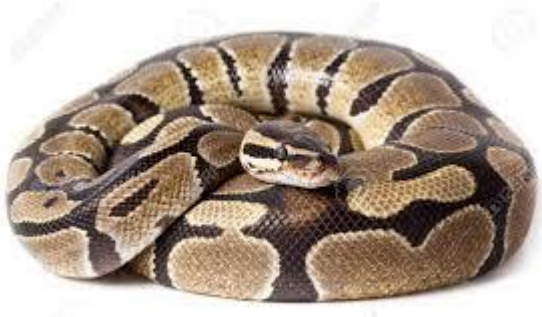
Python and Numpy

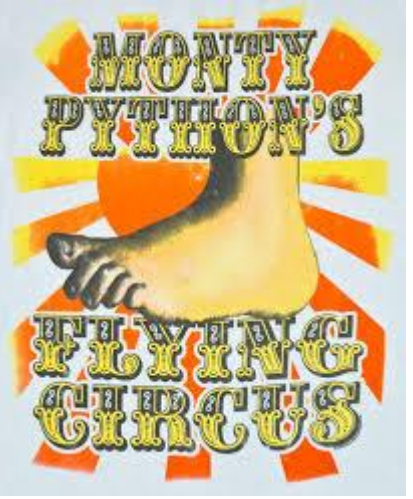
Amos Azaria



Python

- Open source
- Interpreter based language, but can be compiled for faster execution.
- Considered to be very concise – usually requires less “lines of code”
- Dynamic language / typing, no need to declare variables
- Object oriented
- Very commonly used by database analyzers
- (Python 2/3 issues…)
- (Personally used IDE: PyCharm, can use also IP[y] notebook/Jupyter)

Python's name is derived from the television series Monty Python's Flying Circus, and it is common to use Monty Python references in example code.



Variables

- Common naming convention:
  - underscores_for_variables_and_functions
  - ClassNames for classes.
- _ commonly used for variables not needed, e.g.: for _ in range(10)

Interpreter – Hello World

```
>>> print("Hello World")
Hello World
>>> a = 5
>>> a
5
>>> a = a + 1.0
>>> a
6.0
>>> 7**2
49
>>> a++
SyntaxError: invalid syntax
>>> a += 1
>>> a
7.0
>>> a = "Hello"
```

```
>>> 'He said:"Are you ok?"'
'He said:"Are you ok?"'
>>> "it's ok"
"it's ok"
>>> "He said: \"aren't you fine?\""
'He said: "aren\'t you fine?"'
>>> """He said: "How are you?" """
'He said: "How are you?" '
>>>"ell" in "Hi and hello"
True
>>> 6|3 # or
7
>>> 6&3 #and
2
>>> 6^3 #xor
5
```

If have time give example:
turnOn(int flags)
[lights, stove, oven, refrigerator, food processor, etc.]

Lists

```
>>> a = [4, 3.0, 'Hello']
>>> a
[4, 3.0, 'Hello']
>>> a[1]
3.0
>>> a[1] = 7
>>> a
[4, 7, 'Hello']
>>> a = [3,4,5] * 3
>>> a
[3, 4, 5, 3, 4, 5, 3, 4, 5]
>>> m = [[6,3,4], [6,1,3], [1, 3, 4]]
>>> m[-1]
[1,3,4]
>>> m[:2]
[[6, 3, 4], [6, 1, 3]]
>>> ([2,3,4,5]*3)[1:8:2]
[3, 5, 3, 5]
```

Tuples

```
>>> a = (4, 3.0, "Hello")
>>> a[1]
3.0
>>> a[1] = 7
TypeError: 'tuple' object does not support item assignment
>>> (a,b) = (5,4)
>>> b
4
>>> a = (3,4,5) * 3
>>> a
(3, 4, 5, 3, 4, 5, 3, 4, 5)
```

```
>>> a = (4, [5,4, [5.2, 9.3], "What?"], "Yes", ("It", 3, "Ok"))
>>> a
(4, [5, 4, [5.2, 9.3], 'What?'], 'Yes', ('It', 3, 'Ok'))
>>> a[1][2][0]
5.2
```

Dictionaries

```
>>> a = {7:5, 6:3, "bat":"sat"}
>>> a.get(6)
3
>>> print(a.get(4))
None
>>> a.get(2, -1)
-1
>>> a[7]
5
>>> a["bat"]
'sat'
```

```
>>> a[9]
KeyError: 9
>>> 6 in a
True
>>> 3 in a
False
>>> a.values()
dict_values(['sat', 3, 5])
>>> a.keys()
dict_keys(['bat', 6, 7])
```

Conditional Statement

```
>>> a = 7
>>> if a < 5:  #don't forget the colon (:)
...      b = 10 #note the indented block!!! (no {})
... else:
...      b = 3   #also here!
>>> b  #note the scope of the variable "b"
3
>>> b = 10 if a < 5 else 3 #equivalent to b=(a<5?10:3)
```

For loops

```
>>> mylist = [2, 3, "Hello", "Bye"]
>>> for x in mylist:
...     print(x)
2
3
Hello
Bye
>>> for i in range(10):
...     print(i)
>>> for (i,obj) in enumerate(mylist):
...     print('%d element in list is %s' %(i,str(obj)))
0 element in list is 2
1 element in list is 3
2 element in list is Hello
3 element in list is Bye
>>> a = [x*x for x in range (1,10)]
>>> a
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Equivalent in java: for(int i=0; i<10; i++)
All loops in python are actually foreach

Prime Number Program

```
import math

for num in range(10,20):  #iterate between 10 to 20
  for i in range(2, int(math.sqrt(num))+1): #iterate on candidate factors
    if num%i == 0:      #determine the first factor
      j=num/i          #calculate the second factor
      print('%d equals %d * %d' % (num,i,j))
      break
  else: #else of loop
    print(num, 'is a prime number')
```

A loop else will always be triggered (not related to if statement) **unless** we break the loop

Output:

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

Functions

```
>>> def hello(x):
...     return (x, "Hello World "*x)    #indented block
>>> a = [5, 2.3, hello, "Bye"]
>>> a[3]
'Bye'
>>> a[2]
<function hello at 0x0099F738>
>>> a[2](4)
(4, 'Hello World Hello World Hello World Hello World')
```

Default Parameters

```
>>> def multisay(say="Hello World", times = 2):
...     print(say*times)
...
>>> multisay()
Hello WorldHello World
>>> multisay("Hi")
HiHi
>>> multisay(times=3)
Hello WorldHello WorldHello World
```

Anonymous Functions (lambda expressions)

```
>>> f = lambda x: x*x
>>> f(5)
25
>>> f = [7, lambda x : x*x , "Hi"]
>>> f[1]
<function <lambda> at 0x0099F858>
>>> f[1](3)
9
>>> def o(f,g):
...     return lambda x: f(g(x))
>>> b = o(lambda x: x*x, lambda x: x+1)
>>> b(3)
16
```

Class

```
>>> class Complex:  #naming convention for classes is: ClassName
...     def __init__(self, realpart, imagpart): #constructor
...         self.r = realpart #r is not defined as a class field
...         self.i = imagpart
...     def add(self, numtoadd):
...         self.r += numtoadd.r
...         self.i += numtoadd.i
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
>>> x.add(Complex(1,1))
>>> x.r, x.i
(4.0, -3.5)
```

Exceptions

```
>>> (x,y) = (5,0)
>>> try:
...     z=x/y
... except ZeroDivisionError as e:
...     print(e.args)

('division by zero',)
```

Also legal
except:
except Exception as e:
except ZeroDivisionError:

Files

- Write a program that will receive three arguments:
  - Word to search for
  - Input directory
  - Output file

  and will search for all instances of the world in all filess in the input directory, and save to a file named “output file” including file name and word number.
- Think how many lines of code would this require in C, C++ or even Java and C#

Files (Cont.)

- python searcher.py from mydir out.txt
- cat out.txt (type out.txt)

"from" is word #39 in file bird.txt
"from" is word #104 in file bird.txt
"from" is word #259 in file bird.txt
"from" is word #61 in file earth.txt
"from" is word #254 in file earth.txt
"from" is word #5 in file mars.txt
"from" is word #470 in file mars.txt
"from" is word #136 in file sun.txt

Files - answer

```
import sys
from os import listdir, path
from os.path import join

search_word = sys.argv[1]
requested_dir = sys.argv[2]
output_file = open(sys.argv[3], "w")
for curr_file in listdir(requested_dir):
  for i, word in enumerate(open(join(requested_dir, curr_file)).read().split()):
    if word.strip() == search_word :
      print('\"%s\" is word #%d in file %s' %(search_word,i,curr_file), file=output_file)
outfput_file.close()
```

With statement

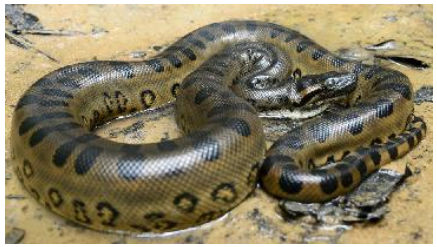
```
import sys
from os import listdir, path
from os.path import join

search_word = sys.argv[1]
requested_dir = sys.argv[2]
with open(sys.argv[3], "w") as output_file:
  for curr_file in listdir(requested_dir):
    for i, word in enumerate(open(join(requested_dir, curr_file)).read().split()):
      if word.strip() == search_word :
       print('\"%s\" is word #%d in file %s' %(search_word,i,curr_file), file=output_file) #can also use output_file.write()
#no need to close when using "with"
```

pip

- Installing library with Python is very easy thanks to pip.
- First install pip
  - Linux: apt-get install pip
  - There is also a version for windows
- Then install the library with:
  - pip install [library_name] (e.g. pip install numpy)
- You can also install Anacoda (https://www.continuum.io/downloads) which is a Python distribution that comes with numpy as well as many other libraries.

ANACONDA
Powered by Continuum Analytics



NumPy

- Enables multi dimension array and matrices operations, (and other math operations).
- Can be installed using “pip install numpy”

```
>>> import numpy as np
```

Numpy Array

```
>>> a=np.array([6,7,8])
>>> a
array([6, 7, 8])
>>> a = np.array([1.0, 2, 3.4])
>>> a
array([ 1. ,  2. ,  3.4])
>>> a.dtype
dtype('float64')
>>> a = np.array([1,  .40 ,"Hello"])
>>> a
array(['1', '0.4', 'Hello'],
    dtype='<U32')
>>> np.array(range(12))
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11])
>>> a = np.arange(12).reshape(3,4)
```

```
>>> a
array([[ 0,  1,  2,  3,],
      [ 4, 5,  6,  7],
      [8, 9, 10, 11]])
>>> a.shape
(3, 4)
>>> a.ndim
2
>>> a.dtype
dtype('int32')
>>> a.size
12
>>> np.array(range(12)).reshape(3,3)
ValueError: total size of new array must be unchanged
```

Numpy Array(cont.)

```
>>> mat = np.array([[3, 7, 5], [4, 1, 2]])
>>> mat
array([[3, 7, 5],
       [4, 1, 2]])
>>> mat[0]
array([3, 7, 5])
>>> mat[0,:]
array([3, 7, 5])
>>> mat[:,1]
array([7, 1])
>>> mat.shape
(2, 3)
>>> np.array([[3, 7, 5], [4, 1, 2, 2]])
array([[3, 7, 5], [4, 1, 2, 2]], dtype=object)
```

Array Operations

```
>>> [2, 4, 5] + [7, 3, 2]
[2, 4, 5, 7, 3, 2]
>>> np.array([2, 4, 5]) + np.array([7, 3, 2])
array([9, 7, 7])
>>> [2, 4, 5] * 3
[2, 4, 5, 2, 4, 5, 2, 4, 5]
>>> np.array([2,4,5]) * 3
array([ 6, 12, 15])
>>> np.array([2, 4, 5]) * np.array([7, 3, 2])
array([14, 12, 10])
```

2D-Array Multiplication

```
>>> np.array([[2, 4, 5], [4, 5, 2]]) * np.array([[6,3,4], [6,1,3]])
array([[12, 12, 20],
       [24,  5,  6]])
>>> np.dot(np.array([[2, 4, 5], [4, 5, 2]]), np.array([[6,3,4], [6,1,3]]))
ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
>>> np.dot(np.array([[2, 4, 5], [4, 5, 2]]), (np.array([[6,3,4], [6,1,3]]).T))
array([[44, 31],
       [47, 35]])
>>> np.array([[2, 4, 5], [4, 5, 2]]) @ (np.array([[6,3,4], [6,1,3]]).T) #Python 3
array([[44, 31],
      [47, 35]])
```