Gestione di File e Directory in Python

Operazioni sui file

Apertura di un file

>>> f = open(filename, mode)

restituisce un oggetto file f:

- **filename = nome del file (stringa)**
- **mode = modalità di apertura:**
  - **'r' : sola lettura (default)**
  - **'w' sola scrittura (un file già esistente verrà sovrascritto)**
  - **'a' append**
  - **'r+' lettura e scrittura (dati originali intatti)**
  - **'w+' scrittura e lettura (dati originali cancellati)**
  - **'b' in aggiunta a una modalità di lettura o scrittura, opera in modalità binaria**

Apertura e chiusura di un file

- **I diversi caratteri di \newline usati nei diversi OS possono creare problemi di portabilità**
  - **Unix '\n', Macintosh '\r', Windows '\r\n'**
- **mode = 'U' in aggiunta alle altre modalità r/w applica un "traduttore universale" ai caratteri di \newline in aperture del file → portabilità su diversi OS**
- **Alternativa compatta di apertura:**

  with open(filename, 'rU') as f:

  … #uso di oggetto file f
- **Chiusura del file e rilascio delle risorse associate:**

  f.close()

Lettura da file

- Diversi modi per leggere il contenuto di un file
- Lettura di tutto il file o di una sua porzione

  f.read(size)

  ritorna il contenuto letto sotto forma di stringa
  - Argomento size opzionale
  - Se omesso, legge l'intero contenuto del file fino al carattere EOF
  - Se presente e positivo, legge dal file un numero di byte pari a size
  - Se il primo carattere letto è EOF, restituisce una stringa vuota

Lettura di un file

- Leggere dal file una linea alla volta

  f.readline()

```
>>> f.readline()
'First line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

Ad ogni invocazione legge la linea successiva

Stringa vuota quando incontra EOF

- Leggere tutto il file ritornando una lista di stringhe, una per ogni linea

  f.readlines()

```
>>> f.readlines()
['First line of the file.\n', 'Second line of the file\n']
```

Lettura di un file

- **Approccio molto comodo per leggere un file una riga alla volta: iterare con un ciclo for sull'oggetto (iterabile) file**

```
>>> for line in f:
...         print line
First line of the file.
Second line of the file
```

Oggetto file: dotato di metodo __iter__()

Es. Ottenere lista delle parole contenute in un file

```
>>> wordList= [ ]
>>> for line in f:
…       for word in line.split():
…           wordList.append(word)
```


Scrittura su file

- **Per scrivere su un file (aperto in scrittura)**
  - **f.write(string) scrive *string* nel file f presso la posizione corrente del cursore - restituisce *None***
  - ***f.writelines(sequence)***

    scrive sul file una sequenza (liste, tuple o set) di stringhe (valore di ritorno *None*)

- **Nota: Per scrivere qualcosa che non sia una stringa su un file di testo occorre prima convertirlo:**

```
>>> value = ('the answer', 42)
>>> s = str(value)
>>> f.write(s)
```

v → tupla
s → "('the answer',42)"

Lettura/scrittura non sequenziale

- **f.tell() restituisce un intero che rappresenta la posizione corrente nel file f**
  - **Misurata in byte dall'inizio del file**
- **f.seek(offset, from_what) sposta la posizione corrente all'interno del file f**
  - **La posizione è calcolata aggiungendo *offset (che può essere negativo) a un reference point from_what:***
    - ***from_what = 0 misura dall'inizio del file (default)***
    - ***from_what = 1 dalla posizione corrente***
    - ***from_what = 2 dalla fine del file***

→ESEMPI:
file.py
file2.py

IOError

Meccanismo delle eccezioni: IOError exception

```
try:
    filename = "prova2.txt"
    f = open(filename)
except IOError as error:
    print("problem reading '"+filename+"': ", error)
    inputText = " "
else:
    inputText = f.readlines()
    f.close()
finally:
    print("Risultato: " + inputText)
```

<type 'exceptions.IOError'>

Modulo pickle

- Come si procede quando si vogliono salvare su file strutture dati complesse come list, dictionary, istanze di classi?
- Python fornisce un modulo standard chiamato pickle, che evita ai programmatori la scrittura di codice per il salvataggio / caricamento dati
- Prende in input praticamente qualunque oggetto Python e lo converte in una rappresentazione testuale (byte stream)
  - Operazione di pickling (o serializzazione)
  - Operazione di ricostruzione dell'oggetto dal byte stream: unpickling

→ESEMPI:
pickling.py

Modulo pickle

- Dato un oggetto x (di qualunque natura) e un oggetto file f aperto in scrittura binaria ("wb"), è sufficiente una sola linea di codice per effettuarne il pickling:

```
import pickle
pickle.dump(x, f)
```

- Unpickling (con f oggetto file aperto in lettura binaria "rb"): x = pickle.load(f)

Es.

```
data = { 'a': [1, 2.0, 3, 4+6j],
'b': ("character string", b"byte string"),
'c': set([None, True, False]) }
with open('data.pickle', 'wb') as f:
    pickle.dump(data, f)
```

Gestione di Directory

File path e portabilità del codice

- Un ostacolo alla portabilità del codice su diversi OS è rappresentata dalle diverse convenzioni per la costruzione dei percorsi dei file (file path)
- Uso di backslash “\” su Windows
- Uso di slash “/” su Linux e Unix-like
- Python mette a disposizione sotto-moduli del modulo os che offrono funzioni per gestire automaticamente i percorsi del file system in modo che il codice sia portabile su diverse piattaforme

Modulo os.path

- os.path: sotto-modulo per la manipolazione dei percorsi all'interno del file system
- Maggiore portabilità del codice rispetto alla manipolazione diretta delle stringhe riguardanti percorsi del file system:

  import os.path

  os.path.join("snakes", "Python")

  Produce:

  - 'snakes\\Python' (windows) (*escape*)
  - 'snakes/Python' (osX, unix, linux, …)

Modulo os.path

- os.path.split: funzione che separa l'ultimo componente di un path
  - Restituisce una tupla con due componenti (stringhe): il path della directory padre e l'ultimo componente del path
  - Ad esempio, in windows:

    os.path.split("C:\\Program Files\\Python25\\Lib")

    produce: ('C:\\Program Files\\Python25', 'Lib')
  - Su Linux (con assegnamento multiplo):

    parent_dir, name = os.path.split("/usr/bin/python")

    parent dir → '/usr/bin'

    name → 'python'

Contenuto di directory

- Modulo os.listdir
  - Restituisce una lista con il contenuto della directory (file, sotto-directory)
  - os.listdir(“.”) per la directory corrente
- Funzioni per determinare se un path si riferisce a un file o a una directory:
  - os.path.isfile(path) → True in caso di file
  - os.path.isdir(path) → True in caso di directory
  - Entrambe restituiscono False se il path specificato non esiste
- os.environ['HOME'] → restituisce il path della home dir

Contenuto di directory

```
>>> import os
>>> path = os.path.join("/","usr","local","bin")
>>> print path    #su Linux
/usr/local/bin
>>> print os.path.split(path)
('/usr/local', 'bin')
>>> print os.listdir(path)
['charm', 'django-admin.pyc', 'django-admin',
   'django-admin.py']
>>> print os.path.isfile(path)   # restituisce False
>>> print os.path.isdir(path)    # restituisce True
```

Esplorare la struttura del file system

- Python mette a disposizione una potente funzione per l'esplorazione del file system

  walk(path) #funzione di os
- Percorre ricorsivamente la struttura del file system e per ogni directory crea una tupla di tre elementi

  1) Il percorso della directory

  2) Una lista dei nomi di directory contenute

  3) Una lista dei nomi di file contenuti
- walk() restituisce un oggetto generatore

```
>>> a = os.walk(“.”)
>>> for i in a: ...    #per elaborazioni successive
```

→ESEMPI:
walk.py

Operazioni su file e directory

- Moduli shutil e os contengono funzioni per operare velocemente sui file
  - Operazioni per rinominare, spostare, copiare e rimuovere file
- shutil.move
  - rinominare un file (spostamento nella stessa directory):

    import shutil

    shutil.move(“server.log”, “server.log.backup”)
  - spostare un file in un’altra directory: shutil.move(“old_mail.txt”, “C:\\data\\archive\\”)

Operazioni su file e directory

- shutil.copy:
  shutil.copy(“important.dat”, “C:\\backups”)
- Per cancellare un file:
  os.remove(“junk.dat”)
- Per creare una nuova directory:
  - os.mkdir(“C:\\photos\\zoo\\snakes”)
    (la directory parent deve essere esistente)
  - os.makedirs(“C:\\photos\\zoo\\snakes”)
    (crea eventualmente tutte le directory parent)
- Per rimuovere una directory vuota:
  os.rmdir(“C:\\photos\\zoo\\snakes”)