Corso di **Python**

Lezione 10

Gestire file e directory

Editor: Davide Brunato
Scuola Internazionale Superiore di Studi Avanzati di Trieste

Premessa per le lezioni sulle librerie ...

- Singole classi e funzioni verranno sempre indicate con il prefisso della libreria a cui appartengono
- I nomi di funzione o classe verranno indicati con parantesi finali `'()'`
- Non sempre saranno illustrate tutte le API disponibili
- Verranno illustrate solo le classi direttamente utilizzabili, evitando quelle di base che sono progettate per costruire oggetti alternativi
- Nelle slide saranno inserite delle URL alle pagine web della documentazione ufficiale
- Meglio seguire le slide con una console Python 3 (meglio se 3.4+ ...), che sarà presa come riferimento per gli esempi
- È utile ricordare di cosa si occupa una singola libreria, non serve memorizzare i singoli nomi delle funzioni:
  - La console è vostra amica:

    ```
    >>> import os
    >>> help(os)
    ```

  - **ipython** e alcuni IDE hanno il completamento automatico

La libreria sys

- Permette di interagire con il runtime del processo mediante attributi e funzioni specifiche
- `sys.argv` : contiene gli argomenti passati al comando:

```
[brunato@localhost ~]$ vi catargv.py
#!/usr/bin/env python
import sys
print(sys.argv)

[brunato@localhost ~]$ python catargv.py --delete --all pippo
['catargv.py', '--delete', '--all', 'pippo']
[brunato@localhost ~]$ ./catargv.py -a .bash_* -p
['./tmp/catargv.py', '-a', '.bash_history', '.bash_logout', '.bash_profile', '-p']
```

- `sys.path` : lista di path passati al processo per il lookup di moduli Python
- `sys.stdin`, `sys.stdout`, `sys.stderr` : stream I/O di default
- `sys.byteorder` : 'big' se la base è *big-endian*, 'little' se è *little-endian*
- `sys.maxint` : delimita il massimo valore per il tipo *int* (sys.maxint - 1)
- `sys.version_info` : ritorna *tupla nominale* contentente le informazioni di versione di Python
- `sys.hexversion` : numero di versione codificato come intero, cresce con ogni nuova versione
- `sys.exit([status])` : funzione di uscita dall'interprete Python, *status* è un intero che indica lo stato di uscita, zero indica una “terminazione con successo”, diverso da zero indica una “terminazione anormale”

Librerie per I/O su file

- **`os`**: interfacce varie per il sistema operativo
- **`os.path`**: gestione dei nomi e dei path
- **`fileinput`**: iterare sulle linee di un file di testo
- **`stat`**: costanti per interpretare lo stato di un file
- **`filecmp`**: comparazione di file e directory
- **`tempfile`**: creazione di file e directory temporanei
- **`glob`**: iterare su alberi di directory
- **`fnmatch`**: pattern matching per filename Unix
- **`linecache`**: cache per accesso random a linee di testo
- **`shutil`:** operazioni di alto livello sui file
- ***`pathlib`****: path di filesystem in modalità object-oriented (Python 3.4+)*
- ***`io`****: strumenti per I/O streaming*

La funzione built-in `open`

- Per aprire un file c'è una funzione built-in apposita:

  `open``(`*`name[, mode[, buffering]]`*`)`

  - *name* : è il nome completo del file da aprire
  - *mode* : stringa che definisce la modalità di accesso
  - *buffering* : ampiezza del buffer (<0 = default, 0 = *unbuffered*)

- La funzione open ritorna un oggetto file, dove poi si possono effettuare le scritture/letture
- Esempio :

```
>>> file_object = open('/tmp/example1', 'wt')
>>> file_object.write('Scrittura di prova su file\n')
>>> file_object.write('Seconda linea per test.\n')
>>> file_object.close()
```

Modalità di accesso

Carattere	Significato
`'r'`	Apre il file in lettura (default)
`'w'`	Apre il file in scrittura, troncandolo se già esiste
`'x'`	Apre il file per la creazione esclusiva, fallendo se già esiste
`'a'`	Apre il file per la scrittura, aggiungendo se già esiste
`'b'`	Modalità binaria
`'t'`	Modalità testo (default)
`'+'`	Apre per l'aggiornamento (lettura o scrittura)
`'U'`	Modalità testo (deprecata) per interpretare i newline in modo universale

- Il modo di default è `'rt'` (lettura di file di testo)
- Esempi:
  - `'w+b'`: Apre file binario per l'aggiornamento, troncandolo se già esiste
  - `'r+b'`: Apre file file binario per l'aggiornamento, senza troncarlo

Funzione **`open`** in Python 3

- In Python 3 cono stati aggiunti altri parametri:

```
open(file, mode='r', buffering=-1, encoding=None,
     errors=None, newline=None, closefd=True, opener=None)
```

  - ***`encoding`***: codifica da utilizzare in text mode
    - Il default: `locale.getpreferredencoding()`
    - Per le codifiche supportate vedere il modulo `codecs`
  - ***`errors`***: modalità di gestione degli errori in fase di codifica e decodifica
  - ***`newline`***: sostituisce il modo 'U' nell'indicazione del tipo di gestione dei newline (può essere: `None, '', '\n', '\r', '\r\n'`)
  - ***`closefd`***: se il file descriptor deve essere eliminato con la chiusura del file
  - ***`opener`***: permette di far gestire l'apertura del file ad altra funzione

La libreria **os**

- Definisce interfacce di base con il sistema operativo
- Le chiamate possono essere raggruppate in diverse parti:
  - Informazioni dal sistema
  - Gestione ambiente del processo in esecuzione
  - Gestione file e directory a livello di sistema
  - Gestione file con file descriptor
  - Gestione dei processi

`os` – Info dal sistema

- Informazioni dipendenti dal sistema operativo:
  - **`os.name`**: nome del modulo di dipendenza caricato con il sistema operativo ('posix', 'nt', 'os2', 'ce', 'java', 'riscos')
  - **`os.uname()`** : ritorna tupla di 5 elementi con i dati riassuntivi del sistema (sistemi Unix)
  - **`os.getloadavg()`** : ritorna tupla con 3 valori di carico del sistema (sistemi Unix)
  - **`os.curdir`** : stringa usata per directory corrente ('`.`')
  - **`os.pardir`** : stringa usata per directory padre ('`..`')
  - **`os.sep`** : separatore nei path ('`/`' per POSIX, '`\\`' per Windows)
  - **`os.linesep`** : separatore di linee ('`\n`' per POSIX, '`\r\n`' per Windows)
  - **`os.urandom`**(*n*) : ritorna una stringa di n byte random, adatta per usi crittografici

`os` - Ambiente del processo

- Funzioni disponibili su diverse piattaforme (almeno Unix e Windows):
  - **`os.environ`** : dizionario contenente le stringhe di environment del sistema
  - **`os.getenv`**(*`varname[, value]`*) : ritorna variabile o valore se non è definita
  - **`os.getpid`**() : ritorna il PID del processo
  - **`os.getcwd`**() : ritorna stringa che rappresenta la directory di lavoro corrente del processo
  - **`os.chdir`**(*`path`*) : cambia la directory corrente del processo
  - **`os.strerror`**(*`code`*) : ritorna il messaggio di errore corrispondente al codice
- Funzioni disponibili solo sotto piattaforme Unix ('posix'):
  - **`os.geteuid`**() : ritorna l'*effective* user id del processo
  - **`os.getresuid`**() : ritorna tupla con *real*, *effective*, *saved* user id del processo
  - **`os.setresuid`**(*`ruid, euid, suid`*) : imposta i *real*, *effective*, *saved* user id del processo

`os` – Gestione file/directory

- Molte funzioni per gestire i file e le directory:
  - `os.link(source, link_name)`: crea un hard link
  - `os.symlink(source, link_name)`: crea un link simbolico
  - `os.listdir(path)`: lista la directory individuata dal path
  - `os.stat(path)` : ritorna le informazioni di accesso e modifica del file individuato dal path
  - `os.remove(path)` : cancella il file individuato dal path
  - `os.removedirs(path)` : rimuove ricorsivamente directory vuote
  - `os.chmod(path, mode)` : cambia permessi di accesso ad un file
  - `os.chown(path, uid, gid)` : cambia uid e gid ad un file
  - `os.chroot(path)` : cambia la directory base del processo
  - `os.walk(top)`: percorre tutto il ramo delle sottodirectory

Esempio: os.walk

```
>>> import os
>>> for files in os.walk('/var/log'):
...     print(files)
...
('/var/log', ['BackupPC', 'ppp', 'samba', 'journal', 'httpd', 'munge',
'pluto', 'speech-dispatcher', 'libvirt', 'gdm', 'anaconda', 'chrony',
'glusterfs', 'cups', 'sssd', 'audit'], ['hawkey.log', 'lastlog',
'dnf.log-20160125', 'java_install.log', 'wtmp', 'firewalld',
'dnf.librepo.log-20160105', 'dnf.log', 'Xorg.0.log', 'dnf.log-
20160201', 'dnf.log-20160105', 'hawkey.log-20160201', 'btmp-20160201',
'hawkey.log-20160119', 'dnf.rpm.log-20160119', 'grubby', 'hawkey.log-
20160125', 'dnf.rpm.log', 'hawkey.log-20160105', 'dnf.librepo.log-
20160201', 'boot.log', 'dnf.rpm.log-20160105', 'btmp',
'dnf.librepo.log', 'dnf.log-20160119', 'Xorg.0.log.old', 'tallylog',
'yum.log', 'dnf.librepo.log-20160119', 'README', 'rsnapshot',
'wpa_supplicant.log', 'dnf.rpm.log-20160125', 'dnf.rpm.log-20160201',
'dnf.librepo.log-20160125'])
('/var/log/BackupPC', [], [])
('/var/log/journal', ['585e1161dee44123bede268ea48f4b18'], [])
......
```

`os` – Gestione con *file descriptor*

- `os.`**`open`**`(`*`file`*`,`*`flags`*`[,`*`mode`*`])`: apre un file, ritornando un file descriptor
- `os.`**`read`**`(`*`fd`*`,`*`n`*`)`: legge fino a n byte dal file, ritorna una stringa con numero di byte letti.
- `os.`**`write`**`(`*`fd`*`,`*`str`*`)`: scrive una string nel file, ritorna il numero di byte scritti.
- `os.`**`close`**`(`*`fd`*`)`: chiude file descriptor
- `os.`**`pipe`**`()`: Apre una pipe, ritorna coppia di file descriptor (r, w)
- `os.`**`fsync`**`(`*`fd`*`)`: sync del file su disco
- `os.`**`dup`**`(`*`fd`*`)`: duplica file descriptor (il file rimane lo stesso)
- `os.`**`fstat`**`(`*`fd`*`)`: ritorna stato del file (come *os.stat()*)
- `os.`**`fchmod`**`(`*`fd, mode`*`)`: imposta il modo di accesso per il file
- `os.`**`fchown`**`(`*`fd, uid, gid`*`)`: cambia il proprietario e il gruppo del file
- `os.`**`isatty`**`(`*`fd`*`)`: Ritorna `True` se il file è una device di tipo *terminale*, `False` altrimenti
- `os.`**`ttyname`**`(`*`fd`*`)`: ritorna stringa identificativa del terminale

`os` – Gestione processi

- Ci sono funzioni per la gestione di processi:
  - `os.fork()`

    Fa un fork generando un processo figlio, ritornandone il PID
  - `os.kill(pid, sig)`

    Kill di un processo con specifico segnale
  - `os.system(command)`

    Esegue un comando di sistema e ritorna il codice di uscita
  - `os.popen(command[, mode[, bufsize]])`:

    Apre una pipe con il comando. Viene ritornato l'oggetto file corrispondente alla pipe aperta. Questa funzione è deprecata a partire da Python 2.6, a favore della funzione `subprocess.call`, ma si può trovare ancora in molti codici.

- Per la gestione dei processi da Python 2.4 c'è la libreria **`subprocess`**

`os.path` – Funzioni per i pathname Posix

- `os.path.`**`abspath`**`(`*`path`*`)`: Ritorna il path assoluto
- `os.path.`**`basename`**`(`*`path`*`)`: Ritorna la parte base del path
- `os.path.`**`dirname`**`(`*`path`*`)`: Ritorna la parte directory del path
- `os.path.`**`split`**`(`*`path`*`)`: ritorna coppia con *dirname* e *basename*
- `os.path.`**`commonprefix`**`(`*`list`*`)`: Ritorna il prefisso comune per lista di path
- `os.path.`**`exists`**`(`*`path`*`)`: ritorna True se il path esiste
- `os.path.`**`isfile`**`(`*`path`*`)`: ritorna True se il path si riferisce ad un file regolare
- `os.path.`**`isdir`**`(`*`path`*`)`: ritorna True se il path si riferisce ad una directory
- `os.path.`**`getsize`**`(`*`path`*`)`: ritorna l'occupazione in byte del file relativo al path
- `os.path.`**`getmtime`**`(`*`path`*`)`: ritorna un *epoch time* relativo all'ultima modifica
- `os.path.`**`join`**`(`*`path, *paths`*`)`: Riassembla i path in maniera opportuna
- Altre API: https://docs.python.org/3/library/os.path.html

La libreria **fileinput**

- Spesso è necessario leggere un file di testo linea per linea
- La libreria fileinput fornisce un'interfaccia adatta per questi casi
- La chiamata base è la funzione `input()`:

```
fileinput.input(files=None, inplace=False,
    backup='', bufsize=0, mode='r',
    openhook=None)
```

  - Percorre linea per linea una sequenza di file
  - Se non si specifica nessuna sequenza allora assume come ingresso gli argomenti dello script o lo standard input
  - Gli argomenti *inplace* e *backup* servono per salvare il vecchio file e sostituirlo con il risultato dell'elaborazione
  - *openhook* serve per specificare una funzione alternativa per l'apertura dei file

La libreria **fileinput**

- Spesso è necessario leggere un file di testo linea per linea
- La libreria fileinput fornisce un'interfaccia adatta per questi casi
- La chiamata base è la funzione `input()`:

```
fileinput.input(files=None, inplace=False,
    backup='', bufsize=0, mode='r',
    openhook=None)
```

  - Percorre linea per linea una sequenza di file
  - Se non si specifica nessuna sequenza allora assume come ingresso gli argomenti dello script o lo standard input
  - Gli argomenti *inplace* e *backup* servono per salvare il vecchio file e sostituirlo con il risultato dell'elaborazione
  - *openhook* serve per specificare una funzione alternativa per l'apertura dei file

Esempi con **fileinput**

- Con input da file singolo:

```
import fileinput
for line in fileinput.input('spam.txt'):
    print(line)
```

- Con input dagli argomenti (*sys.argv*):

```
for line in fileinput.input():
    process(line)
```

- Con elenco e uso dei contesti:

```
myfiles = ('spam.txt', 'eggs.txt')
with fileinput.input(files=myfiles) as f:
    for line in f:
        process(line)
```

API di **fileinput**

- La libreria comprende funzioni per controllare la lettura dei file:
  - `fileinput.`**`filename`**`()`: ritorna il nome del file attivo
  - `fileinput.`**`fileno`**`()`: ritorna il descrittore del file aperto, -1 se non ce nessun file aperto
  - `fileinput.`**`lineno`**`()`: ritorna il contatore cumulativo
  - `fileinput.`**`filelineno`**`()`: ritorna il contatore di linee del file
  - `fileinput.`**`isfirstline`**`()`: ritorna True se sta leggendo la prima linea del file
  - `fileinput.`**`isstdin`**`()`: True se la linea è letta da sys.stdin
  - `fileinput.`**`nextfile`**`()`: chiude il file corrente con l'iterazione in corso
  - `fileinput.`**`close`**`()`: chiude l'intera sequenza di file

La libreria `stat`

- Questa libreria serve per interpretare i risultati delle funzioni *os.stat*, *os.fstat* e *os.lstat*:

```
import os, stat
mode = os.stat(pathname).st_mode
```

- È composta solo da funzioni di test o costanti per la verifica dello stato ritornato
- Dalla versione 3.4 la libreria è implementata con parte retrostante in C
- https://docs.python.org/3/library/stat.html

Esempi d'uso di `stat`

```
>>> import os, stat
>>> fstat = os.stat('testfile')
>>> fstat
posix.stat_result(st_mode=33188, st_ino=262148, st_dev=64770, st_nlink=1,
st_uid=1419, st_gid=800, st_size=0, st_atime=1454388405, st_mtime=1454388405,
st_ctime=1454388405)
>>> dict(fstat)
>>> dir(fstat)
['__add__', '__class__', ... , 'n_fields', 'n_sequence_fields',
'n_unnamed_fields', 'st_atime', 'st_blksize', 'st_blocks', 'st_ctime',
'st_dev', 'st_gid', 'st_ino', 'st_mode', 'st_mtime', 'st_nlink', 'st_rdev',
'st_size', 'st_uid']
>>> fstat[stat.ST_MTIME]
1454388405
>>> fstat.st_mtime
1454388405.8790548
>>>
>>> stat.S_ISREG(fstat.st_mode)
True
```

La libreria **filecmp**

- Serve a comparare file e directory:

  `filecmp.`**`cmp`**`(`*`f1`*`, `*`f2`*`, `*`shallow=True`*`)`

  - Compara due file, tornando *`True`* se i file sono identici
  - Con *`shallow=True`* l'eguaglianza dei due stat può essere ritenuta sufficiente

  `filecmp.`**`cmpfiles`**`(`*`dir1`*`, `*`dir2`*`, `*`common`*`, `*`shallow=True`*`)`

  - Compara i files in due directory i cui nomi sono determinati dalla lista *common*. La comparazione avviene per coppie corrispondenti.

  `filecmp.`**`dircmp`**`(`*`a`*`, `*`b`*`, `*`ignore=None`*`, `*`hide=None`*`)`

  - Classe che serve a costruire un comparatore tra due directory. I parametri opzionali sono liste che servono per specificare liste di nomi da ignorare (default v3.4+: `filecmp.DEFAULT_IGNORES`) e da nascondere (in genere '`.`' e '`..`').

- La comparazione usa una cache per memorizzare i confronti
- Per ripulire la cache c'è un'apposita chiamata:
  - `filecmp.`**`clear_cache`**`()`

Esempi d'uso di **filecmp**

```
>>> os.system('cp -p testfile testfile2')
0
>>> filecmp.cmp('testfile', 'testfile2')
True

from filecmp import dircmp
def print_diff_files(dcmp):
    for name in dcmp.diff_files:
        print("diff_file %s found in %s and %s" % (name,
              dcmp.left, dcmp.right))
    for sub_dcmp in dcmp.subdirs.values():
        print_diff_files(sub_dcmp)

>>> dcmp = dircmp('dir1', 'dir2')
>>> print_diff_files(dcmp)
```

La libreria **tempfile**

- Permette di creare file temporanei:

`tempfile.`**`TemporaryFile`**`(`*`mode='w+b', buffering=None, encoding=None, newline=None, suffix='', prefix='tmp', dir=None`*`)`

  - Crea un file temporaneo. Ritorna un oggetto file. Il file viene eliminato quando il file viene chiuso o l'oggetto viene rimosso dalla memoria.

`tempfile.`**`NamedTemporaryFile`**`(...)`

  - Identica alla precedente solo che il file creato sarà visibile a livello di file system con un nome, reperibile attraverso l'attributo *name* dell'oggetto file ritornato.

`tempfile.`**`SpooledTemporaryFile`**`(`*`max_size=0, ...`*`)`

  - Identica alle precedenti ad esclusione che il file temporaneo viene tenuto in memoria per quanto è possibile. Il passaggio su disco è definitivo e avviene se il file supera la dimensione indicata da max_size o viene fatta una chiamata fileno(). C'è anche un metodo apposito rollover() aggiunto allo scopo per questo tipo di file.

`tempfile.`**`gettempdir`**`()`: ritorna il path della directory temporanea

`tempfile.`**`gettempprefix`**`()`: ritorna il prefisso usato per creare i file temporanei

Esempio d'uso di **tempfile**

```
>>> import tempfile
>>> tempfile.gettempprefix()
'tmp'
>>> tempfile.gettempdir()
'/tmp'
>>> filetemp = tempfile.NamedTemporaryFile()
>>> filetemp.name
'/tmp/tmpwaiu8v0u'
>>> filetemp.write(b"Dati da scrivere")
16
>>> filetemp.seek(0)
0
>>> filetemp.read()
b'Dati da scrivere'
>>> filetemp.close()   # Il file viene automaticamente rimosso ...
```

La libreria `glob`

- La libreria glob riproduce un *listing* sui pathname che rispettano uno specifico pattern

  `glob.glob(`*`pathname, recursive=False`*`)`

  - Ritorna una lista dei path che corrispondono al pathname specificato. Il pathname può essere assoluto o relativo e può contenere i caratteri *shell wildcard* per corrispondenze multiple.
  - Il parametro *recursive* è stato introdotto in Python 3.5.

  `glob.iglob(`*`pathname, recursive=False`*`)`

  - Come glob ma ritorna un iteratore invece di una lista

  `glob.escape(`*`pathname`*`)`

  - Introdotta in Python 3.4. Effettua un escape dei caratteri speciali usati nei path ('*', '?', '[').

Esempi d'uso di **glob**

```
>>> import glob
>>> glob.escape('/file[nome1]/?/nome2/*')
'/file[[]nome1]/[?]/nome2/[*]'
>>> glob.glob('/var/log/')
['/var/log/']
>>> glob.glob('/var/log/*.log')
['/var/log/hawkey.log',
'/var/log/java_install.log',
'/var/log/dnf.log', '/var/log/Xorg.0.log',
'/var/log/dnf.rpm.log', '/var/log/boot.log',
'/var/log/dnf.librepo.log', '/var/log/yum.log',
'/var/log/wpa_supplicant.log']
```

La libreria **fnmatch**

- Questa libreria implementa un pattern matching sui nomi dei file

`fnmatch.`**`fnmatch`**`(`*`filename`*`, `*`pattern`*`)`

- Ritorna True se il nome del file corrisponde al pattern specificato, ritorna False altrimenti.

`fnmatch.`**`fnmatchcase`**`(`*`filename`*`, `*`pattern`*`)`

- Come fnmatch ma con comparazione case-sensitive

`fnmatch.`**`filter`**`(`*`names`*`, `*`pattern`*`)`

- Effettua un filtro su una sequenza di nomi, in forma più efficiente di:

```
[n for n in names if fnmatch(n, pattern)]
```

`fnmatch.`**`translate`**`(`*`pattern`*`)`

- Converte il pattern in espressione regolare ...

Esempio d'uso di **fnmatch**

```
>>> import os, fnmatch
>>> for filename in os.listdir('/var/log'):
...     if fnmatch.fnmatch(filename, '*.log'):
...         print(filename)
...
hawkey.log
java_install.log
dnf.log
Xorg.0.log
dnf.rpm.log
boot.log
dnf.librepo.log
yum.log
wpa_supplicant.log
```

La libreria **linecache**

- Implementa una cache intermedia per l'accesso di tipo random ai file

`linecache.`**`getline`**`(`*`filename, lineno, module_globals=None`*`)`

  - Ritorna la linea *lineno* dal file specificato. Ritorna stringa non vuota (almeno '\n' ...) se l'operazione ha successo, altrimenti ritorna una stringa vuota. Il file viene cercato secondo il path specificato e secondo i sys.path, o in alternativa nell'elenco specificato da *module_globals* (uno zip o altra sorgente non legata al filesystem).

`linecache.`**`clearcache`**`()`

`linecache.`**`checkcache`**`(`*`filename=None`*`)`

  - Effettua un check di validità della cache

`linecache.`**`lazycache`**`(`*`filename, module_globals`*`)`

  - Effettua un precaricamento nella cache di dati dell'oggetto da zip o altra sorgente

La libreria `shutil`

- Questa libreria offre operazioni di alto livello sui file, in particolare per quanto riguarda copia e rimozione

  `shutil.`**`copy`**`(`*`src`*`, `*`dst`*`, `*`follow_symlinks=True`*`)`
  - Copia il file sorgente sulla destinazione. Il parametro *follow_symlinks* è stato aggiunto in Python 3.

  `shutil.`**`copy2`**`(`*`src`*`, `*`dst`*`, `*`follow_symlinks=True`*`)`
  - Come copy, ma cerca di preservare tutti i metadati.

  `shutil.`**`move`**`(`*`src`*`, `*`dst`*`, `*`copy_function=copy2`*`)`
  - Muove un file usando una specifica modalità di copia

  `shutil.`**`copytree`**`(`*`src`*`, `*`dst`*`, `*`symlinks=False`*`, `*`ignore=None`*`, `*`copy_function=copy2`*`, `*`ignore_dangling_symlinks=False`*`)`
  - Copia un'intero albero di una directory su un path di destinazione. Non sovrascrive se la destinazione esiste già.

  `shutil.`**`rmtree`**`(`*`path`*`, `*`ignore_errors=False`*`, `*`onerror=None`*`)`

  `shutil.`**`disk_usage`**`(`*`path`*`)`

  `shutil.`**`chown`**`(`*`path`*`, `*`user=None`*`, `*`group=None`*`)`

  `shutil.`**`which`**`(`*`cmd`*`, `*`mode=os.F_OK | os.X_OK`*`, `*`path=None`*`)`

- Riferimento: https://docs.python.org/3./library/shutil.html

Libreria `io`

- Questa libreria mette a disposizione una serie di strumenti per lo streaming input/output:
- Libreria introdotta in Python 3 con backport anche per le versioni 2.6 e 2.7
  - In Python 2 StringIO era definita in un'omonima libreria StringIO, rimossa in Python 3

  `io.StringIO(initial_value='', newline='\n')`
  - per I/O formato testo in memoria. Il valore iniziale è una stringa. Il parametro *newline* definisce come considerare il fine linea.

  `io.BytesIO([initial_bytes])`
  - per I/O binario in memoria

- Riferimento alle API: https://docs.python.org/3/library/io.html

Esempio di StringIO

```
import io

output = io.StringIO()
output.write('First line.\n')
print('Second line.', file=output)

# Retrieve file contents -- this will be
# 'First line.\nSecond line.\n'
contents = output.getvalue()

# Close object and discard memory buffer --
# .getvalue() will now raise an exception.
output.close()
```