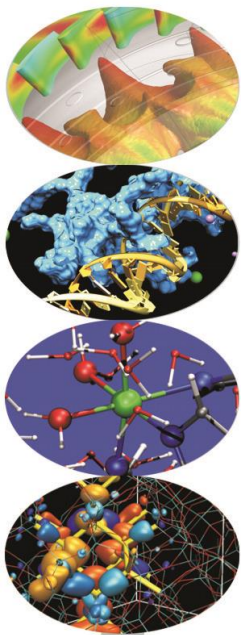
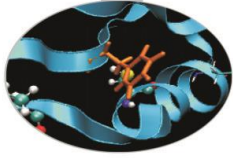




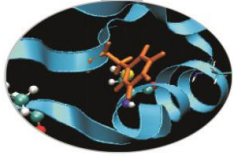
IDE



Ordine del Modulo



- Interactive Development Environment (IDE) e installazioni disponibili in ambito scientifico e non:
 - IDLE (non specifico)
 - Enthought
 - Python-XY
 - I-Python
- Download e Installazioni
- Hands-on

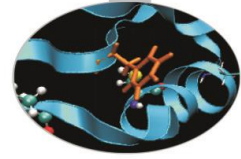


Installazione Python

- Python può essere utilizzato su diverse piattaforme Linux, Macintosh, Windows.
- La distribuzione principale di python è scaricabile <http://www.python.org/>

In Linux Python solitamente è installato con il sistema operativo

```
[alinve@lagrange ~]$ which python
/usr/bin/python
[alinve@lagrange ~]$ man python
[alinve@lagrange ~]$ python -V
Python 2.4.3
```

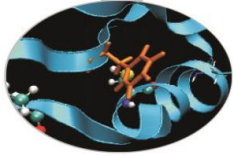


Installazione Python

- I pacchetti di Python vengono solitamente distribuiti usando il modulo Distutils.
- L'installazione si fa in pochi passi.
- Installazione del modulo test-1.0.tar.gz

```
gunzip -c test-1.0.tar.gz | tar xf - # unpacks  
into directory test-1.0  
cd test-1.0  
python setup.py install
```

- Il comando `python setup.py install` costruisce e installa i moduli.
- I moduli vengono installati sotto
`/pythonHOME/lib/python2.6/site-packages`

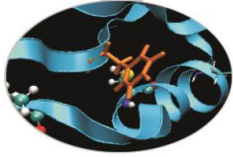


Installazione Python

- I pacchetti di Python vengono solitamente distribuiti usando il modulo Distutils.
- L'installazione si fa in pochi passi.
- Installazione del modulo test-1.0.tar.gz

```
gunzip -c test-1.0.tar.gz | tar xf - # unpacks  
into directory test-1.0  
cd test-1.0  
python setup.py install
```

- Il comando `python setup.py install` costruisce e installa i moduli.
- I moduli vengono installati sotto
`/pythonHOME/lib/python2.6/site-packages`

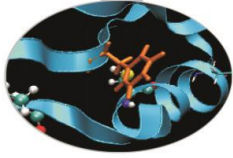


Installazione Python

- Per specificare ulteriori path di ricerca è possibile modificare la variabile d'ambiente PYTHONPATH

```
$ export PYTHONPATH=$HOME
>>> import sys
>>> sys.path
['', '/home/interni/alinve',
 '/usr/lib64/python24.zip', '/usr/lib64/python2.4',
 '/usr/lib64/python2.4/plat-linux2',
 '/usr/lib64/python2.4/lib-tk',
 '/usr/lib64/python2.4/lib-dynload',
 '/usr/lib64/python2.4/site-packages',
 '/usr/lib64/python2.4/site-packages/Numeric',
 '/usr/lib64/python2.4/site-packages/PIL',
 '/usr/lib64/python2.4/site-packages/gtk-2.0',
 '/usr/lib/python2.4/site-packages']
```

Installazione Python

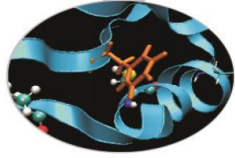


- Per modificare il prefix path della home di installazione di Python è si modifichi la variabile PYTHONHOME
- Per vedere la lista dei moduli installati

```
>>>help('modules')
```

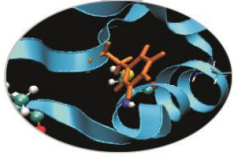
```
Please wait a moment while I gather a list  
of all available modules...
```

Installazione base: IDLE



IDLE: è l'interfaccia built-in di Python sviluppata con la libreria base GUI (**tkinter**). IDLE ha seguenti caratteristiche:

- 100% puro Python
- cross-platform: windows/linux
- text editor multi-window con multipli undo
- colorazione del testo basato sulla sintassi Python ed indentazione automatica del testo.
- completamento delle funzioni
- interprete interattivo
- debugger
- necessita della libreria Tk



Installazione with bacterias included

Esistono installazioni specifiche per il calcolo scientifico e le applicazioni ingegneristiche che contengono già numerosi moduli di utilità:

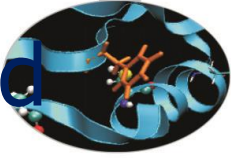
- Enthought (largamente la più diffusa e con un livello di mantenimento e di investimenti significativo)
- L'installazione la si può scaricare dal link:

<http://www.enthought.com/>



Cosa contiene:

<http://www.enthought.com/products/epdlibraries.php>



Installazione with bacteria included

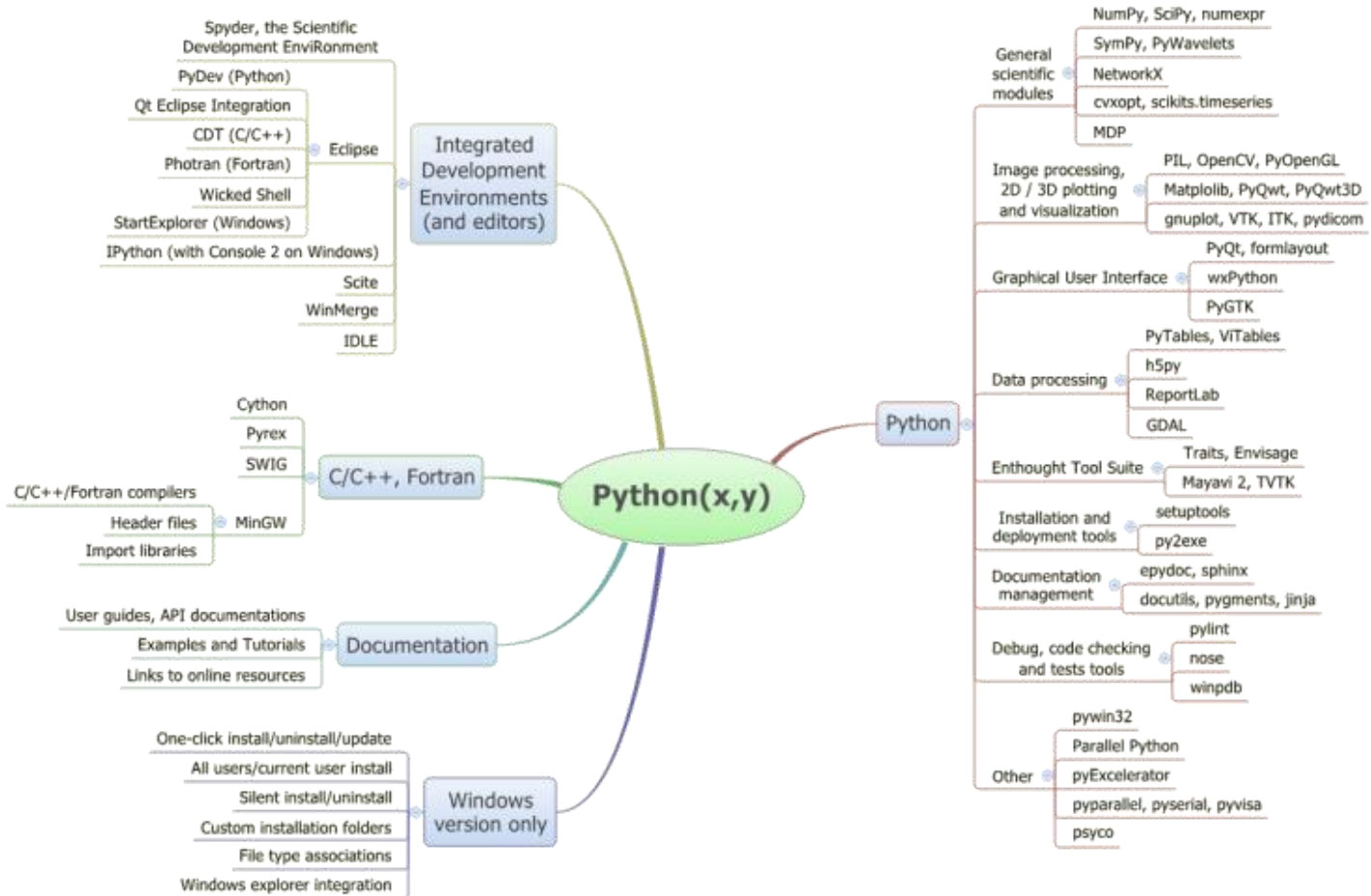
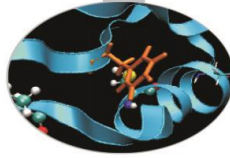
Python-XY (di recente sviluppo e molto promettente) esiste per ora solo per Windows e per Ubuntu/linux

- L'installazione la si può scaricare dal link:

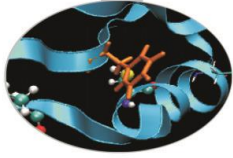
<http://www.pythonxy.com/>



Installazione with bacteriaes included



Installazione with batteries included

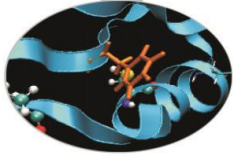


IP[y]:

- I-Python (la shell interattiva più potente per questo ambito)
- L'installazione la si può scaricare dal link:

<http://ipython.scipy.org/>

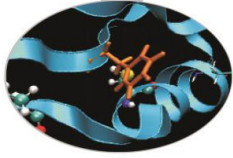
IPython



Caratteristiche:

- Visualizzazione della lista dei metodi e delle proprietà di ogni variabile premendo il tasto TAB.
- Visualizzazione della documentazione di ogni oggetto e funzione in maniera semplice e immediata: `nome_funzione?<enter>`
- Definizione di comandi personalizzati e di alias (es. `%alias !ls -la`).
- Possibilità di eseguire comandi di sistema (es. `!ls`) e di recuperare il loro output sotto forma di stringa o lista (`ipython -p pysh, $$v=ls`).
- Mantiene uno storico dei comandi eseguiti (`%hist, _ih`).
- Integrabile nei programmi come utile strumento per il debugging.
- molto altro...

Demo / Hands on



Connessione ai pc linux

- apertura Ipython
- esecuzione comandi
- lancio di script (`#!/bin/python/` e `PYTHONPATH`)
- comando `sys.argv[*]`
- comando `help()`
- introspezione (`dir module`)
- `import pylab`
- `simple-plot`



Note: indentazione/blocchi

- python si basa sulla indentazione del codice
- blocchi logicamente connessi hanno lo stesso livello di indentazione

```
x = 2.3
```

```
y = 1.2
```

```
def test(x,y):
```

```
    if x==y: print 'the two number are equal'
```

```
    elif x > y: print ' the first number is the greater'
```

```
    else: print ' the former number is the greater'
```

```
test(x,y)
```

```
for I in range(2,5,1):
```

```
    for J in range(5,1,-1):
```

```
        print 'now testing : ', I,J
```

```
        test(I,J)
```



Modalità di lancio

Uno script python può essere lanciato in vari modi:

1) [ponzini@prompt] python script.py

(implica che nel PATH o nel PYTHONPATH sia indicata la locazione di installazione di python)

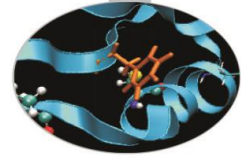
2) [ponzini@prompt] ./script.py (implica che lo script sia in modalità eseguibile (chmod +x))

1) dalla shell di i-python

```
In [3]: !ls *.py
```

```
21/10/2009 11.54          376 pie-hours-py.py
```

```
In [4]: run pie-hours-py.py
```

Nota parametri di input: `sys.argv`

- Uno script python può avere o meno parametri di input necessari alla corretta esecuzione dello script:

```
[prompt@ponzini] python script.py parametro_1 parametro_2 parametro_3
```

- Per prevedere questa modalità di lancio si può fare uso degli appositi parametri `argv[*]` contenuti nel modulo `sys`:
 - `argv[0]`: nome dello script che stiamo eseguendo
 - `argv[i]`: qualsiasi parametro_*i*

Esempio



script che prende 2 parametri di input

```
import sys
```

```
usage="""necessita di due parametri di input (param1, param2)
```

```
correct usage: python script.py param1 param2"""
```

```
if __name__ == '__main__':
```

```
    if len(sys.argv) < 2:
```

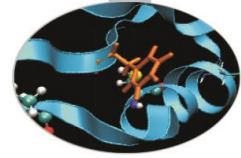
```
        print 'lo script: ',sys.argv[0],usage
```

```
        sys.exit(0) # termina dopo aver stampato la stringa di usage
```

```
    param1 = sys.argv[1]
```

```
    param2 = sys.argv[2]
```

```
    print 'uso I due parametri passati al momento dell'invocazione dello  
script:\ ',param1, param2
```



raw_input()

Nel caso si volesse invece avere una interazione con l'utente dello script è possibile fare uso di una funzione predisposta per accettare parametri a run-time dallo stdin.

Esempio:

script che prende 2 parametri di input

```
import sys
```

```
if __name__ == '__main__':
```

```
    while(True):
```

```
        print 'PLEASE INSERT AN INTEGER NUMBER IN THE RANGE 0-10'
```

```
        param1 = raw_input()
```

```
        if int(param1) in range(11): # notare che raw_input restituisce una stringa
```

```
            while(True):
```

```
                print 'PLEASE INSERT A CHAR PARAMETER IN [A,B,C]'
```

```
                    param2 = raw_input()
```

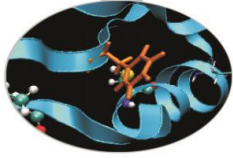
```
                    if param2 in ['A','B','C']:
```

```
                        print 'uso I due parametri passati dall utente: ',param1,  
                            param2
```

```
                        sys.exit()
```

```
                    else: print 'TRY AGAIN PLEASE'
```

```
        else: print 'TRY AGAIN PLEASE'
```



input()

Come *raw_input* ma con un *eval()*: `eval(raw_input(prompt))`

Esempio:

script che prende 2 parametri di input

```
import sys
```

```
if __name__ == '__main__':
```

```
    while(True):
```

```
        print 'PLEASE INSERT AN INTEGER NUMBER IN THE RANGE 0-10'
```

```
        param1 = input() #notare che c'è eval e int() non va messo
```

```
        if param1 in range(11):
```

```
            while(True):
```

```
                print 'PLEASE INSERT A CHAR PARAMETER IN [A,B,C]'
```

```
                param2 = input() # causa eval bisogna usare il simbolo di carattere
```

```
                if param2 in ['A','B','C']:
```

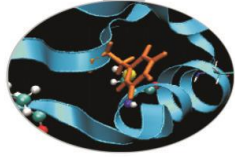
```
                    print 'uso I due parametri passati dall utente: ',param1, param2
```

```
                    sys.exit()
```

```
                else: print 'TRY AGAIN PLEASE'
```

```
            else: print 'TRY AGAIN PLEASE'
```

eval()



Come visto le due funzioni sembrano praticamente equivalenti ma eval() che cosa produce:

```
>>> help(eval)
```

```
eval(...)
```

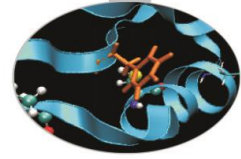
```
eval(source[, globals[, locals]]) -> value
```

Evaluate the source in the context of globals and locals.

The source may be a string representing a Python expression or a code object as returned by compile().

The globals and locals are dictionaries, defaulting to the current globals and locals. If only globals is given, locals defaults to it.

Diventa strategico quando voglio passare da input qualcosa che venga valutato (es: nome di una funzione).



input() versione2

script che prende 2 parametri di input; il secondo parametro è il nome di una funzione

```
import sys
def f(x):
    print x
def g(x):
    print -x
if __name__ == '__main__':
    while(True):
        print 'PLEASE INSERT AN INTEGER NUMBER IN THE RANGE 0-10'
        param1 = input()
        if param1 in range(11):
            while(True):
                print 'PLEASE INSERT THE NAME OF A FUNCTION'
                param2 = input() #grazie alla presenza di eval ha senso l'istruzione che segue
                if param2 in [f,g]:
                    print 'uso i due parametri passati dall utente: ',param1, param2
                    param2(param1)
                    sys.exit()
                else: print 'TRY AGAIN PLEASE'
            else: print 'TRY AGAIN PLEASE'
```