2015-05-13: Introduction to Python

Introduction:

Why Python?

We are teaching python because it is an excellent language for you to learn. It is *RELATIVELY* easy to grasp the concepts and flow of the language.

It helps that it's popular.

It also helps that it is not compiled.

What is Python?

Python is an interpreted language, it is read, line by line, by the python executable, and each line is executed in its proper order.

This means that is has more flexibility than compiled languages. It also means that errors are not ALWAYS found when you run it.

What can you Do with Python?

Whatever you want (mostly).

So, what are *WE* going to do with it?

How Does This Webpage Work?

We will have three types of text. Material that is part of a discussion, which will be in plain text, like this:

Hello.

See what I did there?

We will have code that you can copy and paste into a file:

```
#!/usr/bin/perl
for my $i in (@temp){
  print "$i
}
```

```
#!/usr/bin/python

str = raw_input("Enter your input: ");
print "Received input is : ", str
```

Expected output will be displayed in "info" boxes:

```
Recieved input is : something you typed
```

We will also have commands to copy and paste into a command line. They will look like this:

> *ls -latrh*

Easy, right?

So, what are we going to do?

Logging in to HPCC

For Windows Users:

1) Install MobaXterm:

http://mobaxterm.mobatek.net/MobaXterm_v7.4.zip

2) Open MobaXterm

3) Connect to hpcc:

> *ssh -X yourusername@hpcc.msu.edu*

For Mac Users:

1) Install Xquartz:

2) Open a terminal

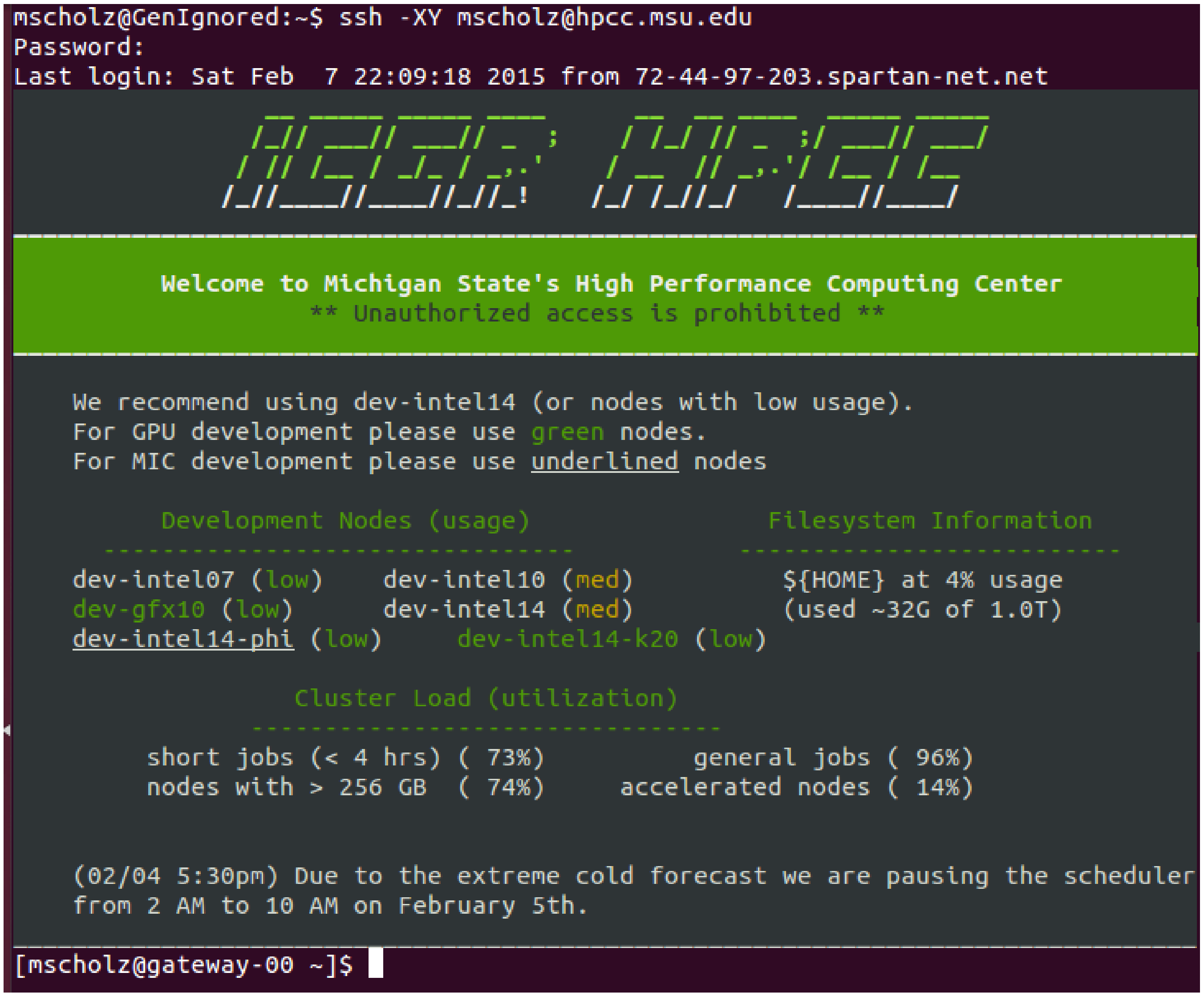
3) Connect to HPCC:

> *ssh -XY yourusername@hpcc.msu.edu*

For Linux Users:

> *ssh -XY yourusername@hpcc.msu.edu*

You should see something similar to this:



```
mscholz@GenIgnored:~$ ssh -XY mscholz@hpcc.msu.edu
Password:
Last login: Sat Feb  7 22:09:18 2015 from 72-44-97-203.spartan-net.net
                 __ _____ _____ ____      __  __ ____  _____ _____
                /_// ___// ___// _  ;    / /_/ // _  ;/ ___// ___/
               / // /__ / _/_ / _,.'    / __  // _,.'/ /__ / /__
              /_//____//____//_//_!    /_/ /_//_/    /____//____/

--------------------------------------------------------------------------------

          Welcome to Michigan State's High Performance Computing Center
                    ** Unauthorized access is prohibited **

--------------------------------------------------------------------------------

    We recommend using dev-intel14 (or nodes with low usage).
    For GPU development please use green nodes.
    For MIC development please use underlined nodes

          Development Nodes (usage)                 Filesystem Information
      --------------------------------          --------------------------
    dev-intel07 (low)     dev-intel10 (med)          ${HOME} at 4% usage
    dev-gfx10 (low)       dev-intel14 (med)          (used ~32G of 1.0T)
    dev-intel14-phi (low)      dev-intel14-k20 (low)

                   Cluster Load (utilization)
                --------------------------------
         short jobs (< 4 hrs) ( 73%)          general jobs ( 96%)
         nodes with > 256 GB  ( 74%)     accelerated nodes ( 14%)


    (02/04 5:30pm) Due to the extreme cold forecast we are pausing the scheduler
    from 2 AM to 10 AM on February 5th.

--------------------------------------------------------------------------------
[mscholz@gateway-00 ~]$
```

This is the "gateway" DO NOT DO ANY WORK ON THIS MACHINE.

Well except for this:

> *cd*
>
> *module load powertools*
>
> *getexample introToPython*
>
> *cd introToPython*
>
> *./login.sh*

What is happening is that you are being logged in to a small portion of the cluster, where you can do work, and not get in eachother's way. When it works you should see this:

Once you are logged in, we can start doing "real" work

Let's Play With Python:

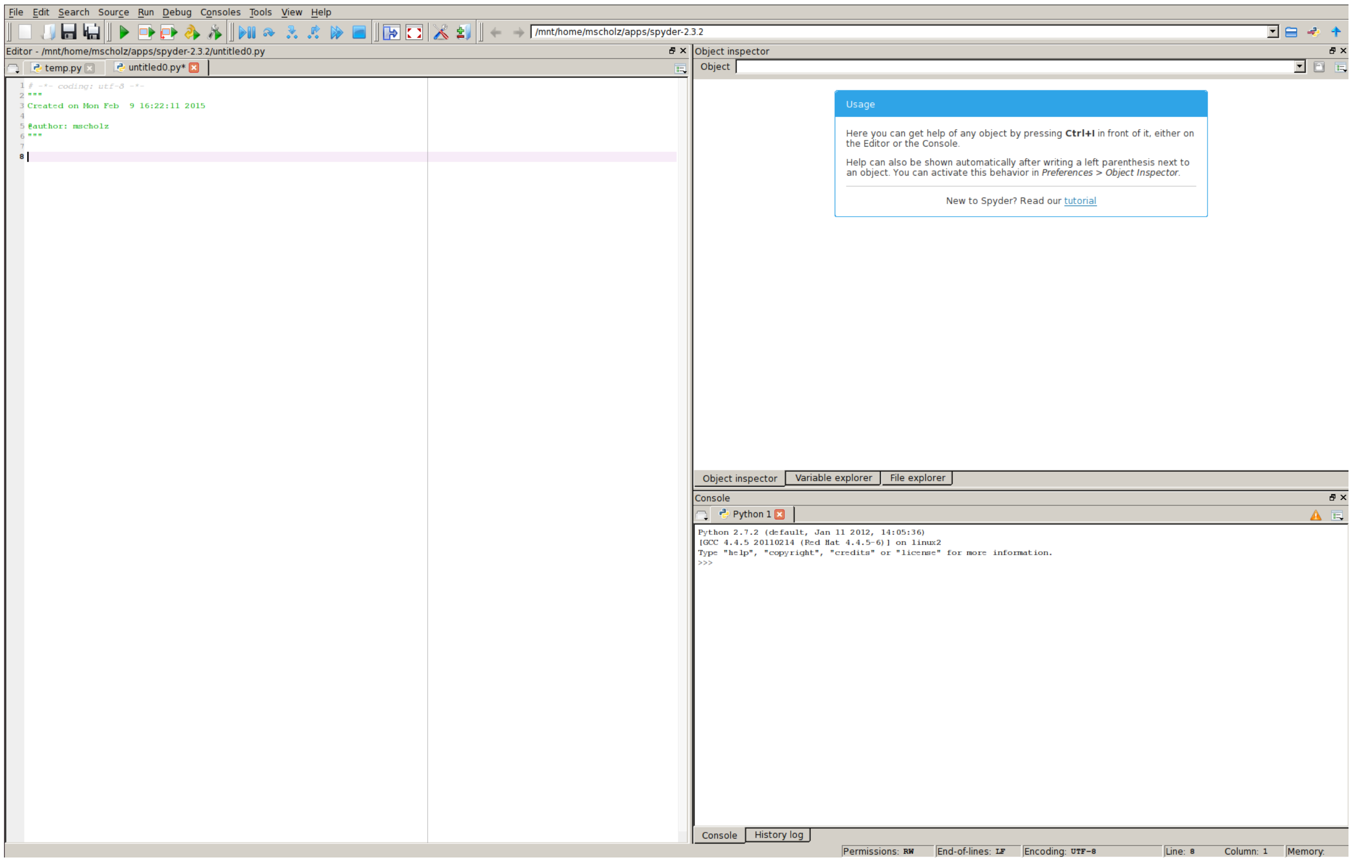
First, we are going to load the necessary modules to be able to DO anything with python (Most of these will be used LATER):

> *module load spyder scipy numpy matplotlib*

now, let's open up a python only Interactive Development Environment (spyder)

> *spyder &*

You SHOULD get a new window that looks like this:



```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb  9 16:22:11 2015

@author: mscholz
"""

```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Object Inspector*.

New to Spyder? Read our tutorial

```
Python 2.7.2 (default, Jan 11 2012, 14:05:36)
[GCC 4.4.5 20110214 (Red Hat 4.4.5-6)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

On the left is an open area for your code (temporary file).

On the right is an "object inspector", and a "console"

Object inspector will show you objects as you create them in the console, console will run your code.

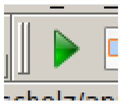
So, let's try it out:

Hello World:

Copy this into the left hand box:

```
print "Hello World"
```

and press the "play" button:



In the lower Right hand box we should have:

Hello World

Well, that was easy.

Everybody can go home.... RIght?

What's that, you want to do more?

OK:

Python Variables

Variables are created when assigned value

Variables are much similar to most languages, except you do not explicitly declare type (unless you really really want to)

See here:

(ignore the parts you don't understand)

```
#string
city = "taipei"
#Numbers
 #integer or long
population = 2000000
 #float
avg_temp = 67.5
 #Compled
pi=3.14j
#lists
cities = ['taipei','hong kong', 'New York']
cities.append ('lansing')
#tupples
planets = ('earth','mars')
#cannot add elements to tupples
```

Typing of variables

Variables are automatically typed, if you don't tell Python differently:

```
variable = "Hello World"
print variable
```

Python is given a variable name (variable), and told to make it equal to "Hello World"

So, now it treats variable as a string. Python can do a lot of things to strings. More on that later.

What CAN'T Python do to strings? (Math)

```
variable = "Hello World"
variable += 3
print variable
```

```
Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

  File "/opt/software/spyder/2.3.2--GCC-4.4.5/lib/python2.7/site-packages/spyderlib/widgets/externalshell/sitecustomize.py", line 601, in runfile

    execfile(filename, namespace)

  File "/opt/software/spyder/2.3.2--GCC-4.4.5/lib/python2.7/site-packages/spyderlib/widgets/externalshell/sitecustomize.py", line 73, in execfile

    builtins.execfile(filename, *where)

  File "/mnt/home/mscholz/apps/spyder-2.3.2/untitled1.py", line 9, in <module>

    variable += 3

TypeError: cannot concatenate 'str' and 'int' objects
```

So, it won't do it. But this:

```
variable = "hello world"
print variable
variable = 1
variable += variable
print variable
```

gives:

```
>>> runfile('/mnt/home/mscholz/apps/spyder-2.3.2/untitled1.py', wdir=r'/mnt/home/mscholz/apps/spyder-2.3.2')

hello world

4
```

So what happened?

Dynamic Variable Typing.

Types of variables cannot be combined, but you CAN recast variables, or have variables retype. I wouldn't recommend it most of the time though.

So, what can we do next?

Input

Information by command line

As with most programs, any information following the command is parsed by the program as arguments

Let's make a program. Paste this into a new file in your main spyder window:

```
#!/usr/bin/python

import sys

for argument in sys.argv:
        print argument
```

Now select save as, and navigate to ~/introToPython

save the file as: printArgs.py

Go to the terminal and:

> *python printArgs.py one two three*

```
python printArgs.py one two three
printArgs.py
one
two
three
```

Or, we can get information WHILE the program is running

Reading from STDIN

This starts to show some of the specifics of how python does things. Bear with me, all will be made clear(er).

To read, we have to OPEN the stdin handle. In other words, we need to read FROM the terminal, and tell Python we want to.

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

print "what is your name?"
name = sys.stdin.readline()

print "your name is: ",name
```

Reading from a file:

```
#!/usr/bin/python

filehandle = open("foxRabbit.csv", "r")
```

Now the fun begins.

15 minute break

Interacting with the System

To do anything useful, you need to be able to do some basic things:

change directories

open files

run commands (maybe)

To do this, you will need to use a python library. (os).

To use the things in that library, you need to understand how to call functions from libraries.

So, this example:

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


print os.getcwd()
os.chdir("../introToPython")
f = open ("rabbitFox.csv","r")
rpop = []
fpop = []
for line in f:
#    print line
    (rabbit,fox) = line.split(' ')
    rpop.append(rabbit)
    fpop.append(fox)

print rpop
print os.getcwd()
subprocess.call("ls")
```

does what?

well,

1. We import libraries of functions (os and subprocess)
2. Then we print the result of os.getcwd()

   > */mnt/home/mscholz/.spyder2*

3. Then we use subprocess.chdir to change the working directory, to ../introToPython (where the file is)
4. then we open the file as a FILEHANDLE (that you can perform functions on)
5. Then we declare two lists
6. THEN we read the whole file in a for loop (this loop generates a new value for the variable 'line' for every line in filehandle f it reads
7. Then we do fancy functions (split) on that string (line.split(' ') returns a list of all values in variable line separated by ' ') and assign it to a temporary set of values.
8. THEN we append each of those values to the lists
9. THEN we print those lists

Now, your turn.

Assignment:

The file "rabbitFox.csv" is a two column file with rabbit and fox populations over time.

1) open the file

2) read each line

3) Find the difference between rabbit and fox populations for each generation

4) save it to a list

5) print that list

(for bonus points, request a number from STDIN and either multiply or divide the value by it before you move on)

Functions and Control

Conditional Statements

For tests, or for general statements, you DON'T NEED PARENTHESES:

But, you also don't need curly brackets, blocks of code (denoted by the previous line having a :) are INDENTED

```
if x== 4:
        print x
        x=2
elsif x < 0:
        print "X was negative"
        x=0
else:
        print x
        x=1
```

Loops

Loops function the same way (we've already seen one):

```
filehandle = open ("rabbitFox.csv","r")
for line in filehandle:
        print line
```

But wait, there's more.

Who recognizes this:

```
for (i = 1; i < 10; i++){

}
```

???

Python has a function called RANGE to deal with this:

```
for number in range(10):
        print number

for number in range (10,100,10):
        print number
```

so, this steps through the ranges.

```
>>> runfile('/mnt/home/mscholz/apps/spyder-2.3.2/untitled1.py', wdir=r'/mnt/home/mscholz/apps/spyder-2.3.2')
0
1
2
3
4
5
6
7
8
9
10
20
30
40
50
60
70
80
90
```

OK, so now to the next challenge:

Assignment:

Write a program that runs through ONE for loop, and prints every even number between 1 and 25

Functions

as with most languages, sometimes, you want to do the same thing in multiple places, so you want to write a function, and call it in both places. Functions are just blocks of code, that are DEFINED.

```
def squareIt(input):
        output = input**2
        print output
        return output

value = squareIt(2)
value2 = squareIt(value)

print value2
```

```
>>> runfile('/mnt/home/mscholz/apps/spyder-2.3.2/untitled1.py', wdir=r'/mnt/home/mscholz/apps/spyder-2.3.2')

4

16

16
```

So, now we can play a little:

in Spyder, open the files

~/introToPython/generate_fox_data.py

~/introToPython/graph_fox_data.py

Discussion:

Generate Fox Data:

(this example is borrowed)

1) functions everywhere

2) uses the random() function to determine if individuals survive, starve, or get eaten

3) steps through 200 generations and saves

graph_fox_data.py:

1) loadtext() is WAY easier to get data from tables than the for loop from before

2) Graphing is done using the matplotlib library

Final Assignment:

generate_fox_data.py:

1) Change the number of generations from a number to a variable.

2) Allow that variable to be set by using stdin or a command line argument

3) make the probablities variables

4) Make the save file a variable, so that you can see multiple result sets

5) Run the program

graph_fox_data.py

1) graph your results

2) what can you change?

Resources:

Software Carpentry:

http://software-carpentry.org/lessons.html

http://software-carpentry.org/v4/python/index.html

http://software-carpentry.org/v5/novice/python/index.html

Other Python resources:

https://docs.python.org/2/tutorial/

http://www.learnpython.org/

Numpy/Scipy/matplotlib:

http://people.duke.edu/~ccc14/pcfb/numerics.html

https://scipy-lectures.github.io/intro/matplotlib/matplotlib.html

And of course:

www.google.com