Linguaggi di programmazione nella fusione

Edmondo Giovannozzi

Introduzione a Python.

Introduzione

- Interpretato
- Orientato agli Oggetti
- Vasta Libreria
- Molto usato nella analisi dei dati
- Specifiche del linguaggio:
  - 2.x compatibile con il passato.
  - 3.x ha alcune incompatibilità con la 2.x (in particolare l'istruzione print è diventata una funzione).
- https://www.python.org/
- http://www.scipy.org/

python vs Matlab, IDL, etc.

- Open source.
  - Matlab, IDL, etc. sono a pagamento
- Interpretato.
  - Ma si può facilmente chiamare funzioni esterne scritte in linguaggio compilato (Fortran, C, etc.)
  - disponibili compilatori JIT, o interpreti con all'interno dei JIT (PyPy).
- Orientato agli oggetti fin dall'inizio.
- Vastità delle librerie disponibili.
  - Attualmente google ha rilasciato la sua libreria per l'intelligenza artificiale con interfaccia per python (https://www.tensorflow.org/)
- La grafica è simile a quella di Matlab.

Per iniziare

- Installare Versione(3.x):

  ipython, matplotlib, numpy, scipy

- Su linux:

  ipython --pylab

  matplotlib, numpy, scipy importati automaticamente

  - l'opzione --pylab è equivalente ad eseguire:

    ```
    from matplotlib.pylab import *
    ```

- Su windows installare winpython o equivalente e laniare **spyder**:
  - http://winpython.github.io/
  - http://python-xy.github.io
  - http://continuum.io/
  - https://www.enthought.com/
  - http://www.activestate.com/activepython

Come eseguire Python

- Spyder (IDE)
  - Console, editor dei programmi, varie viste (variabili, outline, etc.)
- Ipython (Jupiter) notebook
  - All'interno di un browser, celle esegibili e di testo (supporta LaTex per le formule)
- IPython + editor esterni (geany, etc.)
- Su linux la prima riga di un file se contiene "#!/*path*/python" permette di eseguire il file con python.
  - gli argomenti della linea di comando sono disponibili in **sys.argv**, ma è meglio usare il modulo **argparse**.

come calcolatore

```
>>> 3+5
    8
>>> 3.5 * 6.7
    23.4499999
>>> 7/3
     2.33   2
>>> 7.0/3
     2.33
>>> 7.0 // 3.0
     2.0
>>> 3**2
     9
>>> 8 % 3
     2
>>> 1 + 5j
   (1+5j)
```

Divisione intera in Python 2.x

Divisione intera

Resto della divisione intera

Numero complesso

stringhe

```
>>> "Ciao come stai"
  'Ciao come stai'
>>> 'Bene e tu?'
   'Bene e tu'
>>> print(' Bene! \n Bene!')
 Bene!
 Bene!
>>> print(r' Bene! \n Bene!')
 Bene! \n Bene!
>>> "Ciao, ciao".upper()
 'CIAO, CIAO'
>>> "NON Mi dire".lower()
 'non mi dire'
>>> "ma che dici".split()
 ['ma', 'che', 'dici']
```

Singole o doppie quote equivalenti

Come in C "\n" viene interpretato come un carattere di line-feed (a capo su unix).

La r disabilita l'interpretazione della barra inversa.

Maiuscolo minuscolo

Spezzetta in parole

Stringhe (2)

```
>>> a = """Ciao,
come va?
Bene!"""
>>> print(a)
Ciao,
come va?
Bene!
>>> 3 * "Ciao, " + "Roma!"
  'Ciao, Ciao, Ciao, Roma!'
>>> s = "abcdef"
>>> s[3]
  'd'
>>> s[3:5]
  'de'
>>> s[3] = 'K'
TypeError: 'str' object does not support item assignment
>>> len(s)
  6
```

Le tre doppie virgolette iniziano e terminano delle stringhe che si sviluppano su più linee (mantenendo gli a capo correttamente)

Si ripetono con * e si concatenano con +

Si estrae un carattere od una sottostringa (a partire da zero)

Sono immutabili, non possono essere cambiate

Con len la lunghezza

Conversioni

```
>>> a = 3.2
>>> type(a)
    float
>>> b = str(a)
>>> b
   '3.2'
>>> int(b)
ValueError: invalid literal for int() with base 10: '3.2'
>>> float(b)
   3.2
>>> int('54')
   54
>>> " %d  %5.2f" % (5, 6.7)
 ' 5   6.70'
>>> " {1:02d} {0} {pluto}".format(5.3, 2, pluto=6.7)
 ' 02 5.3 6.7'
```

non è trasformabile in n intero

Python 2 vs Python 3

Principali differenze (utente normale)

- print è una funzione in Py3:
  - Py2: >>> print 'Ciao: ', 34
  - Py3: >>> print('Ciao: ', 34)
- Divisione tra interi:
  - Py2: >>> 3/2 ␣ 1
  - Py3: >>> 3/2 ␣ 1.5
- In Py3 range, map, zip, etc. vengono espansi quando servono:
  - Py2: >>> range(5) ␣ [0, 1, 2, 3, 4]
  - Py3: >>> range(5) ␣ range(0, 5)
  - Py3: >>> range(0,50,10)[3] ␣ 30

Esecuzione di un programma

- All'interno di IPython

  In [1]: %run nomeprogramma

- all'interno di qualsiasi interprete Python

  >>> import nomeprogramma

- Dalla shell del sistema operativo

  python nomeprogramma.py

Nel seguito lanceremo i programmi con %run all'interno di IPython

Primo programma

saluto.py

```
def ciao(cosa):
    print('Ciao ' + cosa)
```

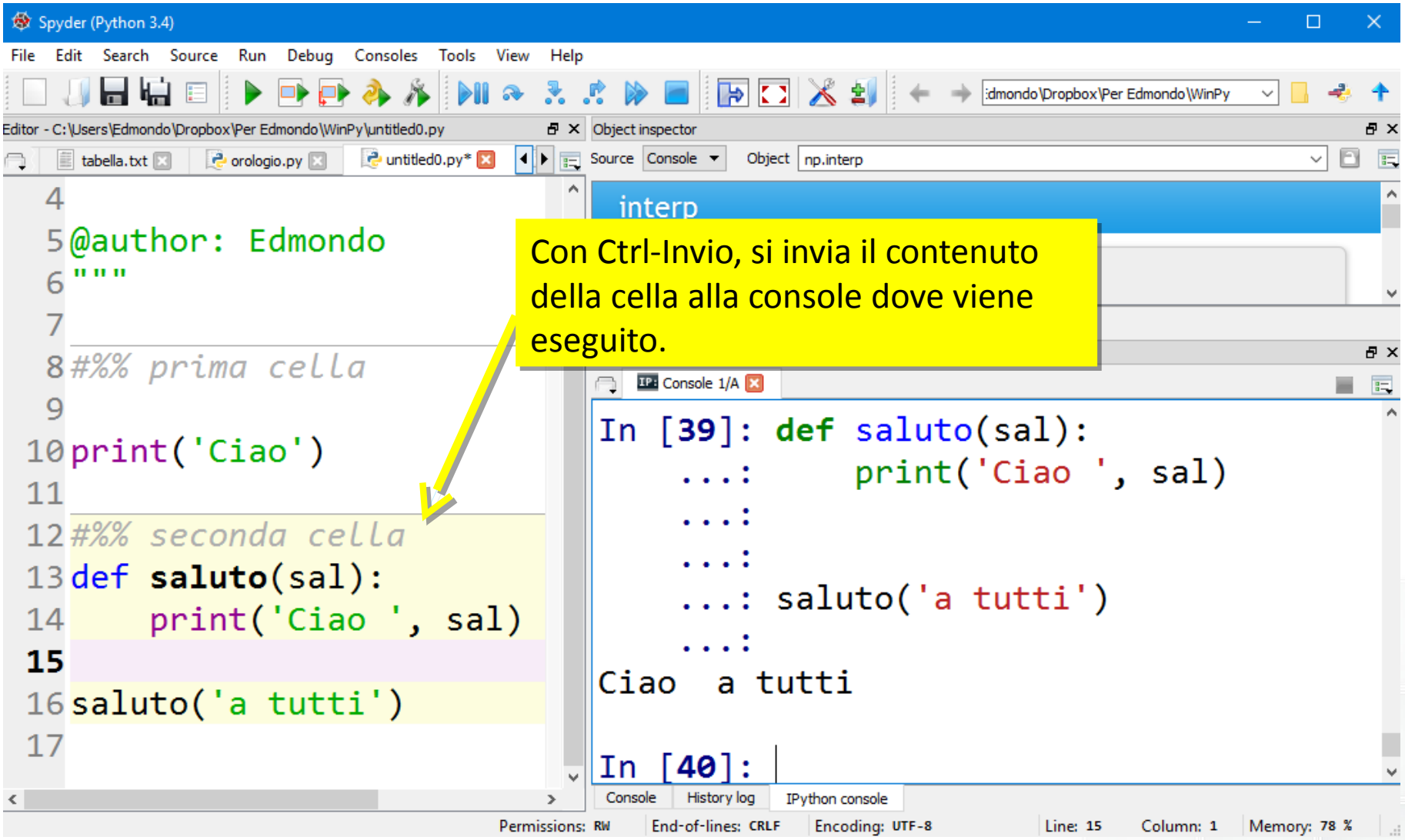
Definizione di una funzione.
cosa è un argomento

```
>>> %run saluto
>>> ciao('a tutti')
Ciao a tutti
```

Con **%run** le definizioni sono direttamente accessibili, con **import** sono qualificate dal nome del **modulo** importato

```
>>> import saluto
>>> saluto.ciao('anche a te')
Ciao anche a te
```

spyder


Spyder (Python 3.4)
File Edit Search Source Run Debug Consoles Tools View Help
Editor - C:\Users\Edmondo\Dropbox\Per Edmondo\WinPy\untitled0.py
tabella.txt
orologio.py
untitled0.py*
4
5 @author: Edmondo
6 """
7
8 #%% prima cella
9
10 print('Ciao')
11
12 #%% seconda cella
13 def saluto(sal):
14 print('Ciao ', sal)
15
16 saluto('a tutti')
17
Object inspector
Source Console Object np.interp
interp
Con Ctrl-Invio, si invia il contenuto della cella alla console dove viene eseguito.
Console 1/A
In [39]: def saluto(sal):
...: print('Ciao ', sal)
...:
...:
...: saluto('a tutti')
...:
Ciao a tutti
In [40]:
Console History log IPython console
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 15 Column: 1 Memory: 78 %

ipython notebook

jupyter testpercorso

File Edit View Insert Cell Kernel Help | Python 3

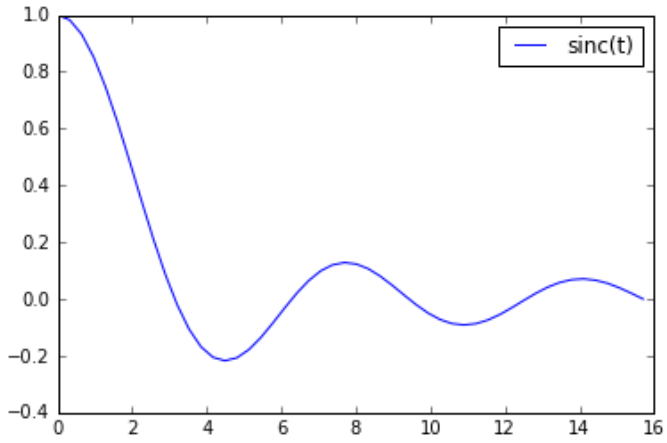
Code | Cell Toolbar: None

```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt

In [6]: t = np.linspace(0.001,5*np.pi)
        y = np.sin(t)/t
```

Plot della funzione:

$$\text{sinc} = \frac{\sin(t)}{t}$$

```
In [10]: plt.plot(t,y,label="sinc(t)");
         plt.legend();
```



All'interno di un browser.

Si possono inserire celle di testo anche con delle formule scritte in LaTex

Le figure sono inserite nel testo.

I file in formato .ipynb possono essere scambiati.

Moduli e package

```
>>> import numpy
```
Importa il package **numpy**

```
>>> import numpy as np
```
Importa il package **numpy** cambiandogli il nome

```
>>> from numpy import linalg
```
Importa il modulo **linalg** del package **numpy**

```
>>> from numpy.linalg import lstsq
```
Importa la funzione **lstsq** del modulo **numpy.linalg**

Python è organizzato con moduli e package, ogni funzionalità aggiuntiva si ottiene importando il relativo modulo:

```
>>> import re
>>> import os
>>> import sys
>>> import argparse
Etc.
```

Espressioni regolari

Accesso al sistema operativo

Informazioni sul sistema

Parsing degli argomenti

Primo programma

saluto.py

```
def ciao(cosa):
    completo = 'Ciao '+cosa
    print(completo)
```

I blocchi iniziano con «:» nella istruzione che precede il blocco.

saluto.py

```
def ciao(cosa):
␣␣␣␣completo = 'Ciao '+cosa
␣␣␣␣print(completo)

def …
```

Il blocco di istruzioni termina quando l'indentazione ritorna al livello precedente.

Non ci sono rispetto agli altri linguaggi dei terminatori del blocco (come } in C/C++).

Un programma così è automaticamente scritto in maniera ordinata.

Assegnazione

Il segno di = associa ad un nome un riferimento ad una variabile che diventa raggiungibile.

```
>>> sonounastringa = 'Ciao come stai'
```

Nome

Variabile

```
>>> quelladiprima = sonounastringa
```

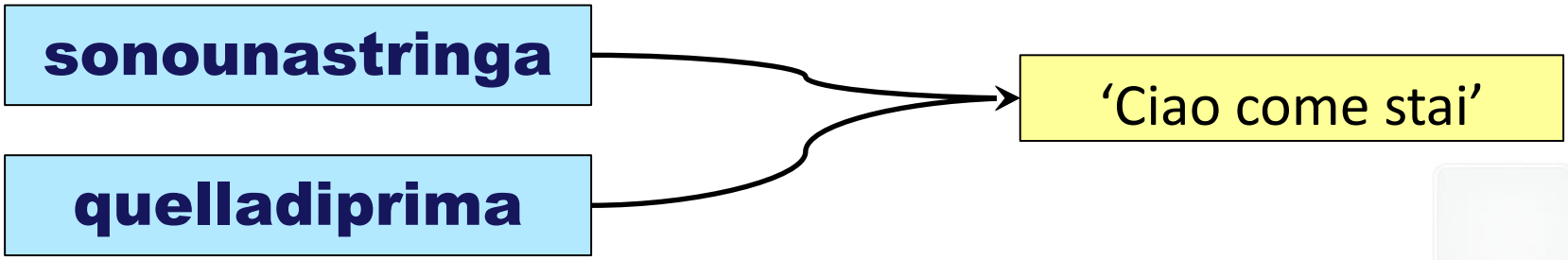
Altro nome

Variabile puntata dal nome

Per le variabili come le stringhe ed i numeri, che sono immutabili, nessuna differenza rispetto alla interpretazione consueta.

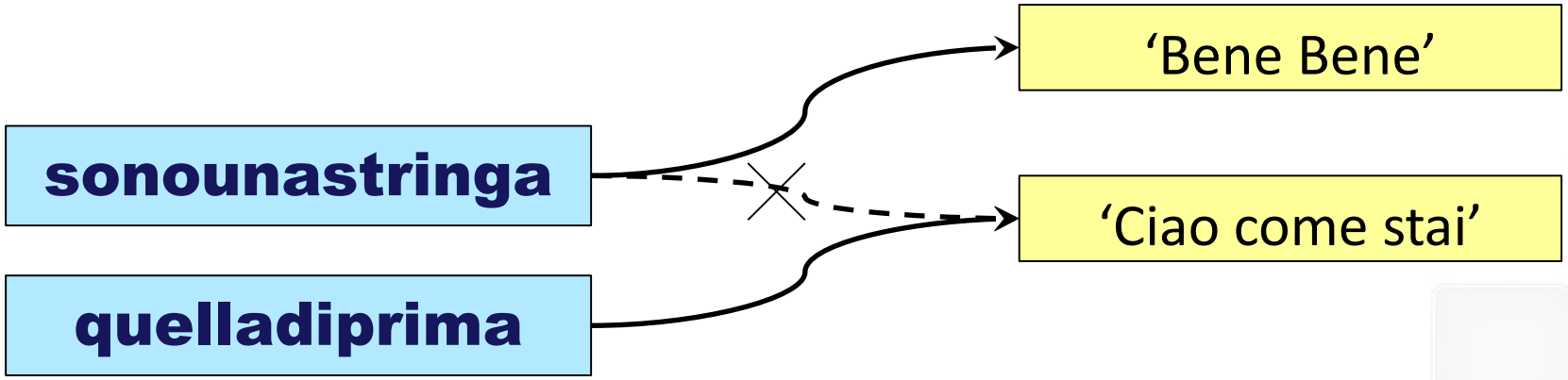
Assegnazione (2)

```
>>>  sonounastringa = 'Ciao come stai'
…
>>> quelladiprima = sonounastringa
```



Assegnazione (3)

```
>>> sonounastringa = 'Ciao come stai'
…
>>> quelladiprima = sonounastringa
...
>>> sonounastringa = 'Bene Bene'
```



Liste

Le liste sono un primo esempio di variabili mutabili.

```
>>> lista_a = [1, 2, 3]
>>> lista_b = lista_a
```

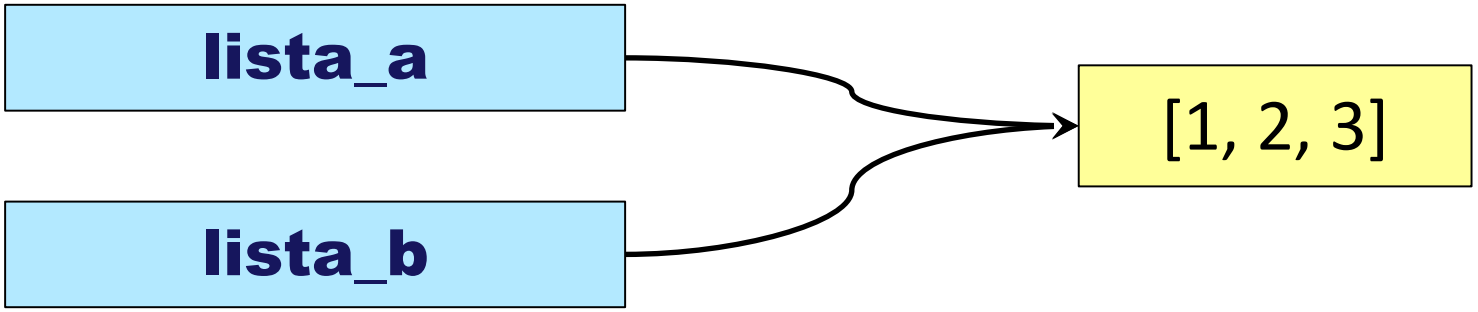
i due nomi **lista_a**, e **lista_b** puntano alla stessa lista

```
>>> lista_a.append(7)
>>> lista_b
    [1, 2, 3, 7]
```

Diversi tipi di variabili sono mutabili: liste, dizionari, set, oggetti (tra cui i numpy array).
Tra i tipi immutabili abbiamo: numeri, stringhe e tuple (ed i frozen_set)

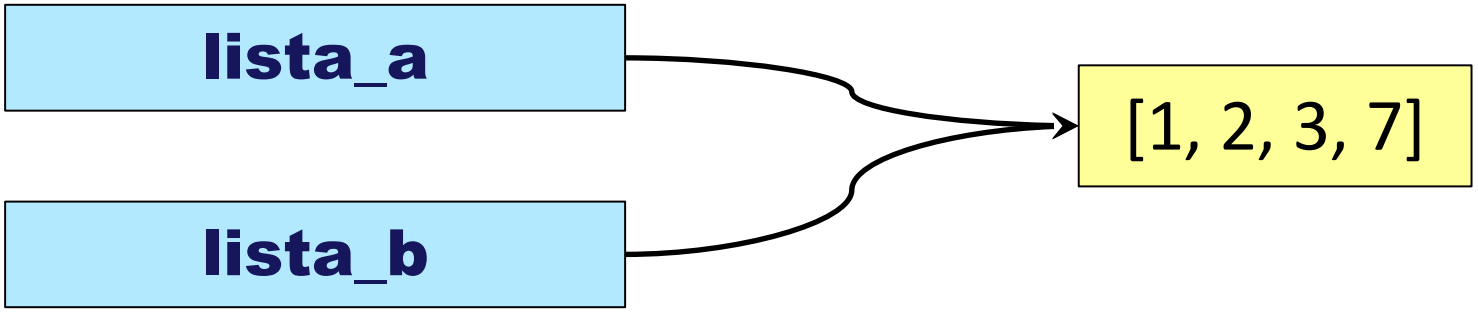
Liste (2)

```
>>> lista_a = [1, 2, 3]
>>> lista_b = lista_a
```



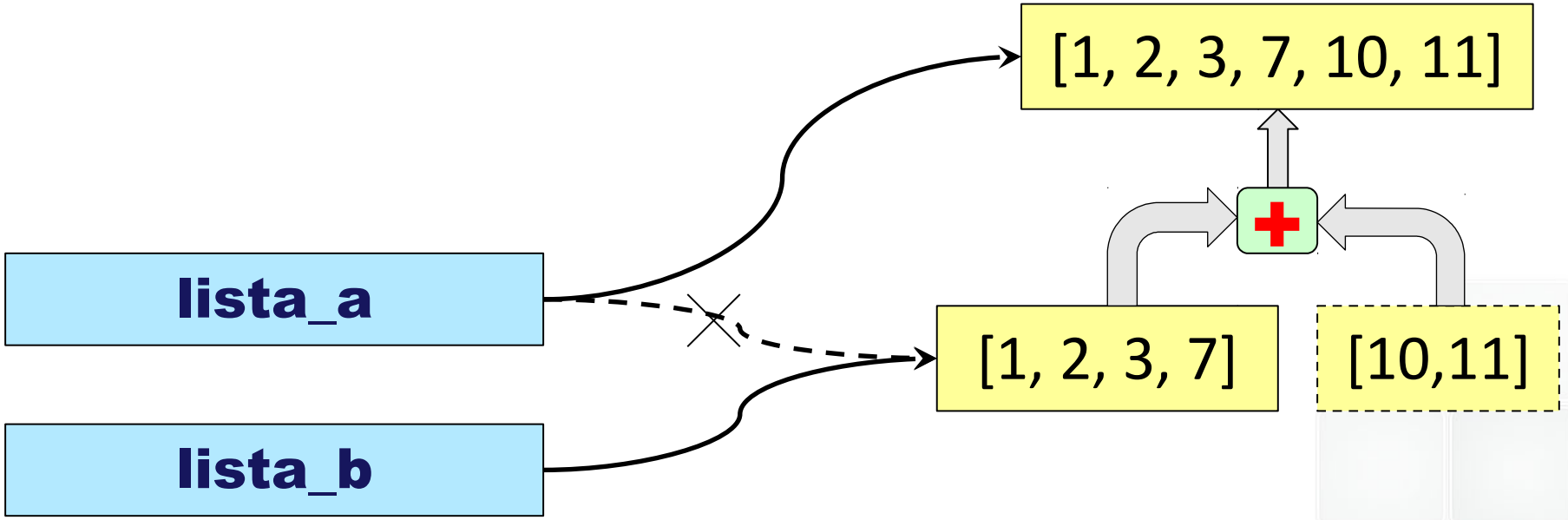
Liste (3)

```
>>> lista_a = [1, 2, 3]
>>> lista_b = lista_a
>>> lista_a.append(7)
```



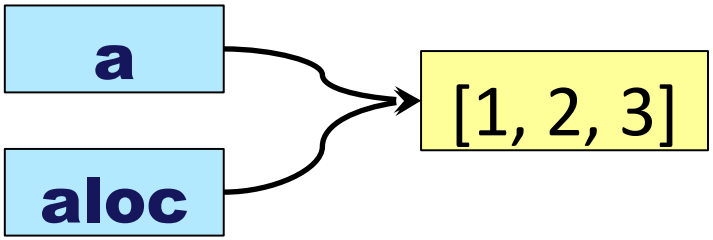
Liste (4)

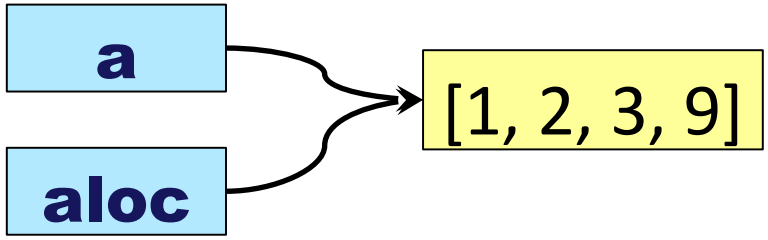
```
>>> lista_a = [1, 2, 3]
>>> lista_b = lista_a
>>> lista_a.append(7)
>>> lista_a = lista_a + [10, 11]
```



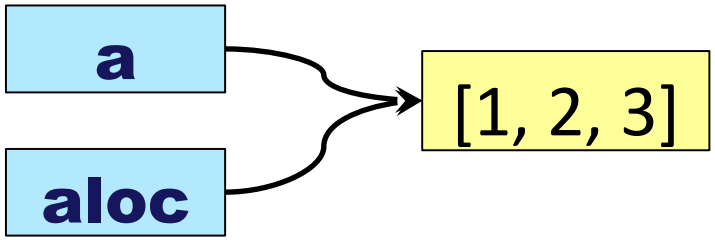
Come argomenti

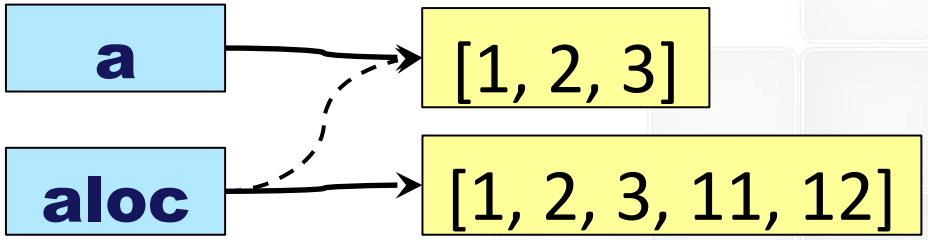
```
>>> def funza(aloc):
        aloc.append(9)
>>> a = [1,2,3]
>>> funza(a)
>>> a
    [1,2,3,9]
```






```
>>> def funzb(aloc):
        aloc = aloc + [11,12]
>>> a = [1,2,3]
>>> funzb(a)
>>> a
    [1,2,3]
```





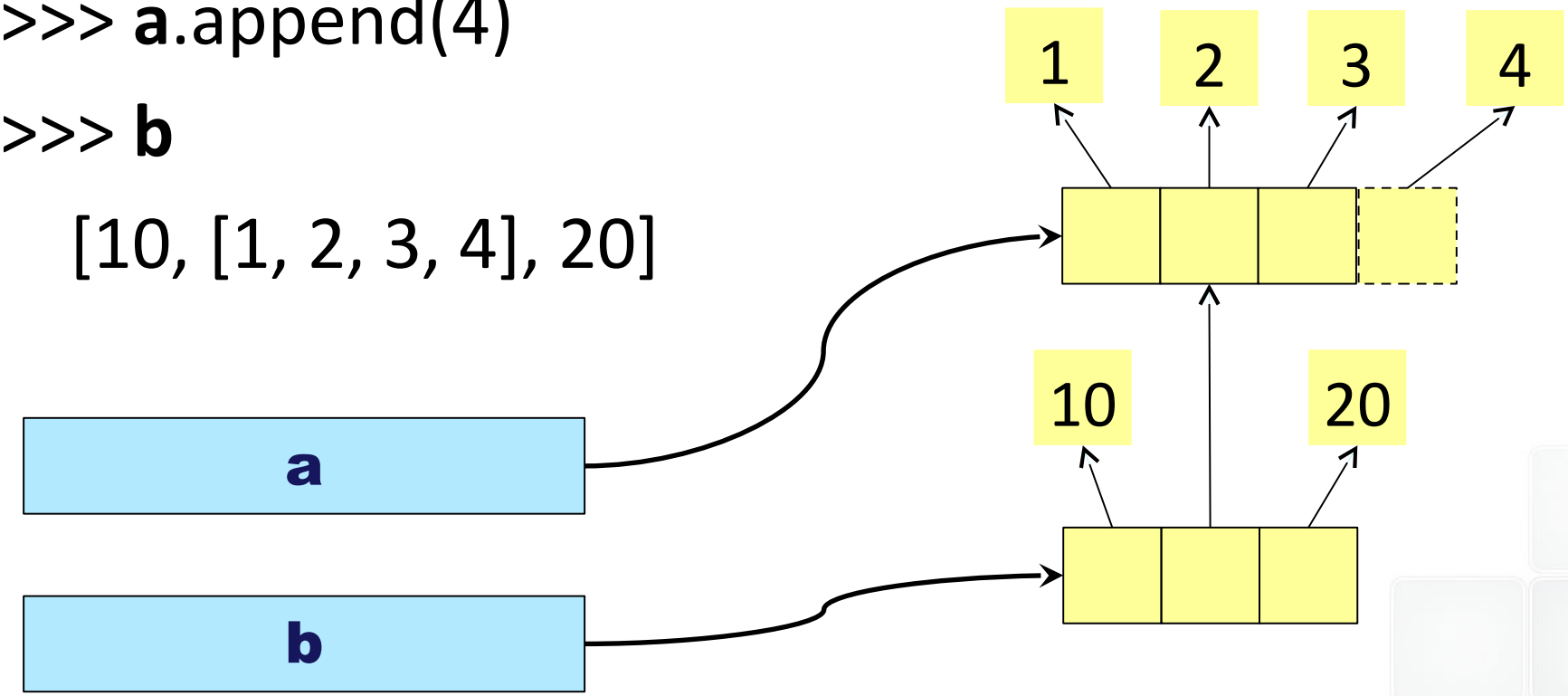
Cosa contengono le liste

```
>>> a = [1, 2, 3]
>>> b = [10, a, 20]
>>> b
   [10, [1, 2, 3], 20]
>>> a.append(8)
>>> b
    [10, [1, 2, 3, 8], 20]
```

La lista **b** contiene un riferimento alla lista **a**.

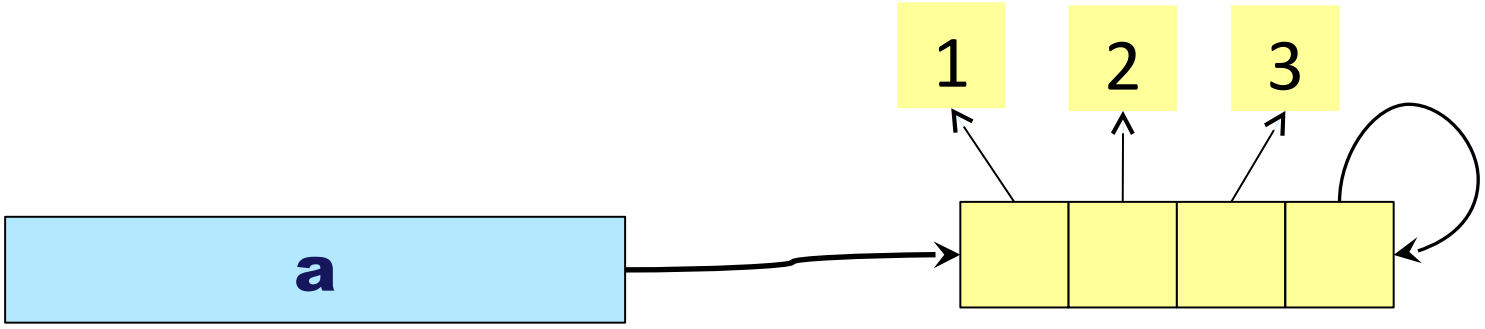
Cosa contengono le liste

```
>>> a = [1, 2, 3]
>>> b = [10, a, 20]
>>> a.append(4)
>>> b
   [10, [1, 2, 3, 4], 20]
```



Cosa contengono le liste

```
>>> a = [1, 2, 3]
>>> a.append(a)
>>> a
    [1, 2, 3, [...]]
```



Liste, e tuple

```
>>> a = [1, 2, 'Ciao', 3.4]
>>> a_singola = [3.4]

>>> b = (1,2, 'ciao')
>>> b_singola = (3.4, )

>>> a[0]
   1
>>> b[-1]
  'ciao'
>>> a[0:2]
  [1,  2]
```

Lista

Lista con un solo elemento

Tupla

Tupla con un solo elemento
notare la virgola finale

Gli indici partono da 0

Gli indici negativi partono dalla fine

In una sezione l'ultimo indice viene escluso

spacchettamento tuple

```
>>> (a, b) = (1, 2)
>>> a
    1
>>> b
    2
```

spacchettamento della tupla

```
>>> aa, bb = 3, 4
>>> aa
    3
>>> bb
    4
```

Le parentesi non sono necessarie

```
>>> aa, bb = bb, aa
>>> aa
    4
>>> bb
    3
```

Usato per scambiare i valori di due variabili e per spacchettare i valori tornati dalle funzioni

Sezioni (slice)

```
>>> a = [1,2,3,4,5]
>>> a[:3]
    [1, 2, 3]
>>> a[2:]
    [3, 4, 5]
>>> a[2:-1]
    [3, 4]
>>> a[::-1]
    [5, 4, 3, 2, 1]
>>> a[:0:-1]
    [5, 4, 3, 2]
>>> len(a)
      5
```

I primi 3 elementi

a partire dal terzo fino alla fine

a partire dal terzo escludendo l'ultimo

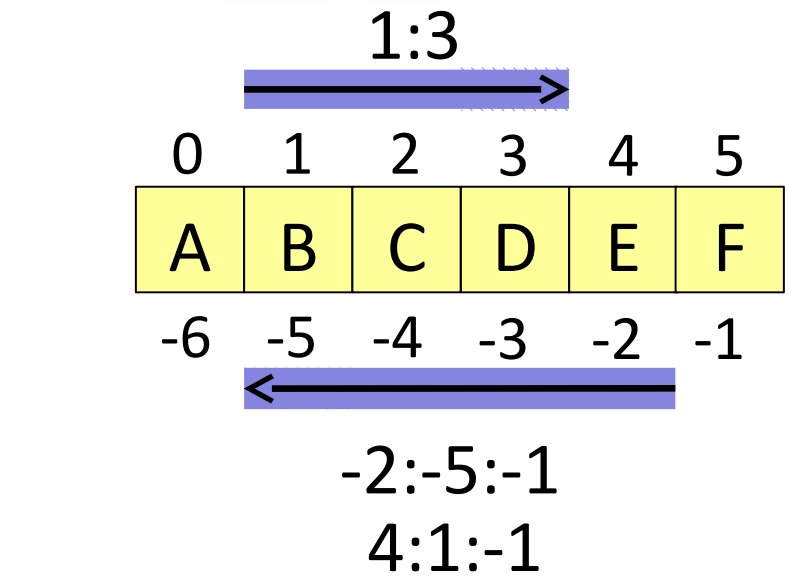
in ordine inverso

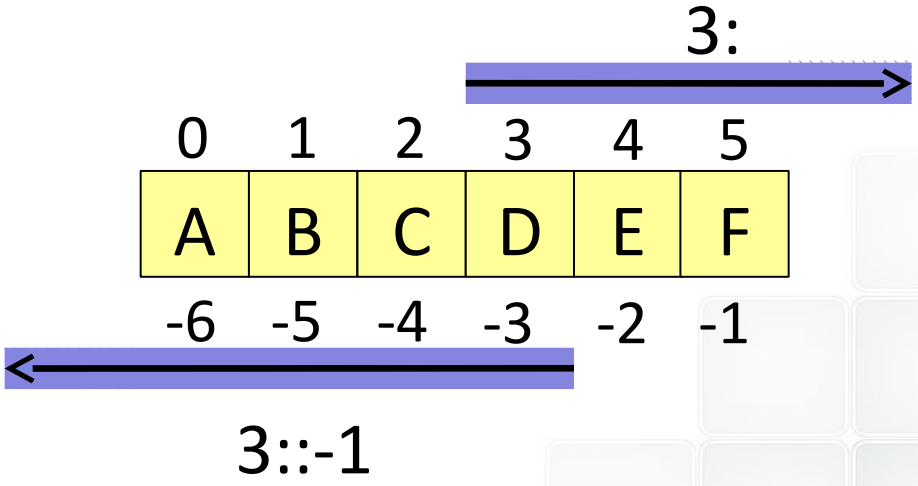
in ordine inverso escludendo il primo

numero di elementi

Sezioni (2)

```
>>> s = "ABCDEF"
>>> s[1:3]
    'BC'
>>> s[-2:-5:-1]
   'EDC'
>>> s[4:1:-1]
   'EDC'
>>> s[3:]
   'DEF'
>>> s[3::-1]
   'DCBA'
```





Liste altre operazioni

```
>>> a = [1,2,3,4]
>>> a[2:2] = [10,20,30]
    [1, 2, 10, 20, 30, 3, 4]
```

inserire elementi in una lista.

```
>>> del a[2:3]
    [1, 2, 20, 30, 3, 4]
```

cancellare elementi da una lista.

L'istruzione del si può usare per cancellare una variabile.

```
>>> del a
>>> a
  NameError: name 'a' is not defined
```

comprehension et al.

```
>>> a = [1, 2, 3, 4, 5]
>>> b = [ i**2 for i in a if i>2]
   [9, 16, 25]

>>> aa = [1, 2, 3]
>>> bb = [10, 20, 30]
>>> cc = list(zip(aa, bb))
    [(1, 10), (2, 20), (3, 30)]

>>> dd, ee = zip(*cc)
>>> dd
  (1, 2, 3)
>>> ee
  (10, 20, 30)
```

List comprehension, per generare una lista a partire da un'altra.

zip, per accoppiare elementi di più liste.
In Py3 ritorna un iteratore

Può anche essere usato per l'operazione inversa.

array

```
>>> import numpy as np
>>> a = np.linspace(0, 2, 5)
>>> a
  array([ 0., 0.5,  1., 1.5, 2.])
>>> a.size
   5
>>> b = np.zeros((2,4))
>>> b
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> b.shape
    (2, 4)
>>> c = np.array([1.0, 2.0, 3.0])
>>> c.dtype
   dtype('float64')
```

spaziati linearmente

il numero totale di elementi dell'array

Notare che le dimensioni sono passate con una tuple

a partire da una lista

il tipo numerico sottostante

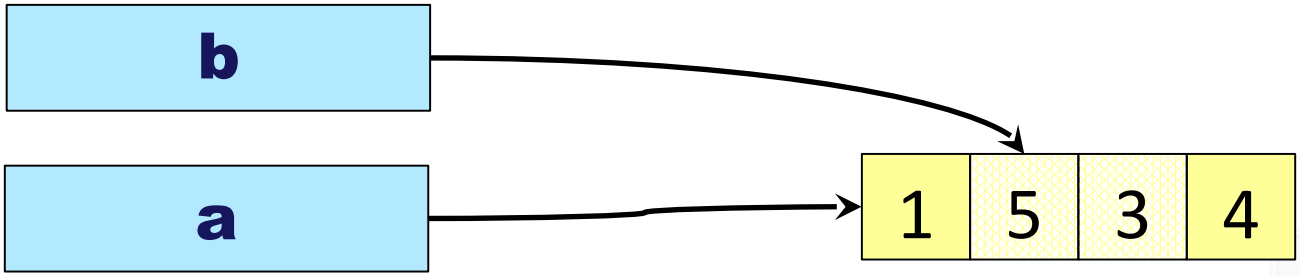
Sezioni e viste

```
>>> a = [1,2,3,4]
>>> b = a[1:3]
>>> b[0] = 5
>>> a
   [1, 2, 3, 4]
```

una sezione di una lista, genera una lista indipendente

```
>>> a = np.array([1,2,3,4])
>>> b = a[1:3]
>>> b[0] = 5
>>> a
   array([1, 5, 3, 4])
```

una sezione di un array, genera una vista sullo stesso array

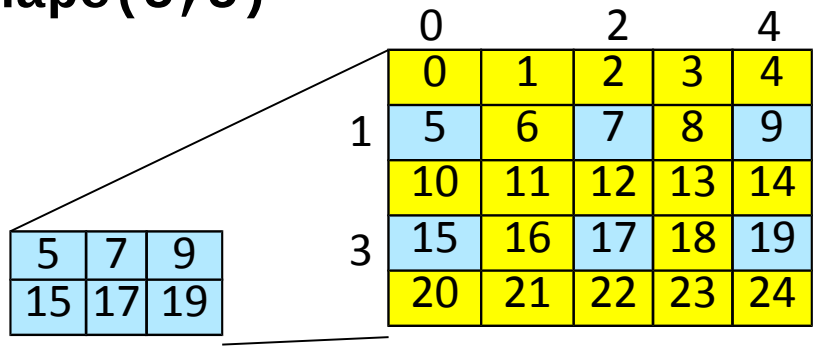


Sezioni (slice) di un array

```
>>> a = np.arange(25).reshape(5,5)
>>> b = a[1:4:2,0:5:2]
```

1, 3

0, 2, 4

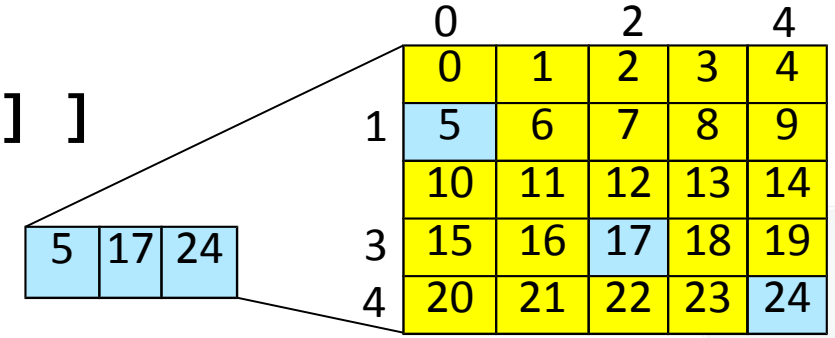


E' una vista, cambiare
b cambia **a**

Indexing Avanzato:

```
>>> c = a[ [1,3,4], [0,2,4] ]
```

I dati vengono copiati.
Modificare **c** non cambia **a**.



operazioni tra array

```
>>> r = np.random.random((2,3))
array([[ 0.90895715,  0.27791328,  0.08318425],
       [ 0.643648  ,  0.83921606,  0.32521858]])
>>> a = np.array([1, 2, 3])
>>> r*a
array([[ 0.90895715,  0.55582656,  0.24955276],
       [ 0.643648  ,  1.67843213,  0.97565575]])
>>> b = np.array([10,100])
>>> r*b[:,np.newaxis]
array([[  9.08957148,   2.77913282,   0.83184253],
       [ 64.36480046,  83.92160635,  32.52185839]])
```

r				*	a			
	0.90	0.27	0.08			1	2	3
	0.64	0.83	0.32			1	2	3

Esteso automaticamente sulle dimensioni iniziali

r				*	b			
	0.90	0.27	0.08			10	10	10
	0.64	0.83	0.32			100	100	100

Esteso tramite **newaxis.** Aggiunge un asse di dimensioni unitarie, che poi si estende automaticamente

set

```
>>> a = {'Ciao', 'b', 1, 2}
>>> b = {'Bene', 2, 'b', 3}
>>> a | b
 {1,2,'b',3,'Ciao','Bene'}
>>> a & b
 {2, 'b'}
>>> a – b
 {1, 'Ciao'}


>>> {1,2} < a
  True
>>> 'Bene' in b
  True
```

Unione

Intersezione

Differenza fra set

sottoinsieme

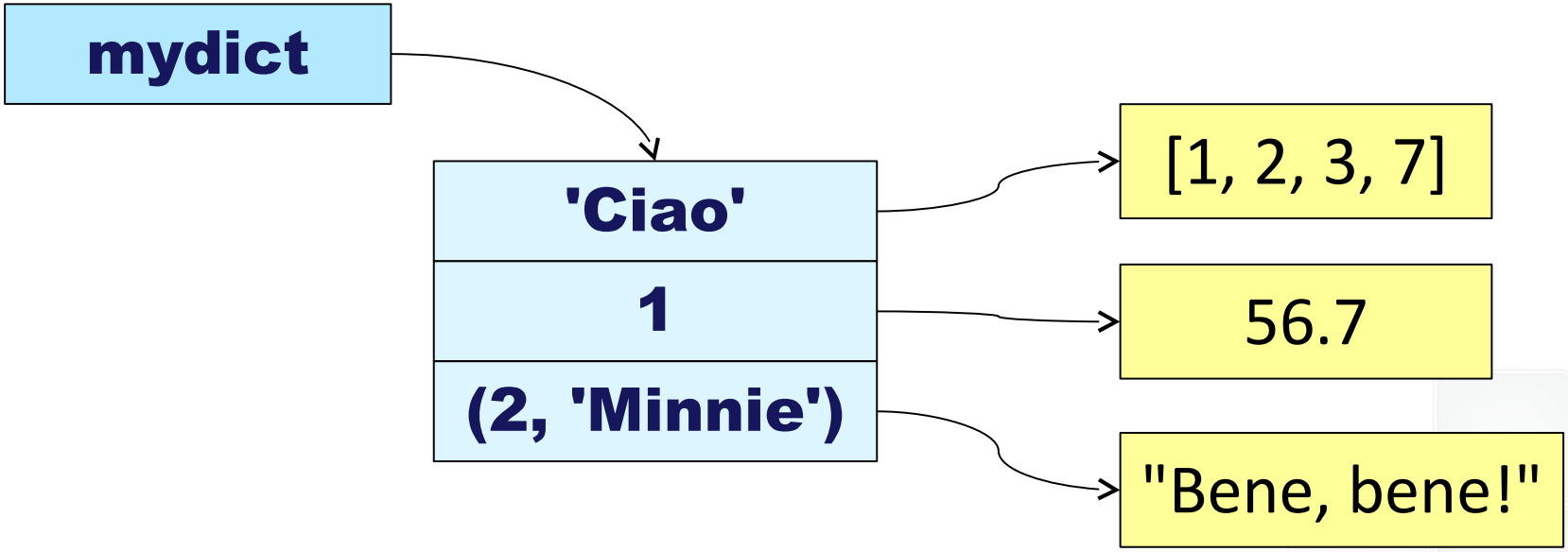
appartenenza

I **set** sono mutabili, i **frozenset** immutabili. Gli elementi di un set devono essere **hashable**, ovvero:

- Immutabili, e se contenitori contenere solo elementi hashable.
- Oggetti definiti dall'utente (salvo alcuni casi), ogni oggetto è diverso da chiunque altro (anche se della stessa classe)

Dizionari

- Associa a delle variabili chiave (a parità di valore) altre variabili (riferimenti ovviamente).
- Usati in molte parti di Python per il funzionamento interno.
- Le chiavi devono essere hashable.



Dizionari

```
>>> aa = {}
>>> aa['ciao'] = 56
>>> aa['test'] =  3.46
>>> aa
   {'ciao': 56, 'test': 3.46}
>>> aa['test']
   3.46
>>>  aa.keys()
   dict_keys(['ciao', 'test'])
>>> aa.values()
   dict_values([56, 3.46])
>>> aa.items()
   dict_items([('ciao', 56), ('test', 3.46)])
>>> len(aa)
   2
>>> 'ciao' in aa
 True
```

Mutabili. Gli elementi si accedono tramite una chiave, che deve essere **hashable**.

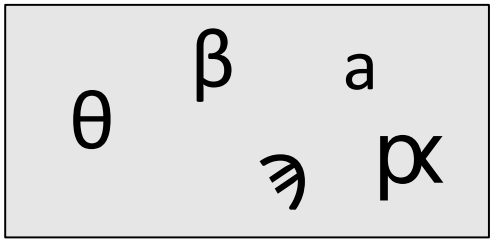
Elenco delle chiavi e dei valori.

In Python 2.x sono delle liste

Esistenza di una chiave

Caratteri (unicode)

In Python 3.x le stringhe sono formate da caratteri unicode.



"Alcuni caratteri: β, θ, ϶, px"

```
>>> stranicaratteri =  "Alcuni caratteri: β, θ, ϶, px"
```

Per trasformarla in una sequenza di byte (ad esempio usando la codifica ASCII):

```
>>> stranicaratteri.encode('ascii',errors='replace')
```

b'Alcuni caratteri: ?, ?, ?, ?'

Di default: UTF8

```
>>> inbytes = stranicaratteri.encode()
```

b'Alcuni caratteri: \xce\xb2, \xce\xb8, \xcf\xa0, \xd4\x97'

```
>>> inbytes.decode()
```

'Alcuni caratteri: β, θ, ϶, px'

```
>>> ord('a'), chr(9782)
    (97, '☖')
```

Grafica

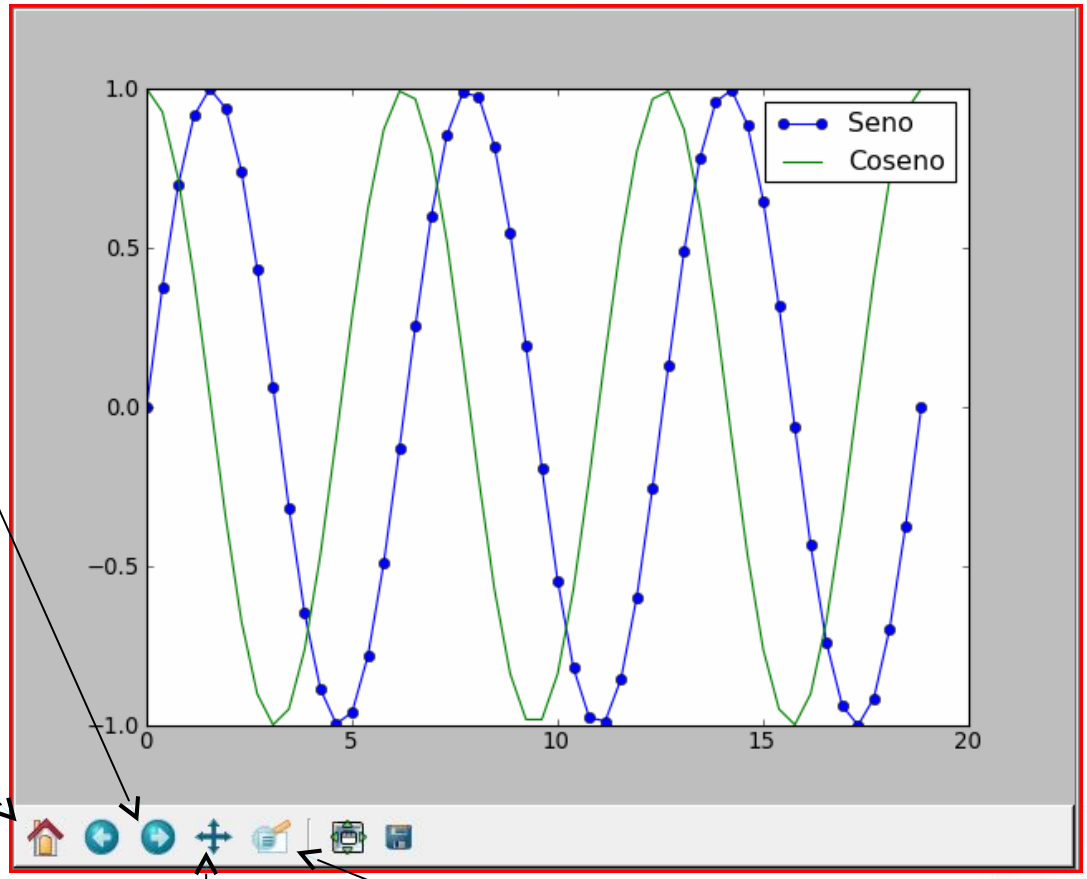
```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> t = np.linspace(0, 6*np.pi)
>>> y = np.sin(t)
>>> plt.plot(t, y, '-o', label='Seno')
>>> plt.plot(t, np.cos(t), label='Coseno')
>>> plt.legend()
>>> plt.show()
```

Con l'opzione **--pylab** gli import non sono necessari come non è necessario qualificare le funzioni con **np** o **plt**. Il comando **show**() finale è anche superfluo, non però all'interno di script.

```
In[1]: t = linspace(0, 6*pi)
In[2]: plot(t, sin(t), label='Seno')
In[3]: plot(t, cos(t), label='Coseno')
In[4]: legend()
In[5]: clf()  # Cancella la figura
```

Figure

Ci si muove avanti ed indietro lungo la storia degli zoom o spostamenti effettuati.



Riporta la figura alle dimensioni originarie

Zoom spostamento dinamico. Spostamento se si muove il mouse con il tasto sinistro abbassato. Zoom se si spinge il tasto destro

zoom rettangolare

Principali strutture (if)

esempio.py

```
def controlla(val):
    if val>0:
        print('val > 0')
    elif val == 0:
        cosa = val + 5
        print(cosa)
    else:
        print('val < 0')
```

If

```
if <condizione1> :
    <istruzioni1>
elif <condizione2> :
    <istruzioni2>
else:
    <istruzioni3>
```

```
>>> %run esempio
>>> controlla(7)
val > 0
>>> controlla(0)
5
```

espressioni logiche

```
>>> (3 > 2 or 5 < 3) and not 7 != 7
    True

>>> 2 in [1, 2, 3]
    True
>>> 2 not in [1, 2, 3]
    False

>>> a = [1, 2, 3]
>>> b = a
>>> b is a
    True
>>> b = a[:]
>>> b is a, b == a
    (False, True)

>>> 1 < 2 < 3
   True
>>> 1 < 4 < 3
   False

>>> 'Grande' if 5 > 3 else 'piccolo'
   'Grande'
>>> 'Grande' if 5 > 8 else 'piccolo'
   'piccolo'
```

I classici operatori

Controlla se un elemento appartiene ad una sequenza

Controlla se due nomi si riferiscono allo stesso oggetto. Vedere la differenza con == che controlla se due oggetti hanno lo stesso valore

Operatori di comparazione possono essere messi uno dopo l'altro, il significato è ovvio.

espressione con if in linea

bitwise «or» «and» «not»

```
>>> a = np.array([1,2,3,4])
>>> b = np.array([8,0,3,7])
>>> (b == a) | (b > a)          | OR
  array([ True, False,  True,  True], dtype=bool)
>>> ~ (b == a)                  ~ NOT
  array([ True,  True, False,  True], dtype=bool)
>>> (b >= a) & (b == a)         & AND
  array([False, False,  True, False], dtype=bool)
>>> (b > a) ^ (b < a)           ^ XOR
  array([ True,  True, False,  True], dtype=bool)
```

Gli operatori bitwise possono essere usati per creare dei vettori logici.
Hanno una precedenza superiore agli operatori di comparazione, perciò devono essere protetti con delle parentesi.

vettori logici et al.

```
>>> a = np.arange(6)*10
  array([ 0, 10, 20, 30, 40, 50])
>>> a[a<20] = -2
  array([-2, -2, 20, 30, 40, 50])
>>> np.flatnonzero(a>20)
  array([3, 4, 5])

>>> b = np.r_[3:7, 21]
  array([ 3,  4,  5,  6, 21])

>>> a = np.arange(0,30,10)
>>> b = np.arange(3)
>>> np.hstack((a,b))
 array([ 0, 10, 20,  0,  1,  2])
>>> c = np.vstack((a,b))
 array([[ 0, 10, 20],
        [ 0,  1,  2]])
>>> c.ravel()
 array([ 0, 10, 20,  0,  1,  2])
```

Un vettore logico può essere usato al posto degli indici per selezionare degli elementi.

La funzione flatnonzero ritorna gli indici stessi (del vettore considerato 1D).

hstack, **vstack** per concatenare i vettori (Attenzione in ingresso una tupla di vettori).

ravel per renderlo monodimensionale (C like)

argmin, argmax, etc.

```
>>> a = np.random.random((5,6))
>>> b = np.random.random((5,6))
```

Genero due matrici di numeri random.

```
>>> idm = a.argmax(axis=0)
>>> i2, = np.indices(idm.shape)
```

Unpacking di una tupla

Cerco, con **argmax()**, la posizione del massimo in ogni colonna della matrice **a**. Non posso usare direttamente l'uscita di **argmax** per trovare il massimo, devo usare **indices** che mi restituisce il valore degli altri indici.

```
>>> print(a.max(axis=0))
>>> print(a[idm, i2])
```

Il metodo **max(axis=0)** mi dà direttamente il massimo lungo le colonne.

```
>>> print(b[idm, i2])
```

Ma l'uso di **indices** mi permette di trovare i valori in **b** corrispondenti ai massimi di **a**.

Altro sugli Array

```
>>> a = np.c_[0:2,2:4,4:6]
    [[0   2   4]
     [1   3   5]]

>>> b = a.ravel()
>>> c = a.flatten()

>>> ac = a.copy()

>>> d = a.T
>>> e = a.reshape((3,2))

>>> a[0,0] = 100
```

se possibile non crea una copia

crea una copia

crea una copia

ritorna una vista della trasposta

se possibile non crea una copia

Controllare che effettivamente in alcuni casi è stata creata una copia

Principali strutture (for)

esempiocicli.py

```
def controlla(vals):
    for v in vals:
        print('v: {0}'.format(v))
        if v<0: break
    else:
        print('tutti > 0')
```

for

```
for <var> in <iterable>:
    <istruzioni>
    … break
    … continue
else:
    <istruzioni else>
```

```
>>> %run esempiocicli
>>> controlla([1,2])
v: 1
v: 2
tutti > 0
>>> controlla([1,-2,3])
v: 1
v: -2
```

Principali strutture (for)

esempiocicli2.py

```
def finoa(n):
    for i in range(n):
        print('i: {}'.format(i))
```

```
>>> %run esempiocicli2
>>> finoa(3)
i: 0
i: 1
i: 2
>>> list(range(3))
 [0, 1, 2]
>>> list(range(2, 8, 3))
 [2, 5]
```

la funzione:

range(start, stop, step)

ritorna un iteratore che genera una lista di interi:

start :0 se non specificato
stop : escluso
step :1 se non specificato

In Py2 ritorna direttamente una lista di interi.

Principali strutture (altro)

```
>>> ipl = 0
>>> while ipl <5:
        ipl += 1
        print(ipl)
1
2
3
4
5
```

Ciclo while: cicla finche l'espressione è vera

Funzione map, ritorna un iteratore in Py3.

Funzioni anonime

```
>>> a = map(lambda x: x**2 + 2, [1,2,3,4] )
  <map at 0x8d40a20>

>>> list(a)
  [3, 6, 11, 18]
```

Iteratori e Generatori

Le liste, le tuple gli array possono essere trasformate in un iteratore ed usate nei cicli for.

Un particolare tipo di iteratori si ottengono tramite i generatori, funzioni che contengono l'istruzione **yield**.

```
>>> %run generatore
2
3
5
8
13
>>> list(fibonacci(7))
 [2, 3, 5, 8, 13, 21, 34]
```

generatore.py

```
def fibonacci(n):
    a, b = 1, 1
    for i in range(n):
        a, b = b, a+b
        yield b

for i in fibonacci(5):
    print(i)
```

Eccezioni (try..except)

esempiotry.py

```
def testtry(n):
    a = [1,2,3]
    try:
        print('a = ', a)
        print('a[n] = ', a[n])
    except IndexError:
        print('Non ci siamo')
```

try

```
try:
    <istruzioni1>
except <eccezioni>:
    <istruzioni2>
```

```
>>> %run esempiotry
>>> testtry(2)
a =  [1, 2, 3]
a[n] =  3
>>> testtry(7)
a =  [1, 2, 3]
a[n] =  Non ci siamo!
```

Ci sono altre possibilità, tipo la clausola **finally:**, **else:**, etc.

Che non mostriamo per semplicità.

Lettura file e **with:**

letturafile.py

```
def leggimi():
    with open('letturafile.py') as f:
        for i, line in enumerate(f):
            print(i, line, end='')
```

```
>>> %run letturafile.py
>>> leggimi()
0 def leggimi():
1     with open('letturafile.py') as f:
2         for i, line in enumerate(f):
3             print(i, line, end='')

>>> list(enumerate(['a','b','c']))
    [(0, 'a'), (1, 'b'), (2, 'c')]
```

Aperto di default in sola lettura e come file di testo.

Si può usare **enumerate** per avere anche l'indice di un iterabile.

Lettura matrici da file

Il modulo numpy provvede funzioni per la lettura di matrici o tabelle di numeri. Esistono anche funzioni per la scrittura o lettura di file matlab

```
>>> import numpy as np
>>> a = np.genfromtxt('tabella.txt',names=True)
>>> a['t']
   array([ 1. ,  2.3,  3.1])
>>> a['te']
   array([ 10.,  35.,  21.])
>>> a['ne']
    array([ 100.,  118.,  250.])
```

tabella.txt

t	te	ne
1	10	100
2.3	35	118
3.1	21	250

Lettura di un file Matlab

```
>>> from scipy.io import loadmat

>>> a = loadmat('test.mat')
>>> a.keys()
 dict_keys(['t', 'co', '__globals__', '__header__',
            '__version__', 'info', 'cella', 'si'])
```

Ritorna un dizionario

In blu le variabili presenti nel file.

```
>>> print(a['t'].shape)
   (1, 100)
```

Tutte le variabili in Matlab sono almeno matrici con 2 dimensioni

Provate a fare un plot di **si** vs **t** e **co** vs **t**

Lettura file altri formati

Si possono leggere files di altri formati:

- Excel
  - http://www.python-excel.org/
  - XlsxWriter, xlrd,xlwt
- Kaleidagraph (con una routine a parte)
- HDF5
  - http://www.h5py.org/: **import h5py**
  - http://www.pytables.org/ : **import tables**
- Netcdf
  - **scipy.io.netcdf.netcdf_file**
- Mdsplus
  - Per accedere ai dati della fusione

Funzioni

```
>>> %run funzioni
>>> pluto(2)
a =  2
b =  3.0
c =  [3.0]
  (2, 3.0, [3.0])
>>> pluto(1,c=[5, 6])
a =  1
b =  3.0
c =  [5, 6, 3.0]
   (1, 3.0, [5, 6, 3.0])
>>> aa, bb, cc = pluto(2)
>>> aa
   2
>>> bb
   3.0
>>> cc
  [3.0]
```

funzioni.py

```
def pluto(a, b=3.0, c=None):

   print('a = ', a)
   print('b = ', b)

   if c is None:
      c = []
   c.append(b)

   print('c =', c)

   return a, b, c
```

Docstring

Una Docstring è una stringa che documenta il codice. Deve essere la prima istruzione dopo la definizione di una funzione, classe, etc.

Tipicamente si usano le stringhe multiline che iniziano e terminano con: """

funzioni.py

```
def pluto(a, b=3.0, c=None):
    """ Test optional argument
    b and c are optional
    """
    print('a = ', a)
    print('b = ', b)

    if c is None:
        c = []
    c.append(b)

    print('c =', c)

    return a, b, c
```

None

- Usato per indicare la mancanza di qualche cosa
- E' un tipo a se stante **NoneType** che ha una sola variabile di quel tipo ovvero **None**.
- Una funzione che non ritorna nulla (ovvero che non ha un istruzione **return**) in realtà ritorna **None**.
- Si controlla l'uguaglianza di un oggetto con None tramite: *nomeoggetto* **is None**.
- Si usa tipicamente se abbiamo un argomento di default di tipo mutabile:

```
def pluto(arg=None)
    if arg is None:
        arg = []
```

- Attenzione diverso dal float NAN:

```
>>> a = np.sqrt(np.array([1.0, 2.0, -2.0, 3.0]))
>>> np.isnan(a)
 array([False, False,  True, False], dtype=bool)
```

Lista di argomenti variabile

```
>>> positionalarg(4, 5, 'Ciao')
(4, 5, 'Ciao')
>>> a = [4, 5, 'Ciao']
>>> positionalarg(*a)
(4, 5, 'Ciao')
```

argumentlist.py

```
def positionalarg(*arg):
    print(arg)

def keywordarg(**kwarg):
    print(kwarg)
```

Gli argomenti sono inseriti in una tupla. Una lista od una tupla con lo * davanti vengono spacchettate negli argomenti.

```
>>> keywordarg(pippo=5,pluto=6,topo='Ciao')
{'topo': 'Ciao', 'pippo': 5, 'pluto': 6}
>>> b = {'primo':3.4, 'sec':7.8}
>>> keywordarg(**b)
{'primo': 3.4, 'sec': 7.8}
```

Gli argomenti passati tramite keyword diventano dizionari (e viceversa).

Programmazione Obj. Or.

- E’ più naturale, siamo abituati ad interagire nel mondo reale con oggetti.
  - Attributi. Dati contenuti nell'oggetto.
  - Metodi. Procedure che interagiscono con l'oggetto.
- Permette di separare meglio le responsabilità.
  - Se scrivo su un foglio, lo posso fare nello stesso modo con una penna, una matita od un pennarello.
  - Ereditarietà
- Schemi di disegno del software (Design Pattern).
  - se ne parla dal 1987, ma diventati famosi dal 1994 a partire dal libro *Design Patterns: Elements of Reusable Object-Oriented Software* gli autori sono spesso citati con l’acronimo GoF (Gang of Four).

Vita di un oggetto

- Creazione.
  - In Python il metodo **__init__**
  - Dopo la creazione un oggetto deve essere completo.
- Vita.
  - Vari metodi possono modificare lo stato di un oggetto o farlo interagire con altri oggetti.
- Distruzione.
  - In C++ si parla di distruttore, in Fortran di routine "final".
  - In Python la deallocazione della memoria è automatica.
  - Se si devono fare delle operazioni "finali", meglio usare i Context Manager con l'istruzione **with** (come accade con i file).

Classi

```
>>> %run myclass
>>> a = MyClass('Pluto')
>>> a
 Nome: Pluto
>>> a.add(' e Clara')
>>> a
 Nome: Pluto e Clara
>>> dir(a)
['__class__',
 ....
 '__init__',
 ....
 'add',
 'nome']
```

myclass.py

```
class MyClass(object):
    def __init__(self, nome):
        self.nome = nome

    def __repr__(self):
        rstr = ['Nome: ' + str(self.nome)]
        return "\n".join(rstr)

    def add(self, suff):
        self.nome += suff
```

MyClass discende da object (sempre consigliabile in Py2, non necessario in Py3).
Ha tre metodi:
__init__ il costruttore.
__repr__ viene invocato quando Python vuole rappresentarlo. ritorna una stringa.
add che aggiunge un suffisso al nome.
Il primo argomento di ogni metodo è **self** corrispondente all'oggetto stesso.

Classe Ereditarietà

```
>>> %run myclass2

>>> a = Point(2,3)
>>> a
 x=2, y=3

>>> b = Square(2,3,5)
>>> b
 x=2, y=3 w=5

>>> isinstance(b, Square)
   True
>>> isinstance(b, Point)
   True
```

myclass2.py

```
class Point(object):
   def __init__(self, x, y):
      self.x, self.y = x, y

   def __repr__(self):
      return 'x={0}, y={1}'.format(self.x,self.y)

class Square(Point):
   def __init__(self, x, y, width=1.0):
      super(Square,self).__init__(x,y)
      self.width = width

   def __repr__(self):
      rst = super(Square,self).__repr__()
      return rst + ', w='+str(self.width)
```

Square discende da Point che discende da object.
I metodi della classe genitore si accedono tramite **super**.
Per controllare se un oggetto appartiene ad una clase che discende da quella data si usa **isinstance**. Anche se si preferisce il duck typing, provare ad accedere ai metodi, al massimo darà un errore che si intercetta con **try…except**.

Classi altro

```
>>> class DueListe(object):
        comune = []
        def __init__(self):
            self.personale = []

>>> a = DueListe()
>>> b = DueListe()
>>> a.numero = 32
>>> a.comune.append('A tutti')
>>> a.personale.append('A me')
>>> print(b.comune)
   ['A tutti']
>>> print(b.personale)
   []
>>> print(a.numero)
    32
>>> b.comune = 'E mo lo cambio'
>>> print(a.comune)
   ['A tutti']
```

due oggetti di classe DueListe

posso sempre aggiungere un attributo ad un particolare oggetto.

la lista **comune** è condivisa da tutti gli oggetti della classe DueListe.

ridefinisco l'attributo **comune** nell'oggetto **b**, che non punta più alla lista condivisa.

Ma c'è un modo per raggiungere la lista condivisa a partire dall'oggetto **b**. Fate
>>> **dir(b)**
e pensate che la lista condivisa appartiene alla classe di **b**.

Convenzioni e metodi standard

- In Python nulla è realmente protetto, anche se il linguaggio definisce delle convenzioni che forzano una sorta di protezione.
- In genere metodi o routine che iniziano con un underscore sono considerati privati (ma non c'è nulla che li protegga realmente).
- Metodi che iniziano con due underscore sono privati e parzialmente protetti quando la classe viene estesa.
- I metodi speciali iniziano e terminano con due underscore:
  - **__init__** costruttore
  - **__repr__** rappresentazione
  - **__getattr__** per emulare l'accesso ad una attributo:
    - a.x --> a.__getattr('x')
  - **__getitem__** per emulare un tipo contenitore (tipo le liste):
    - a[x] --> a.__getitem__('x')
  - **__add__**, **__mul__**, etc. per emulare somme, moltiplicazioni etc.
  - **__call__** l'oggetto può essere chiamato come una funzione

Installazione MDSplus

- Cliccare sull'ultima versione di MDSplus https://pypi.python.org/pypi/MDSplus
- Scaricare l'uovo "egg" corrispondente al python richiesto e metterlo nella stessa cartella dell'eseguibile.
- Aprire una finestra terminale dalla propria installazione di winpython cliccando su "winpython command prompt"
- Lanciare il comando:
  - easy_install nomedell.egg (usare <tab> per completare il nome del file, se non si completa da solo probabilmente non si è nella directory giusta)
- nel menù di spyder "Tools", "PYTHONPATH Manager" aggiungere la cartella "ownCloud\CorsoPython2016/FTUData" e Synchronize
- Test:

```
>>> import tokdata
>>> ipla = tokdata.ftudata(25255,'zzzzed.ipl')
>>> ipla.plot()
```

Alcune note di stile

- Si trovano sul PEP 8
- Indentare usando 4 spazi (non usare tab).
- Rimanere entro gli 80 caratteri (o comunque una lunghezza ragionevole).
- Nomi dei moduli in minuscolo.
- Nomi delle classi in CamelCase.
- Nomi delle funzioni in minuscolo_con_underscore (se necessario).
- Costanti in MAIUSCOLO.
- Nomi dei metodi come quelli delle funzioni
- Attributi e metodi privati iniziano con uno o due underscore

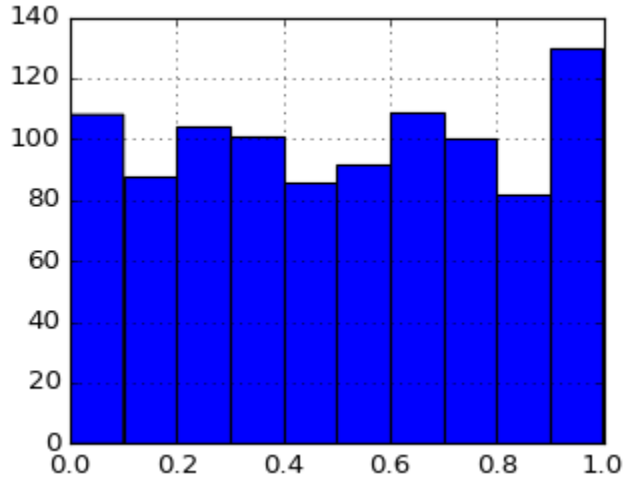
Librerie Scientifiche

Sito web di riferimento: www.scipy.org

- numpy: package di base per array multidimensionali
- scipy: libreria fondamentale per il calcolo scientifico
- sympy: analisi simbolica (non potente come Mathematica)
- pandas: statistica di base
- matplotlib: Grafica 2D

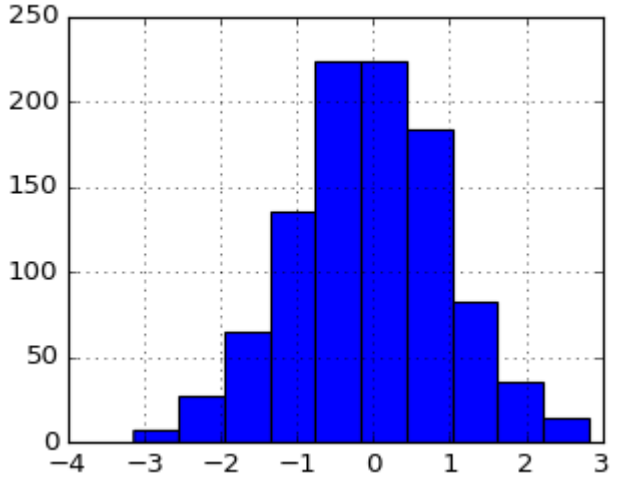
numeri random

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```




```
>>> a = np.random.rand(1000)
>>> plt.figure(figsize=(4.3,3.35),facecolor='w')
>>> plt.hist(a)
>>> plt.grid('on')
```

```
>>> a = np.random.randn(1000)
>>> plt.hist(a)
```



Algebra lineare

```
>>> a = np.array([[3,1], [1,2]])
>>> b = np.array([9,8])
>>> x = np.linalg.solve(a, b)
   array([ 2.,  3.])
>>> np.dot(a,x)
   array([ 9.,  8.])
```

$$\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 9 \\ 8 \end{bmatrix}$$

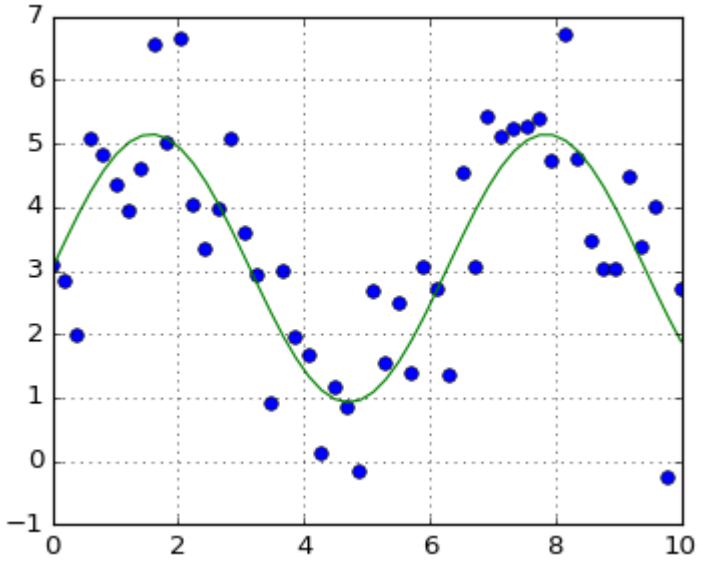
```
>>> a = np.array([[3,1],[1,2],[1,1]])
>>> b = np.array([9,8,11])
>>> x,res,rank,s = np.linalg.lstsq(a, b)
   [ 2.,  4.], 30., 2, [3.8729, 1.4142]
```

$$\begin{bmatrix} 3 & 1 \\ 1 & 2 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ 4 \end{bmatrix} \sim \begin{bmatrix} 9 \\ 8 \\ 11 \end{bmatrix}$$

```
>>> U, s, V = np.linalg.svd(a, full_matrices=False)
```

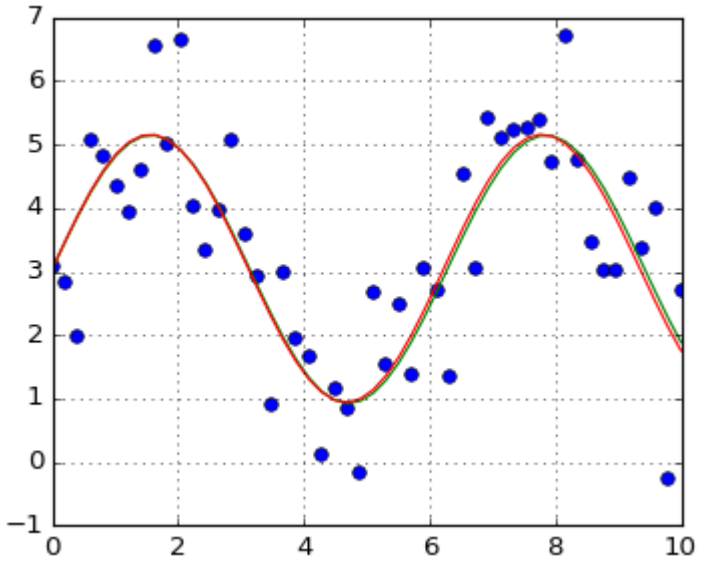
Fit lineare

```
>>> n = 50
>>> t = np.linspace(0,10,n)
>>> y =   3.0   +  2.0 * np.sin(t) + np.random.randn(n)

>>> a = np.c_[np.ones(n),np.sin(t)]
>>> x,res,rank,s = np.linalg.lstsq(a, y)

>>> plt.plot(t, y, 'o')
>>> plt.plot(t, np.dot(a,x))

>>> x
  array([ 3.03484206,  2.1113669 ])
```



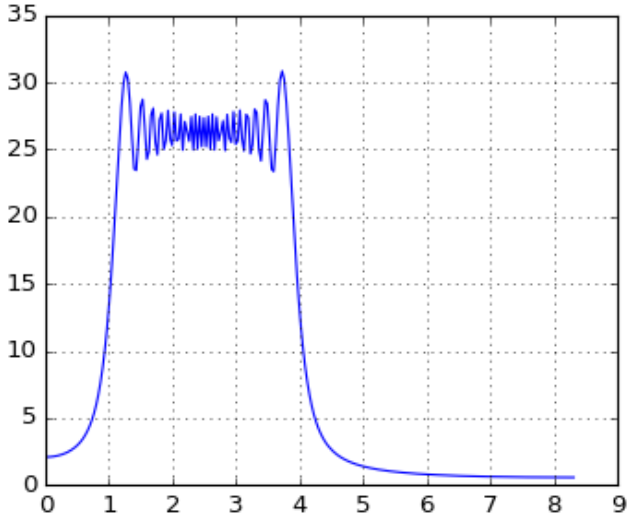
Fit non lineare

```
>>> from scipy.optimize import curve_fit

>>> def funfit(t, a, b, omega):
...     return a + b*np.sin(omega*t)

>>> popt, pcov = curve_fit(funfit, t, y)
   array([ 3.05010557,  2.10975173,  1.00864033])

>>> plt.plot(t, funfit(t, *popt))
```



Fourier

```
>>> n=500
>>> t = np.linspace(0,30,n)
>>> dt = t[1] - t[0]
>>> x = np.sin(2*np.pi*t*(1+t/20.))

>>> xfourier = np.fft.rfft(x)
>>> xfreq = np.fft.rfftfreq(n, dt)

>>> plt.plot(xfreq, np.abs(xfourier))
>>> xfourier
array([ 19.879 +0.000e+00j,  19.909 -5.013e-02j,
        19.995 -1.011e-01j, ...,  -0.047 +9.495e-05j,
        -0.047 +4.747e-05j,  -0.047 +0.000e+00j])

>>> xori = np.fft.irfft(xfourier)
>>> np.angle(xfourier)

>>> xfourier.real
>>> xfourier.imag
```



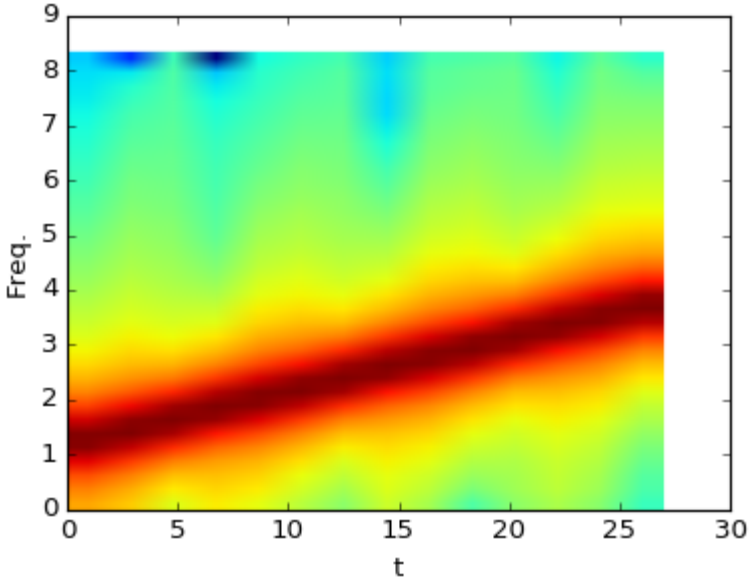
l'inversa

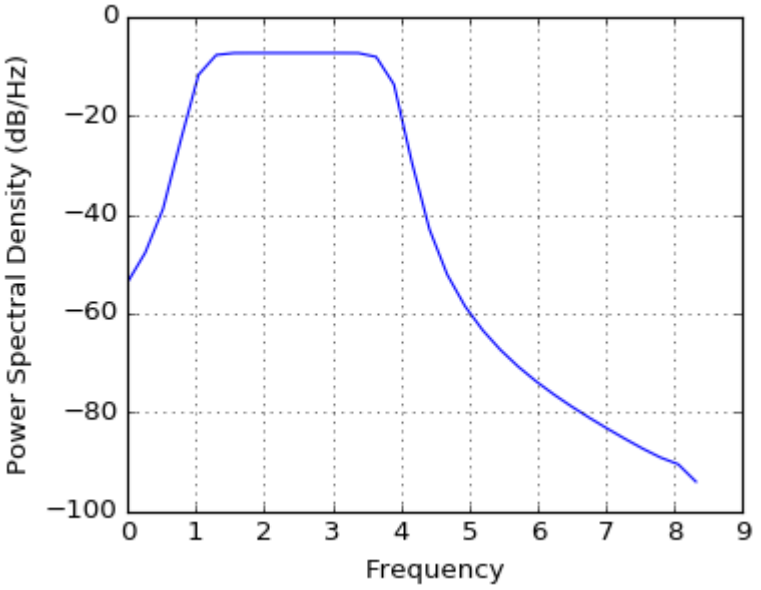
la fase

la parte reale e quella immaginaria

Power Spectrum

```
>>> plt.psd(x, Fs=1.0/dt,NFFT=64,noverlap=32)
```

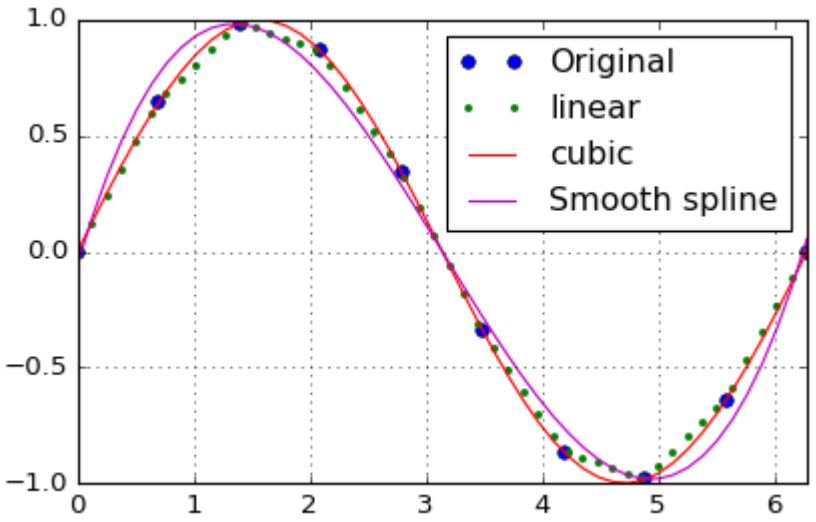





```
>>> plt.specgram(x, Fs=1.0/dt,NFFT=64,noverlap=32)
plt.xlabel('t')
plt.ylabel('Freq.')
```

Interpolazione e spline

```
>>> from scipy.interpolate import interp1d
>>> from scipy.interpolate import UnivariateSpline

>>> x = np.linspace(0,2*np.pi)
>>> xp = np.linspace(0,2*np.pi,10)
>>> yp = np.sin(xp)
```



Le spline tipicamente usano le librerie DIERCKX, ma non solo.
Interpolazioni anche in due dimensioni.

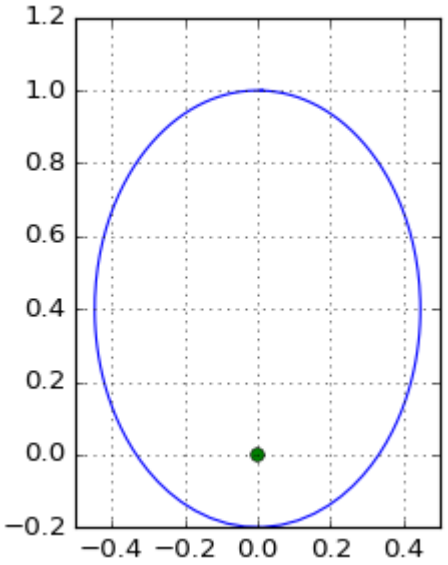
```
>>> y = np.interp(x, xp, yp)
>>> f = interp1d(xp,yp,kind='cubic')
>>> fu = UnivariateSpline(xp,yp,s=0.1)

>>> plt.plot(xp,yp,'o', x,y,'.', x, f(x), x, fu(x))
```

Soluzione di ODE

```
>>> from scipy.integrate import ode

>>> def forzavel(t, y, massa):
      vel = y[0:2); pos = y[2:4]
      r = np.hypot(pos[0], pos[1])
      acc = - massa/r**3 * pos
      return np.r_[acc, vel]
```

Le librerie usate sono la **voda.f** , e simili scritte in Fortran, purtroppo usano dei common internamente e questo implica che solo una integrazione alla volta è possibile

```
>>> r = ode(forzavel).set_integrator('vode', method='adams')
>>> r.set_f_params(3.0)
>>> r.set_initial_value([1.0,0.0,0.0,1.0])
>>> t1 = 1.7; dt = 0.01

>>> sols = []
>>> while r.successful() and r.t < t1:
       sols.append((r.t, r.integrate(r.t+dt)))

>>> t,yy = zip(*sols)
>>> y=np.c_[yy]
```



Serial port Com with Python

- Amongst the many possibilities of Python, communication with the Serial (COM) port of the computer where it is running ranks very high.
- The key library is pySerial, see for details:

➢ https://pyserial.readthedocs.org/en/latest/index.html

- An important introductory example is to detect the available serial ports, on Python prompt enter: (pySerial needs to be installed)

```
>> from serial.tools import list_ports
>> list_ports.comports()
Python OUT >>
[['/dev/cu.Nokia206-COM1', 'n/a', 'n/a'],
['/dev/cu.Bluetooth-Modem', 'n/a', 'n/a'],
['/dev/cu.usbmodem411', 'Arduino Due Prog. Port', 'USB
VID:PID=2341:3d SNR=9523834334351C02112']]
```

In the example an Arduino DUE is connected to port 411. The example is described in:

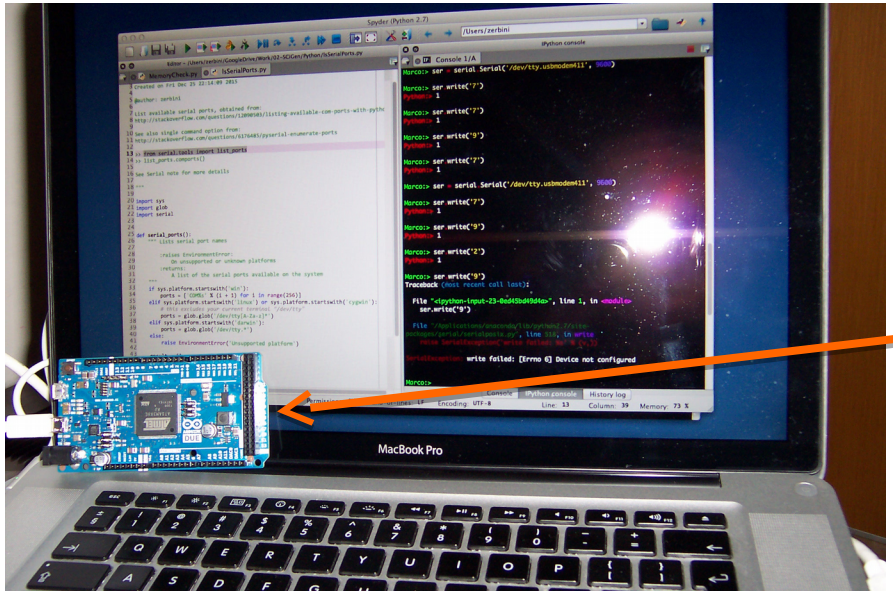
http://stackoverflow.com/questions/12254378/how-to-find-the-serial-port-number-on-mac-os-xhttp://stackoverflow.com/questions/12254378/how-to-find-the-serial-port-number-on-mac-os-x

Python Serial communication, cont.

- Next step is to use Python to control a device, see: www.akeric.com/blog/?p=1140
- Create and upload the Arduino Module SerialPythonRX, as in the webpage.
- Connect Arduino ™ to the computer via USB and Detect the name of serial port
- Open a Python session and enter:

  ```
  >> import serial
  >> ser = serial.Serial('/dev/tty.usbmodem411', 9600)
  >> ser.write('5')
  ```

- the LED connected to pin 13 will blink 5 times. Make sure to use correct Serial port name.



Arduino is a programmable prototyping board based on a ATMega processor, connected via USB to the control computer.

Conclusioni

- Python è un linguaggio molto esteso. Questa è stata solo una introduzione.
- Scipy è una libreria che permette molte altre cose.
- Moltissime librerie sono disponibili su internet.
- Grazie per il vostro tempo!