CS2043 - Unix Tools & Scripting
Lecture 13
Python II

Spring 2015 [1]

Instructor: Nicolas Savva

February 20, 2015

[1] based on slides by Hussam Abu-Libdeh, Bruno Abrahao and David Slater over the years

Announcements

- A3 due today (by 11:59PM)
- A4 coming out this weekend
- grades/feedback for prior assignments on CMS

Back to Python

There are a variety of addons for python we can use. They are called modules. We import them by typing

```
import modulename
```

Python Modules

There are modules for all sorts of activities. For example

- `import MySQLdb` work with SQL databases
- `import wxPython` graphical interface (there are lots of these)
- `import scipi` math stuff
- `import NumPy` more math stuff
- `import matplotlib` more plotting
- `import re` regular expressions

And many many many many more. Some come with python, others can be installed from repositories and others you need to download the source code.

Those pesky regular expressions

You can use regular expressions by importing re:

```
>>> import re
>>> str = 'cool beans'
>>> re.sub('[Cc]o*l','Hot',str)
'Hot beans'
>>> re.search('[Cc]o*l',str)
<_sre.SRE_Match object at 0x7ff33aa0>
>>> re.search('ARG',str)

>>>
```

When a search is successful an object is returned, when it fails nothing is returned. We can easily then use re.search easily inside of if statements.

searching for a string

```
#! /usr/bin/python
import re
infile = file('Frankenstein.txt','r')
lines = infile.readlines()
count=0
for k in lines:
    if re.search('monster',k):
        count=count+1
        print 'Monster Found!'

print 'There are', count, ' monsters in the book'
```

Using a Reg Expression More Than Once

If you are going to use a regular expression many times it is good to compile it:

```
>>>import re
>>> pattern = re.compile('[0-9]3[ -][0-9]3[ -][0-9]4')
>>> pattern.search('213-231 1921')
```

`<_sre.SRE_Match object at 0x7ff33b80>` Thats right, we can create **regular expression objects!**

more re

re contains some other functions as well

- re.findall(pattern,string[,flags]) - return all non-overlapping matches of patternin string as a list of strings
- re.split(pattern,string[, maxsplit=0]) - split string by the occurences of pattern
- and others

Functions

Of course we can use functions in Python. Here we use the `def` statement

```
def sum_diff(a,b):
    return a+b, a-b

x,y = sum_diff(8,3)
```

Unlike matlab, we do not need to give functions two targets if it returns two values.

Functions

- All parameters are passed by value
- Functions may return zero or more values
- Any type can be passed
- Python allows for default values and any number of arguments

```
def f(a,b=3,c='abc'):
    print a,b,c

>>> f(1)
2 1 abc
>>> f(2,'b')
2 b abc
>>> f(2,'b',4)

2 b 4
```

Functions

We can give a function optional parameters: `def g(a,b,*args):`

```
    print a,b,args
```

```
>>> g(2,3)
2 3 ()
>>> g(2,3,4)
2 3 (4,)
>>> g(2,3,4,5)
2 3 (4,5)
>>> g([1,2,3],(4,5),6,7)
[1, 2, 3] (4, 5) (6, 7)
```

Modules

Any python file is a module. To use one Python file in another you use the import statement

```
# mod.py
def f1():
    print 'f1 in mod.py'

def f2():
    print 'f2 in mod.py'
```

```
# main.py
import mod

mod.f1()

from mod import f2
f2()
```

Built-in Modules

Python has a relatively small syntax and relies on modules to extend its basic capabilities. A list of modules and documentation can be found at

http://www.python.org/doc/current/modindex.html

Some of the more commonly used modules include: `os`, `sys`, `string`, `re`, `math`, `scipy`, `numpy`, and `time`
Lets look at how to use sys to pass parameters to python scripts

sys

If we import sys, then the command line content is stored in the sys.argv list. For example:

```
#! /usr/bin/python
# getlist.py
import sys
print sys.argv,
```

Then if we type `./getlist.py file1 file2 file3` the script would print

```
getlist.py file1, file2, file3
```

Note: sys.argv contains the name of the file.

So...

Which means we can now do:

```
#! /usr/bin/python
import re
import sys
if len(sys.argv) != 3:
     sys.exit('Error: This script requires two arguments!')
infile = file(sys.argv[1],'r')
lines = infile.readlines()
count=0
for k in lines:
     if re.search(sys.argv[2],k):
          count=count+1


print 'The regular express', sys.argv[2], 'was found', count, 'times
in the file', sys.argv[1]
```

Variables = References

```
>>> x = 3
>>> y = x
>>> print id(x), id(y)
135278828 135278828
>>> x = 4
>>> y = 4
>>>print id(x), id(y)
135278816 135278816
>>>l1 = [0, 1, 2]
>>>l2 = [0, 1, 2]
>>>print id(l1), id(l2)
2146288652 2146287788
>>>l1 == l2
True
>>> l1 is l2

False
```

os module

The os module contains lots of useful things as well

- os.path.exists('path') - test if a path exists
- os.path.isfile('file') - test if its a file
- os.path.isdir('dir') - you get the gist?

and lots of other things including changing directories, deleting files and changing permissions.

Plotting graphs

`gnuplot`

- A command-line program to generate 2D and 3D plots of functions, data, and data fits.
- Publication quality graphics as well as educational purposes.
- Available on all major operating systems (GNU/Linux, Unix, Windows, Mac OS X, and others).

Gnuplot

- Run `gnuplot` with no command line arguments to get an interactive console.
- Run `gnuplot plot_script` to execute the plotting commands from the given script file.

Important gnuplot commands

- `help` : great help and documentation for the many gnuplot features and commands
- `plot` : plot data or a function

Plot the function `sin(x)`

```
plot sin(x)
```

- `set xrange / yrange` : set the range on the x or y axis.

```
set xrange [0:5]
```

- `set xlabel / ylabel` : set the text label on the x or y axis

```
set xlabel ‘‘Time (s)’’
```

- `set title` : set the graph title

```
set title ‘‘Awesome performance graph!’’
```

Plotting options

- gnuplot can read data files and automatically separate data from multiple columns

Plot data using the first and second column from a file

```
plot ''data.csv'' using 1:2
```

- Plot multiple data series

```
plot ''data.csv'' using 1:2, ''data.csv'' using 1:3
```

- Plot the series using a line

```
plot ''data.csv'' using 1:2 with lines
```

- Plot the series using boxes

```
plot ''data.csv'' using 1:2 with boxes
```

gnuplot references

There are many things that gnuplot can do. To learn more about a particular command or function, use the built in help (which is searchable).

To learn more about the plot command

```
help plot
```

- Many very useful gnuplot tips here: http://t16web.lanl.gov/Kawano/gnuplot/index-e.html
- gnuplot project homepage: http://www.gnuplot.info/

Next Time