CSC321 - Python Tutorial

Kaustav Kundu

January 15, 2015

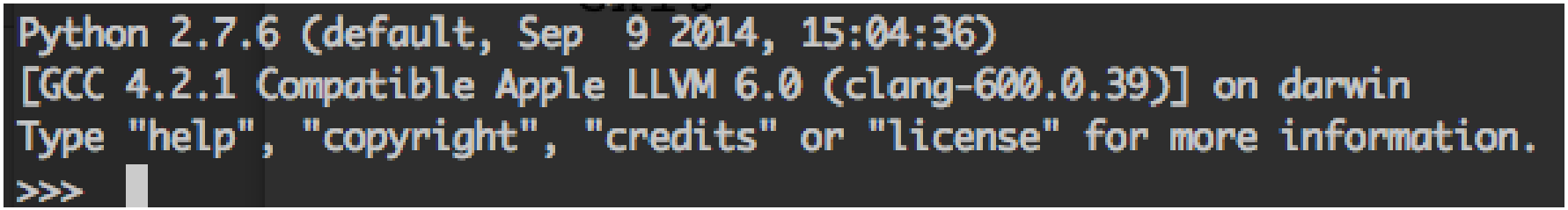
Why Python?

1. High level scripting language.
2. Rich library of modules, including third party modules/add-ons.
3. FOSS (Free and Open Source Software) - unlike Matlab.
4. Extremely good documentation (https://www.python.org/doc/) and support (stackoverflow, etc.).

Python Development Environment

1. **Command Line Interpreter**

   Type `python` from the command line to use the python interpreter



Python Development Environment

1. **Command Line Interpreter**

   Type `python` from the command line to use the python interpreter

```
Python 2.7.6 (default, Sep  9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. **Editor**
   (a) vim, emacs
   (b) IDE: Spyder

Operations

1. Arithmetic
   $+, -, *, /, \%(\text{modulus}), **(\text{exponent}), //(\text{floor division})$
2. Relational
   $>, <, >=, <=, ==, !=$
3. Logical
   `or`, `and`, `not`
4. Bitwise
   $\&, |, \wedge, \sim, <<, >>$

Lists

1. Initialization

```
>>> a = [1, 2, 3, 4, 5]
>>> range(1, 6)
[1, 2, 3, 4, 5]
>>> range(1, 6, 2)
[1, 3, 5]
>>> [1]*5
[1, 1, 1, 1, 1]
```

Lists

1. Initialization

```
>>> a = [1, 2, 3, 4, 5]
>>> range(1, 6)
[1, 2, 3, 4, 5]
>>> range(1, 6, 2)
[1, 3, 5]
>>> [1]*5
[1, 1, 1, 1, 1]
```

2. Methods: `append`, `extend`, `insert`, `remove`, `count`, `index`, `sort`, `reverse`

```
>>> a.reverse()
>>> a
[5, 4, 3, 2, 1]
```

Lists

1. Initialization

```
>>> a = [1, 2, 3, 4, 5]
>>> range(1, 6)
[1, 2, 3, 4, 5]
>>> range(1, 6, 2)
[1, 3, 5]
>>> [1]*5
[1, 1, 1, 1, 1]
```

2. Methods: `append`, `extend`, `insert`, `remove`, `count`, `index`, `sort`, `reverse`

```
>>> a.reverse()
>>> a
[5, 4, 3, 2, 1]
```

3. List comprehensions

```
>>> [x-y for x in range(0, 3) for y in range(0, 5)]
[0, -1, -2, -3, -4, 1, 0, -1, -2, -3, 2, 1, 0, -1, -2]
```

Lists

1. "Sum" of lists

```
>>> b = a[::-1]
>>> b
[1, 2, 3, 4, 5]
>>> a+b
[5, 4, 3, 2, 1, 1, 2, 3, 4, 5]
```

Lists

1. "Sum" of lists

```
>>> b = a[::-1]
>>> b
[1, 2, 3, 4, 5]
>>> a+b
[5, 4, 3, 2, 1, 1, 2, 3, 4, 5]
```

```
>>> [x + y for x, y in a, b]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack
>>> [x + y for x, y in zip(a, b)]
[6, 6, 6, 6, 6]
```

Lists

1. "Sum" of lists

```
>>> b = a[::-1]
>>> b
[1, 2, 3, 4, 5]
>>> a+b
[5, 4, 3, 2, 1, 1, 2, 3, 4, 5]
```

```
>>> [x + y for x, y in a, b]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack
>>> [x + y for x, y in zip(a, b)]
[6, 6, 6, 6, 6]
```

2. 2D lists

```
>>> c = [[x for x in range(0, y)] for y in range(0, 5)]
>>> c
[[], [0], [0, 1], [0, 1, 2], [0, 1, 2, 3]]
>>> c[3][1]
1
>>> c[3][2]
2
>>> len(c[3])
3
>>> len(c)
5
```

Arrays

- Initialization

```
>>> import numpy as np
>>> np.array([1, 2, 3])
array([1, 2, 3])
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
>>> np.ones((5,), dtype=np.int)
array([1, 1, 1, 1, 1])
>>> np.zeros(5)
array([ 0.,  0.,  0.,  0.,  0.])
>>> np.arange(0, 6)
array([0, 1, 2, 3, 4, 5])
```

Arrays

- Initialization

```
>>> import numpy as np
>>> np.array([1, 2, 3])
array([1, 2, 3])
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
>>> np.ones((5,), dtype=np.int)
array([1, 1, 1, 1, 1])
>>> np.zeros(5)
array([ 0.,  0.,  0.,  0.,  0.])
>>> np.arange(0, 6)
array([0, 1, 2, 3, 4, 5])
```

- Element-wise operations

```
>>> a = np.arange(1, 6)
>>> a
array([1, 2, 3, 4, 5])
>>> a[1:3]
array([2, 3])
>>> b = np.random.randint(1, 100, 5)
>>> b
array([48, 99, 76, 26, 71])
>>> a + b
array([ 49, 101,  79,  30,  76])
>>> a / b
array([0, 0, 0, 0, 0])
>>> a*1.0 / b
array([ 0.02083333,  0.02020202,  0.03947368,  0.15384615,  0.07042254])
```

Arrays

- Dot Products

```
>>> c = np.random.randint(10, size=[3, 3])
>>> c
array([[6, 0, 2],
       [1, 9, 9],
       [1, 2, 9]])
>>> b = np.ones([3, 1])*2
>>> b
array([[ 2.],
       [ 2.],
       [ 2.]])
>>> np.dot(c, b)
array([[ 16.],
       [ 38.],
       [ 24.]])
>>> c.dot(b)
array([[ 16.],
       [ 38.],
       [ 24.]])
```

Arrays

- Matrix Multiplication

```
>>> b = np.random.randint(0, 3, [3, 3])
>>> b
array([[1, 1, 2],
       [2, 2, 1],
       [1, 1, 0]])
>>> np.dot(b, b)
array([[5, 5, 3],
       [7, 7, 6],
       [3, 3, 3]])
```

Arrays

- Matrix Multiplication

```
>>> b = np.random.randint(0, 3, [3, 3])
>>> b
array([[1, 1, 2],
       [2, 2, 1],
       [1, 1, 0]])
>>> np.dot(b, b)
array([[5, 5, 3],
       [7, 7, 6],
       [3, 3, 3]])
```

- Transpose

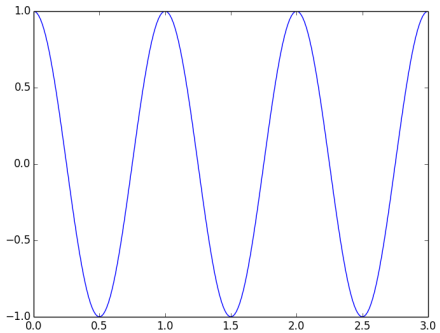
```
>>> c
array([[6, 0, 2],
       [1, 9, 9],
       [1, 2, 9]])
>>> c.T
array([[6, 1, 1],
       [0, 9, 2],
       [2, 9, 9]])
```

Arrays vs Lists

1. Difference between lists and arrays is similar to that between matrices and cell arrays in MATLAB.
2. All elements in arrays have to be of the same data type, specified at the time of creation.
3. Usually memory efficient, may not be time efficient compared to lists.

Arrays vs Lists

```
>>> from pylab import *
>>> t = arange(0, 3, 0.01)
>>> F = 2;
>>> x = cos(pi*t*F)
>>> plot(t,x)
[<matplotlib.lines.Line2D object at 0x109f53b90>]
>>> show()
```



1. `numpy.array` provides a lot of advantages to perform matrix operations, like transpose, inverse, eigen values, etc.
2. For more details look at `numpy` and `scipy` documentations.

Dictionary

Equivalent to `std::map` in C++.

```
>>> d = {x: x**2 for x in range(0, 5)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
>>> d[4]
16
>>> {str(x): x for x in range(0, 5)}
{'1': 1, '0': 0, '3': 3, '2': 2, '4': 4}
```

Modules

```
# Fibonacci numbers module

def fib(n):   # write Fibonacci series up to n
   a, b = 0, 1
   while b < n:
      print b,
      a, b = b, a+b

def fib2(n): # return Fibonacci series up to n
   result = []
   a, b = 0, 1
   while b < n:
      result.append(b)
      a, b = b, a+b
   return result
```

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> f = fibo.fib2(1000)
>>> f
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
>>> fibo.__name__
'fibo'
```

Classes

```
>>> class Complex:
...     def __init__(self, real_part, imag_part):
...             self.r = real_part
...             self.im = imag_part
...
>>> x = Complex(3.0, -4.0)
>>> x.r, x.im
(3.0, -4.0)
```