Corso di **Python**

Lezione 13

Interazione con i Data Base

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

Cosa vedremo oggi

- Collezioni
  - namedtuple
- CSV
- Data Base
  - Oracle
  - PostgreSQL
  - MySQL
  - SQLite

La libreria ***collections***

- Nella libreria collections sono state progressivamente definite una serie di strutture dati interessanti, come ad esempio:

  `collections.`**`deque`**`(`*`[iterable[, maxlen]]`*`)`

  - implementazione thread safe e veloce di una doppia coda generalizzata (Python 2.4)

  `collections.`**`OrderedDict`**`(`*`[items]`*`)`

  - Dizionario che ricorda l'ordine di inserimento (Python 2.7)

  `collections.`**`ChainMap`**`(`*`*maps`*`)`

  - aggregatore di dizionari i un unico dizionario aggiornabile (Python 3.3)

- In particolare per l'accesso a tabelle di dati possono risultare utili le ***namedtuple***, introdotte in Python 2.6

Le *namedtuple*

- Si tratta di una funzione per creare classi basate su una tupla
  - In OOP sarebbe una funzione *factory*
- La caratteristica peculiare di una namedtuple è di creare tuple che hanno dei nomi associati ad ogni elemento
- L'associazione indice-attributo è a livello di classe e perciò non appesantisce le tuple che vengono create
- Sintassi:

```
namedtuple(typename, field_names, verbose=False, rename=False)
```

  - *typename*: Nome del tipo che verrà associato agli oggetti creati
  - *field_names*: Nomi da associare agli elementi della tupla; può essere una sequenza di stringhe o una singola stringa con parole separate da spazi. I nomi validi associabili seguono le stesse regole degli identificatori.
  - *verbose*: visualizza la classe dinamica associata al tipo creato con la funzione
  - *rename*: Se messo a True allora ridenomina automaticamente i nomi non validi

Esempio di *namedtuple*

```
>>> from collections import namedtuple
>>> Point = namedtuple('MyPoint', ['x', 'y'])
>>> p = Point(11, y=22)
>>> p[0] + p[1]          # indicizzabili come le tuple normali
33
>>> x, y = p             # spacchettamento come le tuple
>>> x, y
(11, 22)
>>> p.x + p.y            # attributi accessibili anche con nome
33
>>> p                    # la __repr__() include i nomi ...
Point(x=11, y=22)
>>> type(p)              # il tipo è l'argomento della funzione ...
<class '__main__.MyPoint'>
```

Factory dinamica

- La costruzione di un tipo con namedtuple è aggiornata con la versione di Python
- Per rendersi conto delle differenze provare con *verbose=True* in Python 2.6 e in Python 3.4:

```
>>> from collections import namedtuple
>>> Point = namedtuple('Point', 'x y', verbose=True)
from builtins import property as _property, tuple as _tuple
from operator import itemgetter as _itemgetter
from collections import OrderedDict


class Point(tuple):
    'Point(x, y)'
...
```

- La classe creata con namedtuple contiene anche un costruttore in grado di creare la sequenza di nomi a partire da altri oggetti:
  - Vedremo come questo è particolarmente utile nel caso di tabelle di dati con campi nominali

File CSV

- Formato **CSV** (*Comma Separated Values*):
  - Formato di file usato per l'import/export di dati
  - Excel <==> Data Base
- Da Python 2.3 c'è la libreria *csv*:

```
>>> import csv
>>> with open('eggs.csv', newline='') as csvfile:
...     spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
...     for row in spamreader:
...         print(', '.join(row))
...
Spam, Spam, Spam, Spam, Spam, Baked Beans
Spam, Lovely Spam, Wonderful Spam
```

CSV – *reader* e *writer*

- Il modulo ha due funzioni base per leggere e scrivere i file:

```
csv.reader(csvfile, dialect='excel', **fmtparams)
csv.writer(csvfile, dialect='excel', **fmtparams)
```

  - L'argomento *csvfile* è solitamente un file ma può essere un qualsiasi oggetto iterabile che restituisce stringhe. Per il writer tale oggetto deve implementare un metodo *write()*.
  - L'argomento dialect è relativo ad un settaggio dei parametri specifico per alcuni usi standard.
  - Altri parametri possono essere passati con un dizionario o come *keyword arguments* per fare un override di parametri specifici del dialetto utilizzato:

```
import csv
with open('passwd', newline='') as f:
    reader = csv.reader(f, delimiter=':', quoting=csv.QUOTE_NONE)
    for row in reader:
        print(row)
```

Scrittore di file CSV

```
$ cat csv_writer.py
import csv
import sys

f = open(sys.argv[1], 'wt')
try:
    writer = csv.writer(f)
    writer.writerow( ('Title 1', 'Title 2', 'Title 3') )
    for i in range(10):
        writer.writerow( (i+1, chr(ord('a') + i), '08/%02d/07' % (i+1)) )
finally:
    f.close()

print(open(sys.argv[1], 'rt').read())

$ python csv_writer.py testdata.csv
```

Fonte: Python Module Of The Week: https://pymotw.com/2/csv/index.html#module-csv

Lettore generico di file CSV

```
$ cat csv_reader.py
import csv
import sys

f = open(sys.argv[1], 'rt')
try:
    reader = csv.reader(f)
    for row in reader:
        print(row)
finally:
    f.close()

$ python csv_reader.py testdata.csv
['Title 1', 'Title 2', 'Title 3']
['1', 'a', '08/18/07']
...
```

Fonte: Python Module Of The Week: https://pymotw.com/2/csv/index.html#module-csv

Metodi del *reader*

- L'oggetto reader (csv.DictReader) ha alcuni attributi utili in fase di lettura:

  `csvreader.dialect` : Indica il dialetto utilizzato dal lettore, valore in sola lettura

  `csvreader.line_num` : Numero di linee lette dall'oggetto contenente il testo csv

  `csvreader.fieldnames` : Parametro opzionale che contiene le intestazioni dei vari campi del testo csv

  `csvreader.__next__()` : Ritorna la linea successiva dell'oggetto iterabile, richiamabile con *next(reader)*

- Esempio:

```
import csv
with open('testdata.csv', newline='') as f:
    reader = csv.reader(f, delimiter=',', quoting=csv.QUOTE_NONE)
    for row in reader:
        print(reader.line_num, row)
```

Metodi del *writer*

- L'oggetto writer (csv.DictWriter) ha i seguenti attributi:

  `csvwriter.`**`dialect`** : Sola lettura, indica il dialetto utilizzato dallo scrittore csv

  `csvwriter.`**`writerow`**`(`*`row`*`)` : Scrive una riga sul file destinazione del testo csv, secondo la formattazione definita per il writer

  `csvwriter.`**`writerows`**`(`*`rows`*`)` : Scrive una sequenza di righe sul file destinazione del testo csv, secondo la formattazione definita per il writer

  `DictWriter.`**`writeheader`**`()` : Scrive lo header dichiarato nel file csv

- Esempio:

```
import csv
with open('names.csv', 'w') as csvfile:
    fieldnames = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerow({'first_name': 'Davide', 'last_name': 'Brunato'})
```

Attributi dei dialetti

- I dialetti definiscono degli attributi standard, che possono essere modificati quando definiamo un CSV *reader*/*writer*

  `delimiter``=','` : separatore dei valori

  `quoting``=csv.QUOTE_MINIMAL` : modalità con cui effettuare la quotazione degli elementi (costanti alternative: csv.QUOTE_ALL, csv.QUOTE_NONNUMERIC e csv.QUOTE_NONE)

  `quotechar``='"'` : carattere che viene usato per le quotazioni degl elementi

  `doublequote``=True` : effettua una doppia quotazione dei caratteri usati per quotare

  `lineterminator``='\r\n'` : Terminatore di linea inserito dal writer

  `skipinitialspace``=False` : ignora il primo spazio che segue un delimitatore

  `strict``=False`: solleva un'eccezione (csv.Error) quando il reader trova errori di sintassi

  `escapechar``=None` : carattere di escape da utilizzare sul carattere delimitatore quando il quoting è impostato a QUOTE_NONE (nessun quoting)

Gestione dei dialetti CSV

- Il modulo fornisce un lista di dialetti disponibili:

```
>>> csv.list_dialects()
['excel-tab', 'excel', 'unix']
```

- I dialetti si possono registrare per utilizzare una medesima configurazione in modo semplice:

```
csv.register_dialect(name[, dialect[, **fmtparams]])
```

```
>>> csv.register_dialect('unix-strict', dialect='unix', strict=True)
>>> csv.list_dialects()
['unix', 'excel', 'unix-strict', 'excel-tab']
```

- I dialetti si possono anche ottenere e rimuovere:

```
>>> csv.get_dialect('unix')
<_csv.Dialect object at 0x7f2de5c4b7d8>
>>> csv.get_dialect('unix').strict
0
>>> csv.get_dialect('unix-strict').strict
1
>>> csv.unregister_dialect('unix-strict')
```

Namedtuple e CSV

- Il metodo *namedtuple*._make(iterable) permette di creare una nuova istanza da una sequenza o oggetto iterabile:

```
>>> t = [11, 22]
>>> Point._make(t)
Point(x=11, y=22)
```

- Nel caso dei file CSV si può usare per mappare gli oggetti letti da un reader in una namedtuple opportuna:

```
EmployeeRecord = namedtuple('EmployeeRecord', 'name, age, title,
    department, paygrade')

import csv
for emp in map(EmployeeRecord._make, csv.reader(open("employees.csv", "rb"))):
    print(emp.name, emp.title)
```

CSV – Ultime note

- Le chiamate *`writer.writerow(s)`* ritornano la quantità di byte scritti:

```
>>> with open('eggsout.csv', 'w', newline='') as csvfile:
...     spamwriter = csv.writer(csvfile, delimiter=' ', quotechar='|',
...         quoting=csv.QUOTE_MINIMAL)
...     spamwriter.writerow(['Spam'] * 5 + ['Baked Beans'])
...     spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])
...
40
37
```

- C'è un limite per il singolo campo csv, che può essere variato:

```
>>> csv.field_size_limit()
131072
>>> csv.field_size_limit(200000)
131072
>>> csv.field_size_limit()
200000
```

Python e i database

- Le interfacce per i database di Python seguono la struttura delle DB-API 2.0
  - PEP 249 - https://www.python.org/dev/peps/pep-0249/
- Il supporto non è incluso nella standard library ma con librerie di terze parti
- DB Relazionali supportati direttamente:
  - IBM DB2 e Informix, Firebird, MySQL, Oracle, PostgreSQL, MS SQL Server
- ODBC drivers:
  - https://wiki.python.org/moin/ODBCDrivers
  - pyodbc, mxODBC
- Ref: https://wiki.python.org/moin/DatabaseInterfaces

DB-API 2.0 (PEP 249)

API syntax	API description
`connect``(`*`parameters`*`)`	Costruttore di connessione al database
`InterfaceError` `DatabaseError`	2 gruppi di eccezioni, uno legato all'interfacciamento e l'altro al database
`conn``.`**`close`**`()`	Chiusura della connessione
`conn``.`**`commit`**`()`	Effettua il *commit* di ogni transazione pendente. Non fa nulla se il DB non supporta transazioni.
`conn``.`**`rollback`**`()`	Effettua il rollback delle transazioni pendenti. Opzionale, presente se il DB supporta transazioni.
`conn``.`**`cursor`**`()`	Ritorna un oggetto cursore
`cursor``.`**`description`**	Attributo multivalore di sola lettura, descrittivo del cursore
`cursor``.`**`rowcount`**	Attributo read-only contenente il numero di righe prodotte con l'ultima operazione sul DB

DB-API: Metodi dei cursori

.execute(*operation* [,*parameters*])

- Esegue query o comando sul DB

.executemany(*operation* ,*parameters_sequence*)

- Esegue un'operazione più volte su sequenza di parametri

.fetchone()

- Estrae la riga successiva dal risultato dell'ultima query effettuata; None se non ci sono più righe, Error se la queryha dato risultato vuoto

.fetchmany([*size=cursor.arraysize*])

- Estrae più righe dal risultato (una lista di tuple)

.fetchall()

- Estrae tutte le righe dal risultato (una lista di tuple)

.arraysize

- Attributo scrivibile che definisce il numero di righe lette da .fetchmany()

.close()

- Chiude il cursore, che successivamente non potrà più essere usato per nessuna operazione.

Librerie per Oracle DB e PostgreSQL

- Per database Oracle c'è la libreria *cx_Oracle*:
  - http://cx-oracle.sourceforge.net/
  - Richiede l'installazione del client Oracle (attualmente 11.2 o 12.1)
  - Difficile trovare il pacchetto nelle distribuzioni, installarla con pip:

```
pip install cx_Oracle
```

- Per PostgreSQL usare la libreria :
  - http://initd.org/psycopg/
  - È disponibile in pacchetto sulle distribuzioni Linux, in alternativa è comunque installabile con pip:

```
pip install psycopg2
```

Librerie Python per MySQL

- MySQL-python (MySQLdb):
  - Pacchetto storico, licenza GPL
  - Compatibile 2.3-2.6 (non è compatibile con Python 3!)
  - http://sourceforge.net/projects/mysql-python
- PyMySQL:
  - 100 % compatibile con MySQLdb
  - Python al 100 %, v2 e v3, licenza MIT
  - Buone performance
  - https://github.com/PyMySQL/PyMySQL
- mysqlclient-python (python-mysql):
  - Fork di MySQL-python, che aggiunte il supporto per Python 3
  - https://github.com/PyMySQL/mysqlclient-python
- MySQL-connector di Oracle:
  - Python al 100%, compatibile v2 e v3
  - Licenza GPL v2
  - https://dev.mysql.com/downloads/connector/python/

Uso delle librerie MySQL

- Le si dovrebbe trovare come pacchetti, ma bisogna un attimo districarsi tra nomi diversi
- Ad esempio in Fedora 23 troviamo i seguenti pacchetti:

```
[brunato@direwolf ~]$ dnf list | grep -i mysql | grep python
mysql-connector-python.noarch          1.1.6-4.fc23              @@commandline
mysql-connector-python3.noarch         1.1.6-4.fc23              @fedora
python-mysql.x86_64                    1.3.7-1.fc23              @@commandline
python-mysql-debug.x86_64              1.3.7-1.fc23              updates
python-storm-mysql.x86_64              0.20-5.fc23               fedora
python2-PyMySQL.noarch                 0.6.7-4.fc23              updates
python3-PyMySQL.noarch                 0.6.7-4.fc23              updates
python3-mysql.x86_64                   1.3.7-1.fc23              updates
python3-mysql-debug.x86_64             1.3.7-1.fc23              updates
```

- Poi decidere quale usare, ma ricordasi che eventualmente si possono fare import strutturati con eccezioni e aliasing, sfruttando il fatto che le librerie usano le stesse API

SQLite3

- SQLite: database leggero basato su disco:
  - Disponibile nella libreria standard di Python
  - DB-API 2.0 compliant
  - Si può usarlo per il testing di applicazioni o nel caso di applicazioni che non richiedono un DB con performance elevate

- Esempio con CSV e namedtuple:

```
EmployeeRecord = namedtuple('EmployeeRecord', 'name, age, title, department, paygrade')


import sqlite3
conn = sqlite3.connect('/companydata')
cursor = conn.cursor()
cursor.execute('SELECT name, age, title, department, paygrade FROM employees')
for emp in map(EmployeeRecord._make, cursor.fetchall()):
    print(emp.name, emp.title)
```

DB-API con SQLite3

- Per creare un nuovo database:

```
>>> import sqlite3
>>> conn = sqlite3.connect('example.db')   # Creato il relativo file nella directory corrente
```

- Per creare dati all'interno del database serve poi creare un cursore ed eseguire un comando per creare una tabella

```
c = conn.cursor()

# Crea una tabella
c.execute('''CREATE TABLE stocks
             (date text, trans text, symbol text, qty real, price real)''')

# Inserisce una linea nella tabella
c.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")

# Salva le modifiche con un commit
conn.commit()

# Dopo il commit si può chiudere la connessione al DB.
conn.close()
```

Esempio con *python-mysql*

```
import MySQLdb, collections, datetime
imdb = MySQLdb.connect(IMDB_HOST, IMDB_USER, IMDB_PASSWORD, IMDB_NAME, IMDB_PORT, charset='utf8')
cursor = imdb.cursor()
today = datetime.date.today()

cursor.execute("SELECT * FROM OrganizationalUnit")
results = cursor.fetchall()
UnitaOrganizzativaIMDB = collections.namedtuple('UnitaOrganizzativaIMDB',
         [col[0] for col in cursor.description])

for row in results:
    unitorg = UnitaOrganizzativaIMDB(*row)   #
    if unitorg.startDate is None:
        active = unitorg.endDate is None or unitorg.endDate >= today
    else:
        active = unitorg.endDate is None or \
             ((unitorg.startDate <= today) and (unitorg.endDate >= today))
    if active:
        print(unitorg)

imdb.close()
```

Esempio con *cx_Oracle*

```
import cx_Oracle, collections
ugovdb = cx_Oracle.connect(
    "{0}/{1}@{2}:{3}/{4}".format(UGOVDB_USER, UGOVDB_PASSWORD,
    UGOVDB_HOST, UGOVDB_PORT, UGOVDB_SERVICE_NAME))

cursor = self.ugovdb.cursor()
cursor.execute('select table_name from cat where table_name like \'V_IE_DG10_X_COMM%\'')
tables = cursor.fetchall()
for t in tables:
    try:
        cursor.execute('select * from {0} WHERE ID_DG={1}'.format(t[0], 3332))
    except cx_Oracle.DatabaseError:
        continue

    Table = collections.namedtuple(t[0], [col[0] for col in cursor.description])

    print("### TABELLA: {0} ###".format(t[0]))
    for row in cursor:
        table_row = Table(*row)
        print(table_row)
        print(" ")

ugovdb.close()
```