Python – A Brief Introduction

This presentation was originally done by Austen Hayes as a part of CPSC 4240/6240 Spring 2014

Overview

- Why Python?
- Introduction to Python
- In-class example
- Take-home

(Slides available for later)

Why Python?

- More powerful than Bash
- Easy to learn
- Easy to read
- Python is used (almost) everywhere

Why Python?

- MUCH easier to read
- Count occurrences of word ‘out’ in file ‘function.py’:

Bash: tr " " "\n" < function.py | grep out | wc -w

Python:

```
#!/usr/bin/env python
import sys
data = sys.stdin.read()
print data.count(sys.argv[1])
#Run: ./occurences.py out < function.py
```

Introduction to

Python

Structure of Python

- Byte-code interpreted language, not compiled like C/C++
- Object-oriented
- Dynamically-typed

Writing Python

- Indentation defines blocks, not curly braces
  - Remember: indent the same way everywhere

```
if count==1:
    do_this()
else:
    do_that()
    a = 5+\
    7-3
```

- No semicolons to end line
- Backslash (\) at end of line to continue on to next

Running Python

- Interactive mode: Interpreter
  - “python” command to enter interactive mode
  - “python [Script name]” to run a script
- Executable script file
  - “#!/usr/bin/env python” at the top of file
  - ‘chmod +x script.py‘ to make the file executable
  - ./script.py to run

Script Basics

```
#!/usr/bin/env python

# this is a comment

print ‘my first print’
print “double quotes are the same”

Print ‘’’Look
What I
Can do with triple single quotes’’’

# Just don’t mix quote types like this: print ‘’’bad’
```

C #include statements: Python Style

- Example: Accessing array of command line arguments (argv), like you would in a C program
- Option 1:

```
import sys
if sys.argv…
```

- Option 2:

```
from sys import argv
if argv…
```

- Option 3:

```
from sys import *
if argv…
```

Data Types

- Dynamically-typed
- Strings
  - name = “bob”
  - http://docs.python.org/2/tutorial/introduction.html#strings
- Numbers
  - Integers, floating-point
  - X = 3, Y = 4.567
  - http://docs.python.org/2/tutorial/introduction.html#numbers
- Lists
  - lst = [1,2,3]
  - Not fixed like C or Java

http://docs.python.org/2/tutorial/datastructures.html

If Statements

```
If count==1:
     print 'count is 1!'

if condition == something:
   Function1()
   Function2()
elif:
   Function3()
else:
   Function4()
```

http://docs.python.org/2/tutorial/controlflow.html

Loops

```
listOfNums = [0,1,2,3,4,5,6]

for aNumber in listOfNums:
   Print (aNumber)

for aNumber in range(0,10):
     if aNumber <= 4:
          print '%d is not greater 4' % aNumber
```

http://docs.python.org/2/tutorial/controlflow.html#for-statements

Functions

```
def subNumbers(x,y):
   return x-y

print subNumbers(3,1)

def addNumbers(x,y):
   Pass # do nothing
```

http://docs.python.org/2/tutorial/controlflow.html#defining-functions

Python Magic: Operations On Data Types

List Operations

```
A = [1,2,3,4]
B = [5,6,7,8]
A.append(0)
A.sort()
B.remove(8)
C = A+B # C=[0,1,2,3,4,5,6,7]
print C # [0,1,2,3,4,5,6,7]
```

List Operations

```
B=[0,1,2]
if 2 in B:
    print “it’s there!”

names = ‘bob larry joe’
NamesList = names.split(‘ ‘)
print NamesList  # [‘bob’,’larry’,’joe’]
```

String Methods

```
someString = “This Is A String”
print someString[2,5] # ‘Is ‘
print someString[2:-1] # ‘Is Is A String’
print someString.isalpha() # True
print someString.lower() # ‘this is a string’
Print someString.find(‘Is’) # 5
Print someString.replace(’ ’,’$’) # This$Is$A$String
```

Many more:
http://docs.python.org/2/library/stdtypes.html

String Substitutions

```
a=2
b=4
Name='bob'
print 'I have %d laptops and %d apples' % (a,b)

# Both print 'Bob is a customer':
print '%s is a customer' % Name
print Name, 'is a customer'
```

Reading a File

```
# open file for (r)eading
file f = open('filename', 'r')

print f.read()      # print the whole file
print f.readline() # print one line

for line in f:      # print every line
     print line

f.close()
```

Writing to a File

```
# open file for (w)riting
file f = open('filename', 'w')

f.write('a line in my file.\n')

f.close()
```

Sys Admin Python

Basic tasks of Python Bash scripts

1. Run linux commands as in Bash
2. Capture data and/or return values of commands
3. Process data/return values
   - MUCH more powerful and readable processing in Python

Common packages to use

- os
  - OS environment-specific information and operations
  - http://docs.python.org/2/library/os.html
- os.path
  - Common pathname manipulations
  - http://docs.python.org/2/library/os.path.html
- sys
  - System-specific parameters and functions
  - http://docs.python.org/2/library/sys.html
- subprocess
  - Spawn processes, execute commands, pipe information, etc.
  - http://docs.python.org/2/library/subprocess.html

Some built-in commands

- **`os.environ`** (os.environ['HOME'],['PWD']): information on system and running operations
- **`os.chdir()`:** change the directory
- **`os.getpid()`**: get the process ID of the running program

- **`os.isfile()`, `os.isdir()`**: does the file/directory exist?
- **`os.join(path1,path2)`:** join the path in a smart way

- Many others…

Example: List files in current folder

```
#!/usr/bin/env python
import subprocess

Subprocess.call(“ls -l”, shell=True)
```

Example: Count occurrences of word

Bash command: tr " " "\n" < function.py | grep out | wc -w

```
#!/usr/bin/env python
import sys

word = sys.argv[1]
theFile = sys.stdin.read() # read file passed in
print theFile.count(word)

#Run:  ./occurences.py out < function.py
```

Example: Create list of files in folder

```
#!/usr/bin/env python
import subprocess

# function to run a command  (cmd)
def runBash(cmd):
     splitCmd = cmd.split(' ')
     output = subprocess.check_output(splitCmd)
     return output

files = runBash("ls")
fileList = files.split()
```

In Class Assignment

1. Run the command cat /proc/cpuinfo and observe the output
2. Create a python script file (don't forget: chmod +x [your file name])
3. Add the following code and execute the script.
4. What does each line of the script do, and what is the output?

```
#!/usr/bin/env python
import subprocess
cmd = 'cat /proc/cpuinfo'
output=subprocess.check_output(cmd.split())
output=output.replace(':','')
fields = output.split()
print fields
```

Take home python task

- The **`df`** command gives information on disk space usage. **`df -h`** puts the information in a human-readable format.
- Create a python script that will run the command and check the capacity of each filesystem. Alert the user of all filesystems that are at 90% capacity or more by printing the name and percent used.
- Example output:

```
Warning:
        /dev/sda1 is at 93% capacity (17.67G/19G)
```