Esercitazione Python n. 11 -- 14 Dicembre 2021

Per risolvere gli esercizi in modo che possano essere successivamente corretti **è necessario scrivere la soluzione di ogni esercizio nel file .py relativo**, che trovate nella cartella dell'esercitazione (ad esempio, per l'esercizio 1 scrivete il vostro programma nel file Ex1.py, per l'esercizio 2, nel file Ex2.py, e così via). Notate che ogni file incorpora del codice python per eseguire alcuni test sulla funzione. NON modificate questo codice, ma **SCRIVETE SOLO il contenuto della funzione**. In fase di correzione verranno eseguiti dei test diversi da quelli che trovate attualmente.

Per **consegnare l'esercitazione svolta**, comprimete la cartella LabPython11 (**FATE ATTENZIONE**, la cartella che comprimete deve contenere gli esercizi e NON una ulteriore cartella con dentro gli esercizi) **in un file .zip** e caricatela sulla pagina del corso di https://classroom.google.com/u/0/w/MzkwNTM3Njg2Njc3/t/all (dalla sezione 'Esercitazioni su Python', selezionate 'Esercitazione 11' e successivamente 'Visualizza Compito'; poi cliccate su 'Aggiungi o crea' e scegliete il file da caricare). NON è necessario rinominare il file .zip. E' NECESSARIO NON rinominare i singoli file. Al termine dell'operazione cliccate su 'Contrassegna come completato'. La consegna deve avvenire in maniera inderogabile entro le **23:59 di Mercoledì 15 dicembre**.

A valle dell'esercitazione provvederemo ad effettuare una correzione automatica delle soluzioni consegnate, il cui esito verrà comunicato ad ogni studente tramite un messaggio di posta privato.

Si rammenta che **le esercitazioni consegnate in ritardo o che non rispettano le indicazioni per la consegna saranno automaticamente valutate con punteggio 0 e non contribuiranno all'assegnazione del punto bonus**. In particolare, si ricorda di **comprimere e riconsegnare l'intera cartella dell'esercitazione**, e non singolarmente i file degli esercizi, di **NON usare formati di compressione diversi da .zip**, di **NON rinominare i file** o metterli in sottocartelle.

In ogni esercizio, se non diversamente richiesto, potete sempre assumere che gli input forniti siano coerenti con la traccia (ad esempio, se l'esercizio chiede di dare in input alla funzione un intero positivo, potete assumere che l'input sia sempre un numero intero maggiore di zero, e non è necessario nel codice effettuare controlli per gestire casi diversi da questo).

Per gli esercizi relativi a lettura da file, la stringa in input che identifica il file è sempre comprensiva anche della sua estensione e il file risiede sempre nella stessa directory dell'esercizio.

Esercizi

- **Ex1(m):** scrivere una funzione che riceve in ingresso una matrice NON VUOTA **m** di 0 ed 1, rappresentata come lista di liste, e restituisce la lunghezza della più lunga sequenza orizzontale (per riga) consecutiva di 0. Ad esempio, se la matrice **m** vale `[[1,0,0,1,0],[0,0,0,1,0],[1,0,0,0,0],[1,1,1,0,0]]`, cioè corrisponde alla matrice

```
10010
00010
10000
11100
```

  allora la funzione deve restituire 4, poiché nella riga di indice 2 (terza riga) ci sono quattro 0 consecutivi.

- **Ex2(file):** scrivere una funzione Python che, preso in ingresso il nome di un **file** di testo, calcoli, usando le espressioni regolari, quante sono le sequenze non sovrapposte di 2 parole consecutive con la seguente proprietà:

"Almeno 2 lettere della prima parola sono presenti anche nella seconda ma in ordine inverso, cioè se la prima lettera viene prima della seconda nella prima parola, deve venire dopo nella seconda."

Ignorate la differenza fra maiuscole e minuscole. Ad esempio, prendendo come input il file contenente il seguente testo:

```
tanto va Aldo destinando in giro che era arrestato casa propria
```

la funzione deve restituire come risultato `2`.

Suggerimenti: Usate il flag `re.IGNORECASE` per non fare differenza tra maiuscole e minuscole. Potete usare le funzioni `re.finditer()` o `re.findall()` a vostra scelta. Se usate la funzione `re.findall()`, per contare il numero di soluzioni trovate basta usare la funzione `len()` applicata al risultato della `re.findall()`. Si ricorda che una parola è una sequenza di caratteri alfanumerici più l'underscore, preceduta e seguita da almeno un carattere non alfanumerico e non underscore (se però la parola è all'inizio del file è solo seguita da questo tipo di caratteri, mentre se è alla fine del file è solo preceduta da questi).

- **Ex3(file):** scrivere la funzione Python che, preso in ingresso il nome di un **file** contenente una matrice nel formato:

```
Numero_righe Numero_colonne
Prima riga con valori separati da spazi
…
Ultima riga con valori separati da spazi
```

calcoli l'indice della riga con il numero maggiore di valori (strettamente) negativi. Se ci sono più righe con lo stesso numero di valori negativi, restituire l'indice più grande. Ad esempio, se il file contiene:

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

la funzione deve restituire `3`, poiché le righe di indice 0, 2 e 3 hanno tutte 1 valore negativo, ma 3 è più grande di 0 e 2.

- **Ex4(file)**: Scrivere una funzione Python che prende in ingresso il nome di un **file** csv contenente tutte le eredità di una famiglia nel seguente formato:

```
Oggetto,Antenato,Erede
```

Assumete che il proprietario dell'oggetto sia l'antenato che compare la prima volta (dall'inizio del file) assieme all'oggetto e che se l'antenato NON ha l'oggetto allora l'erede NON lo riceve. Assumete inoltre che l'ordine delle righe conti: in una riga, l'antenato A possiede un oggetto solo se A è il proprietario, oppure se esiste una riga precedente in cui A riceve l'oggetto da un antenato che ha l'oggetto.

La funzione deve leggere il **file** e restituire il dizionario con chiavi i nomi degli oggetti nel **file** e come valore associato a ciascuna chiave *k* una lista contenente due nomi, il nome del proprietario e il nome dell'ultimo erede che ha ricevuto l'oggetto *k*. Ad esempio se il **file** contiene:

```
Oggetto,Antenato,Erede
Anello_di_smeraldi,Maria,Paola
Anello,Silvia,Paolo
Anello_di_smeraldi,Paola,Anna
Anello_di_smeraldi,Anna,Giorgia
```

la funzione deve restituire:

```
{'Anello_di_smeraldi': ['Maria','Giorgia'], 'Anello': ['Silvia',
'Paolo']}
```

Invece se il **file** contiene:

```
Oggetto,Antenato,Erede
Anello_di_smeraldi,Maria,Paola
Anello,Silvia,Paolo
Anello_di_smeraldi,Anna,Giorgia
Anello_di_smeraldi,Paola,Anna
```

la funzione deve restituire:

`{'Anello_di_smeraldi':['Maria','Anna'],'Anello':['Silvia','Paolo']}`
(infatti, alla riga `Anello_di_smeraldi,Anna,Giorgia` Anna non ha l'oggetto, perché non lo ha ricevuto in una riga precedente).

- **Ex5(file)**: un quadrato latino è una scacchiera quadrata con un simbolo su ogni casella, in modo che *ognuno di essi* compaia una e una sola volta in ogni riga ed in ogni colonna.

A	B	C	D	E
B	C	E	A	D
C	E	D	B	A
D	A	B	E	C
E	D	A	C	B

Scrivere una funzione Python che prende in ingresso un file **file** nel seguente formato:

ABCDE
BCEAD
CEDBA
DABEC
EDACB

e restituisce il valore booleano `True` se esso rappresenta un quadrato latino, `False` altrimenti.

- **Ex6(m)** Scrivere una funzione che riceve in ingresso una matrice NON VUOTA **m** di numeri interi, rappresentata come lista di liste, e restituisce il NUMERO DI ELEMENTI della matrice che sono **minimi locali**, cioè tali che tutti gli elementi che gli stanno vicino (a distanza 1 orizzontalmente, verticalmente o obliquamente) sono maggiori od uguali a loro. Assumete che solo elementi che NON sono sui bordi possono essere minimi locali. Ad esempio, se **m** vale `[[3,2,0,1],[4,1,2,1],[2,1,3,1],[2,2,2,4],[3,4,2,2]]`, cioè corrisponde alla matrice:

```
3 2 0 1
4 1 2 1
2 1 3 1
2 2 2 4
3 4 2 2
```

La funzione deve restituire 1 poiché solo l'elemento in grassetto è un punto di minimo locale.