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.

For Windows Users

Windows users will need to install a couple of extra packages to access the shell environments that are available on the Mac/Linux unix-based systems. I recommend using the combination of Windows Subsystem for Linux (WSL) and Visual Studio Code (VSCode). There are instructions here to help set up these environments to work together. Once you have these two working in tandem, you can proceed with the steps below and install the appropriate Linux version of the `conda` environment.

Getting Started Python

You will need to set up a local `anaconda` (a.k.a. `conda`) install of Python, which contains tools relevant for ASTR8080. Conda is a computing environment manager that we will use to install Python and all of the software packages that we will use with it. The difference between Miniconda and Anaconda is that the former includes only the base content that you need for conda and Python, whereas Anaconda also includes many pre-built packages. Miniconda should be sufficient for your needs.

Install Python

If you already have anaconda3 or miniconda3 installed, skip this step. If you haven't downloaded Anaconda before, find the installation instructions on the links page of the course website. The following instructions are tested for Mac; the Linux installation will be very similar.

1. Install Miniconda
   This PDF should have everything you need to install Python, but you can also view detailed installation instructions for Linux or Mac.

   Download the appropriate installer for your operating system from the download page.

   Run the installation. If you downloaded a Mac installer package you can run it and follow the prompts. For Linux (or Mac if you downloaded the .sh script installer) you can install from the command line. The -b flag runs the installation in batch mode with no PATH

modifications to your shell scripts (e.g. .bash_profile). It also assumes that you agree to the license agreement.

```
$ bash Miniconda3-latest-Linux-x86_64.sh -b
```

2. Check your installation:

   Open a *new* terminal window, and make sure your `$PATH` variable points to the conda installation. There are a number of reasonable answers, but the path should include anaconda3 or miniconda3.

   ```
   $ which python
   /Users/runnojc1/opt/anaconda3/bin/python
   ```

   You can also check that conda is working:

   ```
   $ conda list
   # packages in environment at /Users/runnojc1/opt/anaconda3:
   #
   # Name                    Version   Build Channel
   anaconda-client           1.11.2    py310hca03da5_0
   anaconda-navigator        2.4.0     py310hca03da5_0
   anaconda-project          0.11.1    py310hca03da5_0
   ```

3. If Step 2 did not work, it may be because Conda is not in your `$PATH`. Add Conda to your `.bash_profile` by adding the line:

   ```
   export PATH="$HOME/opt/miniconda3/bin:$PATH"
   ```

   Or if you use TCSH instead, you will need the following in your `.tcshrc` file:

   ```
   setenv PATH "$PATH:/Users/runnojc1/opt/miniconda3/bin/"
   ```

   Now try Step 2 again.

4. Once your Conda installation is sorted, you can clean up (once you are sure you will not need the installer anymore):

   ```
   $ rm Miniconda3-latest-Linux-x86_64.sh
   ```
   (be careful with "rm")

Create a Python Environment

Conda is an environment manager, which we will use specifically for managing Python. A Python environment is effectively a Python workspace that is created on top of the base Python installation. You can create multiple environments (e.g. each with unique packages installed) and they will not interfere with each other.

1. Create a new python environment for this class:

   ```
   $ conda create -n astr8080 python=3.9.9 numpy scipy matplotlib astropy
   ```
   When prompted, press `y` to proceed.

2. You can activate this environment from the terminal by typing:

   ```
   $ conda activate astr8080
   ```

3. When necessary, you can deactivate the class Python environment in the terminal by typing:

   ```
   $ conda deactivate
   ```

Using Python

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 type `python` 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.
>>>
```

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://www.arxiv-vanity.com/papers/1905.13189/#S5
https://docs.python.org/3/tutorial/
https://swcarpentry.github.io/python-novice-inflammation/

and try the first tutorial (except for the Python installation), sections 3-6 of the second tutorial, and 1-8 and 12 of the third 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.

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. Use `if __name__=="__main__"` in all your codes.

6. 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!