Esercitazione di Laboratorio Informatica@DSS (2024/2025)

Massimo Lauria

Laboratorio 9 - 02/12/2024

Lo scopo del laboratorio è di esercitarsi e misurare la propria preparazione: gli esercizi non sono troppo difficili, se si sono seguite le lezioni. Non vi viene comunque messo alcun voto.

Modalità di lavoro: gli studenti devono lavorare in autonomia o in piccoli gruppi, senza disturbare i colleghi. Il lavoro di gruppo è fruttuoso solo se tutti partecipano e se ognuno scrive una propria soluzione per tutti gli esercizi.

Il docente cercherà per quanto possibile di non occupare il tempo del laboratorio per introdurre materiale nuovo, anche se a volte questo sarà necessario. Il docente è a disposizione per aiutare gli studenti, che possono iniziare a lavorare anche prima che il docente arrivi in aula, se lo desiderano

Raccomandazioni: leggete bene il testo degli esercizi prima di chiedere chiarimenti. In ogni caso sarò in aula con voi.

Uso dei file di test: per aiutarvi a completare questa esercitazione avete a disposizione dei programmi di test per testare la vostra soluzione. Questi sono simili a quelli che avrete in sede di esame, pertanto vi consiglio di imparate ad usarli. Per portare a termine l'esercizio è necessario

- scrivere un file di soluzione **col nome specificato**;
- avere dentro la funzione **col nome specificato**;
- porre il file di test corrispondente **nella stessa cartella**;
- 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 Restituire il valore associato alla massima chiave

Scrivete una funzione `dict_max_chiave(d)` che:

- si aspetti come argomento un dizionario;
- restituisca il valore associato alla massima chiave in `d`.

Si può assumere che il dizionario contenga almeno una chiave e che tutte le chiavi siano confrontabili tra loro.

Per ottenere la chiave massima si può utilizzare la funzione `max(....)`, che restituisce il massimo tra gli elementi dell'oggetto passato come parametro. Sperimentare l'uso della funzione `max(...)` dando come parametro o l'insieme delle chiavi del dizionario (metodo `.keys()`) oppure direttamente il dizionario.

Ad esempio:

```
diz = {'xx': 44, 'yy': 33, 'zz': 22}
print(dict_max_chiave(diz))
```

```
22
```

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

File di test: `test_dict_max_chiave.py`

2 Carica una lista da un file

Scrivere una funzione `caricalista(nomefile)` che riceva come argomento il nome di un file di testo, e che carichi da quel testo una lista di numeri interi. Per esempio consideriamo il file `caricalista10.txt` contenente

```
7 0 11 -4 5 6 7 10 -9 1
```

Dobbiamo ottenere questo risultato

```
caricalista('caricalista10.txt')
```

```
[7, 0, 11, -4, 5, 6, 7, 10, -9, 1]
```

Importante: i numeri nel file devono essere separati da spazi. Non devono essere necessariamente tutti sulla stessa riga, e tra loro ci possono essere arbitrariamente spazi, a capo e righe vuote. Ad esempio l'esempio precedente potrebbe essere scritto su file anche come

```
7
0    11   -4

5  6  7
10
-9 1
```

Suggeriamo, una volta ottenuto il testo contenuto nel file in forma di stringa, di usare semplicemente il metodo `.split()` per estrarre i dati.

Il programma deve sollevare i seguenti errori:

- `FileNotFoundError` se il file non esiste
- `ValueError` se ci sono sequenze di caratteri che non rappresentano numeri

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

File di test: `test_caricalista.py`

3 Carica Matrice da file

Scrivere una funzione `caricamatrice(nomefile)` che riceva come argomento il nome di un file di testo, e che carichi da quel testo il contenuto di una matrice di interi. La funzione deve interpretare il contenuto del file come una serie di numeri **interi**. I primi due numeri indicano il numero di righe e di colonne della matrice, e il resto dei numeri sono il contenuto delle celle. Ad esempio il file `caricamatrice3x5.txt` contenente

```
3 5

2 0 1 -4 5
6 7 10 -9 1
2 4 5 5 0
```

e rappresenta la matrice

$$\begin{bmatrix} 2 & 0 & 1 & -4 & 5 \\ 6 & 7 & 10 & -9 & 1 \\ 2 & 4 & 5 & 5 & 0 \end{bmatrix}$$

Quindi, per esempio,

```
caricamatrice('caricamatrice3x5.txt')
```

```
[[2, 0, 1, -4, 5], [6, 7, 10, -9, 1], [2, 4, 5, 5, 0]]
```

Importante: i numeri nel file devono essere separati da spazi. Non devono essere necessariamente su righe diverse, e tra loro ci possono essere arbitrariamente spazi, a capo e righe vuote. Ad esempio l'esempio precedente potrebbe essere scritto su file anche come

```
3 5 2 0 1 -4 5 6 7 10 -9 1 2 4 5 5 0
```

Suggeriamo, una volta ottenuto il testo contenuto nel file in forma di stringa, di usare semplicemente il metodo `.split()` per estrarre i dati.

Il formato **corretto** del file richiede che il file inizi con due numeri **interi positivi** R e C, e prosegua con $R \times C$ numeri interi arbitrari. Nell'esempio precedente il numero di righe è 3, il numero di colonne è 5 e quindi ci devono essere esattamente altri 15 numeri, nel file.

Per leggere una riga da un file si può utilizzare il metodo `f.readline()`, dove `f` è l'oggetto ottenuto con la funzione `open`.

Il programma deve sollevare i seguenti errori:

- `FileNotFoundError` se il file non esiste
- `ValueError` se il file non è nel formato corretto

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

File di test: `test_caricamatrice.py`

4 Bilancio finale

Scrivere una funzione `bilanciofinale(testo)` che riceva come argomento una striga di forma simile a questo esempio

```
esempio="""

Mario -100
Luigi 140
Daniela 200

Luigi -50

Daniela -30
Daniela 100 Mario 30

"""
```

che contiene una sequenza di coppie costituite ognuna da un nome e un valore numerico, positivo per le entrate e negativo per le uscite.

La funzione deve restituire un dizionario che associa ad ogni nome presente nel testo il suo bilancio finale (ovvero la somma di tutte le entrate e uscite che gli corrispondono). Ad esempio

```
bilanciofinale(esempio)
```

```
{'Mario': -70, 'Luigi': 90, 'Daniela': 270}
```

Notate il formato della stringa in input:

- un nome e il suo valore corrispondente sono separati da una quantità arbitraria (non vuota) di caratteri di spazio
- un valore ed il nome della coppia successiva sono separati da una quantità arbitraria (non vuota) di caratteri di spazio

Un carattere `c` è condiderato uno spazio se `c.isspace()` è `True`, ed in generale i caratteri di spazio sono , `\n`, `\t`.

Lo stesso esempio di prima è perfettamente equivalente se scritto nella stringa

```
Mario -100      Luigi

140 Daniela 200  Luigi -50
```

```
Daniela -30  Daniela 100 Mario
30
```

o su due righe

```
Mario -100 Luigi 140 Daniela 200  Luigi -50
Daniela -30  Daniela 100 Mario 30
```

Hint: per ottenere la sequenza di nomi e valore provate ad usare il metodo `split()`. La funzione deve sollevare `ValueError` se il testo non corrisponde al formato desiderato, ad esempio se manca un numero intero dove dovrebbe esserci, o se ad un nome non corrisponde un numero.

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

File di test: `test_bilanciofinale.py`

5 Bilancio finale da file

Scrivere una funzione `bilanciofinalefile(nomefile)` che riceva come argomento il nome di un file di testo, e sul testo contenuto in quel file si comporti esattamente come la funzione `bilanciofinale` dell'esercizio corrispondente. Consideriamo ad esempio il file `bilanciofinalefile1.txt` che contiene il testo

```
Mario -100
Luigi 140
Daniela 200
Luigi -50
Giacomo 80
Daniela -30
Daniela 100
Mario 30
Giacomo -40
```

allora

```
bilanciofinalefile('bilanciofinalefile1.txt')
```

```
{'Mario': -70, 'Luigi': 90, 'Daniela': 270, 'Giacomo': 40}
```

La funzione deve restituire un dizionario nello stesso modo di `bilanciofinale`, e deve sollevare `ValueError` se il testo non è adeguato (leggete la descrizione in quell'esercizio). In più `bilanciofinalefile` deve sollevare `FileNotFoundError` se il file non esite.

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

File di test: `test_bilanciofinalefile.py`

6 Somme per colonna di una matrice numerica

Scrivere una funzione `somme_per_colonna_mat(m)` che:

- riceva come parametro una matrice `m`;
- restituisca una lista in cui l'elemento di posto `i` è la somma degli elementi della colonna `i`-esima. La lunghezza di tale lista è uguale al numero di colonne della matrice.

Ricordiamo che in Python noi rappresentiamo una matrice $r \times c$ come una lista di lunghezza r, contenente r liste tutte quante di lunghezza c. Potete sempre assumere che l'input di questo esercizio sia una matrice ben formata, con $r \geq 1$ e $c \geq 1$, e che tutte le liste che rappresentano le righe contengano c valori numerici.

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

File di test: `test_somme_per_colonna_mat.py`