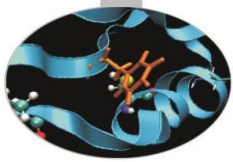
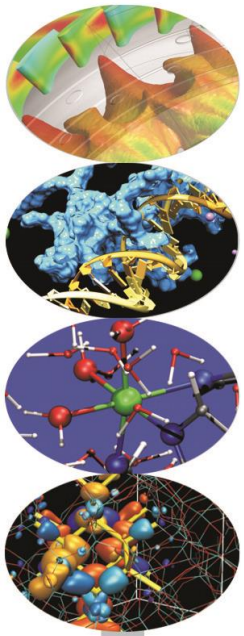
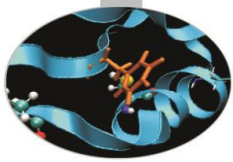


Manipolazione di File e directory



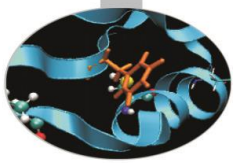
File e Directory



Python dispone di funzionalità per gestire e manipolare files e directories che lavorano su più piattaforme: ricercare files, navigare in directories, manipolare, eliminare e rinominare files.

Le funzionalità più utilizzate sono implementate nei moduli: *os*, *os.path*, *sys, shutil, glob.*

File e Directory

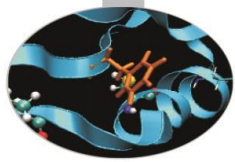


Contenuti:

- *file object*: introduzione all'oggetto *file,* funzioni *built-in* e metodi associati per la manipolazione di un file

- *Handle file and directory*: gestione di file e directory tramite specifici moduli.

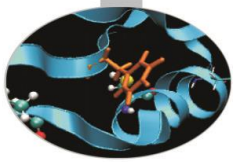
Object File



In Python esiste un oggetto built-in di tipo *file* che utilizzando le API del sistema operativo è in grado di manipolare i file fisici su disco.

Sono disponibili alcune funzioni *built-in* per le operazioni più comuni di manipolazione dei *file object*.

Object File



- **Creazione/Apertura di un file**

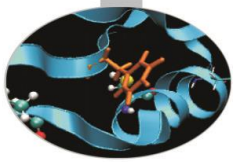
La funzione *file* permette di creare/aprire un *file object* associato al file *filename.*

file (*filename* , [*mode* ,] [*buffering*]) → *file object*

La funzione built-in *open* crea/apre un file *filename* su disco e restituisce un *file object* (come *file*)

open(*filename*[, *mode* [, *buffering*]]) → *file object*

Il *file object* dispone di metodi per la lettura/scrittura e attributi contenenti informazioni sul file corrente.



Object File

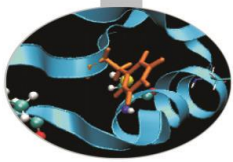
- *filename* specifica il nome del file da aprire: può essere indicato il *path relativo* o il *path assoluto*.
- *mode* indica la modalità di apertura del file*:* '*a*', '*w*', '*r*'.

  Se il file è binario è necessario aggiungere '*b*' al *mode* di apertura: '*rb*', '*wb*'.

  Per permettere contemporaneamente operazioni di lettura/scrittura è necessario aggiungere '+' al *mode* di apertura.

  Solo il parametro *filename* è obbligatorio: se *mode* non viene specificato si assume di default *mode* ='*r*'.
- *buffering* specifica la dimensione desiderata del buffer. Se negativa o omessa viene utilizzata quella di default del sistema.

Metodi Object File



- ***Lettura:***

Per poter usufruire di questi metodi il file deve essere aperto in modalità '*r*'. Al procedere della lettura dei dati la posizione all'interno del file viene incrementata.

- *read* ([*size*]) → *string*

  Lettura di al massimo *size* byte. Se *size* è negativo la lettura prosegue fino a EOF.

- *readline* ([*size*]) → *string*

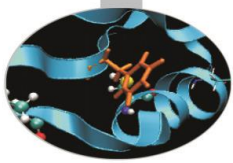
  Lettura della riga successiva di massimo *size* byte.

- *readlines* () → *list of string*

  Lettura delle successive righe.

NOTA Se il file non esiste viene lanciata l'eccezione IOError.

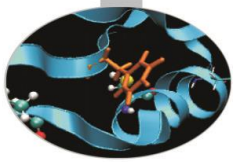
Metodi Object File



Esempio *file_read.py*

```
try:
    f=open('prova_lettura.txt','r')
    l=f.readlines()
    rows=len(l)
    time_istant=l[0:rows:4]
    coord=l[1:rows:4]
    v=l[2:rows:4]
    a=l[3:rows:4]
    print "Istanti di tempo = ", time_istant
    print "Coordinate = ", coord
    print "Velocita' = ",v
    print "Accelerazione = ",a
except IOError,e:
    print "Impossibile aprire file:\n ",e
finally:
    f.close()
```

Metodi Object File



- ***Scrittura:***

Per poter usufruire di questi metodi il file deve essere aperto in modalità '*w*' o '*a*'. Se il file è aperto in modalità *writing* la posizione corrente è all'inizio del file, se aperto in modalità *append* la posizione corrente è EOF.

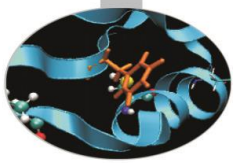
- *flush* () flush del buffer interno di I/O

- *write* (*string*)

- *writelines* (*list*)

- *truncate* (*size*) troncamento del file alla posizione corrente (se *size* non è specificato) o troncamento del file in *size.*

Metodi Object File

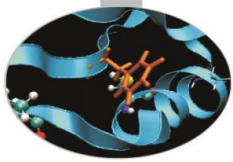


- ***Posizione:***

Le funzioni di lettura/scrittura alterano la posizione corrente all'interno del file. L'*object-file* dispone di metodi per interrogare il file sulla posizione corrente e modificarla.

- *seek* (*offset* , [*whence*]) permette di spostarsi all'interno di un file. Il primo parametro *offset* indica il numero di byte, il secondo parametro *whence* specifica come spostarsi all'interno del file. *whence* può valere:
  - *0* (default): spostamento di *offset* byte dall'inizio del file
  - *1:* spostamento dalla posizione corrente di *offset* byte
  - *2:* spostamento di *offset* byte a partire da EOF.
- *tell* ()→ *integer* opera un'interrogazione sulla posizione corrente.

Metodi Object File



- ***Chiusura***

Un file aperto deve essere chiuso con la funzione:

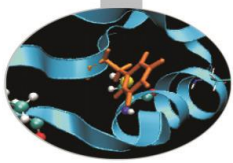
- *close*()

Dopo aver chiuso un file, qualsiasi tentativo di agire sul *file object* a cui era associato genera *IOError*.

- ***Attributi:***

- *closed:* valore *bool* che indica se il file è aperto o chiuso
- *name:* nome del file
- *mode:* modalità di utilizzo del file.

Metodi Object File

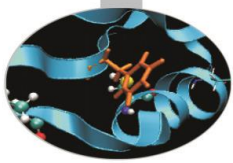


```
try:                    # Esempio file_methods.py
   f=open('file_prova.txt','w+')
   for i in xrange(20):
          f.write("Riga "+str(i)+'\n')
   f.flush()
   f.seek(0)
   print f.read()
   f.seek(0)
   f.truncate(10)
   l=f.readlines()
   print l
   print  "Riga 100", l[100]
except IOError:
          print "Impossibile trovare il file specificato !"
except IndexError,e:
          print "Errore out of range: " ,e
finally:
    f.close()
```

Errore out of range!!

Viene eseguito finally

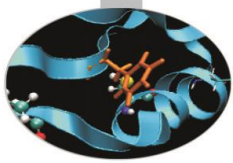
Metodi Object File



Esempio test_lettura.py (efficienza nella lettura di un file di dati)

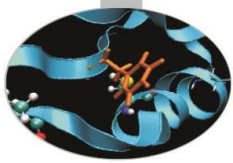
```
import timeit
def read_m1(name):
    f=open(name)
    while 1:
        lines=f.readlines(10000)
        if not lines:
            break
        for line in lines:
            pass
    f.close()
```

Metodi Object File



```
def read_m2(name):
    f=open(name)
    while 1:
        line=f.readline()
        if not line:
            break
        pass
    f.close()
```

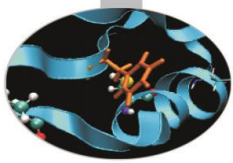
Metodi Object File



```
def read_m3(name):
        f=open(name,’r’)
        l=f.read()
        for i in l:
             pass


def riempie(nomef,kb):
   f=open(nomef,'w')
   for r in xrange(kb*1024/8):
       f.write(format(r,'08d'))
   f.close()
```

Metodi Object File

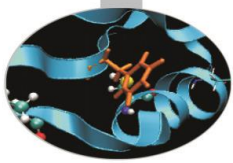


```
size=[1,5,10]
name=[]

for i in size:
   n=format(i,"03d")+"mb.test"
   name.append(n)
   riempie(n,i*1024)

out=open('time_read.txt','a')
```

Metodi Object File

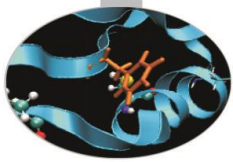


```
for i in name:
    t=timeit.Timer('read_m1(i)', \
        'from __main__ import read_m1,name,i')
    z=t.timeit(100)
    out.write('Tempo di lettura m1 '+ str(i)+': '+str(z)+'\n')

    t2=timeit.Timer('read_m2(i)', \
        'from __main__ import read_m2,name,i')
    z2=t2.timeit(100)
    out.write('Tempo di lettura m2 '+ str(i)+': '+str(z2)+'\n')

    t3=timeit.Timer('read_m3(i)', \
        'from __main__ import read_m3,name,i')
    z3=t3.timeit(100)
    out.write('Tempo di lettura m3 '+ str(i)+': '+str(z3)+'\n')
out.close()
```

Metodi Object File



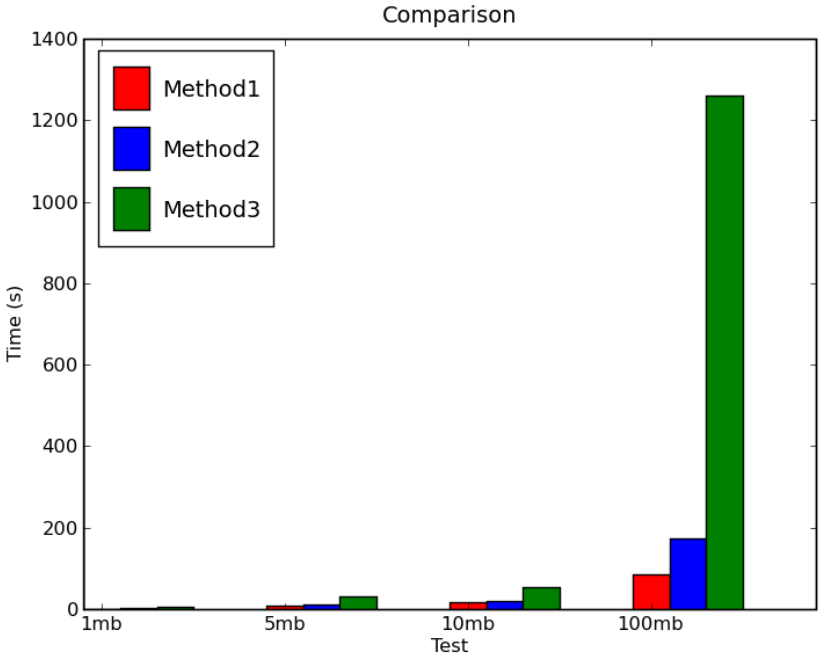
OUTPUT

Tempo di lettura m1 1mb.test: 1.41886830661
Tempo di lettura m2 1mb.test: 1.75900309599
Tempo di lettura m3 1mb.test: 5.46952516591

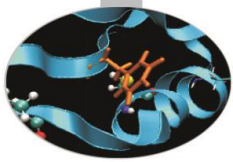
Tempo di lettura m1 5mb.test: 7.76999695747
Tempo di lettura m2 5mb.test: 10.1169012491
Tempo di lettura m3 5mb.test: 32.3493717795

Tempo di lettura m1 10mb.test: 16.0330113541
Tempo di lettura m2 10mb.test: 20.6929195701
Tempo di lettura m3 10mb.test: 53.6444475623

Tempo di lettura m1 100mb.test: 86.4516636965
Tempo di lettura m2 100mb.test: 174.918247282
Tempo di lettura m3 100mb.test: 1259.1668478



File Handling



Le operazioni di processamento di file e directory possono facilmente essere eseguite tramite Python.

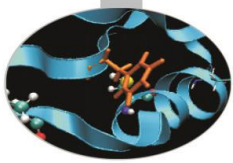
- **Modulo *os.path***

Il modulo *os.path* dispone delle più comuni funzionalità per la manipolazione del *pathname.* Tra cui le principali:

Informazioni sul *pathname*:

- *abspath*(*string*) → *string*
- *basename*(*string*) →*string*
- *dirname* (*string*) →*string*

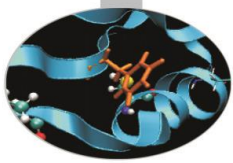
File Handling



Esempi:

```
>>>os.path.abspath('Python')
  'C:\\Programs\\Python'
>>>os.path.basename('C:\Programs\Python\' )
  'Python'
>>>os.path.dirname('C:\Programs\Python\')
  'C:\\Programs'
```

File Handling



Interrogazioni *pathname*:

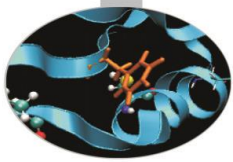
Per verificare se un particolare path fa riferimento al nome di un file, ad un link o ad una directory.

- *isdir* (*string*) → *bool*
- *isfile* (*string*) →*bool*
- *islink* (*string*) →*bool*
- *exist* (*string*) → *bool*

Esempio:

```
>>>os.path.isdir('C:\Program\Python\ciao.txt')
False
```

File Handling



Manipolazione del pathname:

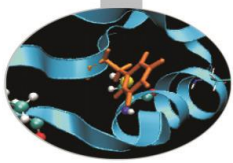
- *normcase* (*string*) → *string*
- *split* (*string*) → *(head,tail)*
- *join* (*a, *p*) → *string*

Esempio:

>>>os.path.join('C','Programs','ciao.txt')

'C\\Programs\\ciao.txt'

File Handling



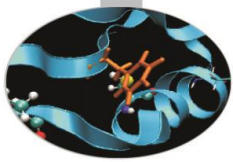
Informazioni sui file:

- *getatime* (*filename*) → *int*
- *getctime* (*filename*) → *int*
- *getmtime* (*filename*) →*int*
- *getsize* (*filename*) → *int*

Esempio:

```
>>> k=os.path.getatime('C:\Programs\tempi_range.txt')
>>> time.gmtime(k)
(2009, 4, 3, 10, 17, 40, 4, 93, 0)
```

Modulo os



Modulo os

Il modulo *os* contiene molte funzioni per la manipolazione di file, directory, processi. L'utilizzo di questo modulo permette lo sviluppo di applicazioni multipiattaforma garantendo la portabilità del codice.

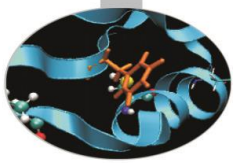
Muoversi tra file e directory

La funzione os.*getcwd( )* ritorna la directory di lavoro corrente.

La funzione os.*chdir* (*path*) cambia la directory di lavoro corrente nella directory *path.*

Per elencare tutti i file presenti in una directory viene utilizzata la funzione *os.listdir* (*path*)

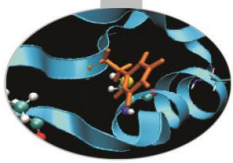
File Handling



Esempio vista_cartella.py

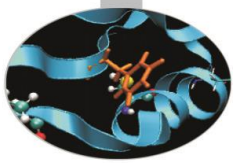
```
from os import *
from os.path import *
def pprint(lista):
    for el in lista: print el
def view_dir(name):
    nabs=abspath(name)
    chdir(nabs)
    l=listdir(getcwd())
    ffile=[]
    flink=[]
    fdir=[]
    for i in l:
        if os.path.isfile(i):
            ffile.append(i)
```

File Handling



```
    if os.path.islink(i):
        flink.append(i)
    elif os.path.isdir(i):
        fdir.append(i)
        view_dir(i)                #chiamata ricorsiva
        os.chdir(nabs)
    else:
        pass
    print '\nLista dei file della cartella: ', name,':'
    pprint(ffile)
    print '\nLista dei link della cartella: ', name,':'
    pprint(flink)
    print '\nLista delle dir della cartella: ', name,':'
    pprint(fdir)
s=raw_input('Inserire nome path: ')
view_dir(s)
```

File Management: Modulo os

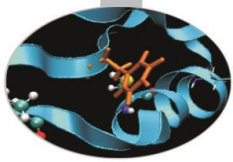


Esempio esplora_cartelle.py

```
def visit(arg,dirname,names):
    os.chdir(dirname)
    files=filter(os.path.isfile,names)
    dire=filter(os.path.isdir,names)
    link=filter(os.path.islink,names)
    print 'File in ',dirname,': ',files
    print 'Dire in ',dirname,': ',dire
    print 'Link in ',dirname,': ',link
s=raw_input('Inserire nome path: ')
os.path.walk(s,visit, ())
```

La funzione *os.path.walk* (*path*, *visit*, *arg*) percorre l'albero delle directory con radice path e per ogni directory chiama la funzione *visit* con argomenti (*arg*, *dirname*, *names*).

File Management : Modulo os



Modifica di file e directory

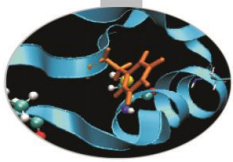
La funzione *os.mkdir* (*path* [, *mode=0777*]) crea una nuova cartella di lavoro

La funzione *os. makedirs*(*path* [, *mode=0777*]) crea le cartelle in maniera ricorsiva.

Esempi:

```
>>>os.listdir(os.getcwd())      # contenuto della cartella corrente
[]
>>>os.makedirs(os.path.join(os.getcwd(),os.path.join('nuova','cartella')))
>>>os.mkdir(os.path.join(os.getcwd(),'nuova2'))
>>>for el in os.listdir (os.getcwd()):
        print os.path.join(os.getcwd(),el)        #os.path.abspath(el)
        os.listdir(os.path.join(os.getcwd(),el))
```

File Management : Modulo os



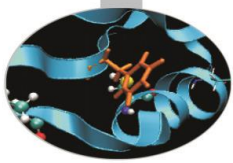
Per eliminare una directory si utilizza la funzione os.*rmdir(path)*, per procedere in maniera ricorsiva si utilizza la funzione *os.removedirs(path).*

La funzione *os.remove* (*path*) permette invece la cancellazione di un file.

Esempio: eliminare in maniera ricorsiva il contenuto di una cartella

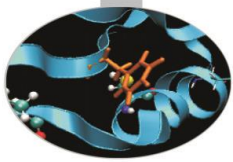
```
def delete(arg,dirname,names):
    os.chdir(dirname)
    files=filter(os.path.isfile,names)
    dire=filter(os.path.isdir,names)
```

File Management : Modulo os



```
for f in files:
        remove(f)
for d in dire:
        try:
           removedirs(d)
        except WindowsError:
           os.path.walk(os.path.join(dirname,d),delete,arg)
try:
      if len(filter(os.path.isfile,os.listdir(arg)))==0:
            os.chdir(arg)
            d=os.listdir(arg)
            for i in d: rmdir(i)
except:
        pass
l=raw_input('Inserisci path: ')
os.chdir(l)
os.path.walk(os.getcwd(),delete,l)
```

File Management



La funzione *rename* (*old,new*) permette di rinominare file e/o cartelle.

La funzione *renames* (*old,new*) si comporta allo stesso modo in maniera ricorsiva.

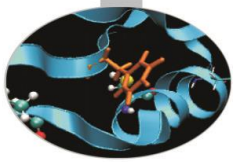
La funzione *os.stat*(*path*) restituisce una tupla con le informazioni inerenti al *path* (dimensione, ultimo accesso, ultima modifica, permessi).

Per cambiare i permessi di accessi ad un a file o ad un cartella il modulo *os* dispone della funzione *os.chmod* (*path*, *mode*).

Esempi:

```
# write-read-exe for user, read for group and other
>>>os.chmod('prova.txt',0744)
# write only for user
>>>os.chmod('prova.txt',0400)
```

Modulo Shutil



In aggiunta al modulo *os*, il modulo *shutil* offre operazioni di alto livello per la manipolazione di files. In particolare sono supportate funzioni per la copia e la rimozione di files.

Copia di file

shutil.copy (*src, dst*) copia il contenuto di *src* in *dst* (file e directory)

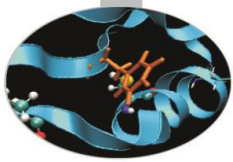
shutil.copyfile (*src*, *dst*) copia il contenuto di *src* in *dst*

shutil.copymode (*src, dst*) copia dei permessi da *src* a *dst*

shutil.copystat (*src*, *dst*) copia dei permessi e dei tempi di ultimo accesso/modifica da *src* a *dst*.

shutil.copytree (*src*, *dst* [,*symlinks*]) copia ricorsivamente un intero albero avente radice *src*.

Modulo Shutil

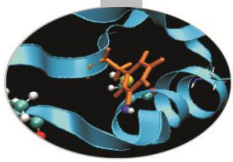


Rimozione di file

shutil.rmtree(*path*[, *ignore_errors*[, *onerror*]]) cancella un intero albero di directory.

shutil.move (*src*, *dst*) sposta ricorsivamente un file o una cartella.

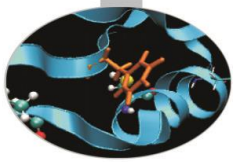
Modulo Shutil



Esempio:

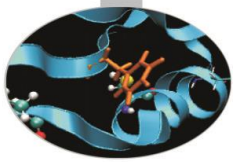
```
from os import * ; from shutil import *
import time , stat
def file_info(name):
    info=os.stat(name)
    print 'fMode ', format(info.st_mode,"o")
    print 'fCreated ',time.ctime(info.st_ctime)
    print 'fAccess ', time.ctime(info.st_atime)
    print 'fModify ', time.ctime(info.st_mtime)
```

Modulo Shutil



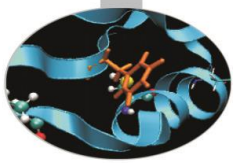
```
def read(name):
    f=file(name)
    while 1:
        l=f.readline()
        if not l: break
    print l
    f.close()
```

Modulo Shutil



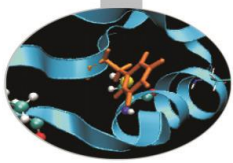
```
file1=raw_input('Inserisci nome file: ')
file2=raw_input('Inserisci nome file2: ')
print '::::::::::::::STAT BEFORE::::::::::::::'
file_info(file1)
print '::::::::::::::read before::::::::::::::'
read(file1)
print '::::::::::::::Copy\n::::::::::::::'
shutil.copy(file2,file1)
print '::::::::::::::STAT AFTER:::::::::::::: '
file_info(file1)
```

Modulo Shutil



```
#OUTPUT
C:\Documents and Settings\user\Desktop\Python_corso\New
::::::::::::::stat before::::::::::::::
fMode  33206
fCreated  Wed Apr 15 12:02:48 2009
fAccess  Wed Apr 15 16:22:11 2009
fModify  Thu Jan 15 11:24:01 2009
:::::::::::::: read before ::::::::::::::
new new
:::::::::::::: Copy ::::::::::::::
:::::::::::::: AFTER: ::::::::::::::
fMode  33206
fCreated  Wed Apr 15 12:02:48 2009
fAccess  Wed Apr 15 16:23:32 2009
fModify  Wed Apr 15 15:32:07 2009
```

Modulo Glob



Python mette a disposizione il modulo *glob* per la manipolazione del path name. Il modulo *glob* opera utilizzando le regole di espansione della UNIX shell : * , ? , [seq].

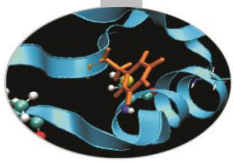
- *glob.glob*(*pathname*) → *list*

  La funzione *glob* ritorna la lista dei path che coincidono con la stringa pathname.

- *glob.iglob*(*pathname*) → *iterator*

  La funzione *iglob* ha la stessa funzionalità della funzione *glob(),* ma ritorna un iteratore anziché una lista di valori.

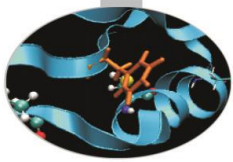
Modulo Glob



Esempio:

```
>>> import os
>>> import glob
>>>os.listdir('.')
['1file.txt', '2.file.log']
>>> glob.glob('*file.*')
['1file.txt','2.file.log']
>>>glob.glob('[0-1]file.*')
['1file.txt']
>>>glob.glob('?file.txt')
['1file.txt']
>>> it = glob.iglob('*file.*')
```

Modulo Glob



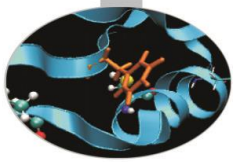
Esempio

```
>>>try:
        while True: print it.next()
    except  StopIteration: pass
```

1file.txt

2.file.log

Modulo Glob



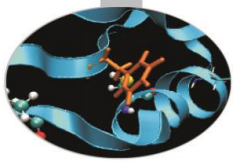
- *glob.fnmatch.fnmatch(filename,pattern) → bool*

  La funzione fnmatch verifica che il *filename* corrisponda a *pattern.* Il confronto è case-insensitive. Per un confronto *case-sensitive* può essere utilizzata la funzione *glob.fnmatch.fnmatchcase*

- *glob.fnmatch.filter (names,pattern) → list*

  La funzione ritorna un sottoinsieme della lista di *names* che soddisfano *pattern*

Modulo Glob



Esempio:

```
l=[]
for ffile in os.listdir(‘.’):
    if glob.fnmatch.fnmatch(ffile, ‘*.txt’):
                    l.append(ffile)
#OPPURE
l=glob.fnmatch.filter(os.listdir(‘.’),’*.txt’)
```