CS2044 - Advanced Unix Tools & Scripting

Spring 2011

Hussam Abu-Libdeh

slides by David Slater

March 4, 2011

Python

An open source programming language conceived in the late 1980s. it is both compiled and interpreted (compilation step hidden).

Since Python is interpreted it is slower than C/C++ but is fast enough for most applications.

Why Python?

In Python we can do a variety of things

- Work with an interactive interpreter that makes it easy to experiment and try code
- Write object-oriented programming... but you do not need to use classes for everything like in java
- Work with built in modules for text processing and regular expressions
- Automatically convert variable types
- Work with scipy and numpy and do scientific computation like in matlab

But most of all...

Python is easy to read as white space is part of the syntax! Instead of enclosing blocks of code in brackets, we simply indent instead.

Python may be the easiest language to pick up and learn because although there may not be 10 ways (cough Perl) to do something, the way you expect to do it works.

Data types

- int : 3
- float: 2.5
- str: 'abc', "abc"
- list: [0,1,2], [0,1,'the']
- tuple: (0,1,2),(0,1,'the')
- dict: {'a': 1, 'Ohio': 'Columbus' 2: 'b'}

strings and tuples cannot be changed once they are created.

The interactive interpretor

Lets go play with the interpretor. To start the basic interpretor type python. If you have ipython installed, type ipython to get python with syntax highlighting, word completition and more!

Becareful with ints

Becareful with ints!

```
>> 1/2
0
>> 1./2
.5
```

Integer division truncates... :(

Working with strings

```
"hello"+"world" "helloworld" # concatenation

"hello"*3  "hellohellohello" # reptition

"hello"[0] "h"  # indexing

"hello"[-1]  "o" # (from end)

"hello"[1:4] "ello"  # slicing

len("hello") 5 # size

"hello" < "jello"  True # comparison

"e" in "hello"  True # search
```

Working with lists

```
somelist = [1, "abc", "5", 2, [3,5,"wewt"]]
somelist[0]
1
somelist[2]
'5'
somelist[4][2]
'wewt'
somelist[1:3]
['abc', '5']   <---- [a:b] starts at a and
     goes up to 1 before b
somelist[:2]
[1,'abc']
del(somelist[2]) <--- remove an element
```

More on Lists

- list.reverse() - reverses a list
- list.append(obj) - appends obj to a list
- list.sort() - sort a list
- list.index(obj) - finds the first occurence of a value in a list
- list.pop() - pop off last element
- help(list) - get documentation
- Everything is an object

Working with Dictionaries

- d = {'a': 1, 'b':2}
- d.keys() - returns list of keys
- d.values() - returns a list of values
- d.items() - returns a list of pairs of keys and values
- `d.has_key(arg)` - is arg a key in d?

```
d = {"duck" : 3 , "geese" : "are pretty"}
d["duck"]
3
d["duck"] = "i like ducks"
```

On Punctuation

- parentheses ( ): defining tuples, calling funtions, grouping expressions
  - t = ('a','b','c')
  - z = func(x,y)
  - z = 2.*(x+3) + 4./(y-1.)
- square brackets []: indexing and slicing (lists, dictionaries, arrays
  - element = lst[i]
  - val = dct['k']
  - y = a[i,j] (numpy array)
  - sublist = list[i:j]
- curly braces {}: dictionary creation
  - dct = {'a': 'apple', 'b': 'bear', 'c': 'cat'}

On variables

- no need to declare
- need to assign
- not strongly typed
- the variable _ in interactive mode stores the most recent output value (good for arithematic)
- **everything** is a "variable" (functions, classes, modules)

Assigment = reference!

When we do
x = y
we are making x reference the object y refers to. So

```
a = [1, 2, 3]
b = a
a.append(4)
print b
[1, 2, 3, 4]
```

Basic I/O

We can get input from the user by using the input and `raw_input` functions:

```
>>> x = input('enter a number:  ')
enter a number:  3+4
>>> x
7
>>> x,y = input('enter two number:  ')
enter two number:  3+4, 2*2
>>> x,y
(7,4)
>>>x,y= raw_input('enter two numbers:  ')
enter two numbers:  3+4, 2*2
>>> x,y
('3+4', '2*2')
```

`input` interprets then returns

`raw_input` returns the exact string typed

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
```

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.

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`?

Writing to a file

To write we just use the mode 'w'.

```
outfile = file('output.txt','w')
count=1
outfile.write('This is the first line of text\n')
count = count + 1
outfile.write('This is line number %d' % (count))
outfile.close()
```

Note: We can run python scripts from the interpretor by typing run script.py

Example Script

```
#! /usr/bin/python

infile=file('Frankenstein.txt','r')
outfile=file('wewt.txt','w')

linelist=infile.readlines()

for i in range(1,len(linelist),10):
outfile.write(linelist[i])
```

Writes to wewt.txt every 10th line of Frankenstein

Next Time In... THE UNIVERSE!!!

We will talk about python modules, which will allow us to pass arguments to our scripts, use regular expressions, and do enough math that we never need to think about buying matlab again!