SCIENTIFIC PYTHON FOR BOTH EXPERT AND NOVICE PROGRAMMERS

By Alex Small


When learning a new language, I want a book that will walk through simplified versions of things that I’d actually do. I’ve written lots of Matlab code, but I wanted to learn Python to teach computational physics in an open source environment. While going through this book, my usual thought was, “Great, now how would I go further and …” only to have my question answered in the next section. Along the way, I even managed to learn a few things about programming.

A Glimpse Inside

This book goes through Python in particular, and programming in general, via tasks that scientists will likely perform. The book starts with the fundamentals, showing the reader how to evaluate and print the value of a simple 1D kinematic formula, even discussing the command line and editor for true novices. The rest of the chapter further develops formulas, simple math (such as temperature conversions and trajectory calculations), and the output of strings and numbers. This chapter would be useful for a novice getting comfortable with scientific computation, yet it would not bore an experienced programmer who wants to skim for Python syntax.

Chapter 2 delves into the meat of scientific programming: loops and lists. A Python list is a bit different from arrays in many languages, and the for loops are handled in a more versatile manner than some might be accustomed to; for instance, you can loop over elements of an array, rather than loop over a range. The author explains the concepts well. The example used repeatedly is temperature conversion, which is appropriate simple for introducing programming to novices.

Chapter 3 covers functions and logic, and the examples are a bit more varied. I appreciated the versatility of Python’s syntax in this chapter. I came away realizing how many different—and readable—ways there are to do the same thing.

The author focuses on input data, errors, and modules (function libraries, essentially) in chapter 4, and continues to illustrate the versatility of Python. The pace quickens somewhat, but the novice has already made the biggest conceptual leaps in chapters 2 and 3.

One more big conceptual leap—arrays—is introduced in chapter 5, along with plotting. I appreciated the author’s decision to stick close to Matlab conventions for plotting syntax, along with his emphasis on good habits, such as vectorizing code.

For novices, graphics will be welcome after four chapters of text output, and array programming is developed with 1D arrays for as long as possible; multidimensional arrays are introduced in the penultimate section. There’s also a rich set of exercises at the end of this chapter (for example, how to animate a wave packet).

With just these chapters and a few elements from subsequent chapters, a beginner could complete programming projects in other courses. An experienced programmer would even be comfortable with Python for many core tasks of scientific computing. The remaining chapters, though, are well worth studying.

For instance, chapter 6’s discussion of files, strings, and dictionaries shows more of Python’s versatility. Perhaps the best example of this is a short program to search through HTML files and extract information for analysis.

I also learned a lot in chapter 7, which focused on an introduction to classes. It opens with excellent motivation—how can you pass functions to programs for doing calculus if different functions have different numbers of parameters? It then develops this sophisticated topic by example.

Related to the book’s treatment of probabilistic concepts, the author uses interval arithmetic as a summarizing example in chapter 7, and he does an excellent job of explaining interval arithmetic’s relationship to error analysis, via examples with...
measurements that students might do in freshman labs. I would love to live in a world where every freshman encounters this example, and I salute the author’s inclusion of it. Although it might not have been intended as such, ending chapter 7 with error analysis provides a great bridge to randomness in chapter 8, if the instructor wishes to emphasize that connection.

My favorite chapter is chapter 8, with its focus on random numbers and simple games. Students are invited to solve probability problems (such as the odds of drawing some combination of cards from a deck) via Monte Carlo simulations rather than formulas with factorials. If we can condition students to try a Monte Carlo simulation to help with a hard question in probability, then we’ve imparted a valuable habit (even if we do annoy purists). In a similar vein, the book introduces the random walk, which is a model of great utility and universality, yet simple enough for almost anybody to program and play with.

Chapter 9’s focus on object-oriented programming expands on the material from chapter 7. More specifically, it introduces the concept of class hierarchy through three examples that science students should care about: Writing a general program to compute numerical derivatives, writing a general program to do numerical integration, and making and plotting geometrical shapes.

Key Strengths
I really appreciate that the author wrote the book in a lucid, thoughtful manner. Everything just made sense as I read along, and most of the examples were things that I would want to do. If you want to learn Python and become a better programmer in the process, this book is for you. Topics are taught in a way that we would teach ourselves: by using lots of examples.

Second, the appendices complement the rich variety of examples in the chapters to make it a self-contained book for a first course on scientific computing. Particularly useful are Appendices B through E, covering discrete calculus and differential equations at increasing levels of sophistication. Instructors can pick their own balance between teaching programming and teaching the beginnings of numerical methods.

Third, in addition to excellent homework exercises, each chapter ends with a challenging summarizing example. My favorites are numerical integration, bisection root finding, animating a time-dependent temperature profile that starts off sinusoidal and gets exponentially damped by heat conduction over time, interval arithmetic, and compounded interest with a random interest rate. Many of the problems have a similar flavor.

Some Concerns
For students who have never programmed before, I might not recommend this book. Heavy use of relevant examples is certainly important. However, learning to program means learning to work in a structured, step-by-step manner with tools that obey strict rules. There aren’t many diagrams, charts, or tables to illustrate concepts such as loops, lists, logic, and arrays. For example, there are two figures in chapter 2 and no figures in chapter 3. Aside from sample graphs, chapter 5 only has one figure.

Even after an introductory programming course, some students still struggle with the orderly and often quite literal thought processes required to construct good code. Without a structured understanding of the concepts and structures, a student working from examples can fall into the same pattern-matching approaches that we see them take in their pencil-and-paper homework. (For example, “This homework problem has a spring in it, and there was a problem in the book with a spring in it, so I will plug in the formulas from the example.”)

Even strong students might appreciate some tables or Venn diagrams to summarize, say, distinctions between lists, tuples, and arrays. More step-by-step lists or pseudocode prior to developing sample code would also help.

Other Features
All of the code in the book is available (along with the author’s lecture slides) on the author’s website (http://vefur.simula.no/intro-programming). The website also has additional instructions for downloading and installing Python and a suitable editor. However, I do wish that the index was a bit more extensive, perhaps with a separate index of all of the Python commands introduced in the book.

In conclusion, this is an excellent book for the experienced programmer who wants to learn a new language and become a better programmer. It contains valuable information for students new to scientific computing, and would be the perfect bridge between an introduction to programming and an advanced course on numerical methods or computational science.

Alex Small is an assistant professor in the Department of Physics and Astronomy at California State Polytechnic University, Pomona. His primary research interests are theoretical and computational issues in superresolution microscopy. Small has a PhD in physics from the University of California, Santa Barbara. Contact him at arsmall@csupomona.edu.