CS2044 - Advanced Unix Tools & Scripting

Spring 2011

Hussam Abu-Libdeh

slides by David Slater

print

The `print` method prints to the screen. Data in quotes is printed exactly as typed, data not in quotes is interpreted first

```
>>> c = 4
>>> print 'a', 1+2, c, "c"
a 3 4 c
```

Working with files

Working with files in python is similar to in Perl:

```
infile=file('Frankenstein.htm','r')
```

So the format is `read(filename,mode)`

Reading a file

- filevar.read() - reads the entire file
- filevar.readline() - reads one line (can use over and over)
- filevar.readlines() - reads entire file creating a list of lines
- filevar.seek(offset,from what) seek offset bytes from either beginning (0), current position (1) or end of file (2). If you don't have a from what it seeks from the beginning of the file

```
#!  /usr/bin/python

infile=file('Text.txt','r')
for i in range(5):
    line = infile.readline()

    print line[:-1] # Remove Newlines
```

What happens if we just do `print line`?

Conditions

Python uses whitespace to mark blocks of code. A colon is placed at the end of lines when the next line needs to be indented:

```
if x > 0:
    print 'x > 0'
    some other command

if x > 0 and y < 0:
    print 'a'
elif y < -10 or x < 5:
    if not z < 10:
        print 'b'
    else:
        print 'c'
else:

    print 'd'
```

The lack of brackets is oddly comforting...

Looping

Python has for and while loops:

```
i = 0
while i < 10 and x > 0:
    x = input('enter a number')
    i = i+1
    total = total+x
print x, total

for i in [0 1 2 3 4]:

    print i
```

range and xrange

for loops can be run over lists, tuples and strings. To generate lists you can use

`range(start,end,increment)`

`xrange(start,end,increment)`

xrange has the advantage that it does not explicitly generate the list.

for loop examples

```
for i in range(4):
    print i,
```

```
(prints 0 1 2 3)
```

```
for i in range(0,10,2):
    print i,
```

```
(prints 0 2 4 6 8 )
```

```
for i in 'the quick':

    print i,
```

```
(prints t h e   q u i c k )
```

Writing Python Scripts

That is all well and good, but we want to write scripts:

```
#! /usr/bin/python

for i in range(4):
          print i,
```

Of course this depends on where you have python installed.

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
```

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.

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.

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.

Built-in Modules

Python as 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