loaded and run as modules or standalone applications.
- Python is general purpose. In today's context, this means that the language includes resources for contemporary applications, including media computing and networks.
- Python is free and is in widespread use in industry. Students can download Python to run on a variety of devices. There is a large Python user community, and expertise in Python programming has great resume value.

To summarize these benefits, Python is a comfortable and flexible vehicle for expressing ideas about computation, both for beginners and for experts as well. If students learn these ideas well in the first year, they should have no problems making a quick transition to other languages needed for courses later in the curriculum. Most importantly, beginning students will spend less time staring at a computer screen and more time thinking about interesting problems to solve.

Organization of the Book

Chapters 1 through 10 constitute the core of a CS1 course. The approach in these chapters is easygoing, with each new concept introduced only when it is needed.

Chapter 1 introduces computer science by focusing on two fundamental ideas, algorithms and information processing. A brief overview of computer hardware and software, followed by an extended discussion of the history of computing, sets the context for computational problem solving.

Chapters 2 and 3 cover the basics of problem solving and algorithm development using the standard control structures of expression evaluation, sequencing, Boolean logic, selection, and iteration with the basic numeric data types. Emphasis in these chapters is on problem solving that is both systematic and experimental, involving algorithm design, testing, and documentation.

Chapters 4 and 5 introduce the use of the strings, text files, lists, and dictionaries. These data structures are both remarkably easy to manipulate in Python and support some interesting applications. Chapter 5 also introduces simple function definitions as a way of organizing algorithmic code.

Chapter 6 explores the technique and benefits of procedural abstraction with function definitions. Top-down design, stepwise refinement, and recursive design with functions are examined as means of structuring code to solve complex problems. Details of namespace organization (parameters, temporary variables, and module variables) and communication among software components are discussed. An optional section on functional programming with higher-order functions shows how to exploit functional design patterns to simplify solutions.

Chapter 7 focuses on the use of existing objects and classes to compose programs. Special attention is paid to the interface, or set of methods, of a class of objects and the manner in which objects cooperate to solve problems. This chapter also introduces two contemporary applications of computing, graphics and image processing—areas in which object-based programming is particularly useful.

Chapter 8 introduces object-oriented design with class and method definitions. Several examples of simple class definitions from different application domains are presented. Some of these are then integrated into more realistic

applications, to show how object-oriented software components can be used to build complex systems. Emphasis is on designing appropriate interfaces for classes that exploit inheritance and polymorphism.

Chapters 9 and 10 cover advanced material related to two important areas of computing: graphical user interfaces and networks. Although these two chapters are entirely optional, they give students challenging experiences at the end of the first semester course. Chapter 9 contrasts the event-driven model of GUI programs with the process-driven model of terminal-based programs. The creation and layout of GUI components are explored, as well as the decomposition of a GUI-based program using the model/view/controller pattern. Chapter 10 introduces multithreaded programs and the construction of simple network-based client/server applications.

Chapters 11-20 cover the topics addressed in a traditional CS2 course. These topics include specialized abstract data types, with a focus on interfaces, implementations, and applications. Other important CS2 topics include recursive processing of data structures, search and sort algorithms, and the tools used in software development, such as complexity analysis, unit testing, and graphical notations (UML) to document designs.

Chapters 11 through 13 explore tools used in software development. Chapter 11 introduces complexity analysis with big-O notation. Enough material is presented to enable you to perform simple analyses of the running time and memory usage of algorithms and data structures, using search and sort algorithms as examples. Chapter 12 examines tools used in the design and testing of software. These include basic UML diagrams, documentation of classes and methods, and unit testing. Chapter 13 begins with an overview of various categories of collection ADTs. The chapter then covers the details of processing arrays and linear linked structures, the concrete data structures used to implement many ADTs. You learn the underlying models of computer memory that support arrays and linked structures and the time/space tradeoffs that they entail.

Armed with these tools, you are then ready to consider the different collection ADTs, which form the subject of Chapters 14-20.

Chapters 14-16 present the linear collections, stacks, queues, and lists. Each collection is viewed first from the perspective of its users, who are aware only of an interface and a set of performance characteristics possessed by a chosen implementation. The use of each collection is illustrated with one or more applications, and then several implementations are developed and their performance is analyzed. Emphasis is placed on the inclusion of conventional methods in interfaces to allow different types of collections to collaborate in applications. For example, one such method creates an iterator, which allows any collection to be traversed in the context of a simple loop structure.

Chapters 17-20 present advanced data structures and algorithms as a transition to later courses in computer science. Chapter 17 visits recursion for the second time in the book. This pass includes an examination of advanced algorithms for sorting, backtracking search, recursive descent parsing, and the processing of recursive data structures such as Lisp-like lists. Chapter 18 discusses various tree structures, including binary search trees, heaps, and expression trees. Chapter 19 examines the implementation of the unordered collections, dictionaries and sets, using hashing strategies. Chapter 20 provides an introduction to graphs and graph-processing algorithms.

Special Features

This book explains and develops concepts carefully, using frequent examples and diagrams. New concepts are then applied in complete programs to show how they aid in solving problems. The chapters place an early and consistent emphasis on good writing habits and neat, readable documentation.

The book includes several other important features:

- Case studies—These present complete Python programs ranging from the simple to the substantial. To emphasize the importance and usefulness of the software development life cycle, case studies are discussed in the framework of a user request, followed by analysis, design, implementation, and suggestions for testing, with well-defined tasks performed at each stage. Some case studies are extended in end-of-chapter programming projects.
- Chapter objectives and chapter summaries—Each chapter begins with a set of learning objectives and ends with a summary of the major concepts covered in the chapter.
- Key terms and a glossary—When a technical term is introduced in the text, it appears in boldface. Definitions of the key terms are also collected in a glossary.
- Exercises—Most major sections of each chapter end with exercise questions that reinforce the reading by asking basic questions about the material in the section. Each chapter ends with a set of review exercises.
- Programming projects—Each chapter ends with a set of programming projects of varying difficulty.
- Software toolkits for graphics and image processing—This book comes with two open-source Python toolkits for the easy graphics and image processing discussed in Chapter 7. These are can be obtained from the student downloads page on *www.course.com*, or at *http://home.wlu.edu/~lambertk/python/*
- Appendices—Three appendices include information on obtaining Python resources, installing the toolkits, and using the toolkits' interfaces.

Supplemental Resources

The following supplemental materials are available when this book is used in a classroom setting. All of the teaching tools available with this book are provided to the instructor on a single CD-ROM.

Electronic Instructor's Manual

The Instructor's Manual that accompanies this textbook includes:

- Additional instructional material to assist in class preparation, including suggestions for lecture topics.
- Solutions to all the end-of-chapter materials, including the Programming Exercises.

ExamView®

This textbook is accompanied by ExamView, a powerful testing software package that allows instructors to create and administer printed, computer (LAN-based), and Internet exams. ExamView includes hundreds of questions that correspond to the topics covered in this text, enabling students to generate detailed study guides that include page references for further review. These computer-based and Internet testing components allow students to take exams at their computers, and save the instructor time because each exam is graded automatically.

PowerPoint Presentations

This book comes with Microsoft PowerPoint slides for each chapter. These are included as a teaching aid either to make available to students on the network for chapter review, or to be used during classroom presentations. Instructors can modify slides or add their own slides to tailor their presentations.

Distance Learning

Course Technology is proud to offer online courses in WebCT and Blackboard. For more information on how to bring distance learning to your course, contact your local Cengage Learning sales representative.

Source Code

The source code is available at *www.cengage.com/computerscience*, and also is available on the Instructor Resources CD-ROM. If an input file is needed to run a program, it is included with the source code.

Solution files

The solution files for all programming exercises are available at *www.cengage.com/computerscience* and are available on the Instructor Resources CD-ROM. If an input file is needed to run a programming exercise, it is included with the solution file.

We Appreciate Your Feedback

We have tried to produce a high-quality text, but should you encounter any errors, please report them to *lambertk@wlu.edu*. A listing of errata, should they be found, as well as other information about the book, will be posted on the Web site *http://home.wlu.edu/~lambertk/python/*.

Acknowledgments

I would like to thank my contributing author, Martin Osborne, for many years of advice, friendly criticism, and encouragement on several of my book projects.

I would like to thank my colleague, Sara Sprenkle, and our students at Washington and Lee University for classroom testing this book over several semesters.

I would like to thank the following reviewers for the time and effort they contributed as they completed their reviews of each chapter: Paul Albee, Central Michigan University; Andrew Danner, Swarthmore College; Susan Fox, Macalester College; Robert Franks, Central College; and Jim Slack, Minnesota State University, Mankato. Also, thank you to the following reviewers who contributed their thoughts on the original book proposal: Christian Blouin, Dalhousie University; Margaret Iwobi, Binghamton University; Sam Midkiff, Purdue University; and Ray Morehead, West Virginia University.

Also, thank you to the individuals at Course Technology who helped to assure that the content of all data and solution files used for this text were correct and accurate: Chris Scriver, MQA Project Leader and Serge Palladino, MQA Tester.

Finally, thanks to several other people whose work made this book possible: Ann Shaffer, Developmental Editor, Shaffer Technical Editing, LLC; Marisa Taylor, Senior Project Manager, GEX Inc.; Amy Jollymore, Acquisitions Editor, Course Technology; Alyssa Pratt, Senior Product Manager, Course Technology; and Matt Hutchinson, Content Project Manager, Course Technology.

Dedication

To my pal, Ken Van Ness

Kenneth A. Lambert
Lexington, VA