Laboratorio 11 - Simulazione esame

Informatica@DSS 2019/2020 — (E-N)

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

Lunedì, 16 Dicembre 2019

Modalità di lavoro: queste è una simulazione della prova d'esame di programmazione. Il testo della prova, i file di test degli esercizi e una tavola sinottica sulle sintassi Python si trovano nella cartella `esame` sulla scrivania, nel sistema Debian Linux. Le soluzioni degli esercizi devono essere messe nella stessa cartella `esame`. **Per prima cosa compilate il file** `studente.txt` con le informazioni di nome, cognome e matricola, secondo la struttura

```
nome: Massimo
cognome: Lauria
matricola: 123456789
```

se non compilate il file `studente.txt` è come se non consegnaste il compito. Per l'esame potete avere al tavolo solo

- un documento;
- una bevanda e uno snack;

e dovete lasciare borse e zaini in un angolo della sala, con cellulari e dispositivi elettronici spenti al loro interno.

Il voto è la somma dei punti degli esercizi, pertanto è possibile avere 30 anche senza completarli tutti.

Uso dei file di test: i test di questa simulazione controllano molto sommariamente l'impostazione delle soluzioni e passare i test non implica affatto che la soluzione sia corretta. Leggete bene il testo degli esercizi prima di iniziare a risolverli. Per portare a termine ogni esercizio è necessario

*http://massimolauria.net/courses/informatica2019/

- scrivere un file di soluzione **col nome specificato**;
- avere dentro la funzione **col nome specificato**;
- porre il file di soluzione nella cartella `esame`;
- eseguire in quella cartella il comando

```
$ python3 <fileditest>
```

dove `<fileditest>` va ovviamente sostituito con il nome del file di test appropriato per l'esercizio su cui state lavorando.

Per ogni esercizio ci sta un file di test indipendente, così da poter lavorare sugli esercizi uno alla volta con più agio. Il risultato di ogni test è una schermata (o più schermate) nella quale si mostra:

- se è stato trovato il file con il nome corretto,
- se il file contiene la funzione con il nome corretto,
- se chiamate alla funzione con diversi valori dei parametri terminano restituendo risultati corretti.

Per ogni funzione scritta vengono eseguite chiamate con diversi valori dei parametri. L'esito dei test viene riportato con il carattere

- `E` se la chiamata non può essere eseguita,
- `F` se la funzione non restituisce il risultato corretto,
- `.` se la funzione restituisce il risultato corretto.

1 Inversione di una stringa (6 punti)

Scrivere una funzione `inversione_stringa(testo)` che, ricevuta come argomento una stringa, restituisca la sua inversione. Potete assumere che l'argomento sia sempre una stringa.

```
s = inversione_stringa("sveglia presto")
print(s)
```

```
'otserp ailgevs'
```

Il programma Python deve essere salvato nel file: `lab11inversione.py`

File di test: `test_lab11inversione.py`

2 Numeri compresi in intervallo (9 punti)

Scrivere una funzione `compresi(L,low,high)` che restituisca il numero di valori nella lista `L` che sono compresi tra `low` e `high` (estremi compresi). Potete assumere che gli argomenti siano di tipo corretto e che la lista contenga solo valori numerici.

```
seq = [3,5,1,8,4,2,5,7,6,3,4,1,9,8,4]
n = compresi(seq,3,5)
print(n)
```

```
7
```

Il programma Python deve essere salvato nel file: `lab11compresi.py`

File di test: `test_lab11compresi.py`

3 Verifica di ordinamento (5 punti)

Scrivere una funzione `ordinata(L)` che data una lista `L` come argomento, verifichi che `L` sia ordinata in modo crescente. Potete assumere che `L` sia costituita da tutti elementi confrontabili e che quindi la nozione di ordinamento abbia senso. Se `L` è ordinata in modo crescente allora la funzione deve restituire `True`, e deve restituire `False` in caso contrario.

```
seq1 = [3,5,1,8,4,2,5,7]
seq2 = ["delfino", "gatto", "renna", "tacchino"]
print(ordinata(seq1))
print(ordinata(seq2))
```

```
False
True
```

Il programma Python deve essere salvato nel file: `lab11ordinata.py`

File di test: `test_lab11ordinata.py`

4 Bordo della Matrice (4 punti)

Scrivere una funzione `bordo_zero(M)` che prenda come argomento una matrice rettangolare di dimensioni $r \times c$ con $r \geq 1$ e $c \geq 1$, che restituisca `True` se il bordo della matrice è costituito esclusivamente da zeri, e che restituisca `False` altrimenti. Potete assumere che `M` sia una lista di liste, tutte contenenti numeri, e della stessa lunghezza. Potete assumere che la matrice, così rappresentata, abbia almeno una riga e una colonna.

```
M1 = [[0]]
M2 = [[0,0,0,0], [0,7,8,0] , [0,0,0,0]]
print(bordo_zero(M1))
print(bordo_zero(M2))
```

```
True
True
```

Il programma Python deve essere salvato nel file: `lab11bordo.py`

File di test: `test_lab11bordo.py`

5 Coppia di distanza minima (5 punti)

Scrivere una funzione `distanza_minima(L)` che riceva una lista `L` contententente solo numeri, e che restituisca una lista contenente le due posizioni `[i,j]` dove (1) `i<j` e (2) `L[i]` e `L[j]` sono i due elementi della lista la cui differenza in valore assoluto è minima rispetto ad ogni altra coppia di elementi in `L`. Potete assumere che la lista `L`

- contenga solo numeri;
- abbia almeno due elementi;
- la coppia a distanza minima sia unica.

```
seq = [-5,4,8,-3,1,3,10]
print(distanza_minima(seq))
```

```
[1, 5]
```

Il programma Python deve essere salvato nel file: `lab11distanzaminima.py`

File di test: `test_lab11distanzaminima.py`

6 Coppie di righe uguali (6 punti)

Scrivere una funzione `righe_uguali(M)` che prenda come argomento una matrice rettangolare di dimensioni $r \times c$ con $r \geq 1$ e $c \geq 1$, e che restituisca un elenco di tutte le coppie di posizioni in cui si presentino righe identiche nella matrice. In sostanza il risultato deve essere

- una lista contenente liste di lunghezza due;

- ognuna di queste liste di lunghezza due è una coppia di indici `[i,j]` tali che le righe `i` e `j` della matrice `M` siano tra loro uguali;
- la lista deve elencare ogni coppia di righe uguali esattamente una volta;
- ogni coppia di indici `[i,j]` deve essere ordinata (ovvero `i<j`);
- la lista di coppie deve essere ordinata.

Potete assumere che `M` sia una lista di liste, tutte contenenti numeri, e con righe della stessa lunghezza. Potete assumere che la matrice, così rappresentata, abbia almeno una riga e una colonna.

```
M = [[6, 2, -5, -7, 4],
     [-6, 10, -4, 10, -2],
     [-4, 6, 1, -8, 1],
     [-4, 8, 7, 2, -7],
     [-6, 10, -4, 10, -2],
     [-4, 6, 1, -8, 1],
     [-3, 8, -5, -9, 7],
     [-4, 6, 1, -8, 1],
     [-3, -3, -9, -5, -9]]
lista= righe_uguali(M)
print(lista)
N = [[3,4]]
lista2 = righe_uguali(N)
print(lista2)
```

```
[[1, 4], [2, 5], [2, 7], [5, 7]]
[]
```

Il programma Python deve essere salvato nel file: `lab11righeuguali.py`

File di test: `test_lab11righeuguali.py`

7 Frequenza di parole (5 punti)

Scrivere una funzione `frequenzaparole(testo)` che dato come argomento un testo, restituisca un dizionario che contenga la frequenza delle parole presenti nel testo. Potere assumere che `testo` sia una stringa, che non contenga punteggiatura, e che tutti i caratteri alfabetici siano minuscoli.

```
testo="""la rana in spagna gracida
         in campagna quanto è felice la rana in spagna"""
F = frequenzaparole(testo)
print(F)
```

```
{'campagna': 1,
 'felice': 1,
 'gracida': 1,
```

```
'in': 3,
'la': 2,
'quanto': 1,
'rana': 2,
'spagna': 2,
'è': 1}
```

Il programma Python deve essere salvato nel file: `lab11frequenzaparole.py`

File di test: `test_lab11frequenzaparole.py`