SKA School 2014: Python + GNU/Linux Lab

For the computer lab exercises in the Introductory Radio Astronomy school, you will need to use python (since CASA, the package used to process radio data, is based around python) and be familiar with the command line (the shell).

We will first take a quick look at using python (actually ipython) in interactive mode, look at some of its features, and make some simple plots. Then we will write a simple program to read in some data, plot it, and fit a line to it. Along the way we will introduce some shell commands (e.g., for navigating the file system). You can also check the "Linux bootcamp" provided by Cynthia Chiang,

http://www.acru.ukzn.ac.za/~ska2014/materials/pythonlab/LinuxBootCamp.pdf

and my condensed "Python Introduction" notes:

http://www.acru.ukzn.ac.za/~ska2014/materials/pythonlab/PythonIntroduction.pdf

For those that are familiar with this already, feel free to skip ahead and try the exercise (or help your friends).

Starting python

First, open a terminal window (on Ubuntu, use the menu on the left hand side to locate the terminal program). Then, to start python in interactive mode, simply type:

```
python
```

This will start python in interactive mode. Here you can type individual python commands, e.g., type:

```
print "Hello world!"
```

You could also use python as your calculator, e.g.,

```
4 + 3
```

but beware numbers like this are treated as integers... e.g., try `4 / 3` and see what the answer is. To get the correct answer, `4.0/3` must be used. This is because python is a *dynamically typed*, interpreted language, and determines the type of the variable (int, float or string), based on "how it looks".

The standard python interpreter is useful, but there is a better, friendlier version called IPython. So, exit the python interpreter by typing `exit()` or press `Ctrl-D`, and then at the shell prompt type:

```
ipython
```

IPython allows you to use common shell commands, and also has tab-completion (which saves typing). For example, let's see what files there are in the current directory using the `ls` command:

```
ls
```

(`ls` is short for 'list'). If you want to see which directory you are currently in, you can use:

```
pwd
```

(which is short for 'print working directory'). Let's make a new folder ('directory' in UNIX parlance) in which we shall do some work:

mkdir pythonlab

We change to this directory using the `cd` command (`cd` = 'change directory'):

```
cd pythonlab
```

You can double-check where you are using `pwd` if you wish.

Modules and functions

You can do almost anything in python, and quickly[1]. Python *modules* group together functions (and sometimes classes, as in C++ for example – see the "Python Introduction" notes if you are interested) that do particular tasks. For example, the `math` module contains (unsurprisingly) math functions. To access these, use the `import` command:

```
import math
```

In IPython, you can use the `Tab` key to see which functions are available in a given module... try typing `math.` (note the `.` - these are used to separate namespaces in python) and pressing `Tab` – you should see a list of all the functions in the `math` module.

You can use the `help()` function to find out how to use any function in a module, e.g.,

```
help(math.sqrt)
```

tells you how to use the square root function. To call the function, try, e.g.,

```
math.sqrt(144)
```

Generally we won't be using the `math` module – instead we will use `numpy` (which works with arrays, and so is usually faster). To access the `numpy` package type

```
import numpy
```

Again, you can see all of the functions that `numpy` provides by typing `numpy.` and pressing the `Tab` key. You can use the `help()` function to read how they work.

Let's make a plot of a sine function. First though, we need an array of angles to feed into it. We can quickly create an array containing 100 elements linearly spaced between 0 and 2π using:

```
x=numpy.linspace(0, 2*numpy.pi, 100)
```

If you type `x` (and press `Enter`), you will see the values contained in the array.

To calculate the sine of the angle and place this in an array called `y`:

```
y=numpy.sin(x)
```

Numpy works quickly using functions which operate on arrays like this. You could instead use a `for` loop to calculate sin for each element of `x`, but this is slower (although that doesn't matter here) and takes more lines of code (and doesn't work so well in interactive mode like we are using here).

1 Python comes with a large standard library – to see everything that is included, go to https://docs.python.org/2/library/

To make a plot, we need to use a plotting package – let's use the `pylab` module (part of `matplotlib`):

```
import pylab
```

Just to be sure that `pylab` is in interactive mode and will display the plot as we make it, type

```
pylab.matplotlib.interactive(True)
```

To plot the sine function,

```
pylab.plot(x, y, 'k-')
```

The last part plots a black (`'k'`) solid line (`'-'`). You can experiment with other colours and line styles, e.g., blue dashed `'b--'`, or use `help(pylab.plot)` to see more.

If you want to plot the values of `x, y` on the curve, try

```
pylab.plot(x, y, 'ro')
```

Let's save our plot, just in case we want to look at it later. To do this, type

```
pylab.savefig("sin.png")
```

If you like, experiment with plotting some more functions. You can clear the plot axes by using

```
pylab.cla()
```

When you are ready, quit IPython by typing `exit()` or by pressing `Ctrl-D`.

Running python programs

The interactive mode is a good way to test out python functions, read the help, and perform quick calculations. Usually however, you will want to run programs (or scripts, there is no difference) to perform a particular job. Let's download some example programs from here:

http://www.acru.ukzn.ac.za/~ska2014/materials/pythonlab/examples.tar.gz

This file is a compressed archive in `.tar.gz` format (it's a small archive, so doesn't really need to be... this is just as an example). Find the directory to which you downloaded it (navigate there using the `cd` command) and unpack the archive using

```
tar -zxvf examples.tar.gz
```

After typing `ls` again, you should see that an `examples` directory has appeared. In there you will find some python programs and ancillary data (we will get to these shortly).

In the shell, most commands are actually programs themselves – including `python`, `ipython`, and the `tar` command you just used. You can get information on each by using the `man` (short for manual) command, e.g.,

```
man tar
```

will tell you about the many options of the `tar` command. The `-zxvf` here are four *switches* – you can read the manual page to figure out what these are (e.g., the `-z` option decompresses the archive, and `-x` extracts all files from the archive). Read Cynthia's "Linux Bootcamp" document for more details.

Now `cd` into the `examples` folder. The files ending in `.py` are python programs. You can view these in any text editor (try launching, e.g., `Kate`, or `Gedit`, from the menu on the left in Ubuntu). You run python programs by using the `python` command followed by the program name, e.g., try:

```
python fetchSDSSImages.py
```

In this case, this will print a message to the console showing how the program should be used: this script needs a few command line arguments: the name of a catalog, an output directory, and an image size in arcmin. So, you can run it with:

```
python fetchSDSSImages.py exampleCatalog.txt SDSS 6.0
```

If you look in the `exampleCatalog.txt` file (type `less exampleCatalog.txt`, and press the `Q` key when you are done, to quit), you'll see it includes the coordinates of two celestial objects – in this case, a group of galaxies and a galaxy cluster. This script uses python's `urllib` module to fetch Sloan Digital Sky Survey (SDSS; see http://www.sdss.org) `.jpg` images. If you look in the `SDSS` directory this script creates, you should find the images. You could do this using the graphical file manager in Ubuntu, or `cd` there and type

```
eog StephanQuintet.jpg &
```

to view the image (`eog` is the default image viewer program in Ubuntu). The `&` symbol here runs the program in the background so that you can continue using the terminal without closing `eog` – you can do this with any program.

After you are done, you probably want to get back to the examples folder... to go up a level, use

```
cd ..
```

(in Unix, `..` means the directory above the one you are in; a single `.` means the current directory).

Feel free to study the `fetchSDSSImages.py` script in a text editor. It has a simple structure, defining one function (using the `def` command) – see Figure 1 in the “Python Introduction” document for an explanation of what each part of the code does.

Indentation and program structure

Indentation, using tabs or spaces (pick one method to use, and stick to it), is a crucial feature of python syntax. Some people love it (in enforces good coding style), and some hate it (and perhaps prefer the C way of using `{}` to mark out code blocks).

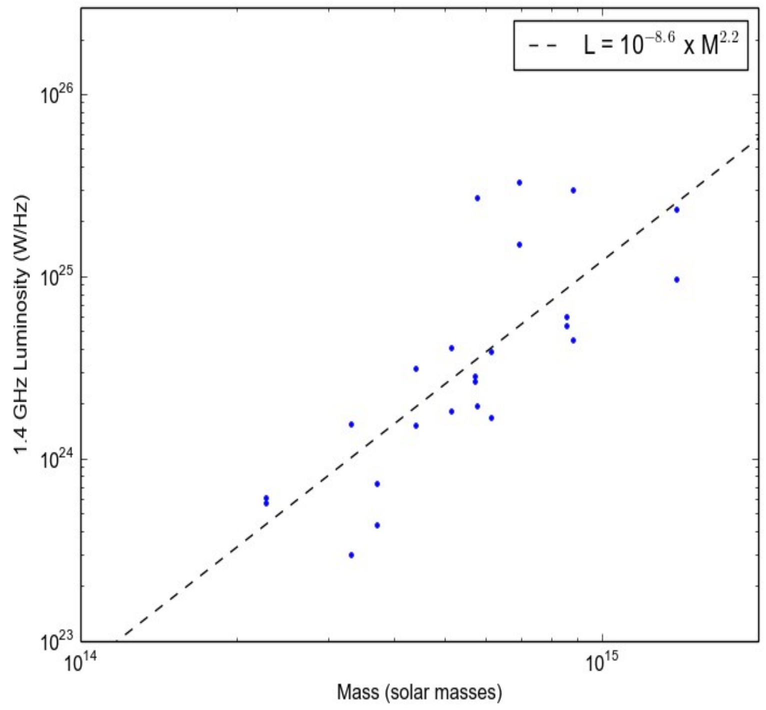
If you write a keyword that is followed by a colon (`:`), then everything under the colon that you want to be executed has to be indented to the same level. You can see this in the example programs – look at the function definition in `fetchSDSSImages.py` for example, or the `forLoopExample.py` script, or `classExample.py`.

If you want to get a feel for this, try moving the line `count=count+1` in the `forLoopExample.py` script between the loops (or indentation levels) and see what happens when you re-run the script. If you try hard enough, or accidentally un-indent the wrong line (directly below where the `for` loop begins), you can perhaps trigger the exception `IndentationError: expected an indented block`.

Since you will be indenting things a lot, it is a good idea to get used to your preferred text editor's keyboard shortcuts. In `Kate`, you can indent a whole block of code (selected/highlighted using the mouse) using `Ctrl+I`, and un-indent it using `Ctrl+Shift+I`. In `Gedit`, you can indent a block by pressing `Tab`, and un-indent using `Shift+Tab`. You may also want to set your text editor to indent using tabs or spaces, and/or make it display tab characters on the screen so that you can see where they are. In `Kate` you can do this using the menu `Settings → Configure Kate... → Editing → Indentation`. You'll also see here that it has a 'python' indentation mode. So, if you are typing a python script, `Kate` will recognise this and indent automatically after you use the `for` or `def` statements (followed by a colon), for example.

Writing a python program

Now would be a good time to write a program... try to write a script that will load in a text file containing an astronomical catalog (in this case, data on radio relics in galaxy clusters), and make a plot that looks like the below:



You can download the text file containing the catalog (in plain-text, tab-delimited format) from:

http://www.acru.ukzn.ac.za/~ska2014/materials/pythonlab/relics/RelicCatalog.csv

Depending on how comfortable you are, you can either attempt to do this all by yourself (perhaps look in `examples/numpyScipyMatplotlibExample.py` for ideas of how to go about this), or start from the template program which already has a routine that can read the catalog file included:

http://www.acru.ukzn.ac.za/~ska2014/materials/pythonlab/relics/plotRelicCatalog_template.py

Listed below are functions you will need to reproduce the plot (use the help function to find out about them):

```
pylab.plot, pylab.xlabel, pylab.ylabel, pylab.loglog, pylab.xlim, pylab.ylim,
pylab.legend, pylab.savefig, numpy.log10, numpy.linspace, scipy.stats.linregress
```

The script which made the above plot will be uploaded at the end of the session.

If you finish early and want to carry on, feel free to try plotting the other table columns, or explore some of the other `scipy`, `pylab` or `numpy` routines (e.g., try plotting histograms, look at some of the other functions in `scipy.stats`, or try to add a command-line interface to the program so that you can easily plot/fit any two columns in the catalog).

If you are interested in the data, they are taken from this paper by de Gasperin et al.:

http://arxiv.org/abs/1408.2677

This paper presents a review of the topic of radio relics and halos:

http://arxiv.org/abs/1205.1919