Practical Python for Data Science

This material is focused on writing complete, command-line programs that you can understand and use to get something done. Most of the programs included as examples and my implementations of the exercises are 100 lines or fewer. By leveraging standard modules and data types in Python, I will show you how to write readable, documented, and testable code. The chapters show many examples of complete programs you can run and alter, and the exercises expect you to write complete programs that satisfy a given test suite. After completing this material, you should be able to:

- Write, test, and document programs in bash and Python
- Use the source code management system Git to version, share, and distribute code
- Use parallelization techniques and hardware (HPC) to run programs faster
- Package and distribute software to create reproducible workflows

Programming Environment

The material begins with the Unix command line. If you are working on Windows, I highly recommend you install Windows Subsystem for Linux and probably GitBash. If you are on an Apple computer, you have a full Unix system available through your Terminal app. The author uses a Mac with iTerm and vim editor to write, debug, and run programs. You may wish to use an editor like Sublime, TextWrangler, or Atom or an integrated develoment environment (IDE) like VSCode or PyCharm. However you choose to *write* code, this material assume you will *run* it from the command line. For many reasons, I have chosen not to use Jupyter Notebooks. Some chapters may include a Notebook, but I would prefer to have students write command-line programs and use a testing framework like PyTest to ensure that code runs correctly, top to bottom.

Python

I personally prefer statically typed and "functional" languages like Rust, Elm, and Haskell, but I concede the dominance of dynamically typed languages such as Perl, Python, and Ruby. This material is intended to steer the student towards best practices when working in Python to avoid what I consider to be dangerous tendencies of the language. I will be using Python version 3.

Author

> "Computer programming has always been a self-taught, maverick occupation." - Ellen Ullman

My name is Ken Youens-Clark, and I'm a Senior Scientific Programmer at the Univrersity of Arizona. While I have a Masters of Science in Biosystems Engineering, my undergraduate trianing was a BA in English Lit with a minor in music. As a kid, I had a RadioShack computer (probably a TRS-80, but I don't really remember) on which I wrote maybe two programs in BASIC. I never got into computing until after completing my bachelor's degree in 1995 when I started playing with computers and databases at my first job. The next year I landed a position where I learned Visual Basic on Windows 3.1, and that was when I actually got hooked on programming and problem solving. Since then, I've worked in several languages on various operating systems, and I've learned by doing – always having to tackle problems I was never trained to solve. I spent the longest part of my career using Perl in a bioinformatics settting, but Python has definitely taken over the data processing and machine learning space, so that's what we'll cover.

Organization

> "The only way to learn a new programming language is by writing programs in it." - Dennis Ritchie

Each chapter is written in Markdown format which you can read directly on GitHub where I host the material. I also use `pandoc` to convert this document to PDF via Latex, but sometimes things don't work perfectly smoothly. Still, there are PDFs you can download and read offline (or print, gasp!) if you like. I mostly like to write lots of programs (there are almost 150 in this repo!) and comment on them. I embed the program directly into the Markdown, but they are all available in the `examples` directory for you to alter and run.

Perhaps owing to my musical training, I approach my teaching with the idea that you need to practice – quite a bit. You cannot learn Python or any language just by reading a text; therefore, most chapters have an `exercises` directory that contain code you should write and tests to help you know when they work sufficiently well. Exercises ending in `_a` are considered easier than those ending in `_b`. Some of the tests are minimal, e.g., I expect something like a "usage" statement under certain circumstances but don't verify anything more than the word "usage" at the beginning of your output. I trust you care enough about your user and your craft to actually create a proper help page. I include my own version of a solution that you can use to compare. I spent many years in the Perl community where "There Is More Than One Way To Do It" (TIMTOWTDI) is something of a mantra whereas the Python community espouses “There should be one– and preferably only one –obvious way to do it.” (https://www.python.org/dev/peps/pep-0020/) I disagree with this notion and believe you can find many creative and beautiful solutions. More than anything, the solution that you figured out, that you understand, and that satisfies the test suite is the “right” one for you. Your style will change as you grow more knowledgeable and confident in your programming skills.

Copyright