

# Data processing in python

Roland Memisevic

January 25, 2013

# Python

## Python according to Wikipedia:

“Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability. Python claims to combine remarkable power with very clear syntax”, and its standard library is large and comprehensive. Its use of indentation for block delimiters is unique among popular programming languages.”

- ▶ Interpreted Language
- ▶ Strict syntax
- ▶ 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 of our data processing needs.
- ▶ “Ipython” *highly* recommended for any kind of interactive work!

# “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)
```

# Everything is an object

- ▶ Functions, too.
- ▶ Objects have member components (functions and attributes).
- ▶ Classes can be defined (if desired or needed) using

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

```
myobject = myclass()  
print myobject.a
```
- ▶ But: Can get far without using classes.

# help

- ▶ When working interactively:

```
a = [1, 2, 3]
help(a)
```

- ▶ help expects the *object* you need help about!
- ▶ Just instantiate one, if you do not have it!

# Scripts, modules, packages

- ▶ Naming convention for scripts: “.py”  
`python myscript.py`
- ▶ In ipython we can use:  
`%run myscript.py`

# Scripts, modules, packages

- ▶ Combine common functionality in “modules” (= “libraries”)
- ▶ Same convention as for scripts: “.py”
- ▶ Can combine modules in “packages”
- ▶ To use a module:

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

- ▶ Remember: Everything is an object. Access the contents of modules accordingly.

```
import mymodul
print mymodul.afunction(1)
print myobject = mymodul.aclass()
```

# 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.
  - ▶ ipython offers the option “-pylab” to give you an environment that contains all the necessary stuff.

# 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 we'll see other ways soon.

# “Pylab”

- ▶ After going

`ipython -pylab`

or

```
from pylab import *; from numpy import * #from within Python
```

we have access to things like

## 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.

- ▶ 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
```

## 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()`

## Accessing your 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)

## Plotting

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

## Broadcasting und *newaxis*

- ▶ How to add a  $2 \times 5$  matrix and a  $1 \times 5$  vector?
- ▶ In Matlab you would use “repmat”. You can in Python, too.
- ▶ However, 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  $2 \times 5$  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](https://docs.python.org/tutorial)
- ▶ [diveintopython](https://diveintopython.com/)

## Data crunching in Python

- ▶ [matplotlib.sourceforge.net](https://matplotlib.sourceforge.net/)
- ▶ [numpy.scipy.org](https://numpy.org/)
- ▶ [scipy.org](https://scipy.org/)

## Python for matlab users

- ▶ [mathesaurus.sourceforge.net/matlab-numpy.html](https://mathesaurus.sourceforge.net/matlab-numpy.html)
- ▶ [scipy.org/NumPy\\_for\\_Matlab\\_Users](https://scipy.org/NumPy_for_Matlab_Users)