Python moduli classi e qualche funzione grafica di base

Loriano Storchi

loriano@storchi.org

http::://www.storchi.org/

I moduli

- I moduli sono codice sorgente, quindi una collezione di dati, funzioni e classi che possono essere importati ed usati all'interno di una sorgente

```
import math
print math.pi

from math import *
print pi

from math import cos
print cos(pi)
bash-3.2$ python testimport.py
3.14159265359
3.14159265359
-1.0
```

Scrivere un modulo

- Scrivere un proprio modulo e'decisamente semplice. Il codice alll'interno del modulo viene eseguito quando questo e' importato

```
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
```

I moduli

- import modulo
  - In questo caso per usareun membro del modulo devo usare modulo.funzione o modulo.dato
- from modulo import func1, func2, classe
  - In questo caso importa dal modulo solo alcune funzioni o classi ad esempio, e posso richiamarle localmente semplicemente: func1(par1, par2)
- from modulo import *
  - In questo caso importa tutte le funzioni classi e dati dal modulo. E’ rischioso visto che rischio di sporcare troppo il namespace locale

I moduli

- Abbiamo visto ad esempio l’uso di sys.path, e’ una normale lista che posso modificare (liste sono mutable) python cerca un modulo con il nome richiesto in tutti i path contenuti nella lista
- All’avvio la lista continen alcuni path di default, la directory corrente ed eventualmente i path contenuto nella variabile di anviroment PYTHONPATH
- Quando iìun modulo (file) e’ importato per la prima volta viene creato un file .pyc . Questo e’ il modulo “compilato” in byte-code. Le volte successive, a meno che il modulo non sia stato modificato, python usera’ questo file gia’ “compilato” , il caricamento sara’ quindi piu’ veloce, ma non l’esecuzione (IMPORTANTE)

If __name__ == “__main__”

- Sempre a proposito di moduli: in python succede alle volte di trovarsi di fronte ad un blocco di codice di questo tipo:

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

Il significato puo’ apparire criptico, ma in realta’ di fatto serve a far eseguire il codice della funzione main solo se il file non e’ importato come modulo. Un semplice esempio dovrebbe chiarire il funzionamento

__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()
```

Cosa succede quando il codice scritto viene importato come modulo ? La funzione main() non viene eseguita, diversamente da ...

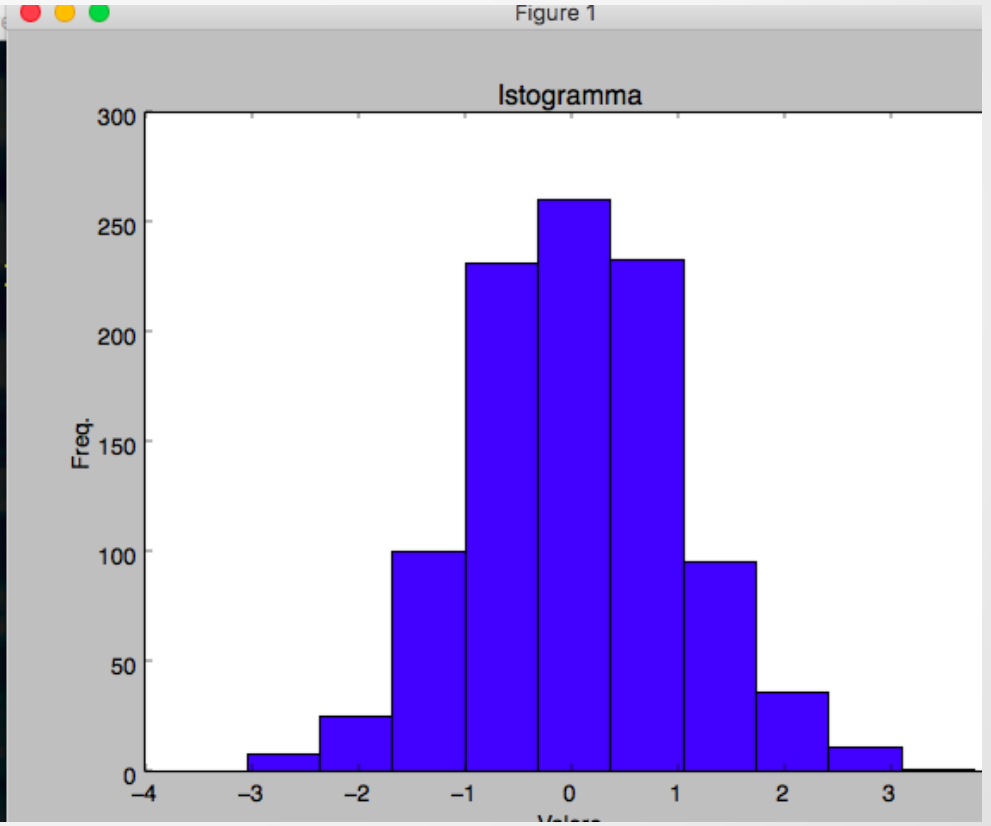
```
  main()
bash-3.2$ python testmain.py
Import del modulo di test
viene eseguito solo se main
adding  2.0
[4.0, 6.0, 7.6, 9.0]
```

```
common          testimport.py   testmain.py     testmtestmod.py
bash-3.2$ python
Python 2.7.10 (default, Jul 30 2016, 19:40:32)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import testmain
Import del modulo di test
>>> ^D
bash-3.2$ ls
common          testimport.py   testmain.py     testmain.pyc    testmtestmod.py
bash-3.2$
```

Matplotlib

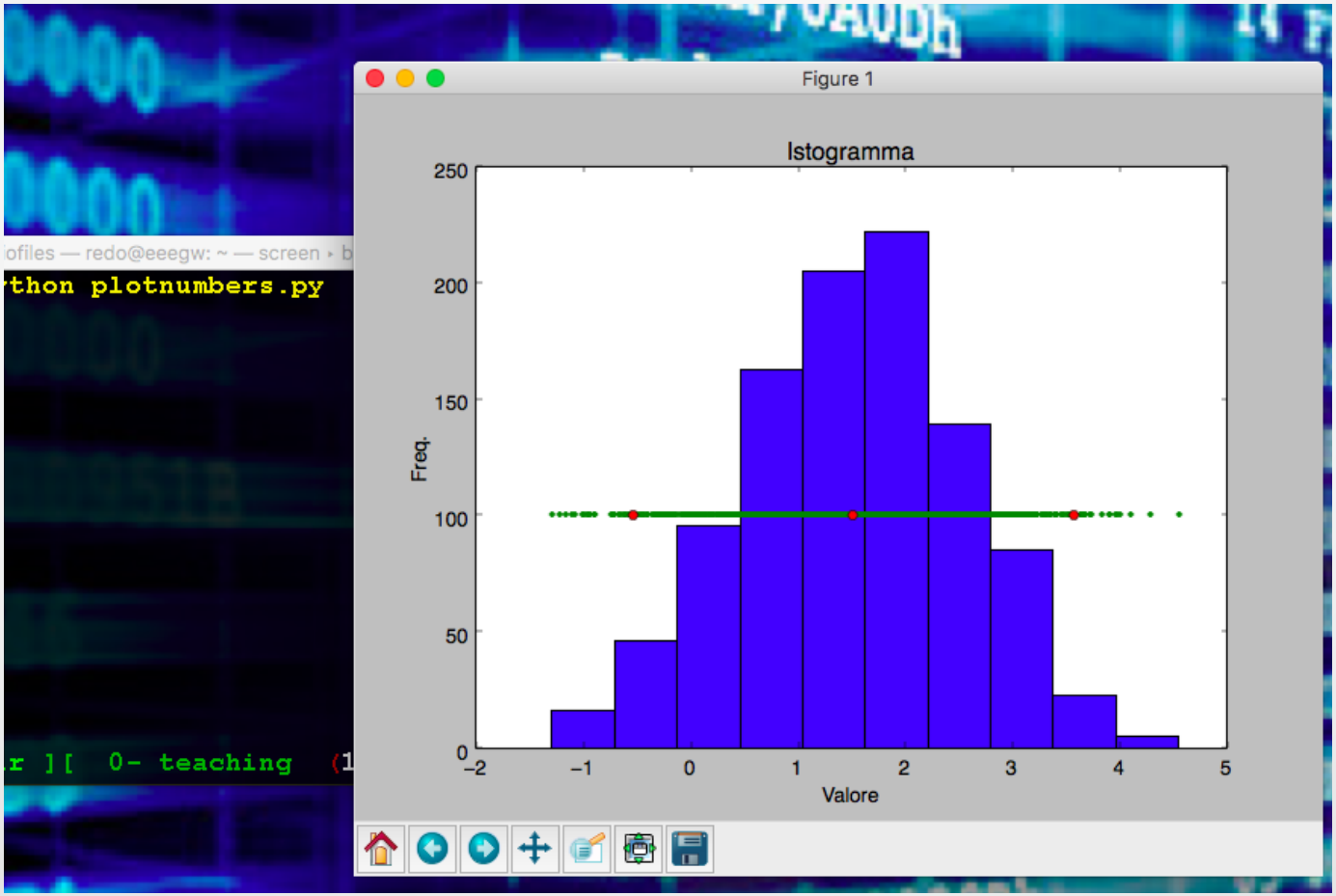
- Rivediamo un modulo che abbiamo gia' visto

```
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
```



Esercizio

- Leggi i punti dal file numbers.txt e plotti i numeri il valore medio e l'istogramma relativo.



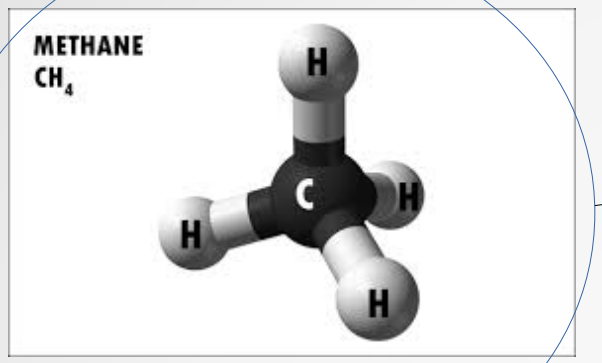
Le classi (OOP)

- I punti salienti dell'OOP. Questo paradigma fa uso di oggetti che sono definiti in base alle loro caratteristiche, quindi attributi, e alle funzioni che posso in qualche modo svolgere, i metodi.
- L'OOP (Object-Oriented Programming) aiuta a strutturare bene programmi di grandi dimensione ed aiuta notevolmente il riuso del codice
- Le classi permettono dunque di definire oggetti in funzione dei loro attributi e in funzione al loro comportamento, quindi metodi. Se volete una classe definisce l'insieme mentre un particolare oggetto e' uno specifico elemento.

Le classi in python

- Affronteremo qui solo l'essenziale dell'uso delle classi in pythom
- La parola chiave **class** introduce la classe
- La creazione dell'oggetto e' semplice **nomeoggetto = nomedellaclasse (attributi_se_necessari)**
- C'e' un metodo speciale nella classe che si chiama **__init__()** e viene richiamato al momento della creazione dell'oggetto
- I metodi di una classe sono richiamati usando, come abbia gia' visto, **nomeoggetto.metodo(eventuali_parametri)**
- Per definire una sottoclasse (vedi ereditarietà) si usa : **class sottoclasse(classe_genitore):**

Classi esempio classe molecola




```
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
```

Vediamo assieme l'implementazione di una classe molecola ed una atomo (vedi nel repo git classes/mol.py) queste due classi ci permetteranno di vedere in pratica gli elementi di base delle classi in python

Una piccola digressione funzioni ricorsive

- Funzioni che richiamano se stesse le funzioni ricorsive

E’ una successione in cui ogni termine e’ la somma dei due precedenti

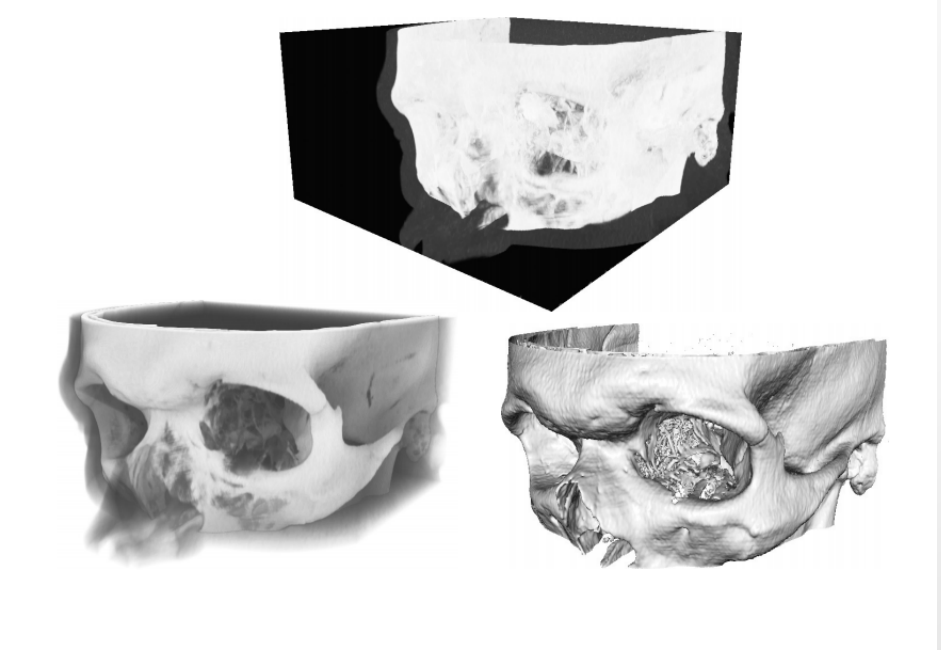
0, 1, 1, 2, 3, 5, 8...

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

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

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)
bash-3.2$ python fibo.py
55
```

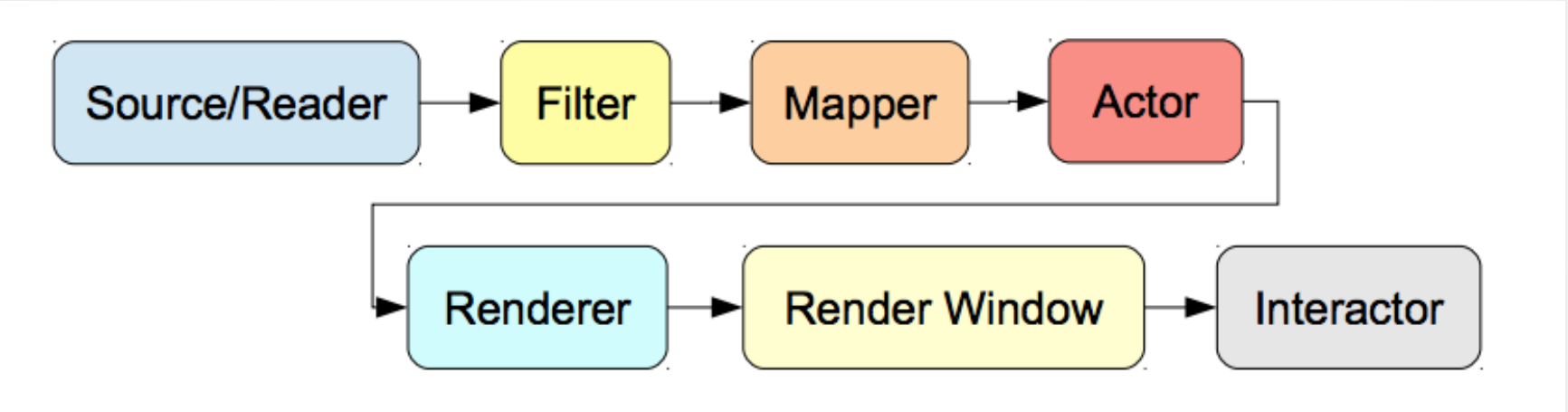
VTK

- Vediamo un framework profondamente OO , sviluppato in C++, ma soprattutto divertiamoci un pochetto
- VTK Visualization Toolkit by Kitware Inc.
- Visualizzazione scientifica 3D , bindings Tcl/Tk, Python, Java, GUI bindings Qt fra gli altri



VTK

- Per visualizzare elementi in una scena in VTK si deve costruire una pipeline



- Noi non usremo tutti gli elementi della pipeline ma solo quelli essenziali

VTK pipeline

- Sources: VTK mette numerose classi che possono essere usate per costruire oggetti geometrici semplici tipo, sfere cubi, etc etc (ad esempio vtkSpehereSource)
- Maps: mappa i dati in primitive come punti e linee che possono poi essere visualizzati dal renderer (ad esempio vtkPolyDatMapper)
- Actors: vtkActor rappresenta un oggetto nella scena
- Rendering: questo e' il processo in cui un oggetto 3D piu' le specifiche ad esempio di materiale e luce oltre che di posizione della camera sono resi in un immagine 2D che puo' essere quindi visualizzata in uno schermo. (vtkRenderer, vtkRendereWindow crea una finestra in cui in redere puo' disegnare, e invece la classe vtkReneder WindowInterator crea una finestra "navigabile" via maouse ad esempio)

Un paio di attori

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

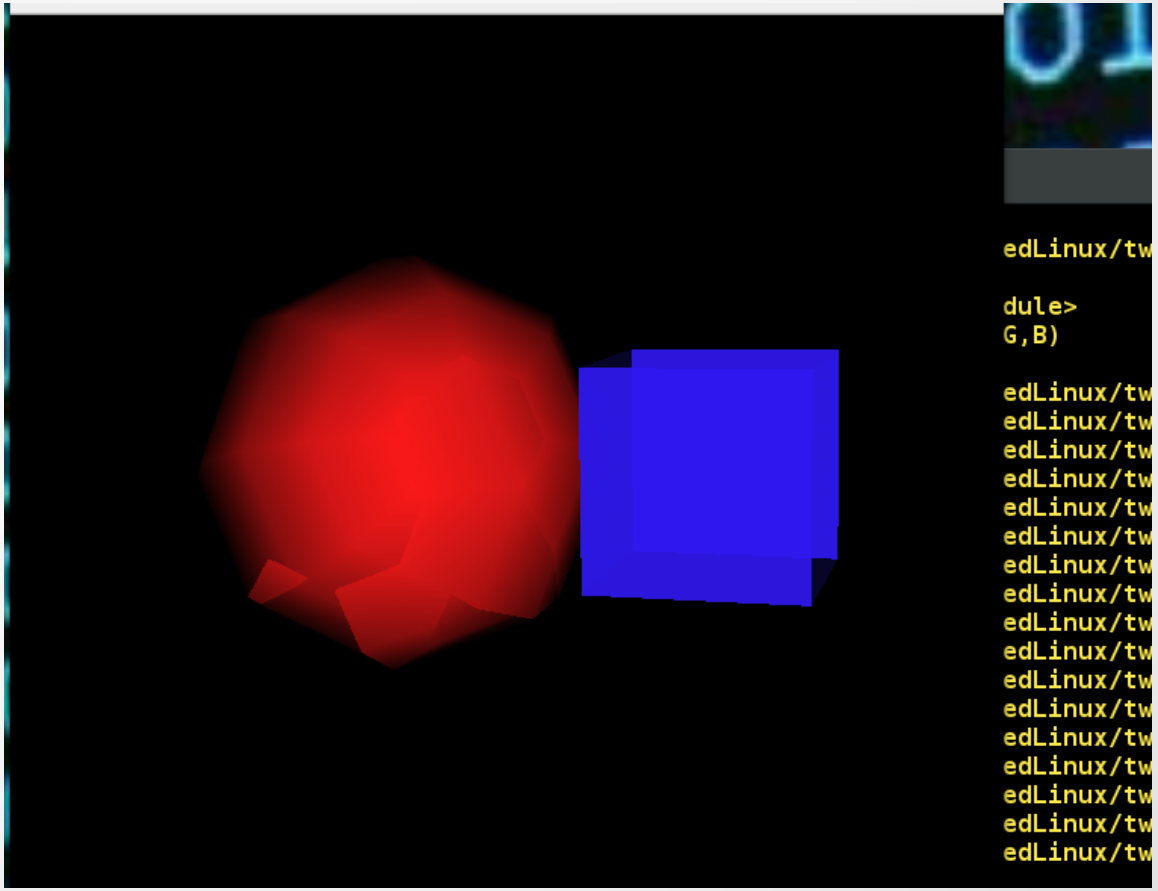
r –reset camera

s/w –surface/wireframe

u –command window

e –exit

Vediamo il source vtk/twoactors.py



Esercizio

- Usando le classi atom e molecule viste in precedenza aggiungiamo un metodo alla classe atom per gestire le dimensioni in angstom dell'atomo (set e get) e se volete anche il suo "colore" in RGB ad esempio. Proviamo poi a rappresentare la molecola i metano