Classes modules and some basic graphical functions

Loriano Storchi

loriano@storchi.org

http:://www.storchi.org/

RECURSIVE FUNCTIONS

Recursive functions

- The adjective "recursive" originates from the Latin verb "recurrere", which means "to run back". And this is what a recursive definition or a recursive function does: It is "running back" or returning to itself.
- Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition fulfils the condition of recursion, we call this function a recursive function.

Recursive functions

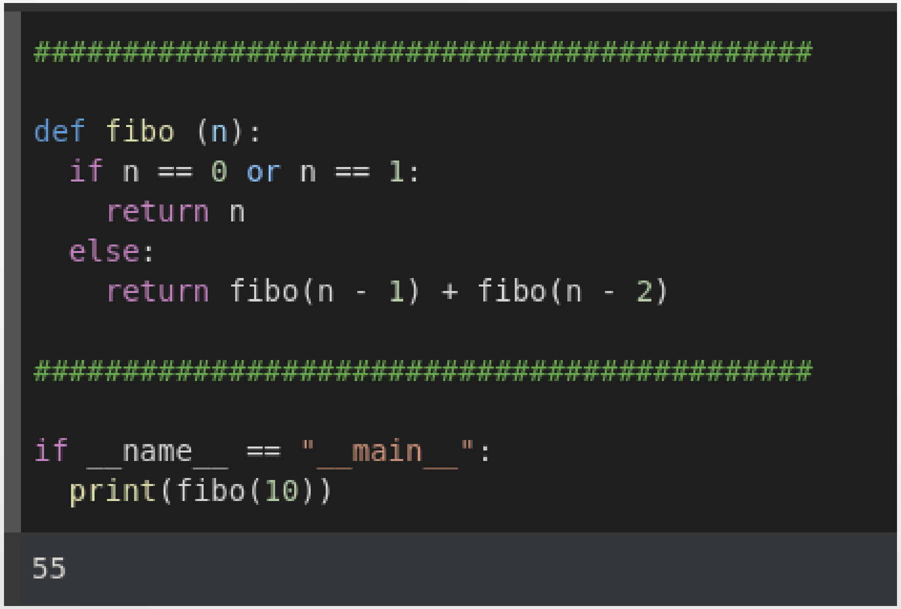
- A function that calls itself

It is a succession in which each term is the sum of the two previous ones

Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1

0, 1, 1, 2, 3, 5, 8...

$$f_0=0$$
$$f_1=1$$
$$f_n= f_{n-1}+ f_{n-2}, \qquad n > 1$$



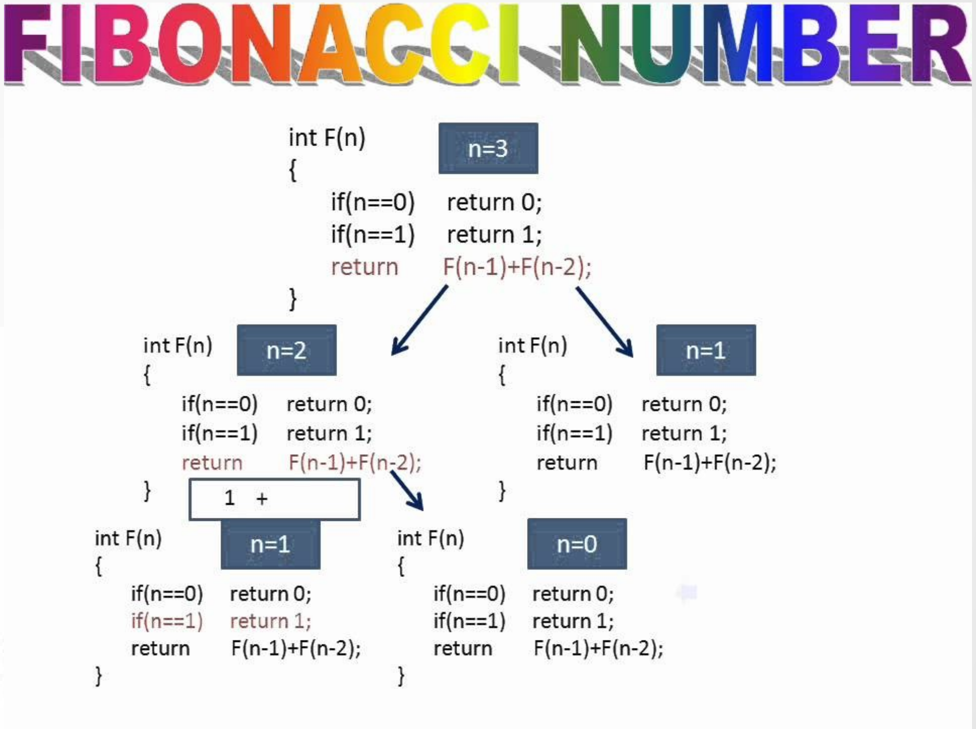
```
############################################

def fibo (n):
  if n == 0 or n == 1:
    return n
  else:
    return fibo(n - 1) + fibo(n - 2)

############################################

if __name__ == "__main__":
  print(fibo(10))
```

```
55
```

Recursive functions

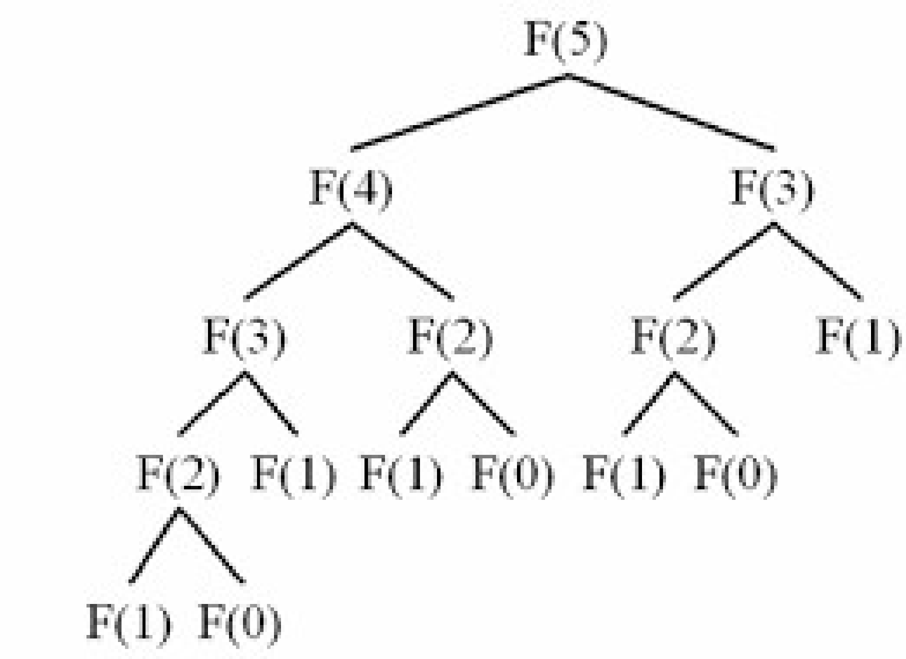
- A function that calls itself

Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1



FIBONACCI NUMBER

```
int F(n)        n=3
{
    if(n==0)   return 0;
    if(n==1)   return 1;
    return     F(n-1)+F(n-2);
}

int F(n)   n=2                      int F(n)   n=1
{                                   {
    if(n==0)   return 0;                if(n==0)   return 0;
    if(n==1)   return 1;                if(n==1)   return 1;
    return     F(n-1)+F(n-2);           return     F(n-1)+F(n-2);
}   1  +                            }

int F(n)   n=1                 int F(n)   n=0
{                              {
    if(n==0)   return 0;           if(n==0)   return 0;
    if(n==1)   return 1;           if(n==1)   return 1;
    return     F(n-1)+F(n-2);      return     F(n-1)+F(n-2);
}                              }
```



MODULES

Modules

- In programming, a definition of library can be following: a collection of pre-compiled and non-volatile routines used by programs. These routines, sometimes called modules, can include configuration data, documentation, message templates, subroutines, classes, values or type specifications.
- Code reuse. Code reuse, also called software reuse, is the use of existing software, or software knowledge, to build new software, following the reusability principles.

Modules

- Modules are source code, so a collection of data, functions and classes that can be imported and used within a source. Consider a module to be the same as a code library.
- Writing your own module it is indeed simple. The code inside the module is executed when this is imported

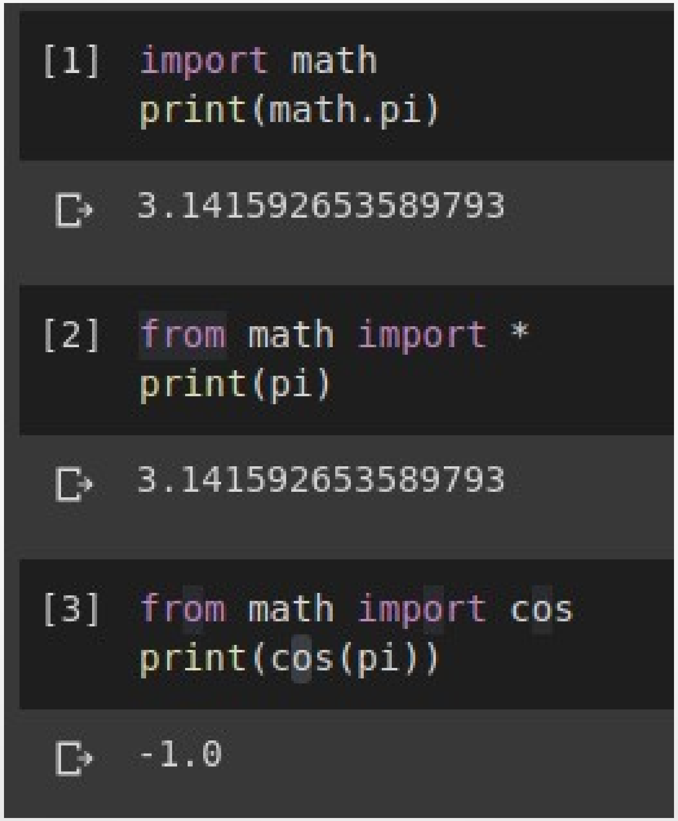
  https://github.com/lstorchi/teaching/tree/master/modules/common

Modules

- **import module**
  - In this case, to use a module member I will write: **module.function or module.data**
- **from modulo import func1, func2, class1**
  - In such a case it imports only some functions or classes from the module, and I can call them locally simply writing: **func1 (par1, par2)**
- **from modulo import ***
  - In this case, it imports all the classes, data and functions from the module. It is risky as quite surely I will “dirty the local namespace”

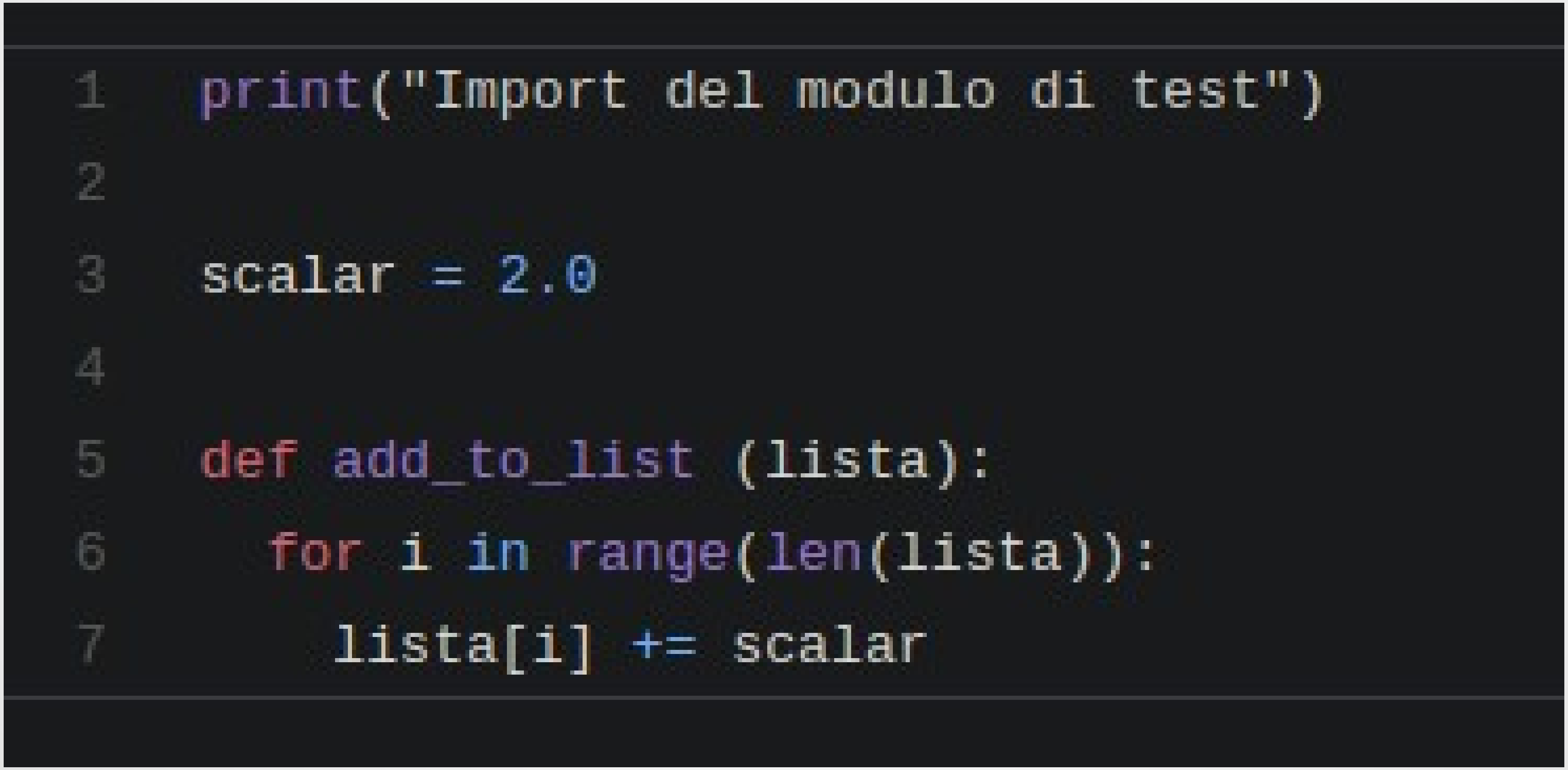
Modules

- Modules are source code, so a collection of data, functions and classes that can be imported and used within a source. Consider a module to be the same as a code library.



```
[1]  import math
     print(math.pi)

     3.141592653589793

[2]  from math import *
     print(pi)

     3.141592653589793

[3]  from math import cos
     print(cos(pi))

     -1.0
```

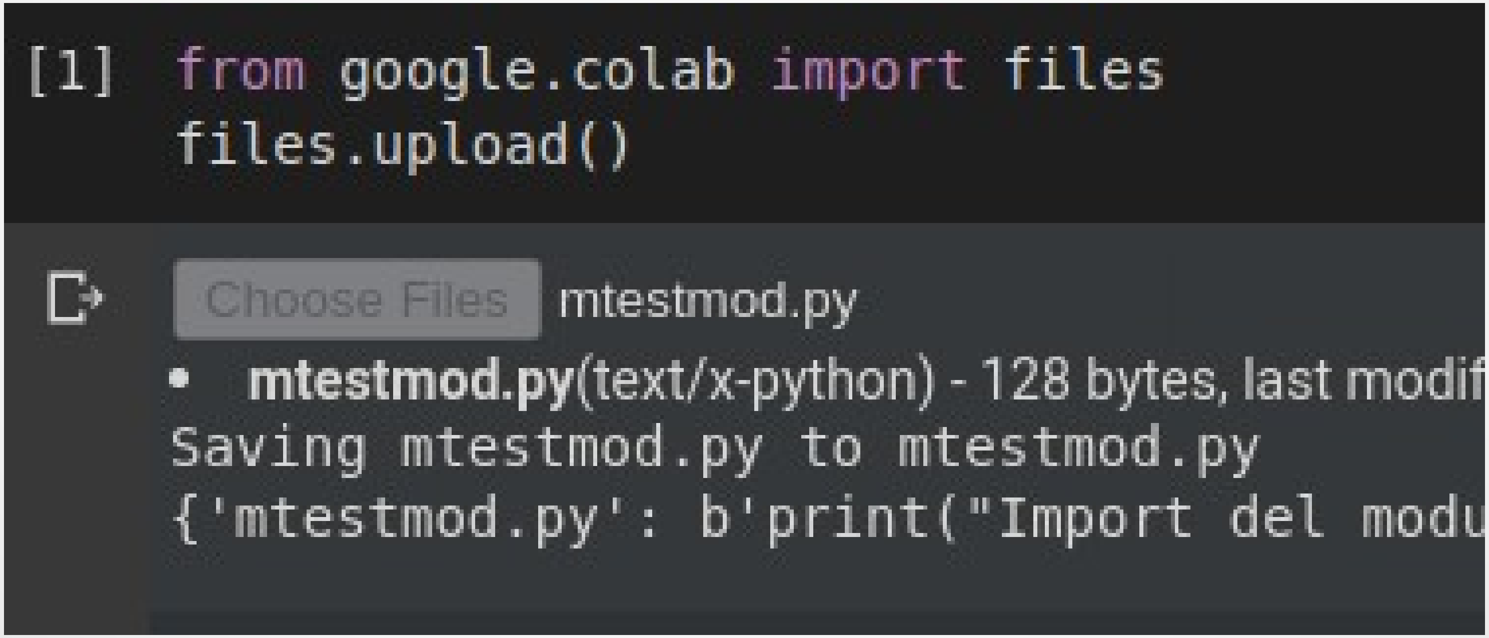
Modules



```python
print("Import del modulo di test")

scalar = 2.0

def add_to_list (lista):
  for i in range(len(lista)):
    lista[i] += scalar
```

Modules

- Import the module in the colab notebook

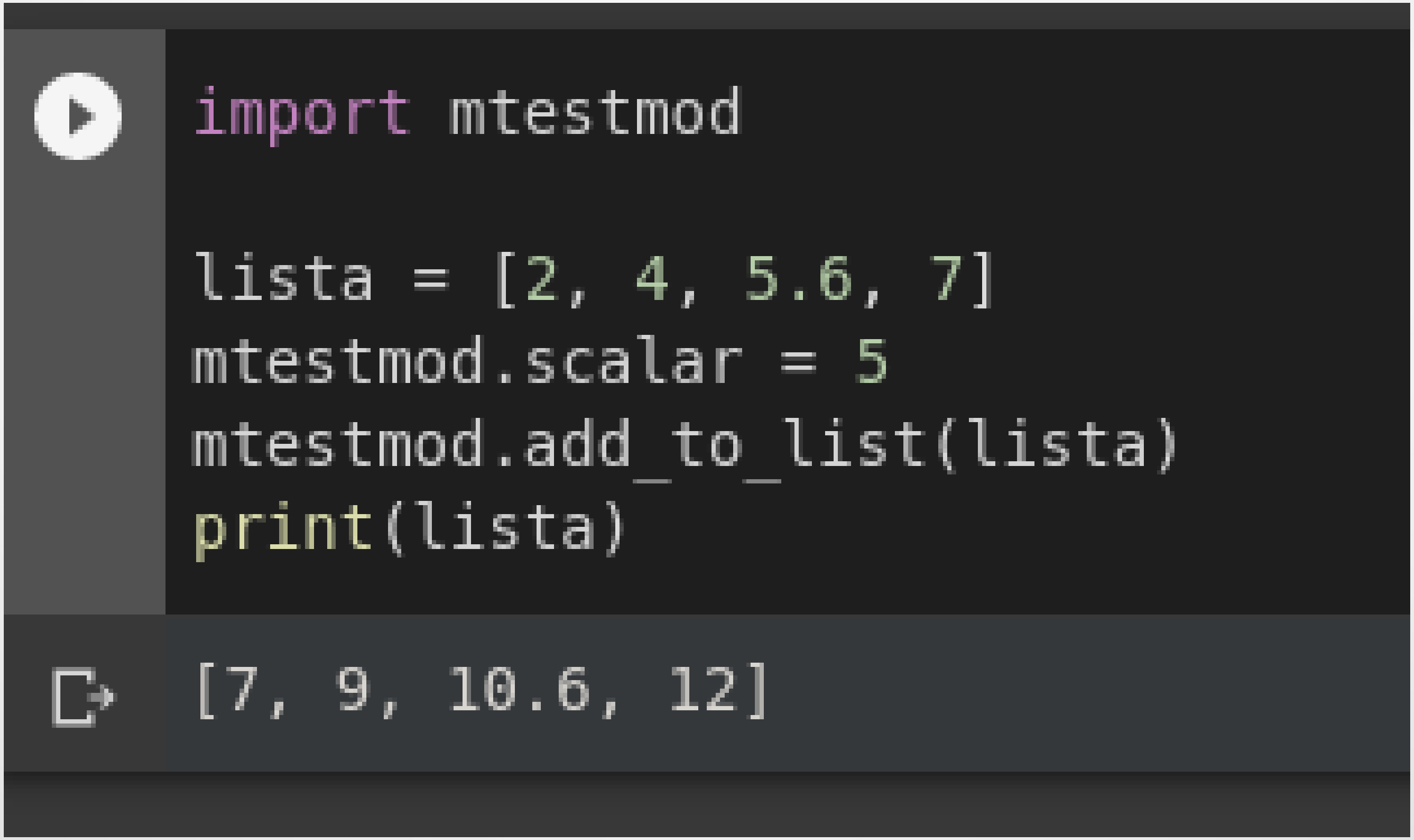


```
[1]  from google.colab import files
     files.upload()
```

```
mtestmod.py
•  mtestmod.py(text/x-python) - 128 bytes, last modif
Saving mtestmod.py to mtestmod.py
{'mtestmod.py': b'print("Import del modu
```

Modules

- Use the module



```python
import mtestmod

lista = [2, 4, 5.6, 7]
mtestmod.scalar = 5
mtestmod.add_to_list(lista)
print(lista)
```

```
[7, 9, 10.6, 12]
```

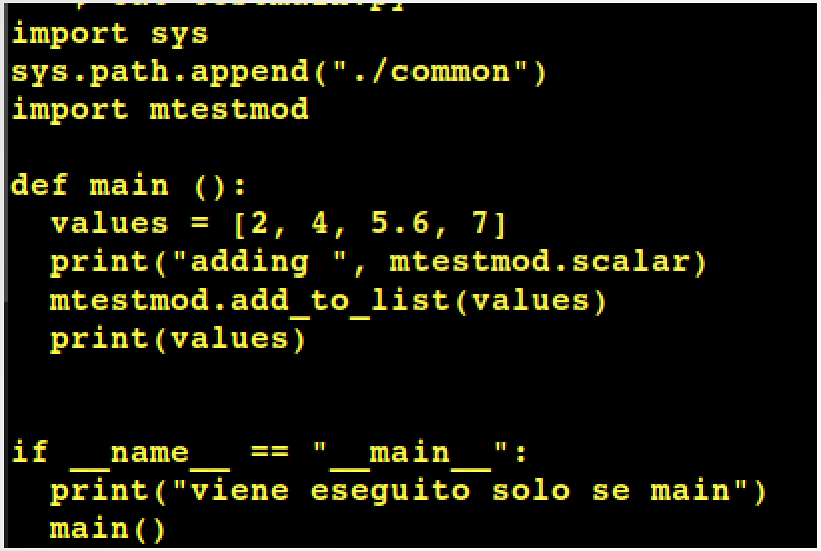
If __name__ == “__main__”

- In python happens sometimes to be faced with a block of code like the following:

```
def main ():
   …
if __name__ == “__main__”:
    main ()
```

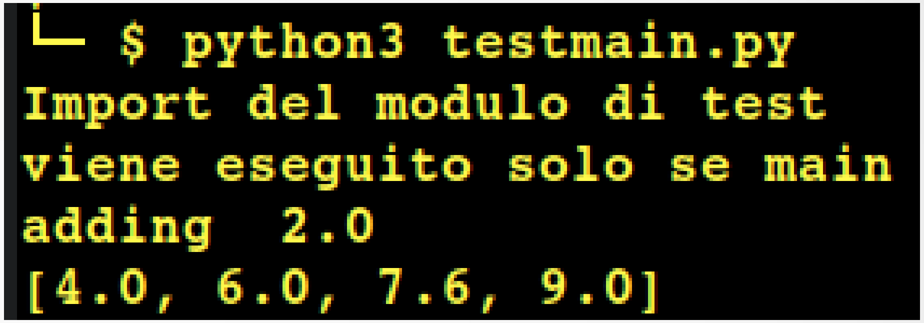
The meaning can appear cryptic, but in reality it actually makes the main function code run only if the file is not imported as a module. A simple example should clarify the operation

__main__

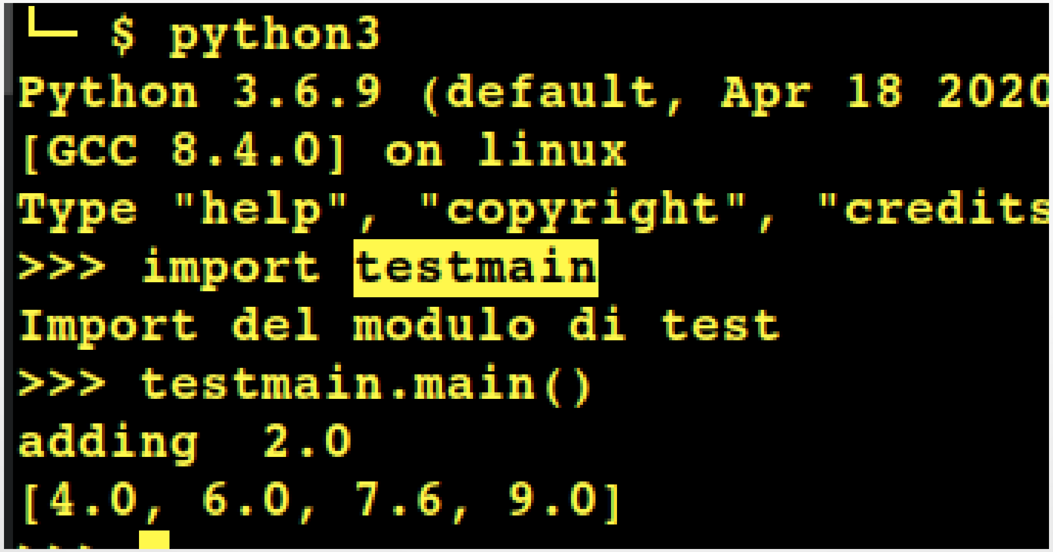


```
import sys
sys.path.append("./common")
import mtestmod

def main ():
  values = [2, 4, 5.6, 7]
  print("adding ", mtestmod.scalar)
  mtestmod.add_to_list(values)
  print(values)


if __name__ == "__main__":
  print("viene eseguito solo se main")
  main()
```

What happens when the written code is imported as a module? The main () function is not performed, unlike ...



```
└─ $ python3 testmain.py
Import del modulo di test
viene eseguito solo se main
adding  2.0
[4.0, 6.0, 7.6, 9.0]
```



```
└─ $ python3
Python 3.6.9 (default, Apr 18 2020
[GCC 8.4.0] on linux
Type "help", "copyright", "credits
>>> import testmain
Import del modulo di test
>>> testmain.main()
adding  2.0
[4.0, 6.0, 7.6, 9.0]
```

CLASSES (OOP)

Classes (OOP)

- OOP: this paradigm makes use of objects that are defined according to their characteristics, then attributes (data) and functions (methods).
- Object-Oriented Programming: it helps to structure large programs well and greatly helps reuse the code
- Classes therefore allow to define objects according to their attributes and according to their behavior, then methods. **A class defines the set while a particular object is a specific element.**

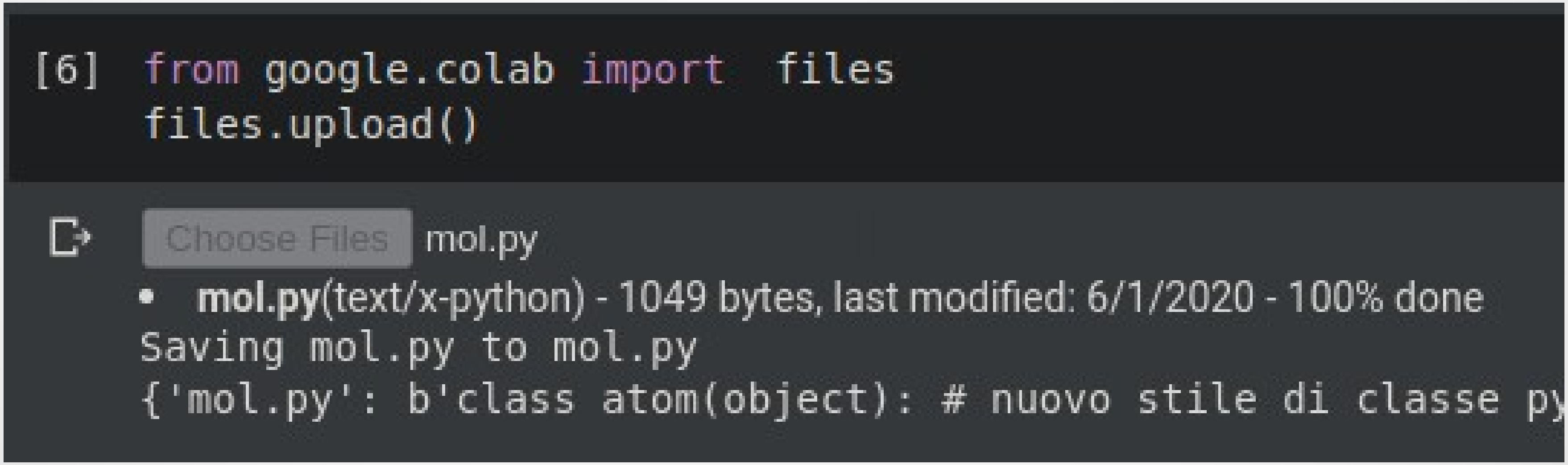
Python classes

- We will deal here only with the essentials of using classes in python
- The **class** keyword introduces the class
- Object creation is simple **object_name = classname (attributes_if_necessary)**
- There is a special method in the class called **__init __ () and is called when the object is created**
- The methods of a class are called using, as has already been seen, **object_name.method (parameters if any)**
- To define a subclass (see inheritance) we use: **class sub2class (parent_class):**

https://github.com/lstorchi/teaching/tree/master/classes

Python classes

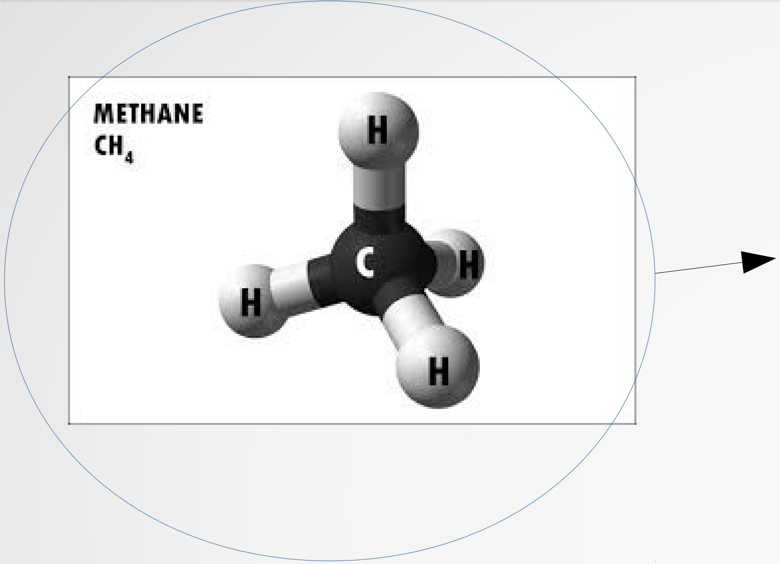
- **Firstly we need to import the file:**



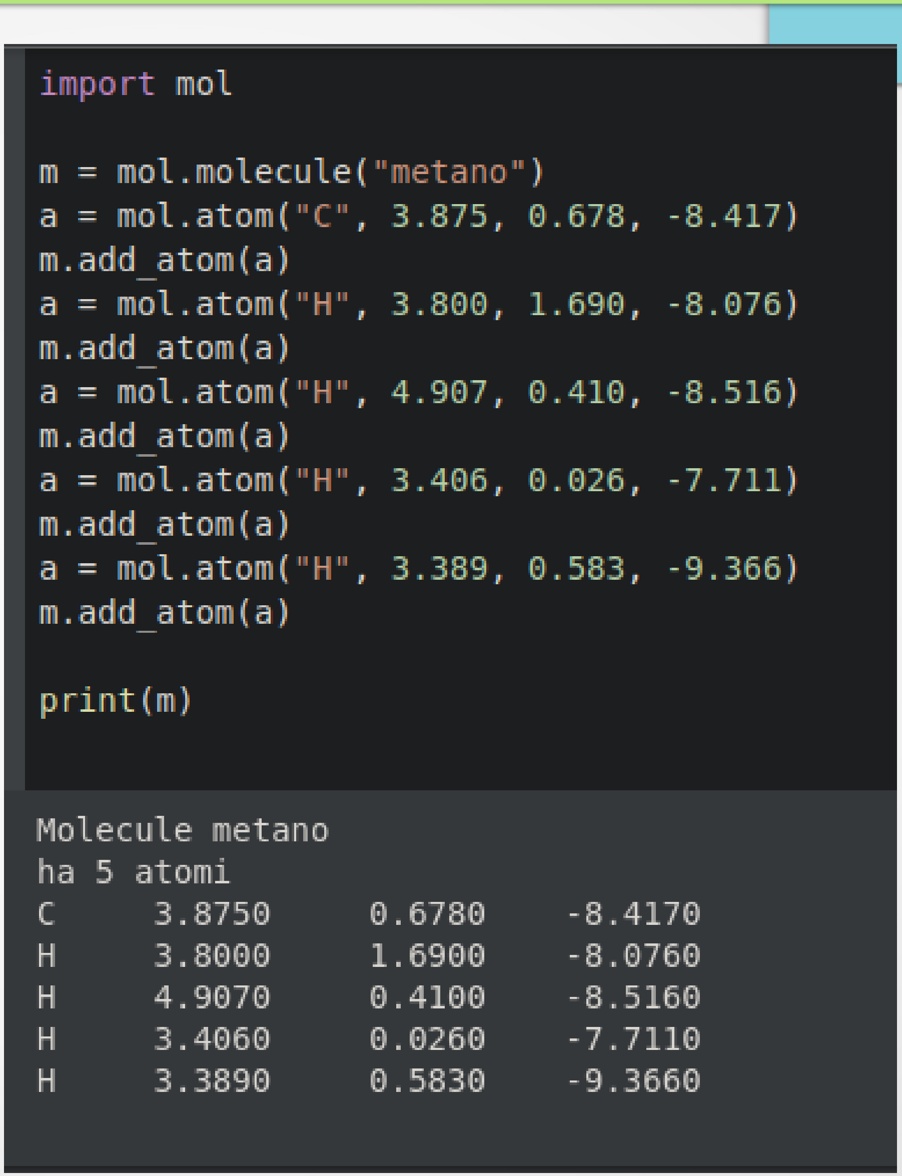
```
[6] from google.colab import  files
    files.upload()
```

```
mol.py
• mol.py(text/x-python) - 1049 bytes, last modified: 6/1/2020 - 100% done
Saving mol.py to mol.py
{'mol.py': b'class atom(object): # nuovo stile di classe py
```

An example



Let's see together the implementation of a molecule class and an atom (see in the repo git classes / mol.py) these two classes will allow us to see in practice the basic elements of the classes in python



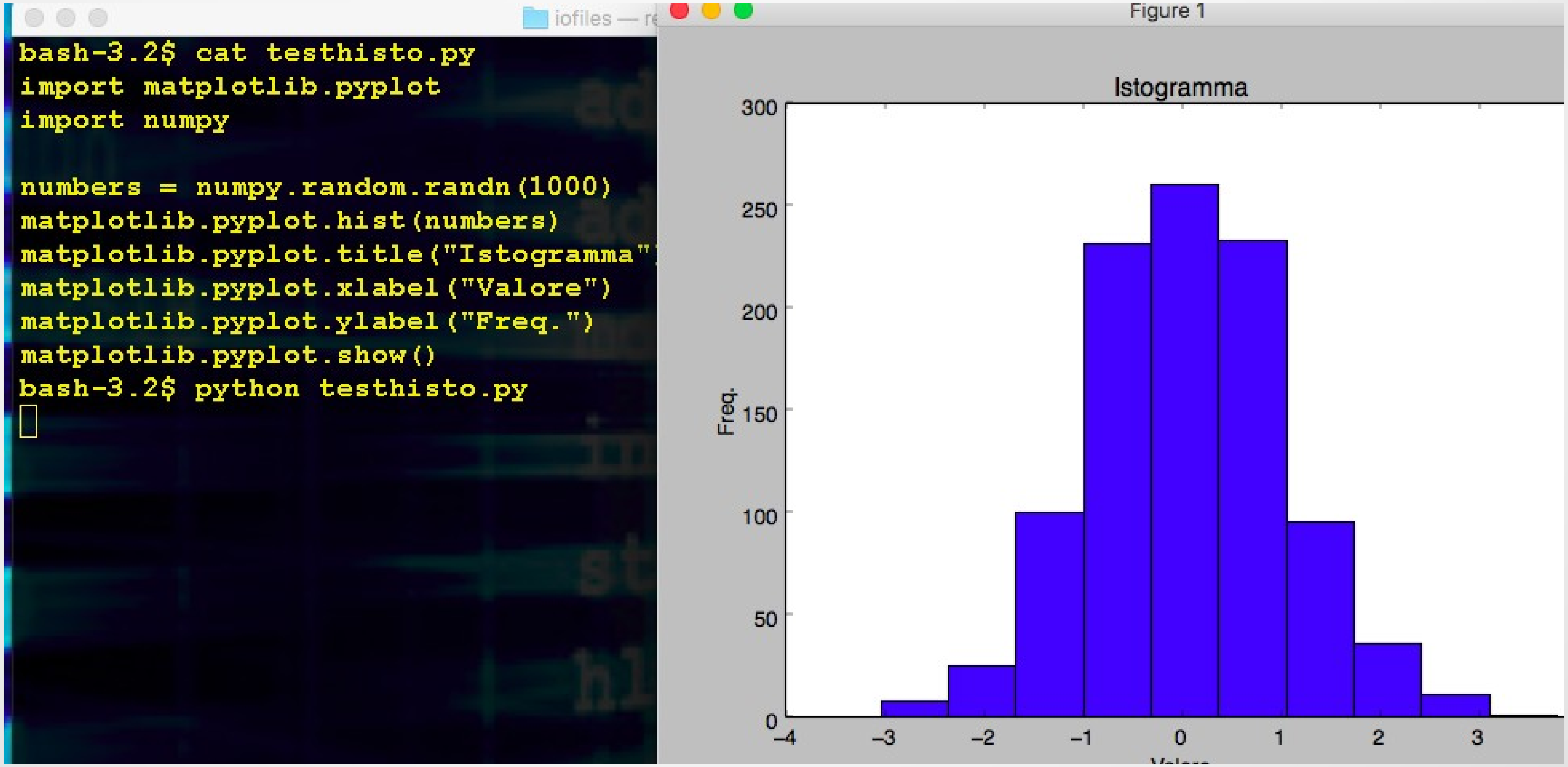
```python
import mol

m = mol.molecule("metano")
a = mol.atom("C", 3.875, 0.678, -8.417)
m.add_atom(a)
a = mol.atom("H", 3.800, 1.690, -8.076)
m.add_atom(a)
a = mol.atom("H", 4.907, 0.410, -8.516)
m.add_atom(a)
a = mol.atom("H", 3.406, 0.026, -7.711)
m.add_atom(a)
a = mol.atom("H", 3.389, 0.583, -9.366)
m.add_atom(a)

print(m)
```

```
Molecule metano
ha 5 atomi
C     3.8750     0.6780     -8.4170
H     3.8000     1.6900     -8.0760
H     4.9070     0.4100     -8.5160
H     3.4060     0.0260     -7.7110
H     3.3890     0.5830     -9.3660
```

A MODULE : MATPLOTLIB

Matplotlib

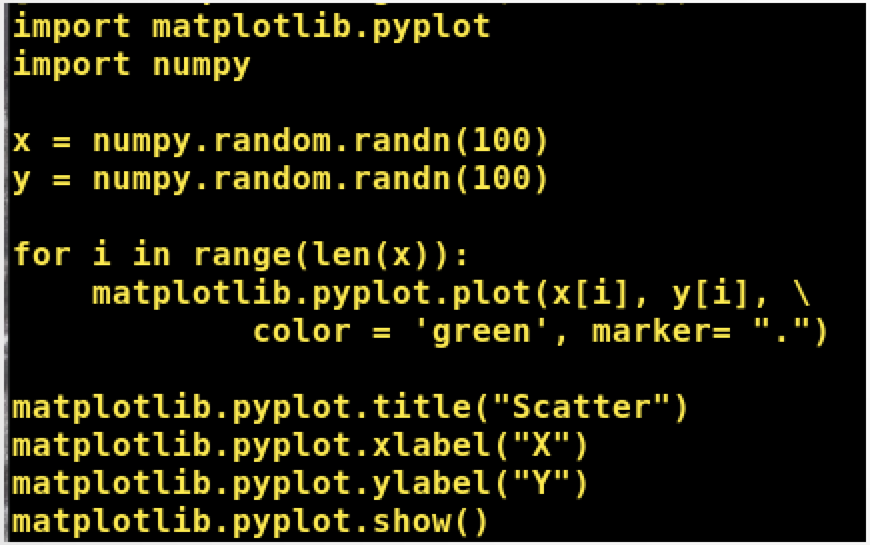
- Let’s try to have a closer look to a module we already used



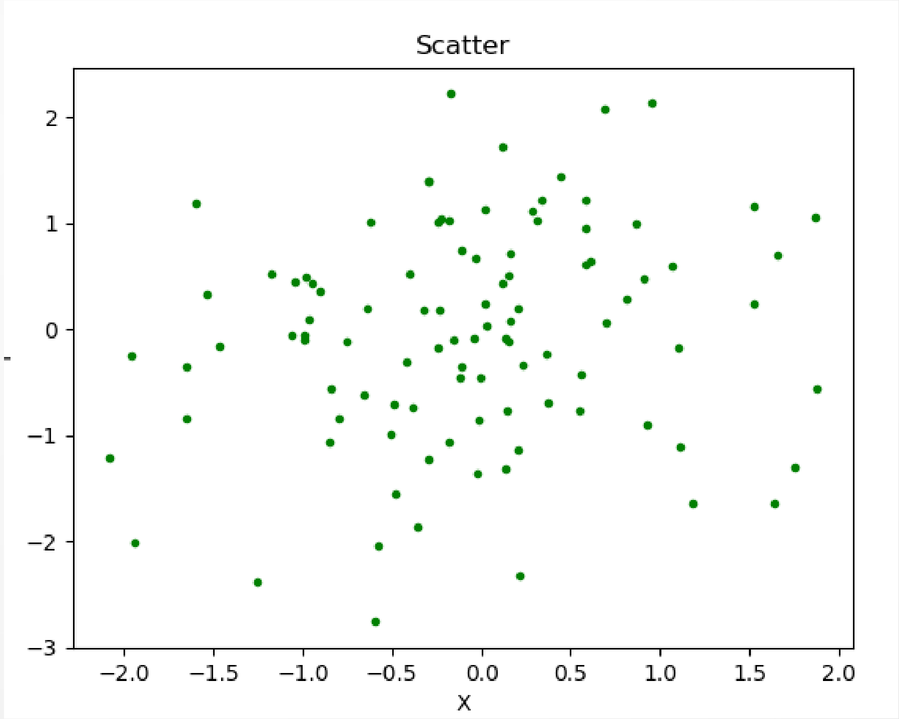
```
bash-3.2$ cat testhisto.py
import matplotlib.pyplot
import numpy

numbers = numpy.random.randn(1000)
matplotlib.pyplot.hist(numbers)
matplotlib.pyplot.title("Istogramma"
matplotlib.pyplot.xlabel("Valore")
matplotlib.pyplot.ylabel("Freq.")
matplotlib.pyplot.show()
bash-3.2$ python testhisto.py
```

Matplotlib

- Let’s try to have a closer look to a module we already used

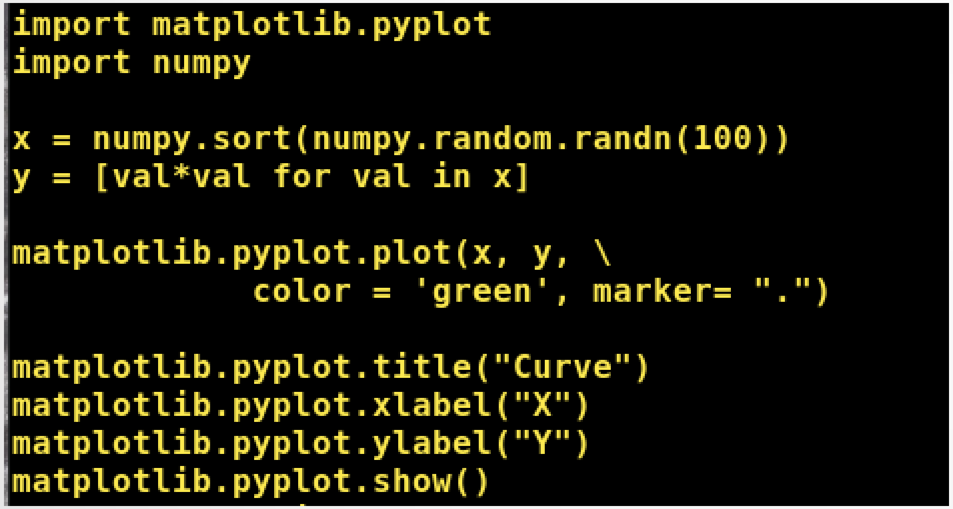


```
import matplotlib.pyplot
import numpy

x = numpy.random.randn(100)
y = numpy.random.randn(100)

for i in range(len(x)):
    matplotlib.pyplot.plot(x[i], y[i], \
            color = 'green', marker= ".")

matplotlib.pyplot.title("Scatter")
matplotlib.pyplot.xlabel("X")
matplotlib.pyplot.ylabel("Y")
matplotlib.pyplot.show()
```

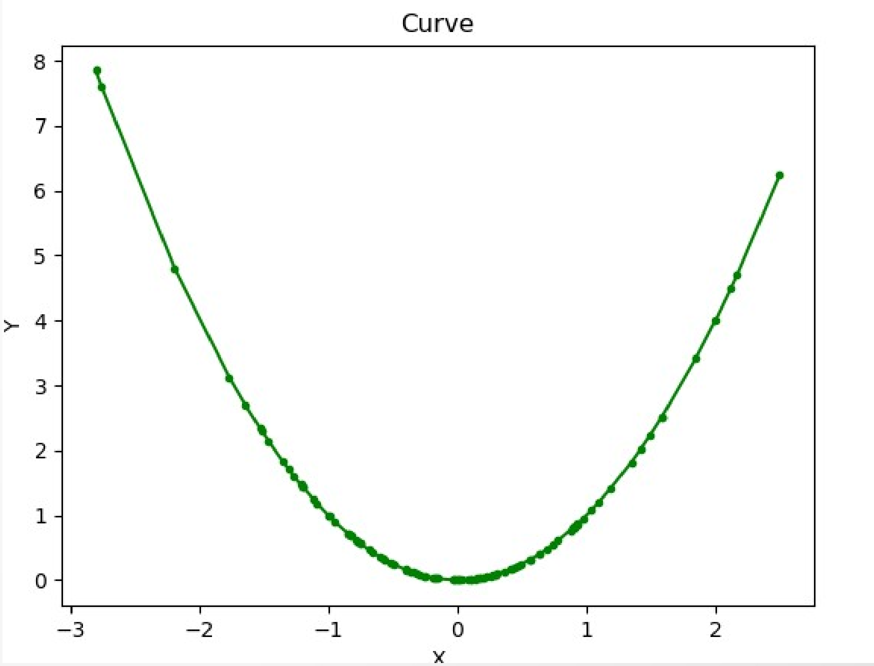


Matplotlib

- Let’s try to have a closer look to a module we already used



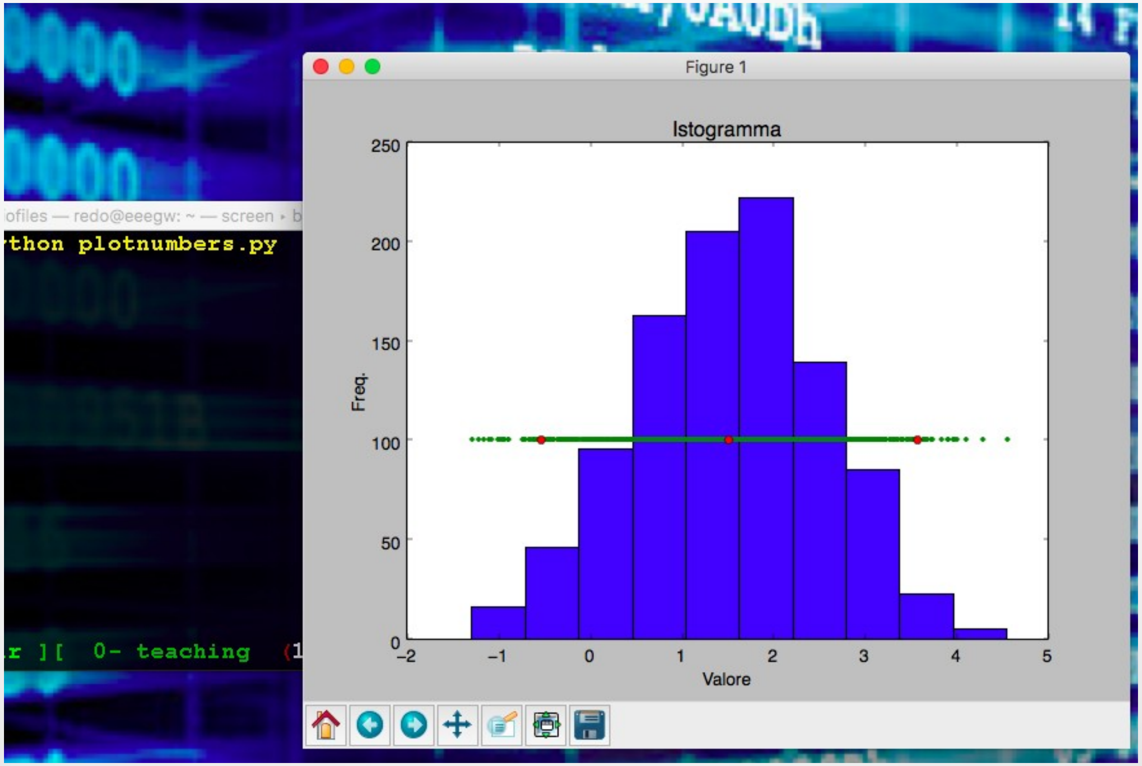
```
import matplotlib.pyplot
import numpy

x = numpy.sort(numpy.random.randn(100))
y = [val*val for val in x]

matplotlib.pyplot.plot(x, y, \
            color = 'green', marker= ".")

matplotlib.pyplot.title("Curve")
matplotlib.pyplot.xlabel("X")
matplotlib.pyplot.ylabel("Y")
matplotlib.pyplot.show()
```



EXERCISE

Exercise

- Write a program that reads all value from numbers.txt and plots the histogram, maybe also all the values and the mean



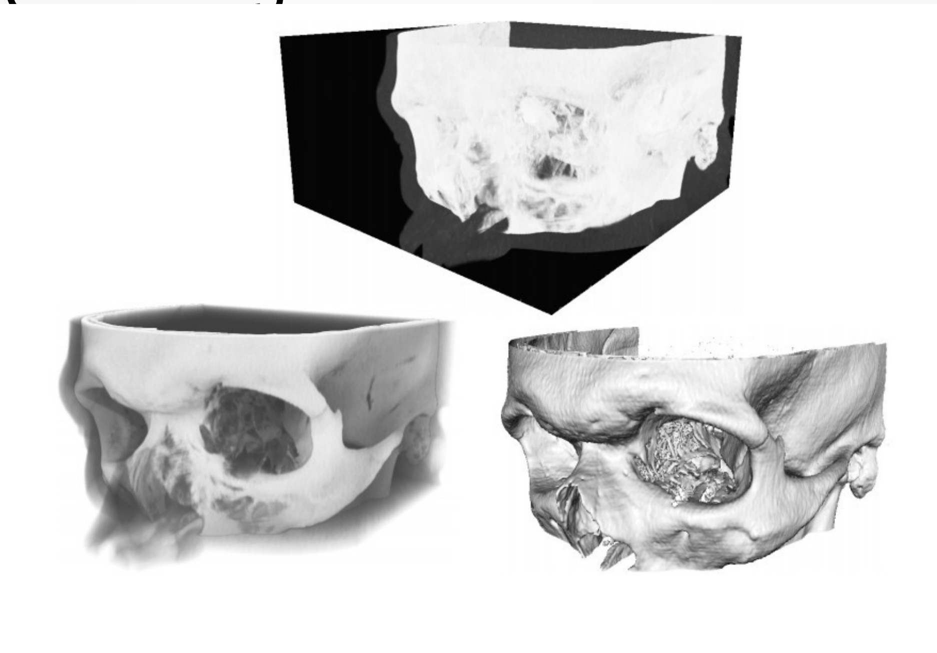
```
ofiles — redo@eeegw: ~ — screen ▸ b
thon plotnumbers.py
```

```
r ][  0- teaching  (1
```

let's have fun with 3D graphics

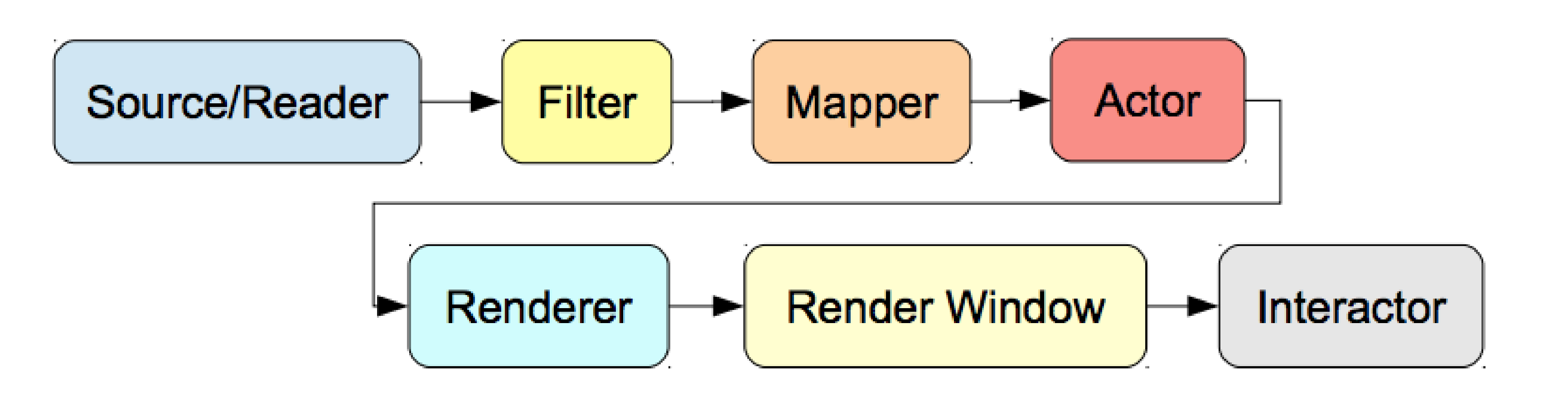
VTK

- We see a profoundly OO framework, developed in C ++, but above all let's have a little fun
- VTK Visualization Toolkit by Kitware Inc.
- 3D scientific visualization, Tcl / Tk bindings, Python, Java, GUI bindings Qt among others



VTK

- To visualize elements in a scene in VTK you have to build a pipeline



- We will not use all the elements of the pipeline but only the essential ones

VTK pipeline

- Sources: VTK puts numerous classes that can be used to construct simple geometric objects such as cubic, spheres, etc etc (for example vtkSpehereSource)
- Maps: maps the data to primitives such as points and lines that can then be viewed by the renderer (for example vtkPolyDatMapper)
- Actors: vtkActor represents an object in the scene
- Rendering: this is the process in which a 3D object plus the specifications of material and light as well as the position of the camera are rendered in a 2D image that can then be displayed on a screen. (vtkRenderer, vtkRendereWindow creates a window in which redere can draw, and instead the vtkRenederWindowInterator class creates a "navigable" window via mouse for example)

A couple of actors

j – joystick (continuous) mode

t – trackball mode

c –camera move mode

a –actor move mode

left mouse – rotate x,y

ctrl - left mouse – rotate z

middle mouse –pan

right mouse –zoom

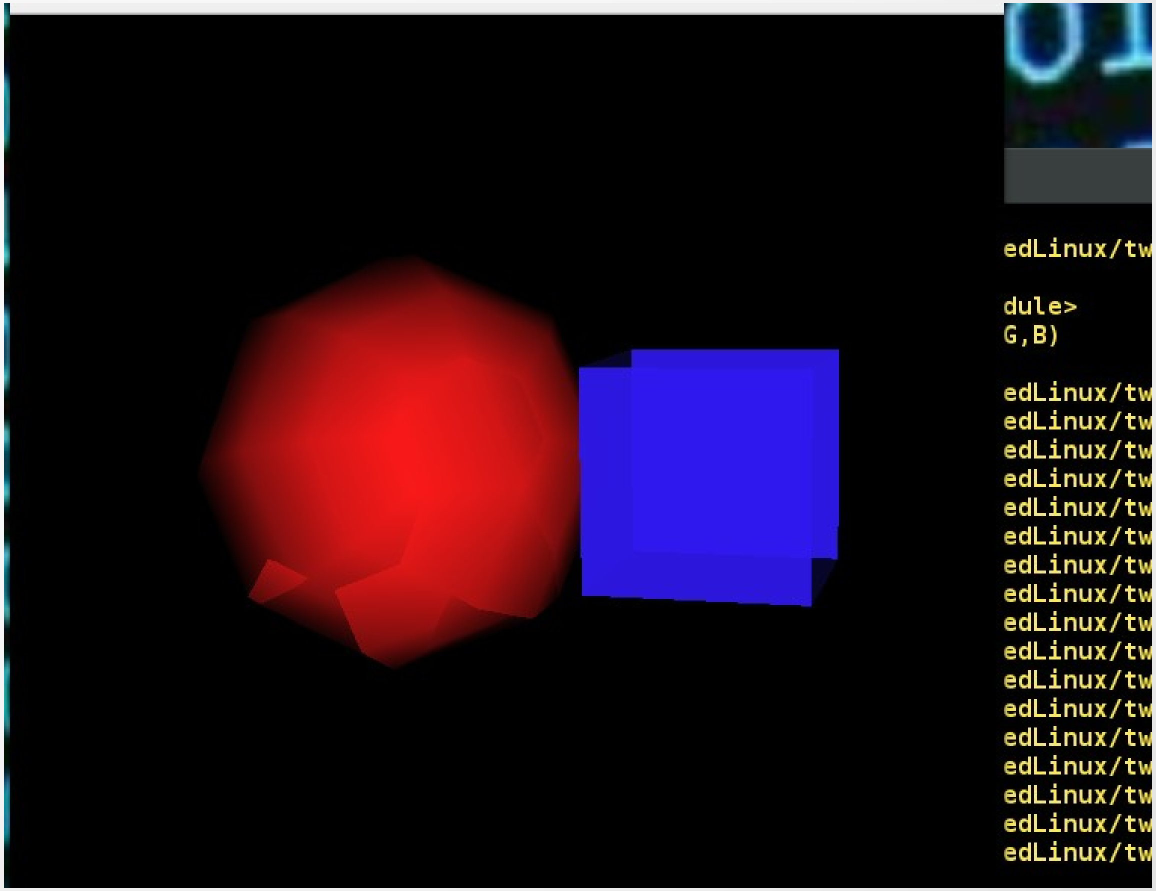
r –reset camera

s/w –surface/wireframe

u –command window

e –exit

Source code vtk/twoactors.py

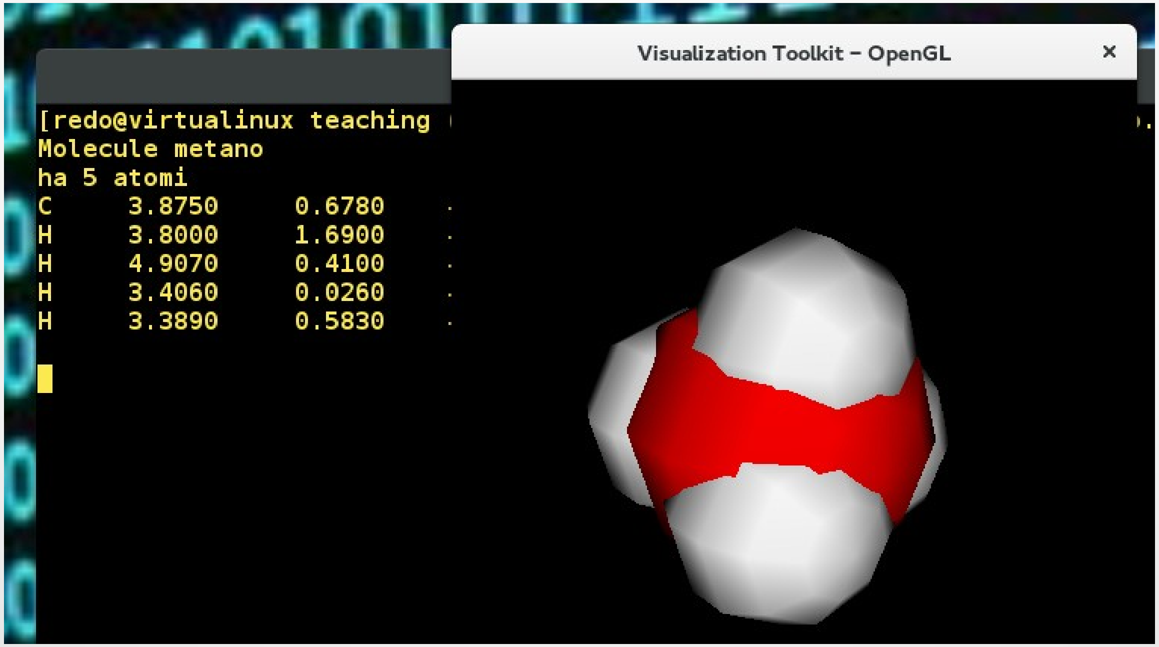


```
edLinux/tw

dule>
G,B)

edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
edLinux/tw
```

EXERCISE

Exercise

- Using the atom and molecule classes seen above, **add a method to the atom class to manage the angstom dimensions of the atom (set and get) and if you want also its "color" in RGB for example**. Then we try to represent the methane molecule



```
[redo@virtualinux teaching
Molecule metano
ha 5 atomi
C      3.8750     0.6780
H      3.8000     1.6900
H      4.9070     0.4100
H      3.4060     0.0260
H      3.3890     0.5830
```

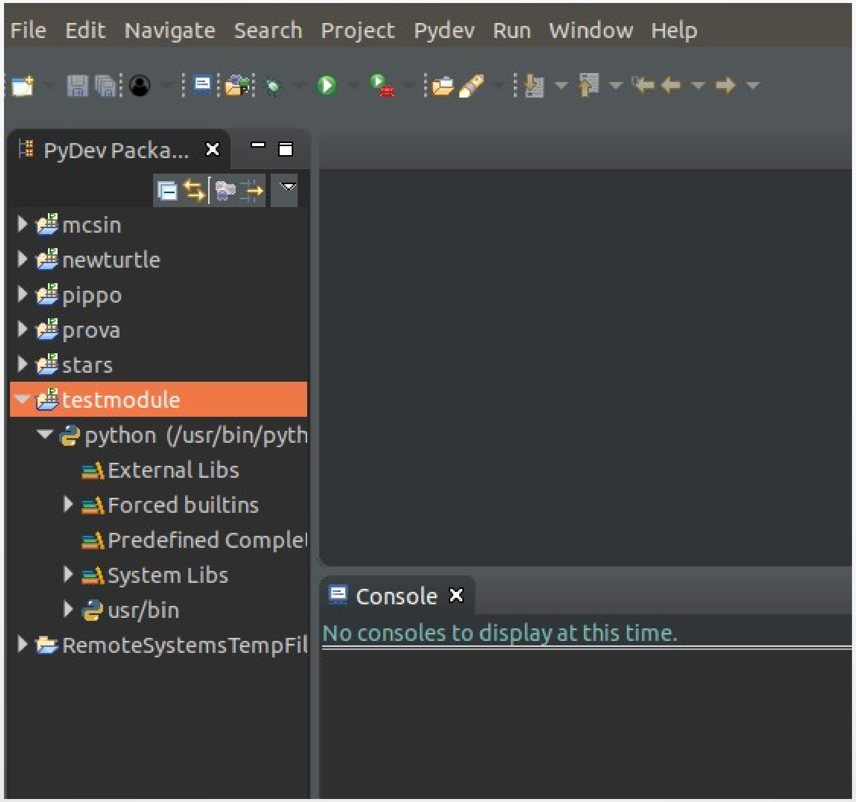
Exercise

```
Create a molecule object
Add all atoms to the molecule
For a in atoms
    Create the sphere source
    Create the mapper ← input source
    Create the actor ← input mapper
    Add the actor to the rederer
```

BACKUP

Write a module

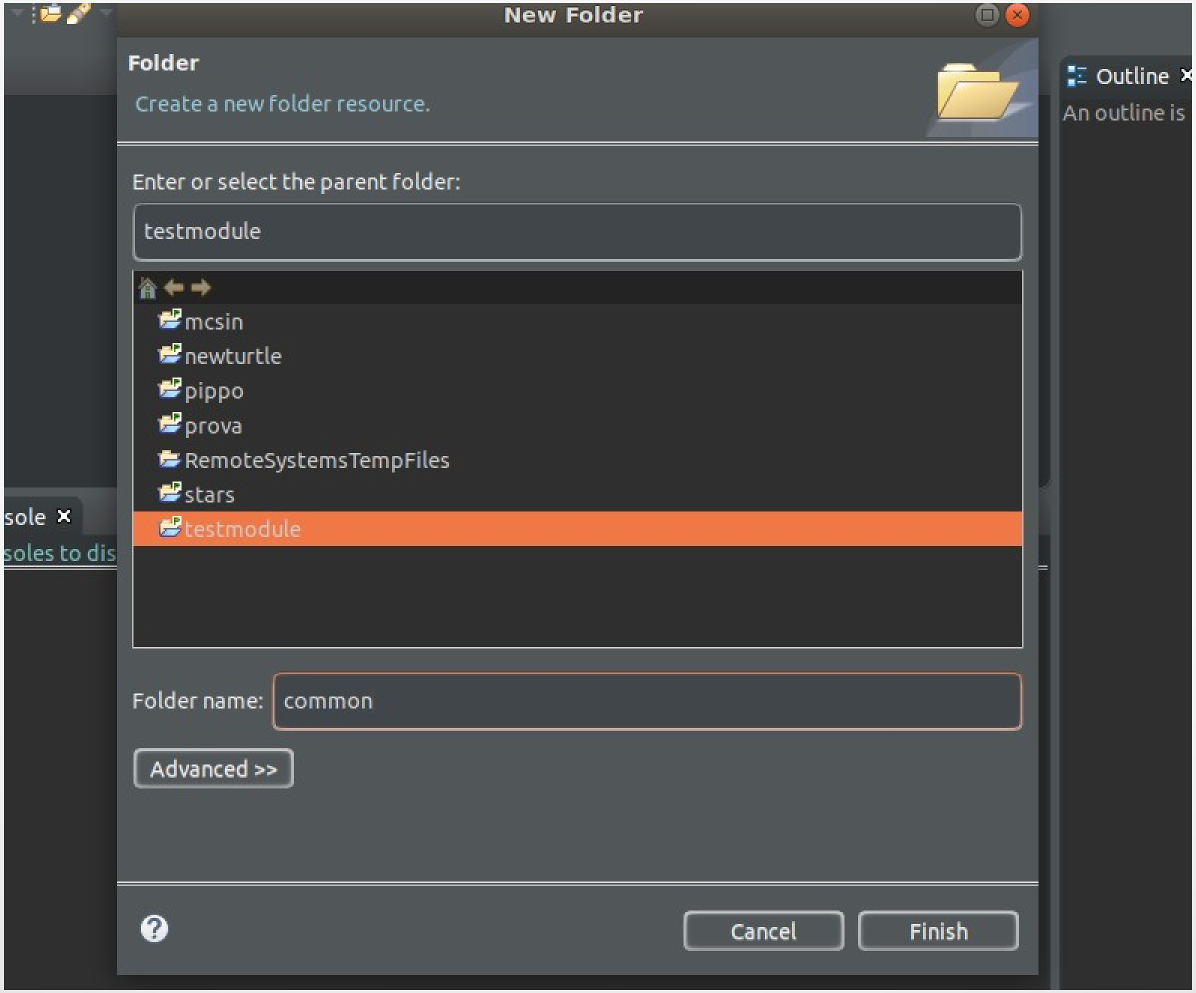
- Writing your own module it is indeed simple. The code inside the module is executed when this is imported
- Start by creating a new PyDev project:



No consoles to display at this time.

Write a module

- Create a new folder



New Folder

Folder

Create a new folder resource.

Enter or select the parent folder:

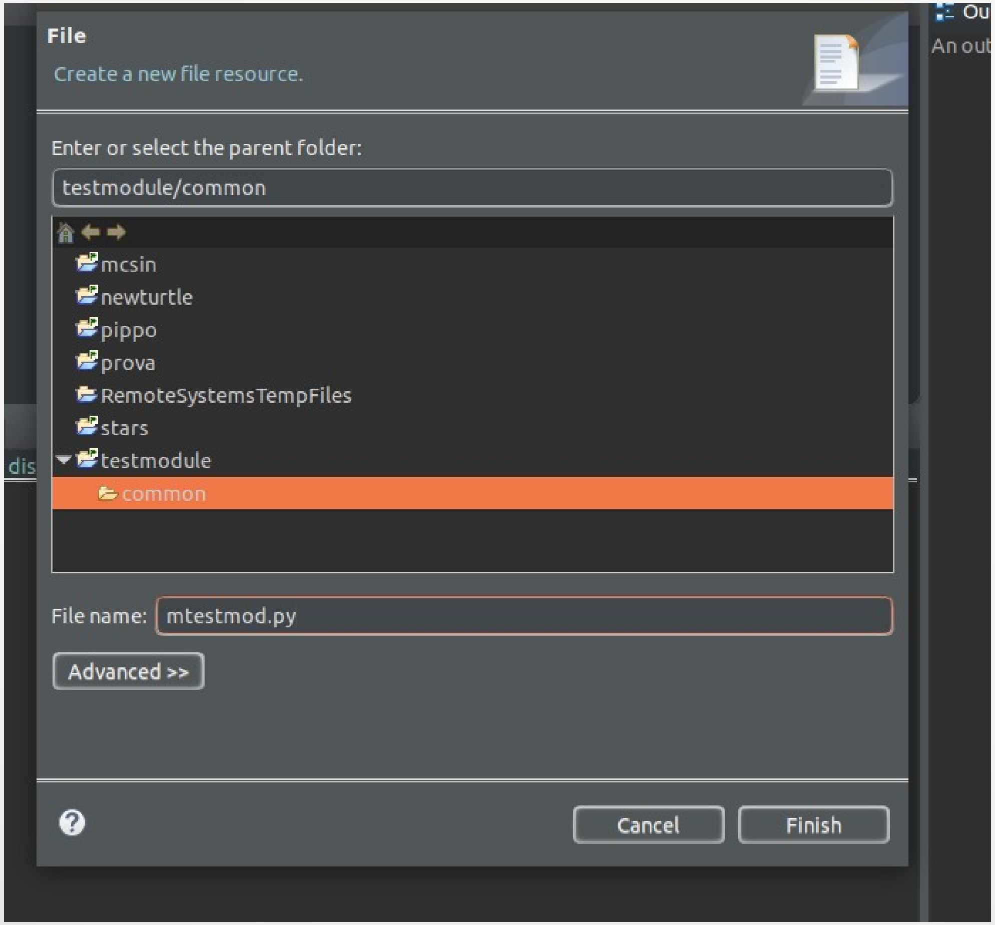
testmodule

- mcsin
- newturtle
- pippo
- prova
- RemoteSystemsTempFiles
- stars
- testmodule

Folder name: common

Write a module

- Create a new file mtestmod.py in common:



File

Create a new file resource.

Enter or select the parent folder:

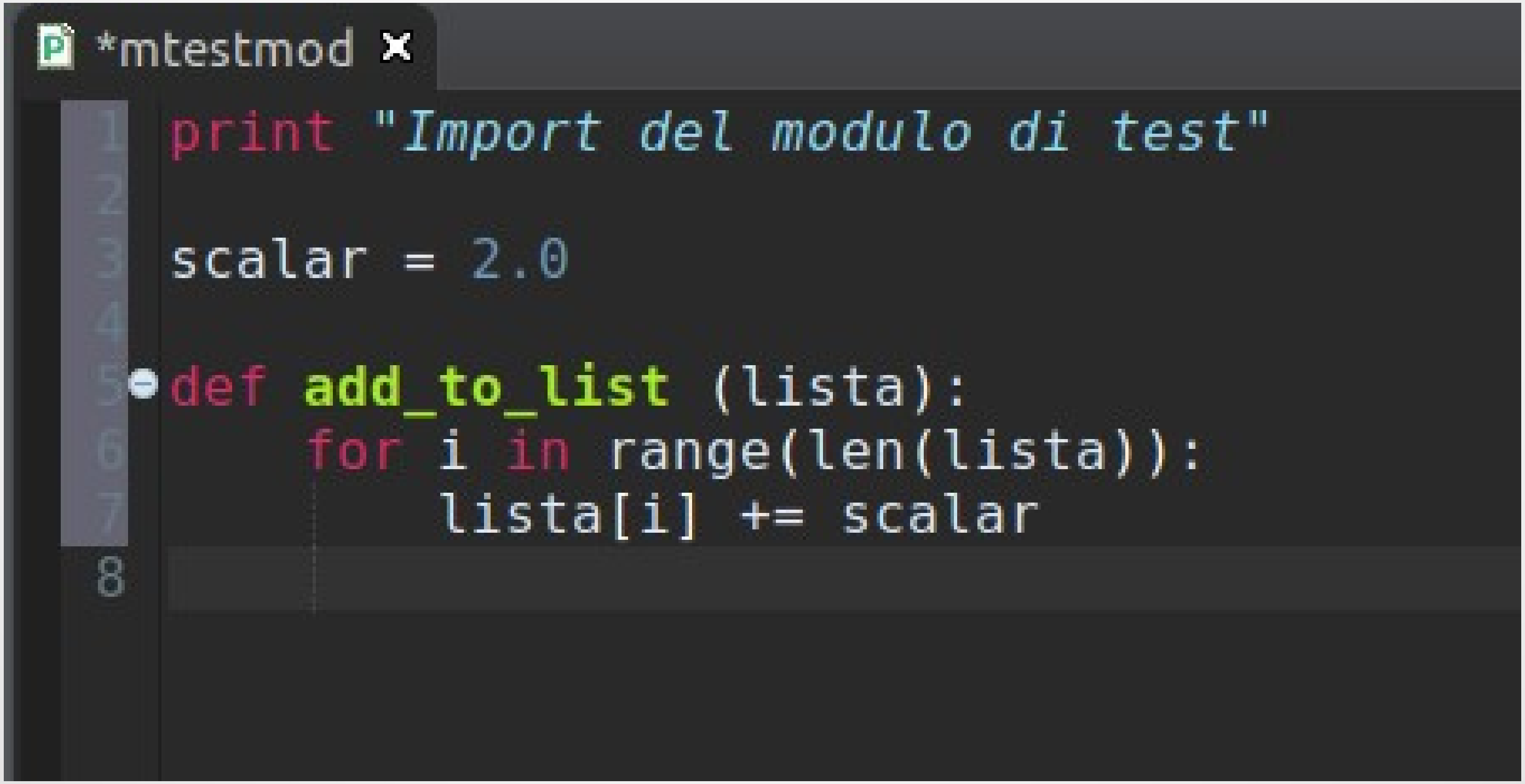
testmodule/common

- mcsin
- newturtle
- pippo
- prova
- RemoteSystemsTempFiles
- stars
- testmodule
  - common

File name: mtestmod.py

Write a module

- mtestmod.py is our test module:

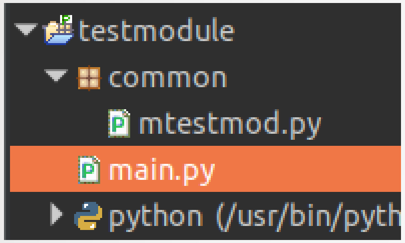


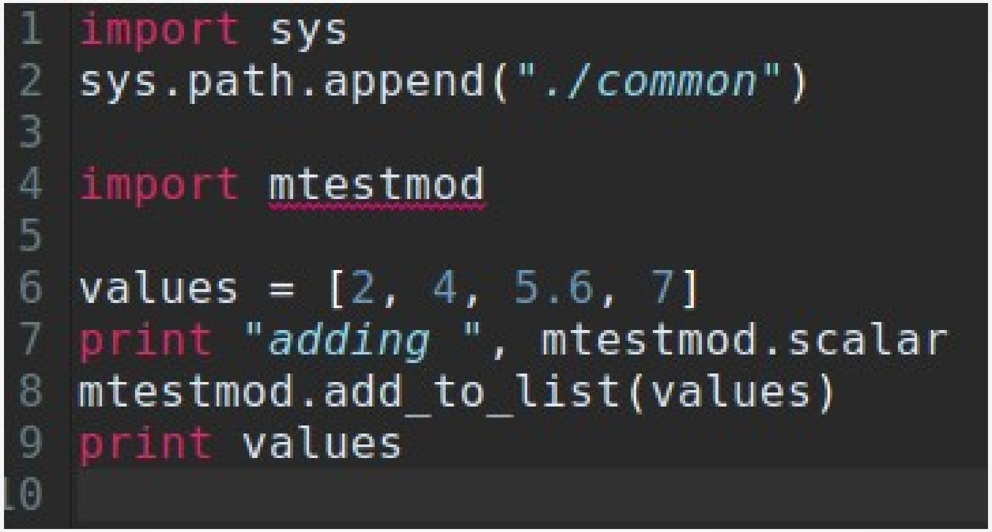
*mtestmod

```
print "Import del modulo di test"

scalar = 2.0

def add_to_list (lista):
    for i in range(len(lista)):
        lista[i] += scalar

```

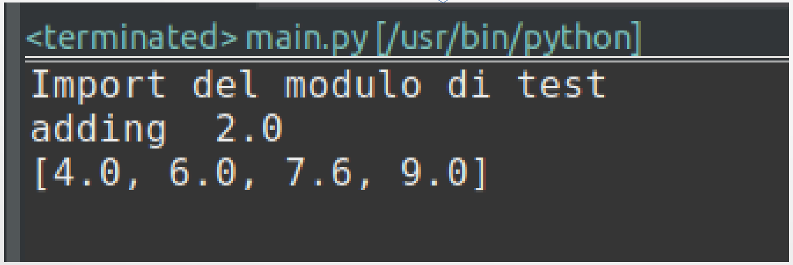
Write a module

- Create the main.py file in the root project directory:





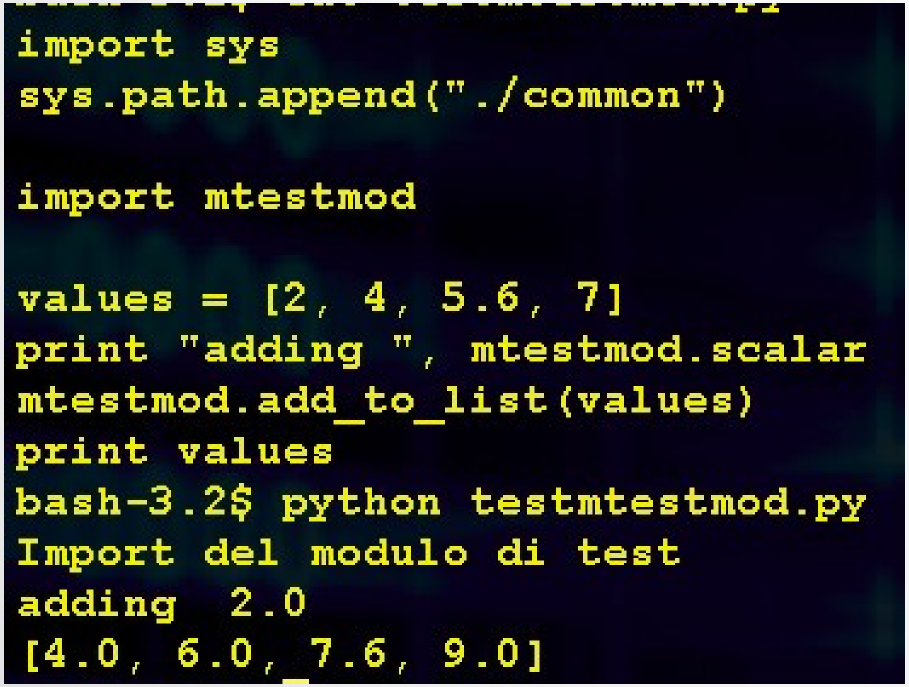
```python
import sys
sys.path.append("./common")

import mtestmod

values = [2, 4, 5.6, 7]
print "adding ", mtestmod.scalar
mtestmod.add_to_list(values)
print values
```



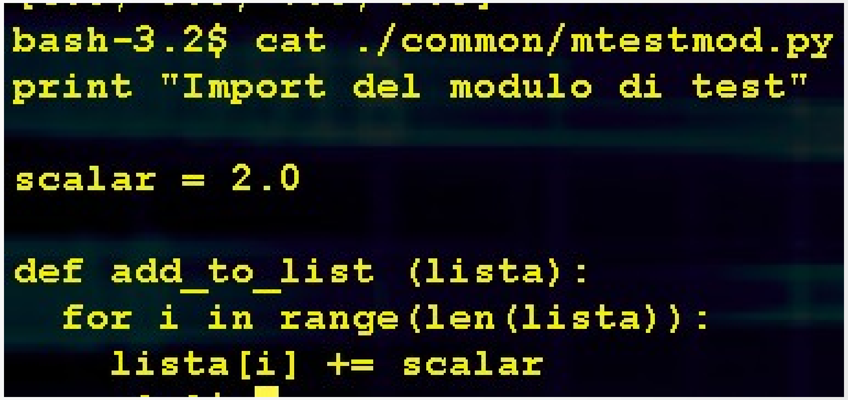
```
<terminated> main.py [/usr/bin/python]
Import del modulo di test
adding  2.0
[4.0, 6.0, 7.6, 9.0]
```

Write a module

- Writing your own module it is indeed simple. The code inside the module is executed when this is imported



```
import sys
sys.path.append("./common")

import mtestmod

values = [2, 4, 5.6, 7]
print "adding ", mtestmod.scalar
mtestmod.add_to_list(values)
print values
bash-3.2$ python testmtestmod.py
Import del modulo di test
adding  2.0
[4.0, 6.0, 7.6, 9.0]
```



```
bash-3.2$ cat ./common/mtestmod.py
print "Import del modulo di test"

scalar = 2.0

def add_to_list (lista):
  for i in range(len(lista)):
    lista[i] += scalar
```

Modules

- We have seen for example by using **sys.path**, it is a normal list that I can edit (lists are mutable), python looks for a module with the name required in all the paths contained in the list
- At start-up, the list will contain some default path, the current directory and possibly the path contained in the PYTHONPATH environment variable
- When a module (file) is imported for the first time **a .pyc file is created**. This is the module "compiled/converted" in byte-code. Subsequent times, unless the module has been modified, python will use this file already "compiled", so the upload will be faster, but not the execution (IMPORTANT)