

SQLite

Informatica@SEFA 2017/2018 - Laboratorio 9

Massimo Lauria <massimo.lauria@uniroma1.it>
<http://massimolauria.net/courses/infosefa2017/>

Lunedì, 4 Dicembre 2017

Ma prima...
soluzioni degli esercizi del
Lab. 8

Esercizio 14

```
parse_dati(testo)
```

La funzione ha in input una stringa di testo, che è costituita da diverse righe. Ogni riga contiene 2 valori numerici float, separati da :

```
x0 : y0  
x1 : y1  
x2 : y2  
...
```

La funzione deve produrre la lista `[(x0,y0), ...]`.

- ▶ ignorete le righe vuote;
- ▶ sollevate `ValueError` se una riga è mal formattata.

Esercizio 14 (esempio)

```
from lab08 import parse_dati
data=''
0 : 1.0
1 : 1.2
2 : 2.5
'''
print(parse_dati(data))
```

1
2
3
4
5
6
7

```
[(0.0, 1.0), (1.0, 1.2), (2.0, 2.5)]
```

Esercizio 14 (soluzione)

```
def parse_dati(testo):
    output=[]
    for line in testo.splitlines():

        if len(line.strip())==0:
            continue

        l = line.split(":")
        if len(l) != 2:
            raise ValueError("Riga non vuota e mal formattata")
    )

    x,y = float(l[0]),float(l[1])
    output.append( (x,y) )
    return output
```

Esercizio 15

```
plot_dati(testo,nomefile)
```

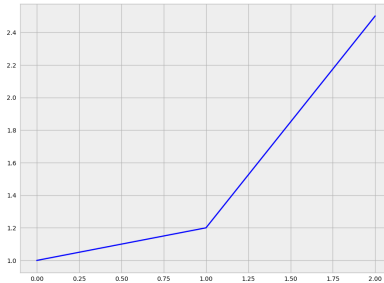
L'esercizio è molto simile al precedente. Dovete analizzare la stringa di testo alla stessa maniera, ma invece di produrre le coppie di valori in output, interpretate le coppie come i punti di una funzione $x \mapsto y$ fate il grafico della funzione con la funzione `plot` di `matplotlib`, salvando l'immagine nel file `nomefile`.

Esercizio 15 (esempio)

```
data=''
0 : 1.0
1 : 1.2
2 : 2.5
''

from lab08 import plot_dati
plot_dati(data,'assets/lab08_esempio.png')
```

1
2
3
4
5
6
7



Esercizio 15 (soluzione)

```
def plot_dati(testo,filename): 1
    x=[] 2
    y=[] 3
    for line in testo.splitlines(): 4
        5
        if len(line.strip())==0: 6
            continue 7
        8
        l = line.split(":") 9
        if len(l) != 2: 10
            raise ValueError("Riga non vuota e mal formattata"11
    ) 12
    13
    x.append( float(l[0])) 14
    y.append( float(l[1])) 15
    16
    plot(x,y) 17
    savefig(filename)
```


Esercizio 16

```
frequenze(testo, lista_parole)
```

La funzione deve prendere in input una stringa e deve restituire una lista della stessa lunghezza di `lista_parole`, nella posizione *i*-esima della lista restituita ci deve essere il numero di occorrenze della parola *i*-esima in `lista_parole`.

- 'Casa' , 'caSa', 'casa' sono la stessa parola

Esercizio 16 (esempio)

```
from lab08 import frequenze 1
2
print(frequenze("Quanta legna taglia un taglia-legna, se vuol 3
    tagliare legna",
    ['legna','taglia','castoro'])) 4
```

```
[3, 2, 0]
```

Esercizio 16 (soluzione)

```
def frequenze(testo, lista_parole):           1
    """Restituisce la lista delle frequenze delle parole  2
    elencate
    """                                         3
    # trova i caratter non alfabetici          4
    noalpha=''                                5
    for c in testo:                             6
        if not c.isalpha() and c not in noalpha: 7
            noalpha += c                        8
    # separa tutte le parole nel testo          9
    for c in noalpha:                           10
        testo = testo.replace(c, ' ')          11
    data=testo.lower().split()                  12
    # restituisci le frequenze                  13
    return [data.count(word.lower()) for word in lista_parole] 14
```

(setq org-latex-default-table-environment "longtable")

SQLite

SQLite

Gestisce una base di dati come un singolo file

- portabile
- comodo da trasferire
- nessun bisogno di configurare un processo server
- incluso nella distribuzione Anaconda.

Basi di dati pre-popolate

Sulla pagina web del corso abbiamo due basi di dati.

- **Registro automobilistico**: la base di dati usata per gli esempi nel libro di testo SQL.
- **Chinook**: una base di dati contenente le informazioni di un negozio musicale.

Entrambe le basi di dati sono fornite come:

1. file di database SQLite
2. istruzioni SQL per rigenerare il DB

Linea di comando

Potete aprire un file già pronto con la base di dati

```
$ sqlite3 registro_automobilistico_db.sqlite
```

Oppure eseguire una lista di comandi SQL per inizializzare

```
$ sqlite3 -init registro_automobilistico.sql nuovofile.sqlite
```

a questo punto vi trovate nel prompt dei comandi di SQLite, che si aspetta istruzioni SQL oppure comandi SQLite.

```
sqlite>
```

Comandi SQLite

I comandi che iniziano con il punto non sono istruzioni SQL ma sono comandi di SQLite per la gestione di DB e file. Potete vederli tutti con il comando `.help`.

```
.help
```

1

<code>.auth ON OFF</code>	Show authorizer callbacks
<code>.backup ?DB? FILE</code>	Backup DB (default "main") to FILE
<code>.bail on off</code>	Stop after hitting an error. Default OFF
<code>.binary on off</code>	Turn binary output on or off. Default OFF
<code>.changes on off</code>	Show number of rows changed by SQL
<code>.check GLOB</code>	Fail if output since <code>.testcase</code> does not match
<code>.clone NEWDB</code>	Clone data into NEWDB from the existing database
<code>[...]</code>	

Ispezionare il DB

```
.tables
```

```
1
```

Categorie	Fabbriche	Proprietari	Veicoli
Combustibili	Modelli	Proprietà	

Lo schema del DB

Lo schema viene descritto da SQLite come la sequenza di istruzioni SQL usate per generare le tabelle.

```
.schema Veicoli
```

1

```
CREATE TABLE Veicoli (  
    Targa            nvarchar(10) primary key,  
    Cilindrata       integer,  
    Cavalli_Fiscali  integer,  
    Velocità         integer,  
    Posti            integer,  
    Immatricolazione date,  
  
    Cod_Categoria    nchar(2) references Categorie(Cod_Categoria),  
    Cod_Combustibile  nchar(2) references Combustibili(Cod_Combustibile),  
    Cod_Modello      nchar(3) references Modelli(Cod_Modello)  
);
```

Lo schema del DB (2)

Se non si inserisce il nome di una tabella come parametro, allora viene stampato lo schema di tutta la base di dati.

```
.schema
```

1

```
CREATE TABLE Categorie (  
    Cod_Categoria nchar(2) primary key,  
    Nome_Categoria nvarchar(30)  
);  
CREATE TABLE Combustibili (  
    Cod_Combustibile nchar(2) primary key,  
    Descrizione_Combustibile nvarchar(30)  
);  
[...]
```

Esecuzione di comandi SQL

Il comando `select` è utilizzato per interrogare la base di dati e leggere informazioni da essa. Nella sua versione più semplice

- mostra i dati di una tabella (`Combustibili`)
- ne mostra tutte le colonne (il simbolo `*`)

```
select * from Combustibili;
```

1

```
01,Benzina  
02,Gasolio  
03,GPL  
04,Metano
```

Miglioriamo la stampa dei risultati

I comandi `.mode` e `.header` possono essere usati per avere un output più leggibile.

```
.mode column      1
.header on       2
select * from Combustibili;  3
```

Cod_Combustibile	Descrizione_Combustibile
-----	-----
01	Benzina
02	Gasolio
03	GPL
04	Metano

Evidenziamo i valori nulli

Le celle delle tabelle che contengono valori nulli non vengono stampate. Per migliorare la leggibilità possiamo modificarne la rappresentazione.

```
.mode column                                1
.header on                                  2
.nullvalue NULL                             3
select Targa,Cilindrata,Posti from Veicoli where Posti<5; 4
```

Targa	Cilindrata	Posti
-----	-----	-----
C78905GT	1998	4
C845905Z	NULL	3
D239765W	NULL	2

Concludiamo la sessione

Si può uscire da SQLite premendo Control+D oppure con
in comando `.exit`