IT Scripting and Automation

Python – Quick Revision
&
Python in the Bash environment

Lecturer: Art Ó Coileáin

Why Python?

- Scripting languages are often used to do repetitive, tedious tasks. Python is one of these languages.
- Python is a valuable tool, specially because it enables you to get your work done efficiently.
- Python is easy to learn. If you are a sysadmin, your work can pile up faster than you can clear it. With Python, you can start writing useful scripts quickly.
- Python is an excellent programming language, it lets you start simply, it also allows you to perform tasks that are as complex as you can imagine.
- Another advantage of Python is its simple support for object-oriented programming

Why Python?

- Python is consider to have an excellent readability.
  - Python relies on whitespace to determine where code blocks begin and end.
- Python is used to do some analysis of genomic sequences, multithreaded web server, and heavy duty statistical analysis.
- Python can be used to read through a logfile line by line and extract information.

E.G.:

- Python can parse through a logfile, extract every piece of information it contains, compare usage from each IP address in this logfile (which can be stored in a relational database) over the past three months, and then store the results to a relational database.

Bash vs Python

```
#!/bin/bash
for a in 1 2
do
    for b in a b
        echo "$a $b"
    done
done
```

```
#!/usr/bin/env python3
for a in [1 , 2]:
        for b in ['a' ,'b']:
                print (a,b)
```

Python – Basic Loops

For Loops:

```
for count in range(3):          #Loop printing from number 0 to 2
        print(count)

for count in range(2,5):        #Loop printing from number 2 to 4
        print(count)

for var in ["a”,”b”,”c”]:       #Loop printing each element in the list
        print (var)
```

Python – Basic Loops

- **For Loops**:

```
for var in [3,5,9]:          #Loop printing each element in the list
        print (var)

for countdown in range(5,1-1):          #Loop Counting down
        print(countdown)
```

- **While Loops**:

```
counter=0                          #Loop counting from 0 to 10
while counter <= 10:
        print counter
        counter += 1
```

Bash vs Python

```
#!/bin/bash
If [ -d “/tmp” ]
then
  echo “/tmp is a directory”
else
  echo “/tmp is not a directory”
fi
```

```
#!/usr/bin/env python3

import os

if os.path.isdir(“/tmp”):
   print (“/tmp is a directory”)
else:
   print (“/tmp is not a directory”)
```

Python – if Statements

- Simple Conditions

```
if temp > 22:
        print("Hot day, wear shorts.")
else:
        print ("Wear long pants.")
```

Python – if Statements

- Arithmetic comparisons:

Meaning	Math Symbol	Python Symbols
Less than	<	<
Greater than	>	>
Less than or equal	≤	<=
Greater than or equal	≥	>=
Equals	=	==
Not equal	≠	!=

Python – if Statements in a Function

```
def letterGrade(score):
        if score >= 90:
                letter = 'A’
        elif score >= 80:
                letter = 'B’
        elif score >= 70:
                letter = 'C’
        elif score >= 60:
                letter = 'D’
        else: letter = 'F’
        return letter
```

Obtaining Python

- https://www.python.org/
- http://ipython.org/
- https://python.ie/

Python subprocess

- To print a message
  - print "Welcome to Python!"
- We can execute a Bash command with Python
  - In Bash, we type: **$ ls -l**
  - In Python, we type:
    - **Import subprocess**
    - **Subprocess.call([“ls”,”-l”])**
- One of the powerful features of Python is its ability to import modules or other files that contain code and reuse them in a new program (import statement).

Syntax:

subprocess.call(["some_command", "some_argument", "another_argument_or_path"])

Python subprocess : pyls.py

```
#!/usr/bin/env python3
#Python wrapper for the ls command
import subprocess
subprocess.call(["ls","-l"])
```

- It must be executable (**chmod u+x pyls.py**)
- To execute the script: **./pyls.py**

Python subprocess: pysysinfo.py

```
#!/usr/bin/env python3
#A System Information Gathering Script
import subprocess

#Command 1 -> uname -a
uname = “uname”
uname_arg = “-a”
print ("Gathering system information with %s command:\n" % uname)
subprocess.call([uname, uname_arg])

#Command 2 -> df -h
diskspace = "df"
diskspace_arg = "-h"
print ("Gathering diskspace information %s command:\n" % diskspace)
subprocess.call([diskspace, diskspace_arg])
```

Functions in Python

The next step to automating our code execution is to create functions.

- A function allows you to create blocks of statements that get called in groups that live inside of the function.
- You can have multiple functions that group statements together in a script, and then that group of statements can be called to run a miniprogram at the proper time in your script.
- The 'Whitespace', in Python, a uniform level of indentation must be maintained in nesting code.
- We will use iPython shell, it is a bit like Bash shell and can execute commands such as ls, cd and pwd.

Functions in Python

```
>>> def pyfunc():
        ... print ("Hello function")
        ...
>>> pyfunc()
Hello function

>>> for i in range(2):
        ... pyfunc()
        ...
        ...
Hello function
Hello function
```

Functions in Python (pysinfo_func.py)

```
#!/usr/bin/env python3
# This is a comment
#A System Information Gathering Script

import subprocess

#Command 1
def uname_func():
        uname = "uname"
        uname_arg = "-a"
        print ("Gathering system information with %s command:\n" % uname)
        subprocess.call([uname, uname_arg])
```

Functions in Python (pysinfo_func.py) (2)

```
#Command 2
def disk_func():
        diskspace = "df"
        diskspace_arg = "-h"
        print ("Gathering diskspace information %s command:\n" % diskspace)
        subprocess.call([diskspace, diskspace_arg])


#Main function that call other functions
def main():
        uname_func()
        disk_func()


main()
```