

Data processing in python

Roland Memisevic

January 20, 2015

Python

Python according to Wikipedia:

“Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability.”

- ▶ Interpreted Language
- ▶ Strict syntax, eg. scope defined by indentation
- ▶ “Batteries included”
- ▶ Dynamic types
- ▶ Mixes imperative, object-oriented, functional programming elements

Python

- ▶ Various implementations.
- ▶ Several (incompatible) versions available.
- ▶ Common Python 2.x suitable for most data processing needs.
- ▶ (Python 3.0 exists but doesn't seem to be widely adopted)
- ▶ *ipython* highly recommended for any kind of interactive work (such as most machine learning research).

“Hello World”

```
print "hello world"
```

```
print "hello world", 1, 2, 1 + 2
```

```
a = 1  
b = 1  
print "hello world", a + b
```

```
a = 1  
b = "hello"  
print "hello world", a + b # oops
```

Built-in data structures

Tuples

```
T = (1, 2, 3, "hello")
print T[0]
```

Lists

```
L = [1, 2, 3, "hello"]
L[0] = "lists are mutable"
print L[0]
```

Dictionaries ("Hashes")

```
L = {"a": 1, "b": 2}
print L["a"]
```

- ▶ In Python, indexing starts at 0 !

Functions, Control structures

- ▶ Function definition:

```
def timesfour(x):
    return 4*x
```

- ▶ Control structures:

- ▶ if-then-else:

```
if s == "y":
    print("y")
elif s == "z":
    print("z")
else:
    print("b")
```

- ▶ while:

```
a = 1.0
while a != 10.0 and s == "hello":
    a = a + 1.0
```

- ▶ for-loops:

```
for i in [1,2,'x',3,4,'h',5]:
    print(i)
```

Call-by-reference

- ▶ To understand function calls you need to remember two things:
 - ▶ All function calls are call-by-reference.
 - ▶ Some things are mutable and some are not.

In python everything is an object

- ▶ Objects have member components (functions and attributes).
- ▶ Classes can be defined using

```
class myclass(object):
    a = 1
```
- ▶ Instantiate objects with

```
myobject = myclass()
print myobject.a myobject.b = 2
```
- ▶ But: Can get far without using classes.
- ▶ Functions are objects, too (it's worth giving a try!)

help

- ▶ When working interactively:

```
a = [1, 2, 3]
help(a)
```
- ▶ pass the *object* you need help about
- ▶ instantiate one, if you do not have one handy

Scripts, modules, packages

- ▶ There are two ways to use code from a file:
- ▶ Scripts and modules
- ▶ For both we typically use a filename ending “.py”
- ▶ To run a script:

```
python myscript.py
```
- ▶ In Ipython we can also use:

```
%run myscript.py
```

Scripts, modules, packages

- ▶ Modules are like libraries, used to keep common functionality in one place.
- ▶ One can combine multiple modules in “packages” (which are just directories containing modules, and some files with meta-data)
- ▶ To access the contents of a module:

```
import mymodul
print mymodul.somefunction(1)
print myobject = mymodul.someclass() ...
```

- ▶ To use a module:

```
import mymodul
#or:
from import mymodul myobject #imports a single object
#or:
from import mymodul * # imports everything
```

Packages for data crunching

“pylab” =

- ▶ numpy (for “computing”) +
 - ▶ matplotlib (for “plotting”) +
 - ▶ scipy (for more specific needs, like Fourier transforms, special functions, etc.)
-
- ▶ After installing matplotlib, use “import pylab” or “from pylab import *” to get direct access to the most important functions and objects.
 - ▶ Or use Ipython with the option “-pylab”

Packages for data crunching

numpy arrays

- ▶ The central object for representing *data* is the numpy array.
- ▶ It is an n-dimensional generalization of a matrix.
- ▶ It can hold data of various types, like int, float, string, etc.
- ▶ The most common use is for representing vectors and matrices filled with numbers (such as float's).
- ▶ One way to generate an array is to use `numpy.array`. But there are many other ways:

Pylab

- ▶ numpy arrays have many useful member components: `a.T`, `a.mean(0)`, `a.std(0)`, `a.max()`, ...
- ▶ One of the most important ones is `a.shape`
- ▶ Example:

```
print randn(3,3).shape
print randn(2,5,7,3).shape print array([1,2,3,4]).shape
```

Pylab

Generate data

- ▶ `array()` generate a numpy array from a list
- ▶ `ones()` get an array filled with ones
- ▶ `zeros()` get an array filled with zeros
- ▶ `eye()` get an identity matrix
- ▶ `randn()` get a matrix filled with random numbers drawn from a Gaussian
- ▶ `load()`, `save()`, `loadtxt()`, etc.

Pylab

Doing computations

- ▶ To do computations on arrays, numpy has functions like
- ▶ `exp()`, `log()`, `cos()`, `sin()`, ...
- ▶ `+`, `-`, `*`, `/`
- ▶ These typically operate *elementwise*
- ▶ Other useful matrix operations: `dot()`, `svd()`, `eig()`

Pylab

Accessing data

- ▶ `a[0]`, `a[1]` use for 1/0-d arrays
- ▶ `a[0, 0]`, `a[0,1]`, `a[37, 50]` use for 2-d arrays
- ▶ `a[0, 0, 0]`, `a[0, 1, 5]`, use for 3-d arrays, etc.
- ▶ `a[0, :]` a “slice”
- ▶ `a[:, 0]` another “slice”
- ▶ `a[1:5, 0]` another “slice”
- ▶ `a[1:5, :]` another “slice” (this is a 2d-block)

Pylab

Plotting

- ▶ `plot`, `scatter`, `hist`, `box`, `bar`
- ▶ `imread`, `imsave`, `imshow`
- ▶ `subplot`
- ▶ `legend`, `annotate`, `xlabel`, `ylabel`, `title`
- ▶ etc.

Broadcasting und *newaxis*

- ▶ How to add a 2×5 matrix and a 1×5 vector?
- ▶ In Matlab you would use “`repmat`”. In python you can, too.
- ▶ But python comes with another possibility:
- ▶ Numpy will always try to copy every dimension in each array as often as it needs to make the dimensions fit
- ▶ Example:
`(randn(2,5) + randn(1,5)).shape` # result is (2,5)

Broadcasting und *newaxis*

- ▶ How about a 2×5 matrix plus a $1 \times 5 \times 3$ tensor?
- ▶ For this, we first need to make the number of dimensions match. Solution: numpy’s “`newaxis`”
- ▶ Example:

```
randn(2,5).shape           # result ist (2,5)
randn(2,5) + randn(1,5,3)  # doesn't work!
randn(2,5)[:,:,newaxis].shape # result is (2,5,1)
randn(2,5)[:,:,newaxis] + randn(1,5,3) # works!
```

Useful information online

Python in general

- ▶ docs.python.org/tutorial
- ▶ [diveintopython](http://diveintopython.com)

Data crunching in Python

- ▶ matplotlib.sourceforge.net
- ▶ numpy.scipy.org
- ▶ scipy.org
- ▶ pandas.pydata.org

Python for matlab users

- ▶ mathesaurus.sourceforge.net/matlab-numpy.html
- ▶ scipy.org/NumPy_for_Matlab_Users

Python for deep learning

- ▶ <http://deeplearning.net/software/theano/>
- ▶ <http://deeplearning.net/tutorial/>