Python Primer

v.2 Jessie C. Runnoe
v.1 Adam D. Myers

Introduction

Python is a high-level scripting language that is useful for manipulating data. As with any programming language, Python has some undesirable features, such as some relatively slow processes and a vast library of complicated dependences. But Python can be used to wrap faster code such as C (using, e.g., the `os.system` or `subprocess.call` routines), and, perhaps most usefully, it has a large number of existing packages for manipulating large datasets.

Getting Started

You will need to set up a local `anaconda` (a.k.a. `conda`) install of Python, which contains tools relevant for ASTR8020. If you haven't downloaded Anaconda before, find the installation instructions on the links page of the course website. For this class, we will all use a Python 3.7 Anaconda environment. In the terminal window, you can define a new environment like this:

```
conda create -n python37 python=3.7.0.
```

This will make an environment named "python37" with a 3.7.0 version of Python. Once you have Anaconda installed and an environment defined, the way that you access it depends a little bit on how your computer is set up. On my Mac, I placed the following commands in the `.cshrc` file in my home directory:

```
alias sconda ‘‘source /Applications/anaconda2/etc/profile.d/conda.csh"
alias gopy3 ‘‘conda activate python37"
```

If you use the bash shell, add these to your `.bashrc` file:

```
alias sconda = ‘‘source /Applications/anaconda2/etc/profile.d/conda.sh"
alias gopy3 = ‘‘conda activate python37"
```

then issues `sconda` and `gopy3` at the UNIX command line. You may also need to add the `anaconda` distribution to your path or python path.

Most often, Python is used interactively, or by writing code in a text file and running that text file at the UNIX prompt. To use Python interactively, you can either type `python` at the UNIX prompt, or you can use the interactive jupyter notebook tool by typing the following:

```
jupyter notebook
```

at the UNIX prompt. For the simple `python` option you should see, e.g.:

```
(python37) Jess@Tissiack:~> python
```

```
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

For the `jupyter` option, a web browser should be launched that displays your directory structure. Navigate to your Git directory of interest (by clicking on folders) and then, in the top right corner, click on `New` and then launch a Python kernel.

As an example of how to run a Python script directly from the command line, create a text file called, e.g., `myprogram.py` that contains the following lines of Python code:

```
def myprogram():
    print('hello world')

if __name__ == "__main__":
    myprogram()
```

and then run it from the UNIX command line as follows:

`python myprogram.py`.

Note that *any piece of code you submit as a homeowrk answer should be able to run from the UNIX command line, and the homework directory should include a README file indicating exactly how to run the code.*

Python is heavily documented online. Usually, if you need a command that performs a specific task, you'll find something on the web. Science is a collaborative endeavor. I have no problem with you using other people's Python modules in this class, including modules from other student's homework submission submitted *in weeks prior to the week of the current homework* (e.g., in week 2 of the class it is fine to use other student's functions and procedures from week 1). If you are completely unfamiliar with Python, then you should visit the Python tutorials at:

https://docs.python.org/3/tutorial/
https://swcarpentry.github.io/python-novice-inflammation/

and try sections 3-6 of the first tutorial and 1-8 and 12 of the second before the next class.

The PYTHON_PATH

You can import any modules to use in code if they are in your current directory or in a directory that is listed in your `PYTHON_PATH`. Look online for the UNIX commands to see how you can change your `PYTHON_PATH` to access other directories. For instance, in the event that you want to directly import each other's work from previous weeks.

Writing Python Packages

You can also write your own Python packages that contain associated functions and can be imported by others. To do this, create a directory called `my_package/`. In it, create a python script called `my_functions.py` containing all of the functions you would like to include. You could even split these into different files to help organize different categories of functions (e.g., to separate plotting functions from functions that calculate things). Then create a file called `__init__.py` with the contents:

```
from my_functions import *
```

If it is in your working directory or the path to the package is explicitly listed in your python path, you will be able to call your package from a python script with `import my_package` and have access to all of the functions in `my_functions.py`. As an alternative, you can use the `sys` package to add a path to your code in runtime rather than adding it permanently to your path. For an example, see my `tues` package in my week2/ directory in our shared repository.

Common Commands and Plotting

For plotting, we will use the matplotlib library. Try the first few examples at the matplotlib tutorial:

http://matplotlib.org/users/pyplot_tutorial.html

Note that if you choose to use the `jupyter` notebook option for interactive Python, then adding the command:

```
%matplotlib inline
```

at any point in your `jupyter` notebook before you import matplotlib will allow you to plot figures inline in the actual notebook.

Good Practice with Python and Git

1. *Write short pieces of code.* Longer code is less likely to be used by somebody else. For example, if you need to write code to; a) determine the area in a box, then b) randomly populate that box at a particular number density, then c) measure the clustering of points in that box; the *write three separate functions* and combine the three tasks using a single short procedure.

2. Give each piece of code *an informative name*, and separate the words in the name by underscores. So, for instance, if you have code to populate the area on a sphere at random, call that code `populate_sphere_at_random`. To distinguish file names from code, separate file names by dashes, as in `random-points-ra-0-to-23-hrs.txt`.

3. Give your files containing your Python tasks from each class meeting an informative name. When you have 15 instances of `wed_tasks.py`, you will wish you'd named them something like `wed_aug26_pyplotting.py` so you can navigate them more easily.

4. Give each piece of code *an informative header*. See my week2/ repo for examples.

5. Comment your code well and always use your initials when you write a comment. This will help to distinguish from comments in your classmate's codes that you might adopt later in the semester. When you adopt code from your classmates or the internet, it is good practice to reference where it came from in comments as well.

This approach not only makes the code more transparent to others, you'll also thank me in week 10 when you want to use the code you wrote in week 2 but you can't remember what it does!