# BRIEF CONTENTS

# CONTENTS

# PREFACE

A FIRST COURSE IN COMPUTER SCIENCE IS ABOUT A NEW WAY OF SOLVING PROBLEMS: computationally. As a result, the focus of a first course in computation should be on problem solving. Practically speaking, however, the teaching of problem solving is inexorably intertwined with the computer language used. Thus, the choice of language for this first course is very important.

We have chosen Python as the introductory language for beginning programming students—majors and nonmajors alike—based on our combined 30 years of experience teaching undergraduate introductory computer science at Michigan State University. Having taught the course in Pascal, C/C++, and now Python, we know that an introductory programming language should have two characteristics. First, it should be relatively simple to learn. Since a first course in programming is mostly about problem solving and learning to be more rigorous in applying problem solving to new problems, the programming language should get in the way as little as possible. Python's simplicity, powerful built-in data structures, and advanced control constructs allow students to focus more on problem solving and less on language issues. Second, it should be practical. Students should see that they might use what they are learning while they are learning it. Python supports learning not only fundamental programming issues such as typical programming constructs, a fundamental object-oriented approach, common data structures, and so on, but also more complex computing issues such as threads and regular expressions. Python can also be used to address practical problems such as Web access, database manipulation, and thousands of others from nearly any discipline imaginable using tools provided by an active and generous Python community (see http://pypi.python.org). From having taken *one* course in programming, a student—any student (major or not)—should walk away from a CS1 course and somewhere later in his or her career address a problem by thinking, "Hey, I'll just write a program to solve that." For us, that defines computational thinking. We have seen many examples of this kind of success every semester since the move to Python—in particular, students returning to tell us how useful Python has been to them in their studies and jobs.

We emphasize both the fundamental issues of programming and practicality by focusing on data manipulation and analysis as a theme—allowing students to work on real problems

using either publicly available data sets from various Internet sources or self-generated data sets from their own work and interests. We also emphasize the development of programs, providing multiple, worked out, examples and three entire chapters for detailed design and implementation of programs. As part of this one-semester course, our students have analyzed breast cancer data, catalogued movie actor relationships, predicted disruptions of satellites from solar storms, and completed many other data analysis problems. We have also found that concepts learned in a Python CS1 course transitioned to a CS2 C++ course with little or no impact on either the class material or the students.

Our goals for the book are as follows:

1. Teach problem solving within the context of CS1 to both majors and nonmajors using Python as a vehicle.
2. Provide examples of *developing* programs focusing on the kinds of data analysis problems students might ultimately face.
3. Give students who take no programming course other than this CS1 course a practical foundation in programming, enabling them to produce useful, meaningful results in their respective fields of study.

## BOOK ORGANIZATION

At the highest level our text follows a fairly traditional CS1 order, though there are some differences. For example, we cover strings rather early (before functions) so that we can do more data manipulation early on. We also include elementary file I/O early for the same reason, leaving detailed coverage for a later chapter. Given our theme of data manipulation, we feel this is appropriate. We also "sprinkle" topics like plotting and drawing throughout the text in service of the data manipulation theme.

We use an "object-use-first" approach where we use built-in Python objects and their methods early in the book, leaving the design and implementation of user-designed objects for later. We have found that students are more receptive to building their own classes once they have experienced the usefulness of Python's existing objects. In other words, we motivate the need for writing classes. Functions are split into two parts because of how Python handles mutable objects such as lists as parameters; discussion of those issues can only come after there is an understanding of lists as mutable objects.

Three of the chapters (3, 10, and 13) are primarily program design chapters, providing an opportunity to "tie things together," as well as showing how to design a solution. A few chapters are intended as supplemental reading material for the students though lecturers may choose to cover these topics as well. For background, we provide a Chapter 0 that introduces some general concepts of a computer as a device and some computer terminology. We feel such an introduction is important—everyone should understand a little about a computer, but this material can be left for reading. The last chapters in the text may not be reached in some courses.

# BOOK FEATURES

## Data Manipulation

Data manipulation is a theme. The examples range from text analysis to breast cancer classification. Along the way, we provide some analysis examples using simple graphing. To incorporate drawing and graphing, we use established packages instead of developing our own: one is built in (Turtle graphics); the other is widely used (matplotlib with numpy).

We have tried to focus on non-numeric examples in the book, but some numeric examples are classics for a good reason. For example, we use a rational numbers example for creating classes that overload operators. Our goal is always to use the best examples.

## Problem Solving and Case Studies

Throughout the text, we emphasize problem solving, especially a divide-and-conquer approach to developing a solution. Three chapters (3, 10, and 13) are devoted almost exclusively to program development. Here we walk students through the solution of larger examples. In addition to design, we show mistakes and how to recover from them. That is, we don't simply show a solution, but show a *process of developing* a solution.

## Code examples

There are over 180 code examples in the text—many are brief, but others illustrate piecemeal development of larger problems.

## Interactive Sessions

The Python interpreter provides a wonderful mechanism for briefly illustrating programming and problem-solving concepts. We provide almost 250 interactive sessions for illustration.

## Exercises and Programming Projects

Practice, practice, and more practice. We provide over 275 short exercises for students and nearly 30 longer programming projects (many with multiple parts).

## Self-Test Exercises

Embedded within the chapters are 24 self-check exercises, each with 5 or more associated questions.

## Programming Tips

We provide over 40 special notes to students on useful tips and things to watch out for. These tips are boxed for emphasis.

# SUPPLEMENTARY MATERIAL ONLINE

- For students
  - All example source code
  - Data sets used in examples

The above material is freely available at www.pearsonhighered.com/cssupport.

- For instructors
  - PowerPoint slides
  - Laboratory exercises
  - Figures (PDF) for use in your own slides
  - Exercise solutions

Qualified instructors may obtain supplementary material by visiting www.pearsonhighered.com/irc. Register at the site for access. You may also contact your local Pearson Education representative or send e-mail to computing@aw.com.

## Python 2.x versus 3.x

The text is based on Python 2.x (for x > = 6) rather than Python 3.x since the latter is not backward compatible with standard Python packages at this time. In an appendix we outline the differences. For what we cover in the text, the differences are trivial, but the support of current Python packages is critical to our approach, and we will not be able to move to 3.x until standard packages are more widely available in that release.

# ACKNOWLEDGEMENTS

We are particularly appreciative of the reviewers who provided valuable feedback at multiple stages of this endeavor: Claude Anderson (Rose-Hulman Institute of Technology), Chris Brooks (University of San Francisco), Judy Franklin (Smith College), Alan Garvey (Truman State University), Ronald I. Greenberg (Loyola University), Andrew Harrington (Loyola College), Steve Harrison (Virginia Tech), Christopher Haynes (Indiana University), Cinda Heeren (University of Illinois/Urbana-Champaign), Brian Howard (DePauw University) Janardhan Iyengar (Franklin & Marshall College), Andree Jacobson (University of New Mexico), John Lasseter (Willamette University), Jim Mahoney (Marlboro College), Joe Oldham (Centre College), Holly Patterson-McNeill (Lewis Clark State College), John

Rabung (Randolph-Macon College), Ben Schafer (University of Northern Iowa), David G. Sullivan (Boston University), David A. Sykes (Wofford Colllege). Here at Michigan State University, Erik Eid (now of Bowling Green State University) provided valuable feedback on the first draft. Laurie Dillon provided feedback when she taught from a draft. C. Titus Brown read a draft from a Pythonic perspective and provided encouragement. As a high school student, Angus Burton provided valuable feedback from a novice's perspective. Srikanth Vudayagiri provided many excellent exercises. Scott Buffa made corrections to an early draft. The summer CSE 231 class provided many exercises. Members of the class were: Mohammed Alwahibi, Younsuk Dong, Michael Ford, Gabriel Friedman, Adam Hamilton, Meagan Houang, Ruba Jiddou, and Adam Palmer. The organization of our course was established by Mark McCullen, who is always available for advice. Larry Nyhoff (Calvin College) shared valuable insights over a dinner that started it all.

W. F. PUNCH
R. J. ENBODY