Day 7: NumPy

Python – General Overview

“*High Level*” programming language – what exactly does this mean?

- **Consists of bits of code that are *precompiled*. Programmer is responsible for arranging the small pieces into program**

- **Distinct from *compiled* languages. Python (and BASH) are “*interpreted*” languages.**
- **Abstracts many more difficult aspects away from the user**
- **As a result, much faster to write a program in a high level language. But the price is *overhead***
- **Supports more complex structures and functions, makes things nice and “easy”**

- ***Object Oriented Programming!***

How is Python different vs the BASH shell you have used?

- **BASH is built for interacting with an operating system. It is not specifically designed to process major data and make calculations**
- ***Moving, copying files, viewing files and folders, rapidly editing***
- ***Creating helpful system utilities, streamlining workflow***

- **Python is built to process and describe data. It is *extensible* and contains many *language features***
- ***Creating applications, processing data, making complex calculations, making plots***

When to use one vs the other is a matter of experience and the specific task at hand

The Python Interpreter

```
brandon@brandon-P34 ~ $ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2*2
4
>>>
```

At any time in BASH

>>> Indicates PYTHON

The Interpreter **is a fully functioning python shell – you can theoretically write an entire program here. But more useful for basic** ***check and test***

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> def print_hello():
...     print("hello world!")
...     return
...
>>> print_hello()
hello world!
```

There are ***variables***, ***functions***, **and** ***data structures*** **within PYTHON. They are all what are considered** ***objects***

The Python Script

Longer, more complex python programs should be in text files

```
brandon@brandon-P34 ~/Desktop/demo $ vi myscript.py
```

Here is how you would accomplish the same exact goal:

```
# This script is meant to print the word
# hello using a function

def print_hello():
        print("hello")
        return

print_hello()
```

Call the script through the python program. Bash knows about Python if it is installed!

```
brandon@brandon-P34 ~/Desktop/demo $ python myscript.py
hello
brandon@brandon-P34 ~/Desktop/demo $
```

A brief refresher on Lists

A *list* is a structure that holds data sequentially in many programming languages

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> myArray = [17, 24, 62, 13, 3, 18, 34, 95, 45, 71]
>>> myArray
[17, 24, 62, 13, 3, 18, 34, 95, 45, 71]
>>> myArray[4]
3
>>> len(myArray)
10
>>> myArray[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

***Arrays* support *indexing*, which is a scheme for organizing and retrieving *elements, Or items* from the list**

Remember, lists are objects too! And the items of a list are all also objects

A brief refresher on Lists

Python's default list object does not really perform the functions that you as a mathematician or scientist might be interested in

```
>>> myArray*2
[17, 24, 62, 13, 3, 18, 34, 95, 45, 71, 17, 24, 62, 13, 3, 18, 34, 95, 45, 71]
>>> myArray + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
>>>
```

You can define the operations you want by writing your own functions

```
>>> def multiply_array(array, constant):
...     for i in range(0, len(array)):
...             array[i] = array[i] * constant
...
...     return
...
>>> myArray
[17, 24, 62, 13, 3, 18, 34, 95, 45, 71]
>>> multiply_array(myArray, 2)
>>> myArray
[34, 48, 124, 26, 6, 36, 68, 190, 90, 142]
```

NumPy

Thankfully there is a *Library* that is widely used specifically designed to tackle matrix and list operations in an intuitive manner – this is NumPy

Third party *Libraries* do not come built into Python, you must download and install them manually. You may then access the functions using an *Import*

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> old_list = [0,1,2,3,4,5,6,7,8,9]
>>> type(old_list)
<type 'list'>
>>> import numpy as np
>>> new_list = np.arange(10)
>>> new_list
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> type(new_list)
<type 'numpy.ndarray'>
>>>
```

Object.method

NumPy arrays simply objects written in standard python – behind the scenes they are no more then lines of simple python code written to produce the functionality that you see here

Common NumPy Routines

Here are some very basic NumPy routines

```
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> list = [1,2,3]
>>> list
[1, 2, 3]
>>> import numpy as np
>>> nparr = np.array(list)
>>> nparr
array([1, 2, 3])
>>> nparr*2
array([2, 4, 6])
>>> nparr + 1
array([2, 3, 4])
>>> np.append(nparr, 4)
array([1, 2, 3, 4])
>>> nparr = np.append(nparr,4)
>>> nparr
array([1, 2, 3, 4])
>>> nparr.reshape((2,2))
array([[1, 2],
       [3, 4]])
>>>
```

A Basic List

Convert to ndarray

Multiply each element by 2

Add 1 to each element

Append 4 to the end of the array

Convert 1x4 into 2x2

More Basic NumPy Routines

There are many NumPy commands for quickly creating and manipulating matrices as well as multidimensional support and intuitive operations

```
>>> matrix
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> ident3 = np.eye(3)*3
>>> ident3
array([[ 3.,  0.,  0.],
       [ 0.,  3.,  0.],
       [ 0.,  0.,  3.]])
>>> ident3 * matrix
array([[  0.,   0.,   0.],
       [  0.,  12.,   0.],
       [  0.,   0.,  24.]])
```

A full course can be taken on NumPy if desired – but today we will simply cover Some basic exercises designed to get you used to simple NumPy routines