# Python for astronomers

# Presentation

- Young language (1989,  Guido van Rossum) but well tested
- Already installed on Linux and OSX. You still may need some upgrade.
- Huge community and web site (just Google “python scientific pdf” to have a lot of manuals.
- A lot of packages:
  - Numerical, graphical, scientific, GUI, SQL, HTML, etc tools
  - Browse the pypi library: 28675 packages!

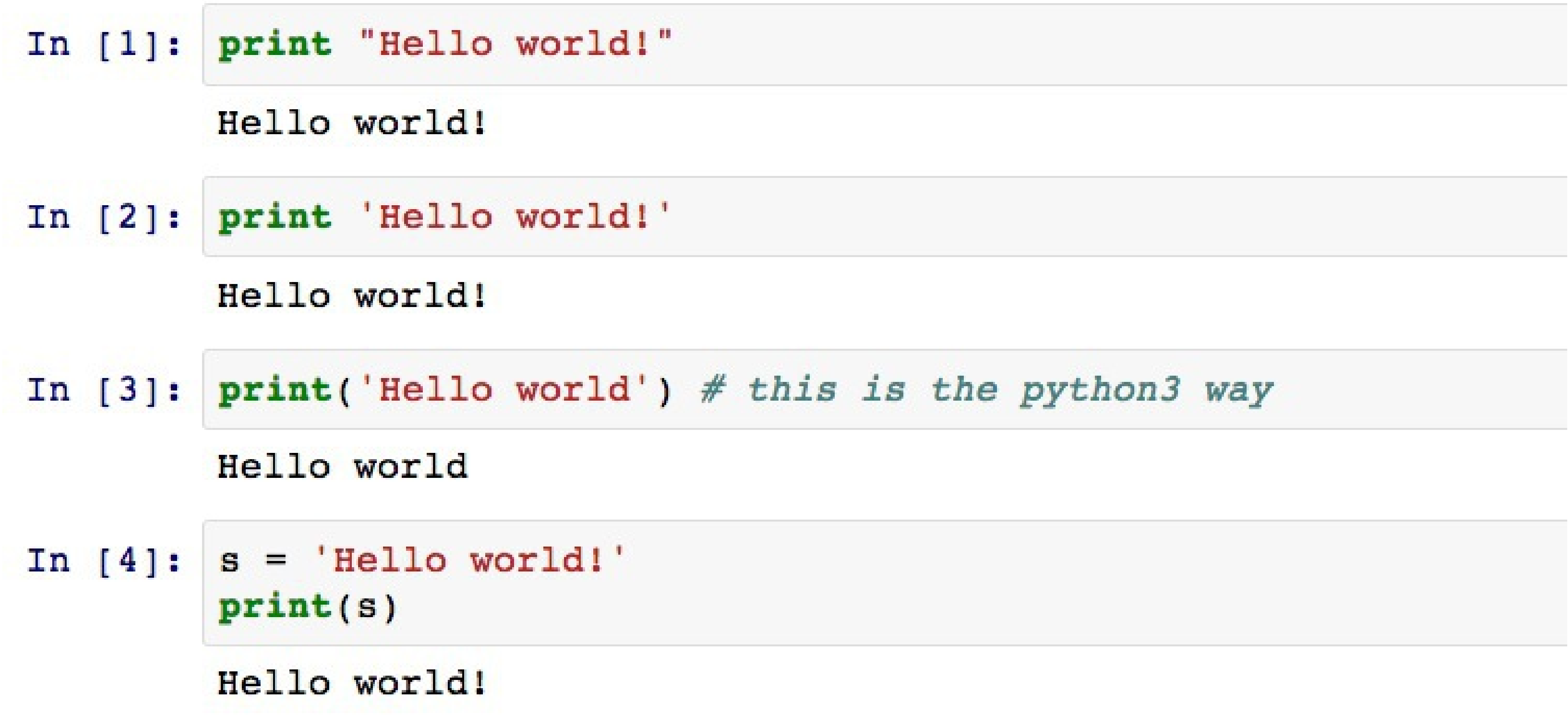
# “best” configuration

- Python >= 2.6 (but not > 3)
- Ipython: interactive python
- Numpy: numerical tools, arrays, vectorization
- Matplotlib: graphical tools
- Scipy: scientific tools (integrations, inter- extra-polation)
- → EPD is the easiest way
  - Free distribution
  - Academic distribution (more complete)

# Interactive session

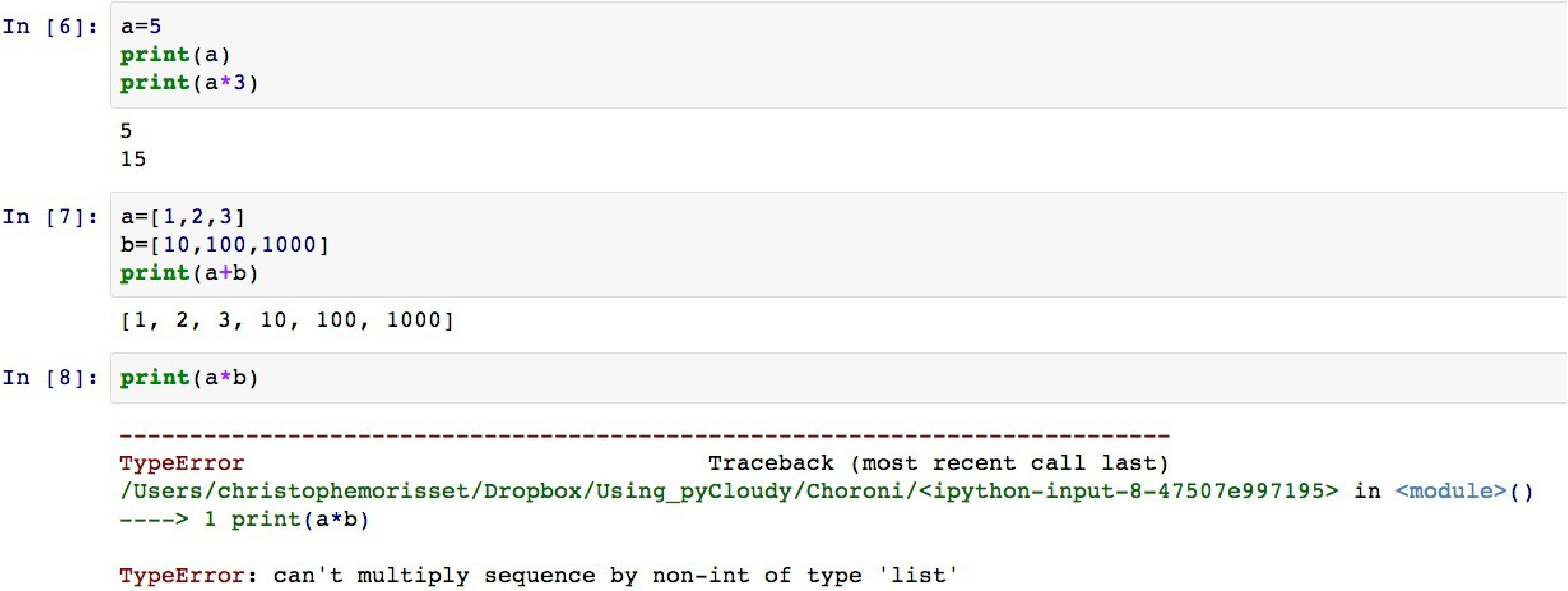
- Using ipython with the -- pylab option:
  - Load numpy and matplotlib.pyplot as *np* and *plt* respectively. This is the same as:
    - import numpy as np
    - Import matplotlib.pyplot as plt
  - Configure the graphical output (backend) so that graphical windows appears without efforts (otherwise a *plt.show()* call is needed)
  - Some magic commands start with %:
    - %run ex1 : execute what is in ex1.py on the main level
    - %paste : paste what is in the clipboard to the session, dealing with the indentation
    - Others...

# Hello world

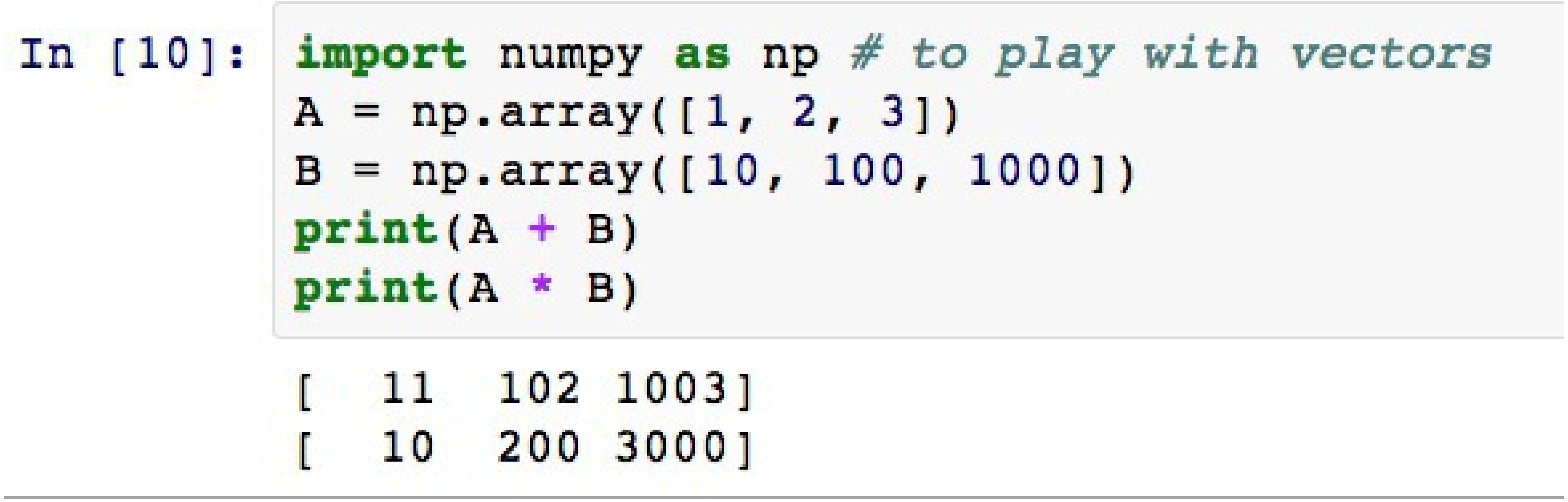


```
In [1]: print "Hello world!"

        Hello world!

In [2]: print 'Hello world!'

        Hello world!

In [3]: print('Hello world') # this is the python3 way

        Hello world

In [4]: s = 'Hello world!'
        print(s)

        Hello world!
```

# Very quick look at python



```
In [6]: a=5
        print(a)
        print(a*3)

        5
        15

In [7]: a=[1,2,3]
        b=[10,100,1000]
        print(a+b)

        [1, 2, 3, 10, 100, 1000]

In [8]: print(a*b)

        ---------------------------------------------------------------------------
        TypeError                                 Traceback (most recent call last)
        /Users/christophemorisset/Dropbox/Using_pyCloudy/Choroni/<ipython-input-8-47507e997195> in <module>()
        ----> 1 print(a*b)

        TypeError: can't multiply sequence by non-int of type 'list'
```



```
import numpy as np # to play with vectors
A = np.array([1, 2, 3])
B = np.array([10, 100, 1000])
print(A + B)
print(A * B)
```

```
[  11  102 1003]
[  10  200 3000]
```

# Types

## Int, real (float), complex, strings.

```
A = 3

print(type(A))

B = 4.5

print(type(B))

C = A * B

print(C, type(C))
```

# Blocks

- Blocks are defined by indentation. Looks nice and no needs for end :-)
- if, elif, else
- for X in list:
- while <condition>:
- List comprehension
  - A = [i**2 for i in range(4)]
  - print(A)

# Functions, procedures, methods

- In a script or “on the fly”

```
def func1(x):
    print(x**3) # use TAB to indent
func1(3)
def func2(x):
    return(x**3)
print(func2(3))
```

In the file Test1.py:

```
import Test1
```

Or

```
run Test1
```

# Functions parameters

Mandatory parameters and optional parameters (default value):

```
def func3(x, y, z, a=0, b=0):
   return a + b * np.sqrt(x**2 + y**2 + z**2)
D = func3(3, 4, 5)
E = func3(3, 4, 5, 10, 100)
F = func3(x=3, y=4, z=5, a=10, b=100)
G = func3(3, 4, 5, a=10, 100) # ERROR!
H = func3(3, 4, 5, a=10, b=100)
I = func3(z=5, x=3, y=4) # quite risky!
```

# Loading a program/script

- Import ex1: this will execute what is in *ex1.py* in the namespace *ex1*.
- If a function *f1()* is defined in *ex1.py*, it is accessible as *ex1.f1()* once test1 is imported:

```
import test1
print(test1.f1(3))
from test1 import f1
print(f1(3))
import test1 as tt # alias to the package
A = tt.f1(3)
```

- You can also execute the script test1:

```
%run test1.py
f1(3)
%whos
```

# Command line help

- Once a package is imported, you can access all its components by TAB after the last point:

```
import numpy as np
np.<TAB>
import pyneb as pn
pn.<TAB>
```

- You access the help for a given function/class by :

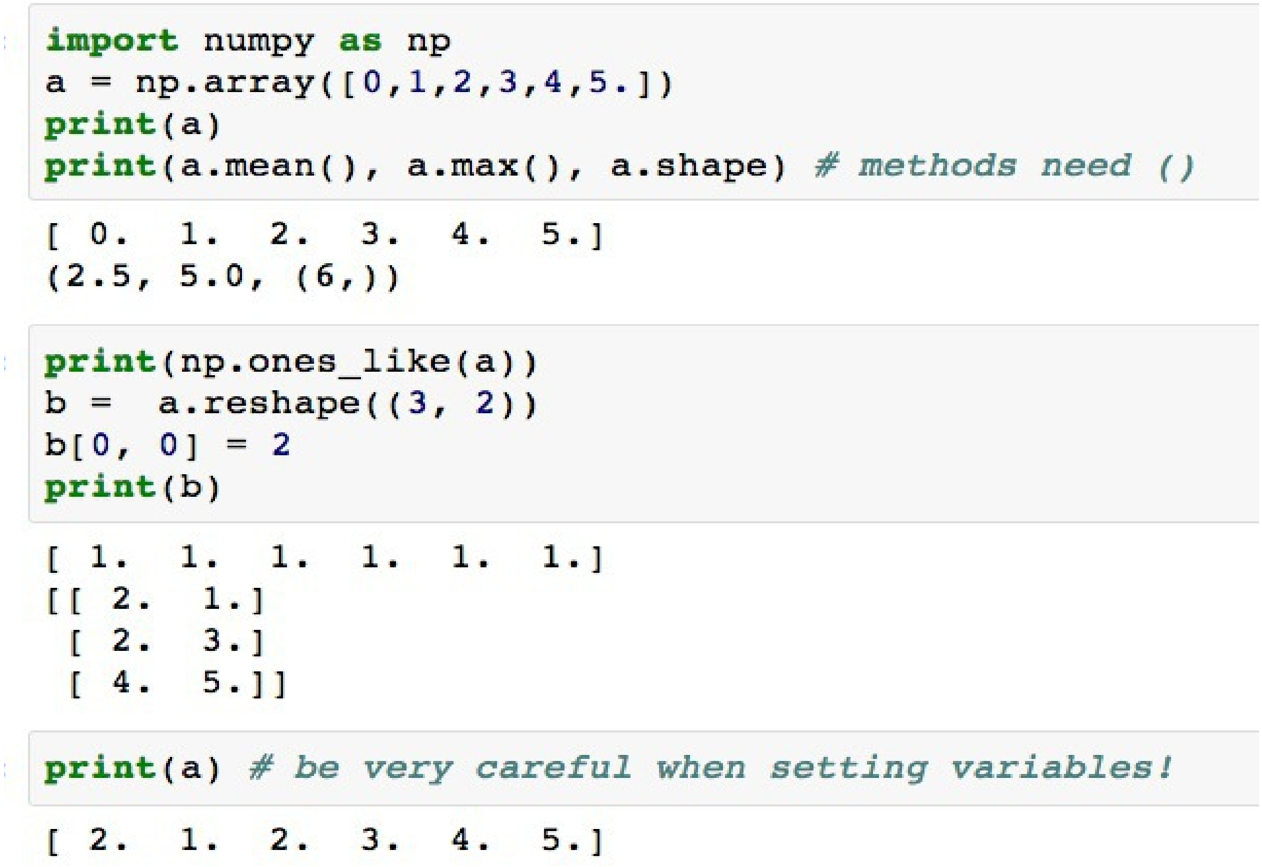
```
pn.Atom?
```

- This works for any object within ipython and goes “recursively” inside the objects:

```
pn.Atom.<TAB>

pn.Atom.getA?
```

# numpy

- Easy way to deal with arrays (1D, 2D, nD)
- Vectorization of most of the operations (not parallel)



```
import numpy as np
a = np.array([0,1,2,3,4,5.])
print(a)
print(a.mean(), a.max(), a.shape) # methods need ()
```

```
[ 0.  1.  2.  3.  4.  5.]
(2.5, 5.0, (6,))
```

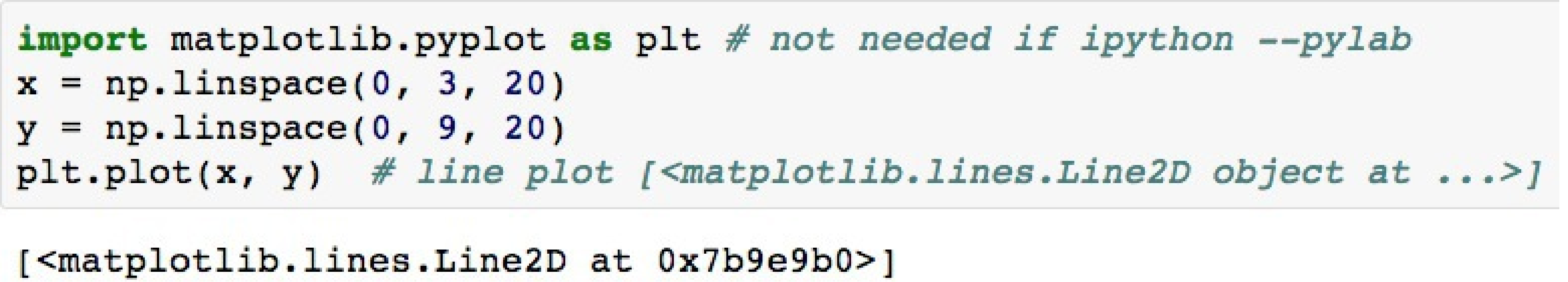
```
print(np.ones_like(a))
b =  a.reshape((3, 2))
b[0, 0] = 2
print(b)
```

```
[ 1.  1.  1.  1.  1.  1.]
[[ 2.  1.]
 [ 2.  3.]
 [ 4.  5.]]
```

```
print(a) # be very careful when setting variables!
```

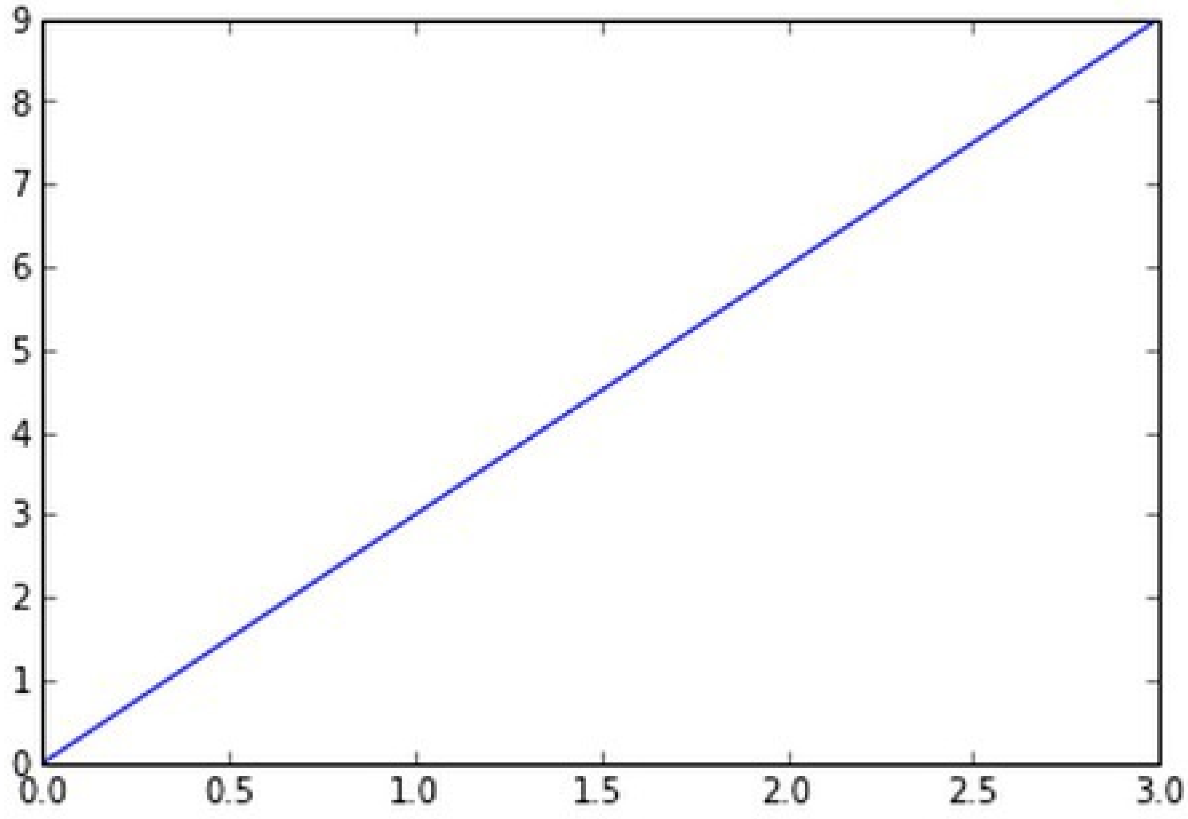
```
[ 2.  1.  2.  3.  4.  5.]
```

# Visualization



```
import matplotlib.pyplot as plt # not needed if ipython --pylab
x = np.linspace(0, 3, 20)
y = np.linspace(0, 9, 20)
plt.plot(x, y)  # line plot [<matplotlib.lines.Line2D object at ...>]
```

```
[<matplotlib.lines.Line2D at 0x7b9e9b0>]
```



# Vizualisation

The web site of matplotlib is full of examples you can adapt to your problem.

Obviously, you can add axes label, make log plots, change colors, make multiplots in a single window, etc...

# Object oriented

- Python allows to manage classes and the corresponding objects (class instantiation).
- Methods (class functions) and attributes (class variables) are accessed by: `name_of_the_object.method`