PREFACE

A FIRST COURSE IN COMPUTER SCIENCE IS ABOUT A NEW WAY OF SOLVING PROBLEMS: computationally. Our goal is that after the course, students when presented with a problem will think, "Hey, I can write a program to do that!"

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. 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. 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. Finally, Python is "industrial strength," forming the backbone of companies such as YouTube, DropBox, Industrial Light and Magic, and many others.

The main driver for the second edition of this textbook came from requests for a Python 3 version. We began our course with Python 2 because Python 3 hadn't been released when we started in 2007 and because we expected that it would take some time for important open-source packages such as NumPy and matplotlib to transition to Python 3. When NumPy and matplotlib converted to Python 3 in 2011, we felt comfortable making the transition. Of course, many other useful modules have also been converted to Python 3—the default installation now includes thousands of modules. With momentum building behind Python 3, it was time for us to rewrite our course and this text.

Why Python 3? Python 3 is a major step in the development of Python and is a new and improved version. Some nagging inconsistencies in the Python 2 branch required fixing, and the Python community decided that these changes were worth breaking backward

compatibility. One glaring example is that `print` acted like a function but didn't have standard function syntax. Another important change was moving the default character encoding to Unicode, recognizing the worldwide adoption of the Python language. In many ways beyond the introductory level, Python 3 is a better language, and the community is making the transition to Python 3.

At the introductory level, the transition to Python 3 appears to be relatively small, but the change resulted in touching nearly every page of the book. With a major rewrite in play, we made a number of other improvements as well. We tried to incorporate every aspect of Python 3 that was reasonable for an introductory book. Here are some of the many changes:

- For Python 3:
  - `print` is now a function requiring parenthesis.
  - We adopted the new string formatting approach.
  - Unicode UTF-8 has replaced ASCII as the default character encoding.
  - The "/" operator returns float even for `int` operands. This has implications for students moving on to C++/Java.
  - Sets and their operations have nearly equivalent binary operators. Both are discussed.
  - Comprehensions have been extended to include sets and dictionaries. Though not emphasized, comprehensions are introduced.
  - Many functions now return iterators (e.g., `range` , `map`)
  - Dictionary methods such as items return dictionary views.
  - `input` has replaced `raw_input` .
  - Comparison of dissimilar types results in an error.
  - Literal base specifications changed.
  - File reading and writing changed with the adoption of Unicode default.
  - Some exception code has changed (e.g., `as`).
  - Use of `next` function instead of the .next method for iterators.
- We redid every program and session so that they all met the Python's PEP 8 naming conventions. We introduced some extensions as well to help new students read and write code better.
- Many programs were repaired or wholesale replaced based on the changes to Python 3.
- We added a set of nine **Rules** to guide novice programmers.
- Each chapter now has a reference summary to make it easier to refer to syntax and semantics.
- The Quick Check exercises now have answers in the back of the book.
- We inserted an early chapter that briefly introduces file reading and exceptions, because our approach uses file reading throughout. A more detailed coverage occurs later.
- We removed as many forward reference elements (talking about a feature before it was fully presented) as possible.
- We added a chapter at the end on Python features that more advanced students will find useful and interesting.

- All the exercises were ordered, from easy to hard, with a line marking the transition from easy to hard.
- Of course, we fixed many errors.

As in the first edition, 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 analyzed breast cancer data, cataloged 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 negative impact on either the class material or the students.

Our goals for the book are as follows:

- Teach problem solving within the context of CS1 to both majors and nonmajors using Python as a vehicle.
- Provide examples of *developing* programs focusing on the kinds of data-analysis problems students might ultimately face.
- 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 outside 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 five 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
  - VideoNotes (icons in the margin indicate when a VideoNote is available for a topic)

The above material is available at www.pearsonhighered.com/punch.

- **For instructors**
  - PowerPoint slides
  - Laboratory exercises
  - Figures for use in your own slides (PDF)
  - 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 representative.

- **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 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 grade book 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.

ACKNOWLEDGMENTS

We acknowledge and thank the following reviewers for their contribution to improving the second edition:

- Ric Heishman – George Mason University
- Erik Linstead – Chapman University
- John S. Mallozzi – Iona College
- Daniel D. McCracken – City College of New York
- Deborah S. Noonan – College of William and Mary
- Jeff Ondich – Carleton College
- Leon Tietz – Minnesota State University, Mankato

We are also appreciative of the reviewers who provided valuable feedback for the first edition: 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 College). 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