

# Python and Object Oriented Programming

Antonio Pons  
Pau Rué

November 12, 2009

# python

- 1** Introduction
- 2** python syntax
  - strings
  - introspection
  - conditional statements
  - loops
  - lists
  - tuples
  - dictionaries
  - string formatting
  - functions
  - modularity
  - classes
  - external modules
- 3** Hands on

# python

clear syntax

introspection

intuitive object orientation

natural expression of procedural code

modularity

batteries included

# python resources

Python help function

Python documentation

<http://docs.python.org/>

Code examples

<http://sandbox.mc.edu/~bennet/python/code/>

Useless Python

<http://www.uselesspython.com/>

Python recipes at Activestate

<http://code.activestate.com/recipes/langs/python/>

# starting python

## Interactive mode (Command line interface)

```
$ python
```

```
Python 2.6.3 (r263:75183, Oct  4 2009, 09:36:16)
```

```
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
```

```
Type "help", "copyright", "credits" or "license"  
for more information.
```

```
>>> print "Hello World!"
```

```
Hello World!
```

```
>>>
```

## From a file

```
$ python helloworld.py
```

```
Hello World!
```

```
$
```

## From an IDE (IDLE)

# data Types

```
>>> a = 3.1415
>>> b = 10
>>> c = 3 + 4j
>>> d = 'Hey!'
>>> e = ['eggs', 'spam', 15, [2, 3]]
>>> f = ('eggs', 'spam', 15, [2, 3])
>>> g = {'USD':0.6745, 'GBP':1.1189, 'JPY':0.0075}
```

## names (instead of variables)

```
>>> a = 'Hello World'
```

```
>>> print a
```

```
Hello World
```

```
>>> a = 3
```

```
>>> print a
```

```
3
```

```
>>> print pow(2, 10)
```

```
1024
```

```
>>> a = pow
```

```
>>> print a(2,10)
```

```
1024
```

# strings

```
>>> a = 'Hello World'
```

```
>>> print a[1:6]
```

```
ello
```

```
>>> b = a.upper()
```

```
>>> print b
```

```
HELLO WORLD
```

```
>>> c = a.replace('e', 'a')
```

```
>>> print c
```

```
Hallo World
```

```
>>> print len(a)
```

```
11
```

```
>>> print a.count('o')
```

```
2
```



# introspection

```
>>> a = 'Hello World'
```

```
>>> dir(a)
```

```
...
```

```
'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace',  
'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',  
'split', 'splitlines', 'startswith', 'strip', 'swapcase',  
'title', 'translate', 'upper', 'zfill']
```

```
>>> help(range)
```

```
range([start,] stop[, step]) -> list of integers
```

Return a list containing an arithmetic progression of integers.  
range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults  
When step is given, it specifies the increment (or decrement).

```
...
```

# conditional statements

```
>>> i = 12
>>> if i % 3 == 0:
>>>     # some code here
>>>     pass
>>> elif i % 2:
>>>     # some other code here
>>>     pass
>>> else:
>>>     # code for the default case
>>>     pass
```

# for loops

```
>>> x = 1
>>> for i in range(4, 8):
>>>     x *= 3
>>>     print "3 to the", i , "is", x
```

3 to the 4 is 3

3 to the 5 is 9

3 to the 6 is 27

3 to the 7 is 81

# while loops

```
>>> n = 1
>>> x = 0
>>> while x < 0.99:
>>>     x = x + 0.5**n
>>>     n = n + 1
>>>     print n, x
```

2 0.5

3 0.75

4 0.875

5 0.9375

6 0.96875

7 0.984375

8 0.9921875

# lists

lists are **ordered sets** of objects

```
>>> A = ['Hannibal', 'B.A. Baracus', 'Faceman']
```

```
>>> print A[0], A[-1]
```

```
Hannibal Faceman
```

```
>>> print A[1:3]
```

```
['B.A. Baracus', 'Faceman']
```

```
>>> print len(A)
```

```
3
```

```
>>> A[1] = 'Mr. T'
```

```
>>> print A
```

```
['Hannibal', 'Mr. T', 'Faceman']
```

# lists

```
>>> A = ['Hannibal', 'B.A. Baracus', 'Faceman']  
>>> A.append('Murdock')  
>>> A.append('Murdock')  
>>> print A.count('Murdock')
```

2

```
>>> for a in A:  
>>>     print a
```

Hannibal

B.A. Baracus

Faceman

Murdock

Murdock

# lists comprehensions

```
>>> a = [x**2 for x in range(5)]
```

```
>>> print a
```

```
[0, 1, 4, 9, 16]
```

```
>>> noprimers = [j for i in range(2, 8) \
```

```
>>> for j in range(i*2, 50, i)]
```

```
>>> primes = [x for x in range(2, 50) if x not in noprimers]
```

```
>>> print primes
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

# tuples (immutable data type)

```
>>> printerIP = (147,83,94,16,9100)
>>> print printerIP[3]
```

16

```
>>> try:
>>>     printerIP[3] = 12
>>> except TypeError, err:
>>>     print 'There was an error:'
>>>     print err
```

There was an error:

'tuple' object does not support item assignment

```
>>> oct1, oct1, oct2, oct3, port = printerIP
>>> print port
```

9100



# dictionaries

dictionaries map keys to values (one-to-one relationships)  
order is not conserved

```
>>> d = {'a':1, 'name':'James', 'grades':['A', 'A', 'C']}
```

```
>>> print d.keys()  
['a', 'grades', 'name']
```

```
>>> print d.values()  
[1, ['A', 'A', 'C'], 'James']
```

```
>>> print d['a']  
1
```

```
>>> d['grades'] = ['C', 'C', 'C']
```

## zipping, combining dictionaries

```
>>> seq1 = ('a','b','c','d')
>>> seq2 = [1,2,3,4]
>>> d = dict(zip(seq1,seq2))
>>> print d

{'a': 1, 'c': 3, 'b': 2, 'd': 4}
```

```
>>> d1 = {'apples': 3, 'oranges': 2}
>>> d2 = {'pears': 4, 'grapes': 1}
>>> d1.update(d2)
>>> print d1

{'grapes': 1, 'pears': 4, 'apples': 3, 'oranges': 2}
```

## looping, deleting dictionaries

```
>>> eur = {'USD':0.6745, 'GBP':1.1189, 'JPY':0.0075}
>>>
>>> for k, v in eur.items():
>>>     print "EUR 1 = %s %g" % (k, v)

EUR 1 = JPY 0.0075
EUR 1 = USD 0.6745
EUR 1 = GBP 1.1189

>>> del eur['JPY']
>>> print eur

{'USD': 0.6744999999999999, 'GBP': 1.1189}
```

# string formatting

```
>>> template = 'All %s and no %s makes %s a %s.'
>>> tuple1 = ('sun', 'snow', 'Hawaii', 'funny place')
>>> # a list is also ok
>>> phrase1 = template % tuple1
>>> print phrase1
```

All sun and no snow makes Hawaii a funny place.

```
>>> tuple2 = ('work', 'play', 'Jack', 'dull boy')
>>> phrase2 = template % tuple2
>>> print phrase2
```

All work and no play makes Jack a dull boy.

# string formatting with dictionaries

```
>>> format = 'results_epsilon%(eps).3fk_%(k).2f_N%(N)d.txt'
>>> params = {'eps':0.125, 'k':2.35, 'N':200}
>>> filename = format % params
>>> print filename

results_epsilon0.125k_2.35_N200.txt

>>> params['k'] = 2.40
>>> filename = format % params
>>> print filename

results_epsilon0.125k_2.40_N200.txt
```

# functions

```
>>> def myFunction(a, b):  
>>>     '''  
>>>     Returns the result of two times 'a' plus 'b'  
>>>     '''  
>>>     c = 2*a + b  
>>>     return c  
>>>  
>>> print myFunction(2, 3)  
  
7  
  
>>> print myFunction('Hey ', 'Ho ')
```

Hey Hey Ho

# positional vs keyword arguments

```
>>> def printArguments(a, b):  
>>>     print 'First arg. ' + a + ', second arg. ' + b  
>>>
```

```
>>> # Positional arguments
```

```
>>> printArguments('1', '2')
```

```
First arg. 1, second arg. 2
```

```
>>> # Keyword arguments
```

```
>>> printArguments(b='1', a='2')
```

```
First arg. 2, second arg. 1
```

# positional arguments

```
>>> def myFunction(*args):  
>>>     ''' Here 'args' is a list of values '''  
>>>     return args  
>>>  
>>> print myFunction(2, 'A', ['Random', 3.15])  
  
(2, 'A', ['Random', 3.1499999999999999])
```

```
>>> def computeMean(*args):  
>>>     t = 0  
>>>     for arg in args: t += arg  
>>>     return t/len(args)  
>>>  
>>> print 'The mean is %.2f' % computeMean(1,2,3,4,5,6,7)  
  
The mean is 4.00
```



# keyword arguments

```
>>> def myFunction(**kwargs):  
>>>     ''' In this case 'kwargs' is a dictionary'''  
>>>     for k,v in kwargs.items():  
>>>         print k, '=', v  
>>>  
>>> myFunction(dt=0.1, c='RungeKutta', epsilon=0.05)  
  
epsilon = 0.05  
dt = 0.1  
c = RungeKutta  
  
>>> dict = {'USD':0.6745, 'GBP':1.1189, 'JPY':0.0075}  
>>> myFunction(**dict)  
  
JPY = 0.0075  
USD = 0.6745  
GBP = 1.1189
```

everything can be put in a module

```
>>> # This is the conversor.py module
>>>
>>> def c2f(Tf):
>>>     '''    Converts Farenheit to Celsius    '''
>>>     return (5.0/9.0)*(Tf-32.0)
```

---

```
>>> import conversor
>>> print conversor.c2f.__doc__

    Converts Farenheit to Celsius
```

```
>>> Tc = conversor.c2f(66)
>>> print Tc
```

```
18.8888888889
```

# Classes

```
>>> class Stack():
>>>     ''' Aqui definim la classe '''
>>>     def __init__(self):
>>>         self.items = []
>>>
>>>     def push(self, item):
>>>         self.items.append(item)
>>>
>>>     def pop(self):
>>>         item = self.items[-1]
>>>         self.items = self.items[0:-1]
>>>         return item
```

## BioSequence Example

## external modules

```
>>> from pylab import *  
>>> A = array([[1, 2, 3], [4, 5, 6]])  
>>> print 2*A
```

```
[[ 2  4  6]  
 [ 8 10 12]]
```

```
>>> print A*A
```

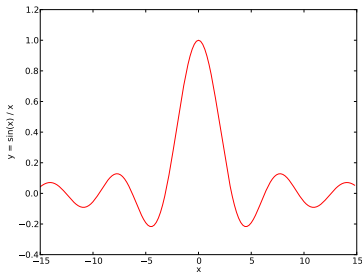
```
[[ 1  4  9]  
 [16 25 36]]
```

```
>>> print dot(A.T, A)
```

```
[[17 22 27]  
 [22 29 36]  
 [27 36 45]]
```

## external modules

```
>>> from pylab import *  
>>>  
>>> x = arange(-15, 15, 0.2)  
>>> y = sin(x) / x  
>>> plot(x, y, 'r-')  
>>> xlabel('x')  
>>> ylabel('y = sin(x) / x')  
>>> savefig('sinc.pdf')  
>>> close()
```



Hands on