ASTR2013 Lab 1: Python as a Calculator

Dr Mike Ireland

July 21, 2019

1 Introduction

The scripting/programming language `python` is a flexible, modern high-level language. It is the basis for the most modern radio astronomy software called CASA and is intended to be the basis for much James-Webb Space Telescope software. We will be using it in OS X, but it works just fine in Windows too. It is certainly not the only popular language in astrophysics: `fortran` is still widely used amongst theorists, and the (expensive) language IDL is (unfortunately) also still used amongst observers and theorists alike. Many observational astronomers do not directly program in python themselves for routine tasks: they will use IRAF, for image analysis, for which people typically use python as a front end for (called PyRAF). The skills you will learn in this lab are therefore not just applicable to the content in ASTR2013, but is more generally applicable to astrophysics and throughout your science career.

We will use `python 3` in this course, although `python 2.7` is still used for some libraries in astrophysics.

2 Interactive Python

Python can be used in an interactive mode. There are three ways to do this - we'll start with the most basic. Start with the `Terminal` app (opened by typing terminal into the macOS search in the top right) and type `ipython --pylab`. This is the most interactive use of a python terminal, and you get colour for your prompt. You can use the `python` prompt as a calculator, for example:

```
x = 1
y = 2
x + y
```

Without any extensions, this use of python has some pitfalls. Variable declaration is implicit, so if you type `a=1`, `a` is an integer, but if you type `a=1.0`, then `a` is a floating point number. In python 3, a single division sign returns a floating point result, while a double division sign returns an integer division (i.e. without the remainder). Floating point numbers in python are by default *double-precision*. This means that each number takes up 8 bytes of memory and has about 15 decimal places stored. For example, try:

```
1 + 1e-15
1 + 1e-16
```

It is possible, but very unlikely, that you will run into this kind of precision error in python.

3 astropy

In recent years, many astrophysics utilities have become part of a package called `astropy`. This is still a work in progress, but has large community investment and is becoming a standard, so we will use this where we can. This week, we'll simply use it as a calculator. Start by loading physical units and constants into short-hand variables:

```
import astropy.units as u
import astropy.constants as c
```

One thing this allows us to do is easily convert between units. Try the following:

```
u.au.si
u.au.to(u.m)
```

You'll notice that a light year is not a standard astrophysical unit. You can get help on the units and constants via `help(u)` and `help(c)`. Press the space bar to move forward in help, `b` to go back and `q` to quit. We can convert the light year to parsec quite easily using:

```
(c.c*u.yr).to(u.pc)
```

If you miss the speed of light in this simple calculation, you won't get the wrong answer by a factor of 3×10^8 - you'll simply get a unit conversion error. Try:

```
(u.yr).to(u.pc)
```

4 Additional Libraries: numpy, scipy and matplotlib

Although there are a large number of built-in commands in python, the greatest strength of python for scientific computing comes from *libraries*, which are sets of programs that can do very useful things. The two main libraries we will be using are `numpy` and `astropy`. Plotting is best done with `matplotlib` but there are other interesting plotting libraries to try out if you like in your own time, e.g. `plotly`. Within `matplotlib` there is an excellent plotting module called `pyplot`. These libraries can be imported into a python script using:

```
import numpy as np
import matplotlib.pyplot as plt
plt.ion()
```

When you start with the '–pylab' option, these lines have been typed in for you when you start. To see an example of how these libraries work, try:

```
im = np.random.random([100,100])
plt.imshow(im, cmap='gray', interpolation='nearest')
```

See `http://matplotlib.org/users/pyplot_tutorial.html` and `http://www.numpy.org/` for plenty of interesting examples.

5 Other ways to run `python`

In addition to running python in a terminal, there are a range of integrated development environments, and also the capability to run experimental code interactively in a web browser. To see these two options, first try typing `spyder &` in a terminal. You'll notice that this is a graphical user interface that includes and ipython terminal as part of it. Additionally, in the main window you can write a series of python commands sequentially under each other, and execute this as a script. For example, try more complex mathematical lines like:

```
import numpy as np
#Try e to the power of i times pi.
z = np.exp(1j*np.pi)
print(z)
#How about trigonometry?
an_angle = np.degrees(np.arctan(1))
print("My angle is: {:5.3f}".format(an_angle))
print("Take the tan of the angle and we get:")
print(np.tan(an_angle))
print("Take the tan of the angle after converting to radians and we get:")
print(np.tan(np.radians(an_angle)))
```

Speak to your neighbours about the output - does it all make sense?

Finally, for the main tutorial today, we will use `jupyter`. Start this by typing `jupyter notebook &` into a terminal. The exercises for this tutorial are in a `jupyter` notebook on the course website. Download this to your ANU homedrive and open it from notebook. This has boxes you can write in. Follow the instructions and talk with your neighbours and the tutors about using the software and the problems down the bottom of the notebook.

6 Additional Python Information

To use python on your own computer and play around in your own time, please install the Anaconda python distribution. If you're low on space, you can use `miniconda` and then only install the packages you want with e.g. `conda install astropy`. Have a look at the following tutorials:
`http://docs.python.org/3/tutorial/introduction.html`
`http://docs.python.org/3/tutorial/controlflow.html`
(only section 4.1, 4.2, 4.3 and 4.6 - come back to the rest later)
`http://docs.python.org/3/tutorial/modules.html`

Make sure you type many of these commands in to the `python` interpreter to understand how they work. Please come back to the tutorials in future weeks whenever you have spare time - although you won't have to be a python expert to succeed in ASTR2013, some familiarity with basics will be essential, and in future years, scientific programming skills will be very useful.