Esercizi di programmazione 2017/2018

Informatica@SEFA 2017/2018 - esame orale

Massimo Lauria <massimo.lauria@uniroma1.it>*

Venerdì, 5 Gennaio 2018

Questi esercizi devono essere svolti prima dell'esame orale. Devo essere consegnati a vista, durante l'esame orale, su un singolo file Python `esercizio2017.py`. Il programma deve superare i test preliminari che si trovano sul sito del corso.

1 Valutazione dell'esercizio

Il file consegnato verrà testato in ambiente Linux o MacOS. L'ambiente python3 di esecuzione sarà dotato del modulo `matplotlib` per completare la parte quarta dell'esercizio, riguardate i plot.

1.1 Autovalutazione

Avete a disposizione un file di test **preliminare**. Il programma deve passare **TUTTI** i test preliminari o non potrete fare continuare con l'esame orale. Il passaggio dei test preliminari **suggerisce solamente** che la soluzione sembra ben impostata, ma non dice nulla sulla sua correttezza.

Il file di test `test_esercizio2017.py` (link cliccabile)

e per essere eseguito, la stessa cartella deve contenere il vostro esercizio `esercizio2017.py` più tre file di dati presenti sulla pagina del corso (i link seguenti sono cliccabili).

1. il database `registro_automobilistico_db.sqlite`

*http://massimolauria.net/courses/infosefa2017/

2. il database `chinook_db.sqlite`
3. il file di testo `alice.txt` (codificato `utf-8-sig`)

1.2 Valutazione durante l'esame orale

Per verificare la correttezza dell'esercizio verranno eseguiti ulteriori test segreti e verranno fatte domande sul codice durante l'esame orale.

Il voto dell'esame dipenderà **anche** dall'esito di queste prove. É permesso discutere con i colleghi le tracce e gli esercizi, per risolvere incomprensioni e dubbi su cosa viene richiesto dalla traccia. **Non è assolutamente permesso** scrivere i programmi con la collaborazione di altri studenti, pena l'annullamento dell'esame orale e la cancellazione di eventuali scritti. Se non siete sicuri di come si debbano comportare le vostre funzioni nei casi limite, provate ad eseguire i test, usate il buon senso e confrontatevi con i colleghi.

2 Prima parte - Liste

Scrivere una funzione `spacchetta` che prende una lista di tuple, tutte della stessa dimensione d, e produce d liste: dove la lista i-esima contiene gli i-esimi elementi delle tuple, nello stesso ordine. Per esempio

```
from esercizio2017 import spacchetta
result=spacchetta([(1,'a'),(4,'b'),('x','ff')])
print(result)
result=spacchetta([(1,'a','casa'),('foglia',4,'b')])
print(result)
```

```
[[1, 4, 'x'], ['a', 'b', 'ff']]
[[1, 'foglia'], ['a', 4], ['casa', 'b']]
```

3 Seconda parte - Matrici

Tutti questo esercizi riguardano matrici, rappresentate in Python. Una **matrice** è una griglia rettangolare di numeri che viene di solito utilizzata per rappresentare elementi dell'algebra lineare. Una matrice con M righe ($M > 0$) e N colonne ($N > 0$) viene detta matrice $M \times N$. Normalmente in

matematica le righe sono indicizzate da 1 a M e le colonne da 1 a N, ma in python è più comodo indicizzare da 0 a $M-1$ e da 0 a $N-1$, rispettivamente.

Per questi esercizi le matrici saranno rappresentate come liste di liste di numeri di tipo `float`. Una matrice $M \times N$ è raprresentata come una lista di M elementi, dove ogni elemento di questa lista è a sua volta una lista di N numeri `float` (osservate bene che le liste che rappresentano le righe devono avere uguale lunghezza). Viene richiesto che M e N siano entrambi numeri maggiori di 0.

Per esempio la matrice $\begin{bmatrix} 1.3 & 2.0 & 0.4 \\ 3.5 & -7.2 & 2.3 \end{bmatrix}$ viene rappresentata in Python come

```
A=[[ 1.3, 2.0, 0.4], [3.5, -7.2, 2.3] ]
```

In questo modo si può accedere all'elemento alla riga `i` e colonna `j` della matrice `A` con l'espressione `A[i][j]`. Dovete scrivere una serie di funzioni che permettano di effettuare operazioni su matrici. Le operazioni da realizzare sono molto semplici e richiedono solo una conoscenza elementare dell'algebra.

- `ismatrix(A)` deve restituire `True` o `False` per indicare se `A` sia effettivamente la rappresentazione di una matrice, secondo il criterio specificato sopra.

Tutte le funzioni successive devono sollevare `ValueError` se la matrice (o le matrici) passate come argomento non sono rappresentazioni corrette.

- `size(A)` restituisce una coppia `(r,c)` dove `r` è il numero di righe della matrice e `c` il numero di colonne.
- `zeromatrix(r,c)` restituisce una matrice piena di zeri con `r` righe e `c` colonne. Verificare che siano entrambi maggiori di 0.
- `transpose(A)` restituisce la matrice trasposta di `A`

Per effettuare alcune operazioni tra due matrici è necessario che le dimensioni siano compatibili o che gli argomenti abbiano il tipo `float`. In caso non lo siano, sollevare un `ValueError`.

- `matrixsum(A,B)` restituisce la matrice che rappresenta la somma $A + B$.
- `scale(v,A)` se v è un numero float ed A una matrice, restituisce vA, ovvero la matrice ottenuta da A moltiplicandone tutti i numeri per v.

- `mult(A,B)` realizza la moltiplicazione tra matrici. Ricorda che la moltiplicazione è lecita solo tra due matrici di dimensioni, rispettivamente, $n \times k$ e $k \times m$. Il risultato è una matrice di dimensione $n \times m$.

```
from esercizio2017 import transpose,scale,ismatrix,matrixsum,size
A=[[ 1.3, 2.0, 0.4], [3.5, -7.2, 2.3] ]
print( transpose(A) )
B=scale(-2.5,A)
print( B )
print( matrixsum(A,B) )
print( size(A))
print( size(transpose(A)))
```

```
[[1.3, 3.5], [2.0, -7.2], [0.4, 2.3]]
[[-3.25, -5.0, -1.0], [-8.75, 18.0, -5.75]]
[[-1.95, -3.0, -0.6], [-5.25, 10.8, -3.45]]
(2, 3)
(3, 2)
```

4 Terza parte - Elaborazione di testi

Scrivere due funzioni. La prima

```
def conteggiotesto(testo,lunghezza):
    ...
```

deve restituire il numero di parole **distinte**, di lunghezza `lunghezza` all'interno della stringa `testo`. Parole uguali ma non differenti maiuscole e minuscole devono essere considerate come ripetizioni della **stessa** parola. La seconda funzione

```
def conteggiofile(nome_file,lunghezza,encoding='utf-8')
    ...
```

deve fare la stessa cosa, ma invece di effettuare il conteggio in una stringa, deve farlo all'interno del file di nome `nome_file`. Il file deve essere aperto con l'encoding `utf-8` se non viene specificato l'argomento opzionale `encoding`. Vediamo degli esempi:

```
from esercizio2017 import conteggiofile,conteggiotesto
print(conteggiotesto('Casa caSa, casa gatto cane.',4))
print(conteggiotesto('Casa caSa, casa gatto cane.',5))
print(conteggiotesto('Casa caSa, casa gatto cane.',-3))
print(conteggiofile('../dataset/alice.txt',5,encoding='utf-8-sig'))
print(conteggiofile('../dataset/alice.txt',12,encoding='utf-8-sig'))
print(conteggiofile('../dataset/alice.txt',16,encoding='utf-8-sig'))
```

```
2
1
0
480
32
1
```

5 Quarta parte - Plot

Dovete realizzare una funzione che, utilizzando il modulo `matplotlib.pyplot`, produca un plot come visto a lezione ed in laboratorio.

```
def produciplot(func1,func2,nome_file):
    ...
```

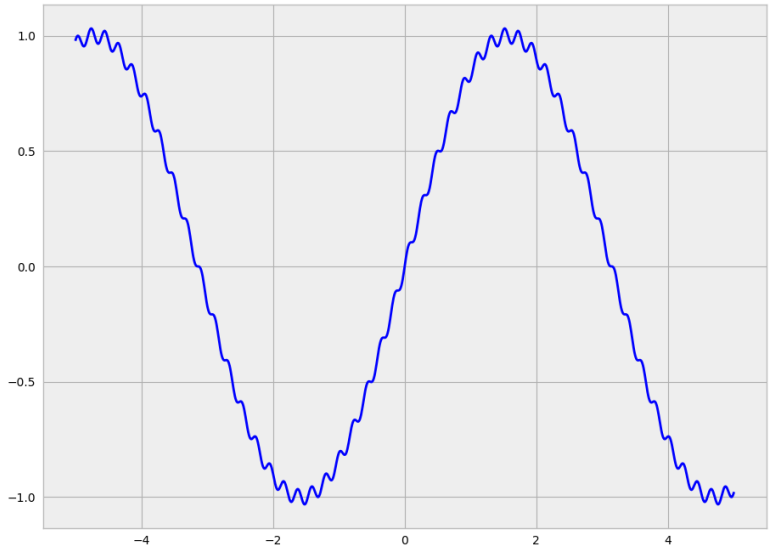
Il plot deve rappresentare la funzione che ad ogni `x` mappa il valore di `func1(x)+func2(x)`. Il valore delle `x` deve andare da -5.0 a 5.0 per intervalli di 0.01 (Il che vuol dire che dovrete calcolare il valore della funzione su 1001 punti).

Non personalizzate troppo il plot. É sufficiente che la funzione venga rappresentata con una linea continua rossa (la scelta di default).

Le due funzioni passate come argomento sono **garantite** essere funzioni che mappano un numero reale in un numero reale (e.g. `sin`, `round`, ...). Il terzo argomento è il nome del file nel quale salvare il plot. In questo esercizio i dati passati in input sono corretti, quindi non c'è bisogno di sollevare eccezioni o di verificare la correttezza degli argomenti passati.

I test verificheranno che i plot vengano prodotti, ma la loro correttezza verrà verificata a vista. Il programma di test produce i plot in un file pdf chiamato `plot_esercizi2017.pdf`. Vediamo un esempio

```
from esercizio2017 import produciplot
from math import sin
def smallsin(x):
    return sin(30*x)/30
produciplot(sin,smallsin,"assets/esempioesame2017.png")
```



6 Quinta parte - Database e SQL via Python

Questa parte del compito prevede che si scrivano delle funzioni python che si interfaccino con i database SQLite e che eseguino certe query. Tutte le funzioni realizzate per questo esercizio devono restituire una lista di tuple che costituiscono la risposta alla query, come visto in una delle esercitazioni. Le funzioni che accettano argomenti devono verificarne la correttezza.

Query sul database `registro_automobilistico_db.sqlite`. Scrivere

- una funzione `query1(N)` che restituisca i nomi di tutte le fabbriche che producono in totale `N` versioni di modelli di auto.

```
from esercizio2017 import query1
print(query1(3))
```

```
[('Ford',), ('Honda',)]
```

- una funzione `query2()` che restituisca la lista dei proprietari di auto nel registro, specificando cognome, nome ed il numero di auto registrate da ognuno. La lista deve essere ordinata per cognome e nome

in modo crescente.

```
from esercizio2017 import query2
print(query2())
```

```
[('Bernocchi', 'Giuseppina', 2),
 ('Cottarelli', 'Cristian', 2),
{...omissis...}
```

Query sul database `chinook_db.sqlite`. Scrivete

- una funzione `query3(a,b)` che produce i nomi degli artisti ed il loro numero di album nell'archivio, ma solo per quegli artisti che hanno almeno `a` e al massimo `b` album. Le righe devono essere ordinate in modo decrescente rispetto al numero di album e (come ordine secondario) in modo crescente rispetto al nome.

```
from esercizio2017 import query3
print(query3(11,20))
print(query3(7,5))
print(query3(6,10))
```

```
[('Led Zeppelin', 14), ('Deep Purple', 11)]
[]
[('Metallica', 10), ('U2', 10), ('Ozzy Osbourne', 6)]
```