Scientific computing with Python

John Livingston

May 2016

1 Introduction

There is a wealth of powerful, free, open source software available within the Python ecosystem. This document will guide you through the basics of setting up a robust and convenient platform-independent Python environment, as well as introduce you to the powerful tool `git`, and its accompanying code repository GitHub. I will also introduce Jupyter notebooks, which are a powerful way to develop and share code ideas, and also provide the capability to interactively learn about various subjects right in your browser using the enormous collections of notebooks already freely available.

2 Python setup

2.1 Installing Anaconda

We will use `miniconda`, a light-weight, commandline-only version of the powerful Python distribution `Anaconda`. First, download the Python 2.7 version of the `miniconda` installer for your platform from here for Mac OS X, or here for Linux. I do not recommend using Windows for scientific computing, so if you have a PC, consider installing one of the many user-friendly flavors of Linux, either natively or in a virtual machine. Once the installer is finished downloading, you'll need to execute it from the terminal with:

```
cd /path/to/Miniconda2-latest-MacOSX-x86_64.sh
chmod +x Miniconda2-latest-MacOSX-x86_64.sh
./Miniconda2-latest-MacOSX-x86_64.sh
```

Of course, if you're installing on Linux you'll need to replace the filename above with the correct filename – the one you just downloaded. It will ask some questions during installation, so you will need to type in your answers or press enter to accept the defaults. I recommend accepting the defaults.

2.2 Creating your first Python environment

Now that you've installed `miniconda`, you have a powerful tool for creating and managing virtual Python environments, which are all installed to your home directory so you don't need to use `sudo` when installing Python packages. You also don't need to worry about running into problems with your system's global Python installation, because any environment you create is fully self-contained and can be easily modified or replicated if need be. To create your first environment, open up a new terminal window. This ensures that your `.bashrc` (or equivalent) is sourced, which places the `miniconda` programs into your path. Now, creating an environment is as simple as:

```
conda create -n test python
```

where 'test' is the name of the environment you've just created. This environment can now be 'entered' with:

```
source activate test
```

You'll notice that the command prompt changes when you're inside an environment, so you know exactly which one you're in at all times (eventually you'll want to have multiple environments side by side so you can easily use different package versions – `conda` makes this easy).
To 'leave' an environment, just type:

```
source deactivate
```

and now you'll see the name of the environment disappear from your command prompt. Now that you've seen the basics, try installing a new environment with all the Python packages you probably want already installed:

```
conda create -n py27 pip python ipython numpy scipy matplotlib
```

When it's finished installing, this environment can now be entered just like before, only the name of the environment has changed:

```
source activate py27
```

2.3 Managing an existing environment

Now you've got a working installation of the Python/numpy/scipy/matplotlib stack, a powerful set of tools for scientific computing. What if you find there's another package out there which you'd like to install? You don't have to create an entirely new environment, rather it can be installed into an existing environment using `conda` (for the most common packages) or `pip` (for anything from the entire PyPI universe). The syntax for installing a package is very simple:

```
conda install packagename
```

where `packagename` is the name of the package you want to install. For example, `pandas` is a very powerful library for reading and manipulating tabular data in Python, and can be installed like this:

```
conda install pandas
```

Remember, you should already be inside the environment you wish to install these packages to when you execute the above commands, so check your command prompt if you're not sure. If `conda` can't find the package you're looking for, it's probably on PyPI, which means you can install it with `pip`. To do so, simply replace 'conda' with 'pip' in the above code.

There are many powerful and convenient capabilities of `conda`, but explaining them in detail is beyond the scope of this document. Simply understand that it is a way to create computing environments on your machine, as well as a package management tool similar to `apt-get` on Linux. To learn more about it, read the documentation here.

3 GitHub and `git`

The best and biggest repository of free and open-source software on the internet is GitHub. The reason for its name is the powerful commandline tool `git`, created by Linus Torvalds for the purpose of developing the Linux kernel, as well as making software development a much more reasonable affair in general. In brief, `git` is a tool for version control management, allowing the user to make changes to their code without the headaches of manually storing different versions of the code as they develop it. In reality, it is much more than that – along with GitHub, it is leading the open-source revolution. This is because `git` also makes it much easier for groups of people to work simultaneously on the same software. For small tasks this is not strictly necessary, but as the need for more sophisticated data analysis continues to grow in science and other fields, so will the need for more people to collaborate.

3.1 Installation

If you don't already have `git` installed (you can check by typing `which git` in the terminal), you'll need to install it. If you're using a Mac, the best system-wide package manager nowadays is `brew`. It can be installed very easily by following the instructions here. Once you have `brew`, it is very easy to install git:

```
brew install git
```

If you're using Linux, you already have a package manager, so you can install it the usual way (i.e. `sudo apt-get install git`).

3.2 Usage

Take the time to learn how to use `git` – it will change your life. The documentation is here, and there are plenty of other great online resources to help you, including tutorials with terminal emulators running right in the browser to help get you started (i.e. this one).

3.3 GitHub

Once you have a basic grasp of what `git` is all about, the concept of the accompanying website will make much more sense. In brief, it is a place for code to live and thrive. In practice, using both `git` and GitHub together is very natural, and you'll soon find it makes a lot of sense to use for your own software projects, even if you're not collaborating with anyone. Also, a GitHub account is fast becoming a *de facto* resume for programmers – you can showcase all your hard work and share it with others.

4 Jupyter Notebooks

If you want to learn about Python, particular Python packages, or even about entire subjects, there exist an enormous number of freely available documents, called Jupyter Notebooks, which can help you. There are some which contain good introductions to the basics of Python, some which contain advanced tutorials about complex data analysis methods and tools, and some which are actually entire books on a particular subject. In all cases, Jupyter Notebooks are interactive – you read them *and run the code contained in them* right in your browser. They are also great for sharing a new idea with a collaborator, or even just for testing your own code and exploring your data.

To get started, install Jupyter to your conda environment with `conda install jupyter`. Next, download this Notebook file (download button located at upper right), and execute the following in the terminal:

```
ipython notebook /path/to/downloaded/notebook
```

You will now see a browser window open up which is rendering the Notebook. This browser window is connected to a running Python kernel in the background, so you can actually *run* the code you see, modify it, etc. This is what makes Jupyter Notebooks so powerful. To understand how to use Jupyter Notebooks, there is plenty of online documentation, but a good starting point is to read this. *Bonus credit:* use `git` to download an entire Notebook repository from GitHub, then run the Notebooks on your machine:

```
git clone https://github.com/jrjohansson/scientific-python-lectures.git
cd scientific-python-lectures
ipython notebook Lecture-1-Introduction-to-Python-Programming.ipynb
```