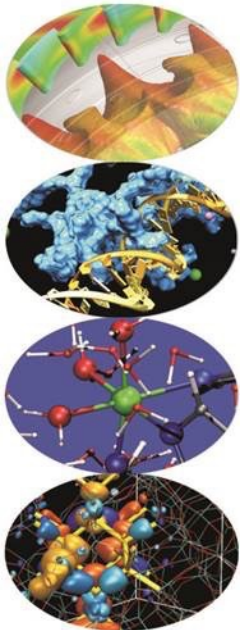
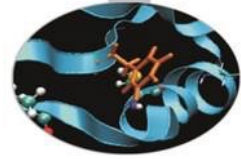


Python: introduzione al linguaggio

seconda parte

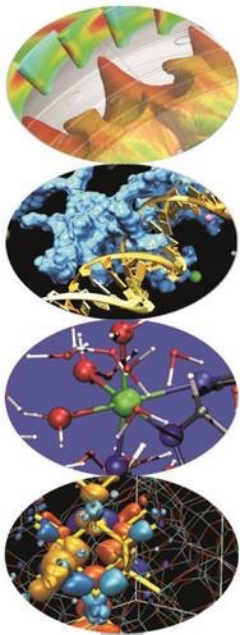
Paolo D'Onorio De Meo
p.donoriodemeo@cineca.it

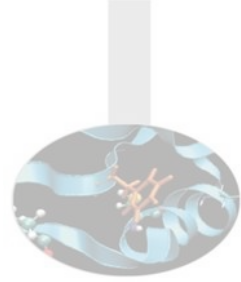




Corso python

Utilizzo di I/O





Esplorare un file: concetti e *parsing*

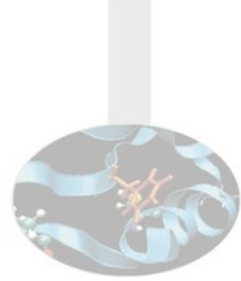
Ipotizziamo di avere un file composto da linee del tipo:

```
1, Joe, Doe, 1976
```

Python legge le righe del file, che sono stringhe.

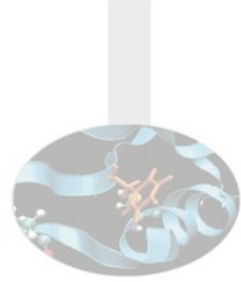
Bisogna applicare un lavoro in più per ottenere e gestire i dati di ogni linea:

questa operazione si chiama **parsing**



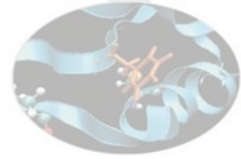
Leggere un file

1. Aprire il file
 - a. Utilizzo di un *filehandler*
 - b. Il filehandler viene creato dalla funzione `open`
 - c. Attenzione al path relativo / assoluto
2. Lettura del contenuto
 - a. Di solito basandoci su un ciclo
3. Chiusura del file
 - a. Liberare le risorse, chiudendo il filehandler



Esempio di file

```
0.0 0.0 0.0  
1.0 0.0 0.0  
1.0 1.0 0.0  
0.0 1.0 0.0  
0.0 0.0 1.0  
1.0 0.0 1.0  
1.0 1.0 1.0  
0.0 1.0 1.0
```



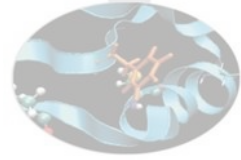
Leggere un file: il codice

1. Aprire
2. Leggere
3. Chiudere

```
>>> fh=open('points.txt')
>>> fh
<open file 'points.txt', mode 'r' at 0xb748df98>

>>> fh.read()
'0.0 0.0 0.0\n1.0 0.0 0.0\n1.0 1.0 0.0\n0.0 1.0
0.0\n0.0 0.0 1.0\n1.0 0.0 1.0\n1.0 1.0 1.0\n0.0 1.0
1.0\n'
```

```
>>> fh.close()
>>> fh
<closed file 'points.txt', mode 'r' at 0xb748df98>
```

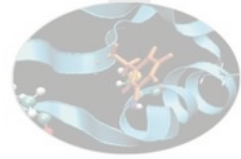


Leggere un file: riga per riga

1. Aprire
2. Leggere
3. Chiudere?

```
>>> fh=open('points.txt')
>>> while True:
...     line = fh.readline()
...     if line=="":
...         break
...     print line.replace('\n','')
... 
```

```
0.0 0.0 0.0
1.0 0.0 0.0
1.0 1.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
1.0 0.0 1.0
1.0 1.0 1.0
0.0 1.0 1.0
```



Leggere un file: parsing

1. Leggere
2. Chiudere

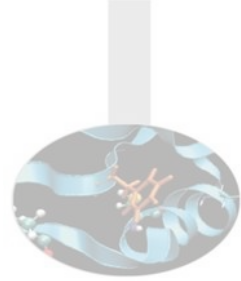
```
>>> fh.seek(0,0)
>>> lista = list()
>>> lista
[]

>>> while True:
...     line = fh.readline()
...     if line=="":
...         break
...     lista.append(line.replace('\n','').split(' '))
...

>>> lista
[['1.0', '0.0', '0.0'], ['0.0', '1.0', '0.0'], ['1.0',
'0.0', '1.0'], ['0.0', '1.0', '1.0']]

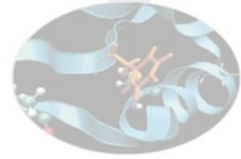
>>> lista[1]
['0.0', '1.0', '0.0']

>>> fh.close()
```

Il formato CSV

- CSV sta per “Comma Separated Values”
- Si tratta di files con dati separati da virgole
 - si possono usare anche altri separatori
 - es punto e virgola, tab, ecc.
- Ogni riga rappresenta un record diverso
 - La prima riga indica solitamente i nomi dei campi
- Tutti i fogli di calcolo possono essere rappresentati con questo formato



Il modulo CSV di Python

1. Import
2. Handler
3. Iterazioni

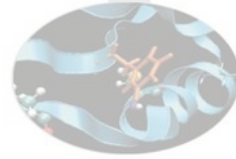
```
>>> import csv

>>> lista = list()

>>> lines = csv.reader(open('points.csv'))
>>> lines.next()
['X', 'Y', 'Z']

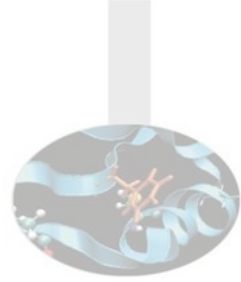
>>> for line in lines:
...     lista.append(line)
...

>>> lista
[['0.0', '0.0', '0.0'], ['1.0', '0.0', '0.0'], ['1.0',
'1.0', '0.0'], ['0.0', '1.0', '0.0'], ['0.0', '0.0',
'1.0'], ['1.0', '0.0', '1.0'], ['1.0', '1.0', '1.0'],
['0.0', '1.0', '1.0']]
```



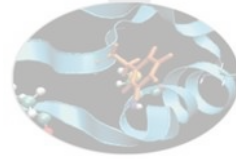
Il formato XLS

```
#Un diverso delimitatore?  
>>> rows = csv.reader(open("passwd"), delimiter=':')  
#Il dialetto XLS  
>>> rows = csv.reader(open("data.csv"), dialect='excel')  
#Indovinare il dialetto  
dialect = csv.Sniffer().sniff(open('data.csv').read())  
rows = csv.reader(open("data.csv"), dialect=dialect)
```



Scrivere un file

1. Aprire il file in modalità scrittura
 - a. Utilizzo di un *filehandler*
 - b. Il filehandler viene creato dalla funzione `open`
 - c. Attenzione al path relativo / assoluto
2. Scrittura del contenuto
 - a. Di solito a partire da una struttura dati
3. Chiusura del file
 - a. Liberare le risorse, chiudendo il filehandler



Scrivere un file

#Come si apre un file in scrittura?

```
>>> help('open')
```

```
Help on built-in function open in module __builtin__:
```

```
open(...)
```

```
open(name[, mode[, buffering]]) -> file object
```

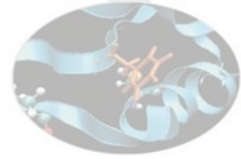
Open a file using the file() type, returns a file object. This is the preferred way to open a file. See file.__doc__ for further information.

#Quali sono i mode a disposizione?

```
>>> print(file.__doc__)
```

```
file(name[, mode[, buffering]]) -> file object
```

Open a file. The mode can be 'r', 'w' or 'a' for reading (default), writing or appending. The file will be created if it doesn't exist when opened for writing or appending; it will be truncated when opened for writing.



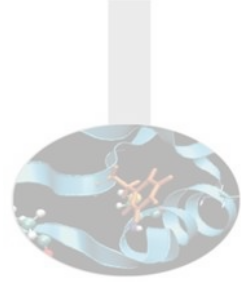
Scrivere un file

1. Aprire
2. Scrivere
3. Chiudere

```
#Il file viene creato
>>> fo = open("/tmp/foo.txt", "w")

#Buffer di scrittura
>>> fo.write( "Python is a great language.\nYeah its great!!\n");

#Il contenuto viene riversato nel file aperto
>>> fo.close()
```



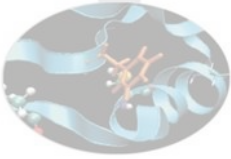
Esercizio: lettura e scrittura file

Aprire il file 'points.txt', ottenere la seconda colonna e moltiplicarne ogni valore per 7,3 .

Scrivere un file di output 'ypoints.csv' che contiene una sola riga con tutti i nuovi valori in formato csv.

(Consigliato l'uso del modulo python apposito.)

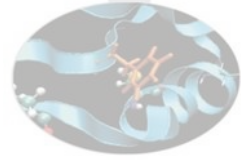
Salvataggio e recupero dei dati



Pickle è il modulo per gestire i dati su file. cPickle è la sua versione in C.

Permettono di scrivere e recuperare su file delle strutture dati di Python.

```
>>> dizionario = { 'A' : 1, 'B' : 2 }  
#Dump di un dizionario  
>>> import cPickle  
>>> fh = open("dizionario.data", 'w')  
>>> cPickle.dump(dizionario, fh)  
>>> fh.close()  
  
#Recupero del dato salvato  
>>> fh = open("dizionario.data")  
>>> recupero = cPickle.load(fh)  
>>> fh.close()  
>>> recupero  
{ 'A': 1, 'B': 2 }
```

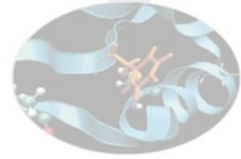
Accesso file system via OS

os è il modulo che consente attraverso le primitive del sistema operativo l'accesso a files e directory con strutture Python.

```
>>> import os
>>> os.getcwd()
'/home/paulie'

>>> os.chdir('../')
>>> os.getcwd()
'/home'

>>> os.listdir("/var")
['metrics', 'opt', 'run', 'mail', 'lock',
'local', 'spool', 'backups', 'lib', 'log',
'cache', 'tmp', 'crash']
```



Accesso file system via OS

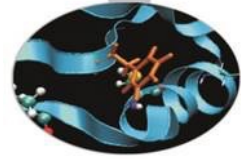
os è il modulo che consente attraverso le primitive del sistema operativo l'accesso a files e directory con strutture Python.

```
>>> import os

>>> os.remove("/tmp/foo.txt")
>>> os.remove("/tmp/foo.txt")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [Errno 2] No such file or directory: '/tmp/foo.txt'

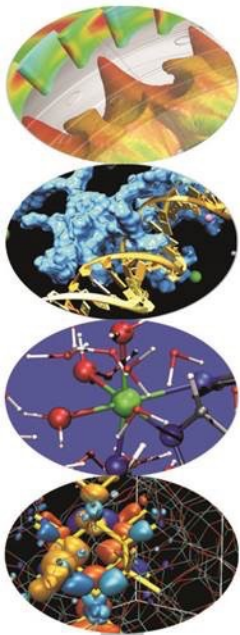
>>> os.rename("/tmp/foo2.txt", "/tmp/foo.txt")
>>> os.mkdir("/tmp/newdir")
>>> os.listdir("/tmp/newdir")
[]

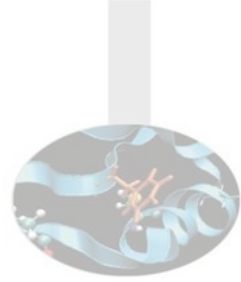
>>> os.path.split("/tmp/foo2.txt")
('/tmp', 'foo2.txt')
```



Corso python

Rendere modulare il codice

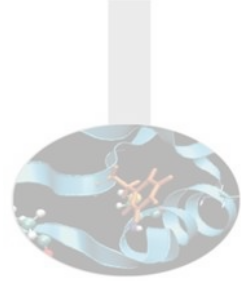




Le funzioni

Le funzioni sono il metodo più diffuso per modularizzare il codice.

- Una funzione prende uno o più parametri come input
- Esegue delle operazioni basate sui parametri ricevuti
- Ritorna il valore calcolato



Le funzioni

Le funzioni sono il metodo più diffuso per modularizzare il codice.

```
def FunctionName(argument1, argument2, ...):  
    """ Optional description """  
    ... FUNCTION_CODE ...  
return DATA
```

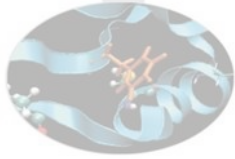


Un esempio semplice di funzione

Vogliamo avere sempre a disposizione nel nostro codice una funzione in grado di raddoppiare un numero.

```
>>> def raddoppia(numero):  
...     """ Moltiplica il numero in input per due """  
...     return numero*2  
...  
>>> raddoppia(4)  
8  
>>> a=3  
>>> raddoppia(a)  
6  
>>> b = raddoppia(a*2)  
>>> b  
12  
  
>>> raddoppia  
<function raddoppia at 0xb74cacdc>  
>>> help(raddoppia)  
raddoppia(numero)  
    Moltiplica il numero in input per due
```

Funzioni: uno o più parametri



Una funzione può avere uno o più parametri. Senza parametri non ha molto senso.

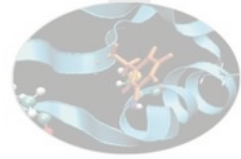
Una funzione può naturalmente chiamare al suo interno altre funzioni.

```
>>> def ucfirst(stringa):  
...     return stringa[0].upper() + stringa[1:]  
...
```

```
>>> def info(nome, cognome):  
...     print "Cognome:\t" + ucfirst(cognome)  
...     print "Nome:\t\t" + ucfirst(nome)  
...
```

```
>>> info("paolo", "d'Onorio De Meo")  
Cognome:      D'Onorio De Meo  
Nome:         Paolo
```

```
>>> def inutile():  
...     return "inutile"  
...  
>>> inutile()  
'inutile'
```



Funzioni: valori di default

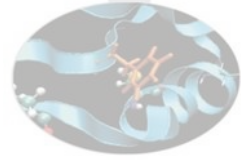
Un parametro puo' essere opzionale, se ne indichiamo il valore di default durante la definizione della funzione.

```
>>> def info(nome, cognome, eta = 18):
...     print "Cognome:\t" + ucfirst(cognome)
...     print "Nome:\t\t" + ucfirst(nome)
...     print "Eta:\t\t" + str(eta)
>>> info("Mario","rossi")
Cognome:      Rossi
Nome:         Mario
Eta:          18
>>> info("luca","Bianchi",30)
Cognome:      Bianchi
Nome:         Luca
Eta:          30
>>> info("Luca")

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: info() takes at least 2 arguments (1 given)

#Dove lo avevamo gia' incontrato?

>>> fh=open("filename.txt")
>>> fh=open("filename.txt",'w')
```

Funzioni: valori di ritorno

Possiamo ritornare
anche un dato
complesso.

```
>>> def ucfirst(stringa):  
...     return stringa[0].upper() + stringa[1:].lower()  
...  
  
>>> def info(nome, cognome, eta = 18):  
...     return [ucfirst(nome), ucfirst(cognome), eta]  
...  
  
>>> info("Mario", "Rossi")  
['Mario', 'Rossi', 18]  
>>> info("Luca", "Bianchi", 30)  
['Luca', 'Bianchi', 30]  
  
>>> def info(nome, cognome, eta = 18):  
...     return {'Nome':ucfirst(nome),  
...           'Cognome':ucfirst(cognome), "Eta'":eta }  
...  
>>> info("Mario", "Rossi")  
{'Cognome': 'Rossi', "Eta'": 18, 'Nome': 'Mario'}
```

Funzioni: lo scope di una variabile

Attenzione al valore
di una variabile in
base al contesto
in cui viene usata

```
>>> def test(numero):  
...     x=3  
...     print("Valore di x = " + str(x))  
...     return numero*2  
...
```

```
>>> x=55  
>>> test(4)  
Valore di x = 3  
8  
>>> x  
55
```

```
>>> numero  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'numero' is not defined
```

Funzioni: numero di valori variabile

Il numero di
parametri passato
a una funzione
può essere
definito variabile!

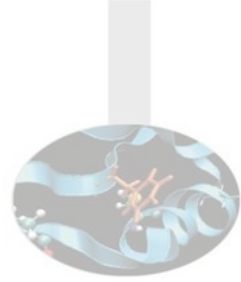
```
def average(*numbers):  
    if len(numbers)==0:  
        return None  
    else:  
        total = sum(numbers)  
        return float(total)/len(numbers)
```

```
>>> average(2,4)
```

```
3
```

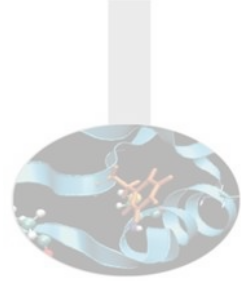
```
>>> average(6,4,8)
```

```
9
```



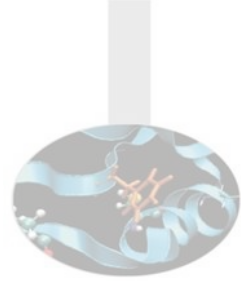
Esercizio sulle funzioni

1. Scrivere e testare una funzione che prenda una lista di parole e ritorni la dimensione della più lunga
2. Scrivere e testare una funzione che calcoli il numero di caratteri contenuti in un file



MODULI: il concetto

- Cos'è un modulo?
 - Un file contenente definizioni di funzioni e tipi di strutture Python
- Namespace
 - Uno spazio indipendente di nomi
 - `math.log` è diverso da eventuale funzione `log` definita da noi
- Il nome del modulo
 - il file `"mio_modulo.py"` contiene il modulo `"mio_modulo"`

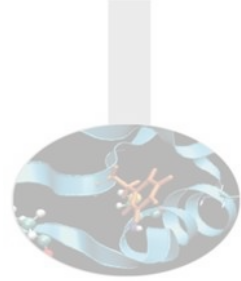


MODULI: import

```
>>> import math
>>> pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
>>> math.pi
3.141592653589793
```

#Don't use "from module import *" unless YOU KNOWN WHAT YOU ARE DOING

```
>>> from math import *
>>> pi
3.141592653589793
>>> dir
<built-in function dir>
>>> dir()
['_builtins__', '__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'math', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```



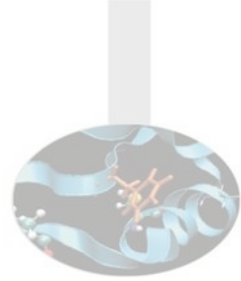
MODULI: dove Python cerca...

1. Nella stessa directory in cui ci troviamo al momento dell'esecuzione dell'interprete
2. Nella stessa directory dove si trovano gli eseguibili del Python
3. In una directory specificata da noi (variabile PYTHONPATH)

```
>>> import sys
```

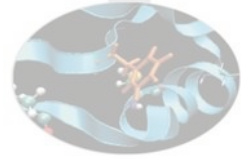
```
>>> sys.path
```

```
['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-i386-linux-gnu', '/usr/lib/python2.7/lib-tk', ... ]
```



MODULI: installazione

- PYTHONPATH
 - Variabile di ambiente del sistema operativo
- Pacchettizzazione di sistema
 - Ubuntu: apt
 - RedHat: yum
 - Suse: yast
- Easy install
 - `$ easy_install`
Il programma "easy_install" non è attualmente installato. È possibile installarlo digitando:
`sudo apt-get install python-setuptools`
 - `sudo easy_install NOMEMODULO`
- Standard
 - `$ python setup.py install`



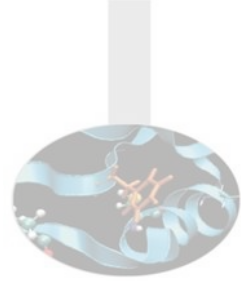
MODULI: creare il proprio

HOME/test.py

```
#un valore di default
zero=0
#lasciare il valore intatto
def lascia(numero=zero):
    return numero
#valore per 2
def raddoppia(numero=zero):
    return numero*2
```

bash:HOME\$ python

```
>>> import test
>>> dir(test)
['__builtins__', '__doc__', '__file__',
 '__name__', '__package__', 'lascia',
 'raddoppia', 'zero']
>>> print __doc__
None
>>> test.lascia()
0
>>> test.lascia(2)
2
>>> test.raddoppia(2)
4
>>> test.raddoppia()
0
>>> test.zero
0
>>> test.raddoppia(test.zero)
0
```



Esercizio

Crea il tuo primo modulo:

L'agenda

Gestire un'agenda telefonica attraverso almeno

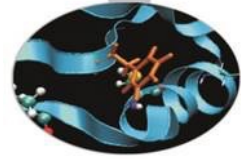
due funzioni:

a) Inserimento dati

(richiesta interattiva di Nome e Numero)

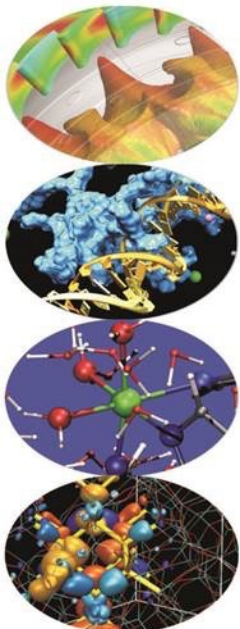
b) Ricerca numero

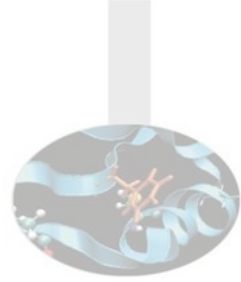
(a partire dal Nome)



Corso python

Gestione degli errori
[Extra]



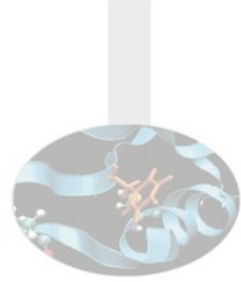


Cos'è l'errore

```
>>> fh = open("nonesisto.csv")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory:
'nonesisto.csv'
```

NAESER'S LAW:

You can make it foolproof, but you can't make it damn-foolproof.

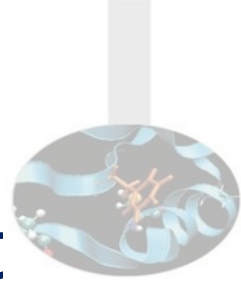


Evitare errori non previsti

Possiamo evitare di cadere in alcuni errori con dei controlli.

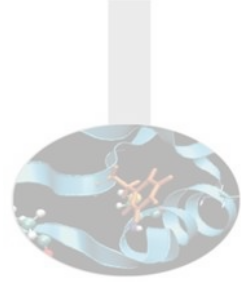
E.g.

```
>>> if os.access("nonesito.csv",os.W_OK):  
...     fh = open("nonesito.csv")  
... 
```



Gestione errori: blocco try/except

```
try:  
    code block 1  
    # ...some error prone code...  
except:  
    code block 2  
    # ...do something with the error...  
[else:  
    code block 3  
    # ...to do when there is no error...  
finally:  
    code block 4  
    #...some clean up code..].
```



Houston abbiamo un problema

```
try:
```

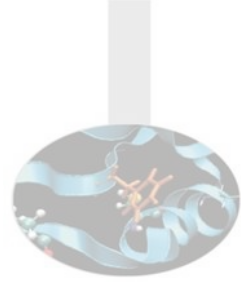
```
    print 0/0
```

```
except:
```

```
    print("Houston, we have a problem...")
```

Il risultato è

Houston, we have a problem...



Caso reale di gestione

```
#Gestione di un vocabolario
```

```
d = {"A": "Adenine", "C": "Cisteine", "T": "Timine", "G": "Guanine"}
```

```
try:
```

```
    print d[raw_input("Enter letter: ")]
```

```
except:
```

```
    print "No such nucleotide"
```

```
#Gestione del tipo di errore!
```

```
d={"A": "Adenine", "C": "Cisteine", "T": "Timine", "G": "Guanine"}
```

```
try:
```

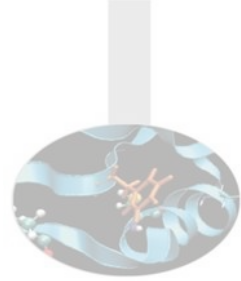
```
    print(d[raw_input("Enter letter: ")])
```

```
except EOFError:
```

```
    print("Good bye!")
```

```
except KeyError:
```

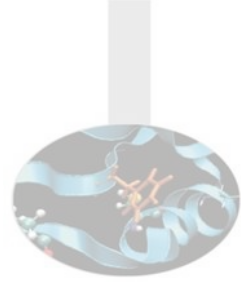
```
    print("No such nucleotide")
```

Tipi di eccezioni

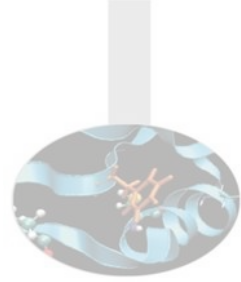
```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError, (errno, strerror):
    print "Errore I/O (%s): %s" % (errno, strerror)
except ValueError:
    print "Non si può convertire il dato in un intero."
except:
    print "Errore inatteso:", sys.exc_info()[0]
    raise
```



Tipi di eccezioni

- exception IOError
- exception ArithmeticError
- exception MemoryError
- exception NameError
- exception OSError
- exception SyntaxError
 - IndentationError
- exception SystemExit
- Un utente può anche definire un proprio tipo di eccezione
 - Richiede conoscenza di OOP



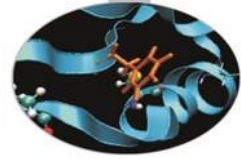
Provocare una *eccezione*

L'istruzione `raise` permette al programmatore di forzare una specifica eccezione.

Per esempio:

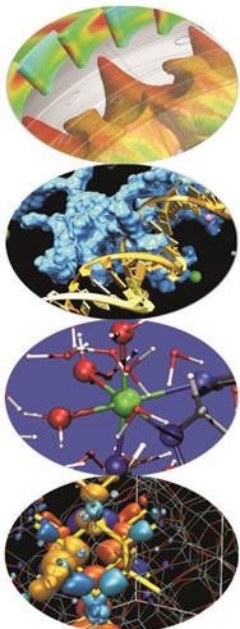
```
>>> raise NameError, 'HiThere'
```

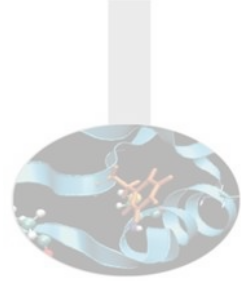
```
Traceback (most recent call last):  
  File "<stdin>", line 1, in ?  
NameError: HiThere
```



Corso python

Come continuare a lavorare sulle basi?





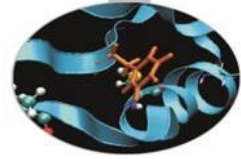
DOCS

Help inline

- introspezione
- documentazione comando `help()`

Help online

- ufficiale del linguaggio
 - <http://www.python.org/doc/current>
- Tradotti in italiano
 - <http://docs.python.it>



Introduzione: fine della seconda parte

Bravi!

