A QUICK INTRODUCTION TO PYTHON

Emmanuel Dormy

Jan. 2024

1 Foreword

I will be using the *Anaconda* distribution of Python available for Linux, Mac or Windows (see the course website for the download link).

This distribution, in addition to its simplicity of installation has the advantage of integrating the majority of it packages necessary for scientific computing.

2 Getting started

I recommend using the following command to start the *Python* interpreter :

`ipython --pylab`

The *iPython* interface offers many features that enhance the standard *Python* interface, including command history, completion (TAB), online help, access to the system ... In addition, the "--pylab" option avoids having to systematically import this module.

The appearance of the *prompt* "In [1] :" confirms that the instruction worked properly.

3 Types of variables

3.1 Numeric types

```
In [1]: x=3           # integer

In [2]: y=30000000000 # long integer

In [3]: y
Out[3]: 30000000000L

In [4]: w=1.23        # real

In [5]: z=1.2+3.4j    # complex, j = sqrt(-1)
```

Some arithmetic operations : +, -, *, /, % (modulo) :

```
In [1]: 3. * 8
Out[1]: 24.0

In [2]: 2**8
Out[2]: 256

In [3]: 8%3    # Modulo
Out[3]: 2

In [4]: 3/2    # Warning: integer division
Out[4]: 1

In [5]: 3/2.
Out[5]: 1.5

In [6]: a=3

In [7]: b=2

In [8]: a/b
Out[8]: 1

In [9]: a/float(b)
Out[9]: 1.5
```

Note : by default *Python* computes using "double precision" (i.e. real numbers are encoded using 64 bits). This accuracy is often very useful (even essential) for scientific computing. It is however possible to force the computation to be performed in single-precision (32-bit), but this is not recommended!

```
In [1]: x=ones((2,5),float32)

In [2]: x[0]/3
Out[2]: array([ 0.33333334,  0.33333334,  0.33333334,  0.33333334,  0.33333334], dtype=↩
    float32)

In [3]: 1./3
Out[3]: 0.33333333333333331
```

3.2 Arrays

```
In [1]: x=arange(0,2.0,0.1)    # From 0 to 2 with step 0.1

In [2]: x                      # The full array
Out[2]:
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9])

In [3]: size(x)                # Its size
Out[3]: 20

In [4]: x[0]                   # First element
Out[4]: 0.0

In [5]: x[1]
Out[5]: 0.10000000000000001

In [6]: x[19]                  # Last element
```

```
Out[6]: 1.9000000000000001

In [7]: x[20]                # Not an element!
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)

IndexError: index out of bounds
```

Another way to declare an array :

```
In [1]: from scipy import linspace

In [2]: x=linspace(-10.,10.,100)

In [3]: size(x)
Out[3]: 100

In [4]: x[0]
Out[4]: -10.0

In [5]: x[99]
Out[5]: 10.0
```

or :

```
In [1]: x=mgrid[-10:10:100j]
```

Use caution the third entry must be complex in order to specify a number of elements.

One can also initialize arrays with a zero, or other explicit values :

```
In [1]: x=zeros((2,5))

In [2]: y=ones((2,5))

In [3]: a = array([[1,2,3], [4,5,6], [7,8,9]])
```

The usual arithmetic operations are defined on arrays. Be careful however, unlike it Matlab, the product of two matrices is a simple term by term product, not a matrix product. One must "dot" to perform a matrix product :

```
In [4]: b = 2 * a    # Multiplying each term

In [5]: c = a + b    # Term by term summation

In [6]: dot(a,b)     # Matrix product
Out[6]:
array([[ 60,  72,  84],
       [132, 162, 192],
       [204, 252, 300]])

In [7]: a*b          # Term by term product
Out[7]:
```

```
array([[   2,    8,   18],
       [  32,   50,   72],
       [  98,  128,  162]])
```

Slices in arrays :

```
In [1]: t=array([1, 2, 3, 4, 5, 6])

In [2]: t[1:3]     # from 1 to 3e element... CAREFUL
Out[2]: array([2, 3])

In [3]: t[:3]      # from the beginning to 3
Out[3]: array([1, 2, 3])

In [4]: t[3:]      # from 3 to the end
Out[4]: array([4, 5, 6])

In [5]: t[::2]     # from the beginning to the end with steps 2
Out[5]: array([1, 3, 5])
```

Be careful with copying arrays!

For a number the behaviour is “as expected” :

```
In [1]: a=1.0

In [2]: b=a

In [3]: b
Out[3]: 1.0

In [4]: a=0.0

In [5]: b
Out[5]: 1.0
```

But for an array...

```
In [1]: a=zeros((2,2))

In [2]: b=a

In [3]: b
Out[3]:
array([[ 0.,  0.],
       [ 0.,  0.]])

In [4]: a[1,1]=10.0

In [5]: b
Out[5]:
array([[  0.,   0.],
       [  0.,  10.]])

In [6]: c=b.copy()
```

```
In [7]: c
Out[7]:
array([[  0.,   0.],
       [  0.,  10.]])

In [8]: b[1,1]=0.0

In [9]: b
Out[9]:
array([[ 0.,  0.],
       [ 0.,  0.]])

In [10]: c
Out[10]:
array([[  0.,   0.],
       [  0.,  10.]])
```

4 Loops and tests

```
for i in range(N):
    # i takes values 0,1,2,...,N-1 (N elements all together)
```

Be extra careful to the indentation. It is absolutely essential in Python.

It is the indentation that defines the extend of a loop.

```
if (x > 1) and (y < 0):
  # do A
elif y >= 0:    # elif (not required)
  # do B
else:           # else (not required)
  # do C
```

Here again : be careful to the indentation.

5 To quit

To quit *iPython* use “Ctrl d”

6 To run a program

You can either enter commands interactively (as done in this tutorial), or write a program (say “dummy.py”) and run it, from the system `ipython --pylab dummy.py` or from the *iPython* interpreter `run dummy.py`.

7 Visualization

7.1 Function graph

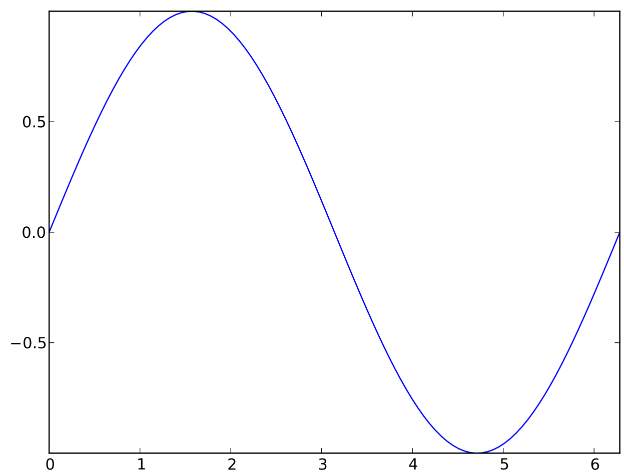
```
In [1]: x=mgrid[0:2*pi:100j]

In [2]: y=sin(x)

In [3]: size(y)
Out[3]: 100

In [4]: plot(x,y)
Out[4]: [<matplotlib.lines.Line2D object at 0x1623e170>]

In [5]: axis('tight')
Out[5]: (0.0, 6.2831853071795862, -0.99987412767387507, 0.99987412767387507)
```



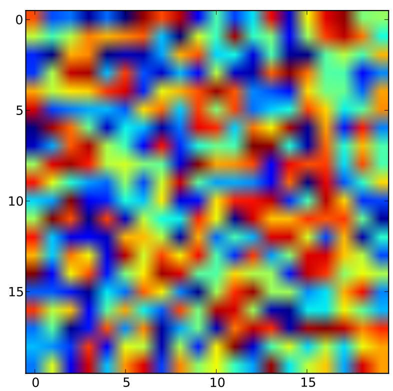
7.2 Vizualisation of 2D data

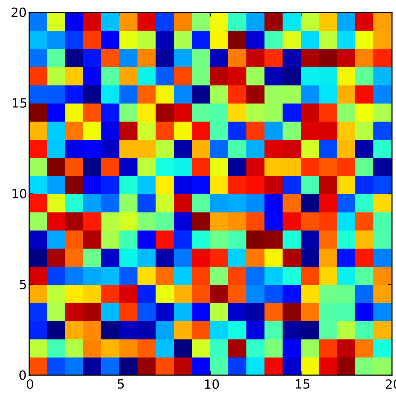
```
In [1]: I=5*rand(20,20)

In [2]: imshow(I)
Out[2]: <matplotlib.image.AxesImage object at 0x16226550>

In [3]: pcolormesh(I)
Out[3]: <matplotlib.collections.PolyCollection object at 0x4ff2b50>

In [4]: axis('scaled')
Out[4]: (0.0, 20.0, 0.0, 20.0)
```





8 To learn more

Use the online help, for example

```
In [1]: help plot
```

(use "q" to exit the online help).

Use the online documentations for Numpy and Scipy

http://docs.scipy.org/doc

as well as the Matplotlib documentation

http://matplotlib.sourceforge.net/ .