Programming with Python (/python-novice-inflammation/index.html): Instructor Notes

About the lesson

Our real goal isn't to teach workshop attendees Python as a programming language, but to teach them the basic concepts that all programming depends on. We use Python in our lessons because:

1. It is free and open-source software.
2. It is well-documented and runs on all platforms.
3. It has a large and constantly growing user-base which includes scientists.
4. It is easier for novices to pick up than most other languages.

Legend

We are using a dataset with records on inflammation from patients following an arthritis treatment.

We make reference in the lesson that this data is suspicious and has been synthetically generated in Python by the imaginary "Dr. Maverick"! The script used to generate the inflammation data is included as `code/gen_inflammation.py` (../code/gen_inflammation.py).

Overall

This lesson is written as an introduction to Python, but its real purpose is to introduce the single most important idea in programming: how to solve problems by building functions, each of which can fit in a programmer's working memory. In order to teach that, we must teach people a little about the mechanics of manipulating data with lists and file I/O so that their functions can do things they actually care about. Our teaching order tries to show practical uses of every idea as soon as it is introduced; instructors should resist the temptation to explain the "other 90%" of the language as well.

The final example asks them to build a command-line tool that works with the Unix pipe-and-filter model. We do this because it is a useful skill and because it helps learners see that the software they use isn't magical. Tools like `grep` might be more sophisticated than the programs our learners can write at this point in their careers, but it's crucial they realize this is a difference of scale rather than kind.

Explain that we use Python because:

- It's free.
- It has a lot of scientific libraries, and more are constantly being added.
- It has a large scientific user community.

- It's easier for novices to learn than most of the mature alternatives. (Software Carpentry originally used Perl; when we switched, we found that we could cover as much material in two days in Python as we'd covered in three days in Perl, and that retention was higher.)

We do *not* include instructions on running the Jupyter Notebook in the tutorial because we want to focus on the language rather than the tools. Instructors should, however, walk learners through some basic operations:

- Launch from the command line with `jupyter notebook`.
- Create a new notebook.
- Enter code or data in a cell and execute it.
- Explain the difference between `In[#]` and `Out[#]`.

Watching the instructor grow programs step by step is as helpful to learners as anything to do with Python. Resist the urge to update a single cell repeatedly (which is what you'd probably do in real life). Instead, clone the previous cell and write the update in the new copy so that learners have a complete record of how the program grew. Once you've done this, you can say, "Now why don't we just break things into small functions right from the start?"

The discussion of command-line scripts assumes that students understand standard I/O and building filters, which are covered in the lesson on the shell.

Frequently Argued Issues (FAI)

- `import ... as ...` syntax.

  This syntax is commonly used in the scientific Python community; it is explicitly recommended in documentation to `import numpy as np` and `import matplotlib.pyplot as plt`. Despite that, we have decided not to introduce aliasing imports in this novice lesson due to the additional cognitive load it puts on students, despite the typing that it saves. A good summary of arguments for and against can be found in PR #61 (https://github.com/swcarpentry/python-novice-inflammation/pull/61).

  It is up to you as an individual instructor whether you want to introduce these aliases when you teach this lesson, but we encourage you to please read those arguments thoroughly before deciding one way or the other.

- NumPy methods.

  We used to use NumPy array methods in the first NumPy topic (../01-numpy/). We switched these methods to the equivalent functions because a majority of instructors supported the change; see PR #244 (https://github.com/swcarpentry/python-novice-inflammation/pull/244) for detailed arguments for and against the change.

- Underscores vs. hyphens in filenames

  We used to use hyphens in filenames in order to signify that these Python files should only be run as scripts and never imported. However, after some discussion (https://github.com/swcarpentry/python-novice-inflammation/pull/254), including an informal Twitter poll, we switched over to underscores because many files that start off as Python scripts end up being imported eventually. For that reason, we also added `if __name__ == '__main__'` guards around `main()` calls, which is how real-world Python scripts ensure that imports do not result in side-effects.

After discussing the challenges is a good time to introduce the `b *= 2` syntax.