Python Starter Pack

Written by Nick Roy, last updated 09/10/16

For everyone who has been wanting to get started with Python, this is the document for you! Below you will find instructions and links that will allow you to install all the relevant software you will need, some high-level info about the language as a whole, more links to tutorials and lectures, and a variety of random optional info that you may or may not find useful.

If you have any questions, feel free to email me and I'll be happy to help the best I can (though it should be noted that one of the best things about Python is that most Python-related questions are extremely google-able).

Step 1: Installation

The first thing to do is install Anaconda. (Download here, make sure to grab to 3.5 version!)

One of the things contained within Anaconda is conda. Conda is both a Python *environment manager* and a *package manager*. Two reasons why this is important to have:

- There are currently two different versions of Python in modern usage, Python 2 and Python 3. I'll explain later why this is the case and why it matters, but for now it's just important to know that conda allows you to create, maintain, and easily switch between multiple Python environments, each with their own distinct version of Python.
- Python is package driven. Packages can be thought of as libraries that can be imported into your scripts to access specialized functions and objects. Some packages are ubiquitous and unavoidable, while others are incredibly specialized. The two great advantages of conda are (1) it comes with 300+ of the most popular packages for scientific analysis and (2) it acts as a centralized platform through which to add additional packages to your environments and keep them updated.

Conda comes with a 30 minute test drive to help you get comfortable and set up your initial environments. There is also an Conda Cheat Sheet that you can save and reference for future use.

Also contained within the Anaconda installation:

- IPython : an interactive shell that gives the user extreme flexibility to explore and interface efficiently with your code (Documentation)
- Jupyter Notebook : an incredibly useful and flexible coding platform that we will be using frequently in the course (Documentation)
- Spyder : The best Python IDE I have come across for working effectively in Python. It integrates a nice text editor with an IPython environment that allows for quick code testing and development. (Documentation)

Step 2: General Information

Python 2 vs. 3

As mentioned previously, there are two different versions of Python commonly used today, Python 2 and Python 3. Why didn't Python 3 just replace Python 2? It was decided around 2008 that there were too many things wrong (bugs, inconsistencies, etc.) in Python 2 that it needed not only an incremental update, but a complete overhaul. The overhaul, Python 3, was unfortunately not backward compatible with Python 2. So not only was basic Python 2 code not guaranteed to work in Python 3, but more problematic was that the thousands of packages on which people depended did not yet support Python 3. Thus, no one used

Python 3 when it first came out and stuck with Python 2.

The Python developers were very aware of this issue and have continued support for Python 2, releasing the final incremental update, Python 2.7, in 2010 and promising continued support for Python 2 through 2020. However, Python 3 is the version in active development (with Python 3.5 being released in September 2015) and over the years people have slowly migrated. The initial problems with Python 3 - no backward compatibility or package support - become less of an issue everyday as new users simply choose to start with Python 3 and package managers update their packages for Python 3 compatibility. (History of Python)

Personally, I started coding in Python 2 a few years ago, but used graduate school as an opportunity to finally make the switch to Python 3. **I would suggest new users simply start with Python 3.** In the increasingly rare instance that you need a package without Python 3 support, there are various workarounds available to help you over that obstacle.

Packages

Python is heavily dependent on it's vast array of packages. The Python Package Index lists 88k+ packages available for download. Your anaconda installation comes with hundreds of the most popular, and almost all other packages can be installed easily using the preinstalled package manager **pip**.

Packages are used by importing the package at the beginning of your script, which allows you to then access their specialized functions or objects. Below is a short list of the most widely used packages:

- NumPy : "the fundamental package for scientific computing with Python," the source of most linear algebra functions, data import functions, and general computation
- SciPy : builds on top of Numpy to include more specific, higher level functionality including image processing, statistics, and optimization
- Matplotlib : the primary way to generate figures in Python, replicates much of the functionality of plotting in MATLAB (hence the name)

Again, things are very google-able if you're looking for something specific, there's almost certainly a python package somewhere that has it.

Debugging

One of the great advantages of Python is it's readability and high-level nature which cuts down on long, indecipherable lines of code, leading in turn to fewer bugs. However, there is a Python debugger if you find yourself needing one called **pdb**. The official pdb documentation is a bit dense (Documentation), but I have found this article to be quite useful in exploring the basic functionality of the debugger.

Step 3: Learning and Tutorials

Relative to most programming languages, learning Python is fairly straightforward and intuitive. To start with the fundamentals, many people have recommended the Code Academy online Python course. This should allow everyone to feel comfortable around basic (i.e. package-less) Python.

For a more high-speed learning experience, I have found the "lecture" from Wakari to be extremely helpful. They're not actual lectures, but instead online Jupyter Notebooks that you can follow along with and even manipulate yourself. There is a lecture covering the basics to Python programming and then it jumps right into lectures for individual Python packages including the 3 mentioned before (NumPy, SciPy, Matplotlib).