Sapienza Università di Roma
Corso di Laurea in Ingegneria Informatica e Automatica A.A. 2021-22

Esercitazione Python n. 10 -- 7 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 LabPython10 (**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 10' 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 9 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(file)**: scrivere la 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 aventi la seguente proprietà:

  "La due parole sono composte da almeno due caratteri ed hanno a stessa lettera iniziale e la stessa lettera finale, ignorando la distinzione fra maiuscole e minuscole."

  Ad esempio, prendendo come input il file contenente il seguente testo:

  `tanto va **A**ldin**o** **a**nnaspand**o** in giro che era andato al bar`

  la funzione deve restituire come risultato `1`.

  Se invece la funzione prende in input

  `Ho trovato delle **o**ss**a** **o**rs**a** ora **v**ad**o** **v**elat**o**`

deve restituire come risultato `2`, infatti le due sequenze sono `ossa orsa` e `vado velato`, mentre `orsa ora` non va bene perché si sovrappone con la prima.

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).

- **Ex2(file):** scrivere la funzione Python che, preso in ingresso il nome di un **file** di testo calcoli, usando le espressioni regolari, quante volte compaiono in una stessa riga tre parole consecutive (senza sovrapposizioni) tutte con la stessa doppia, ignorando la differenza fra maiuscole e minuscole.

  Ad esempio, prendendo come input il file contenente il seguente testo:

  ```
  va Aldo oziando dalla stalla alla casa
  chiedendo solo di andare via da casa
  ```

  la funzione deve restituire come risultato `1`.

  Adottate le stesse assunzioni descritte per l'esercizio Ex1 e seguite gli stessi suggerimenti.

- **Ex3(file)**: scrivere una funzione Python che, preso in ingresso il nome di un **file** di testo calcoli, usando le espressioni regolari, quante volte compare una parola con la seguente proprietà:

  "la lettera iniziale e finale della parola sono uguali ed
  all'interno della parola compare almeno una doppia."

  Si noti che la doppia deve comparire all'interno della parola, senza quindi considerare il primo e l'ultimo carattere della parola stessa. Ad esempio, prendendo come input il file contenente il seguente testo:

  ```
  tanto attacca contro elettore in giro abbastanza era andato a casa
  ```

  la funzione deve restituire come risultato `3`.

  Adottate le stesse assunzioni descritte per l'esercizio Ex1 e seguite gli stessi suggerimenti.

- **Ex4(file):** scrivere una funzione Python che prende in ingresso il nome di un **file** csv contenente le informazioni sulle amicizie e inimicizie che si creano in un gruppo di persone nel seguente formato:

  ```
  Nome1, Nome2, relazione
  ```

  dove la relazione può essere solo un valore tra `'amici'` e `'nemici'`. La relazione è sempre simmetrica ed una relazione riportata ad una certa riga può essere modificata da una relazione indicata in una riga successiva. Ad esempio, se **file** contiene:

  ```
  Nome1, Nome2, relazione
  Paolo, Marco, amici
  Anna, Maria, amici
  Paola, Anna, amici
  Marco, Giorgio, amici
  Giorgio, Marco, nemici
  ```

  la riga 2 dice che Paolo è amico di Marco, e vice-versa (analogamente le altre righe), mentre notiamo che alla riga 5 Marco e Giorgio sono amici, ma alla riga 6 diventano nemici.

La funzione deve leggere il **file** e costruire un dizionario avente come chiavi i nomi di tutte le persone che compaiono nel file, e per valore associato a ciascuna chiave *k* la lista ordinata in ordine lessicografico crescente degli amici di *k* (rimasti al termine della lista). Ogni volta in cui 2 persone diventano amiche dovete aggiungere il nome di ciascuno dei due alla lista degli amici dell'altro, se non era già presente (non ci devono essere duplicati nella lista). Se due persone diventano nemiche dovete eliminare il nome di ciascuno dei due dalla lista degli amici dell'altro (se c'era, altrimenti non dovete fare niente). In riferimento al file d'esempio precedente, la funzione deve restituire il seguente dizionario:

```
{'Marco': ['Paolo'], 'Maria': ['Anna'], 'Paolo': ['Marco'], 'Paola':
['Anna'], 'Giorgio': [], 'Anna': ['Maria', 'Paola']}
```

- **Ex5(file)**: un indirizzo IP è un'etichetta numerica che identifica univocamente un dispositivo all'interno di una rete informatica. Esso è costituito da una sequenza di 4 numeri compresi tra 0 e 255, formati da una, due o tre cifre e separati da un punto, (es. 192.168.0.1). Tra tutti i possibili indirizzi IP, vi è un intervallo dedicato alle reti domestiche, i quali hanno formato 192.168.X.Y, dove X ed Y sono due numeri compresi tra 0 e 255. Scrivere la funzione Python che preso in ingresso il nome di un **file** di testo avente il seguente formato

```
Indirizzo_IP
Indirizzo_IP
    […]
Indirizzo_IP
```

  e restituisca un dizionario contenente le seguenti chiavi: 'invalidi', 'domestici' ed 'altri', e come valori associati il numero di indirizzi letti dal file che sono rispettivamente non validi, domestici oppure validi non domestici.

- **Ex6(file):** scrivere una funzione Python che prende in ingresso un file di testo contenente dei codici fiscali, scritti uno per riga e che possono contenere o meno spazi tra i vari campi, e restituisce la lista (nell'ordine in cui sono nel file) delle date di nascita nel formato dd/mm/aaaa (2 cifre obbligatorie per giorno e mese, 4 per anno). Si assuma che se l'anno xx nel codice fiscale è minore o uguale a 20 allora l'anno corrisponde a 20xx, altrimenti corrisponde a 19xx. Si ricorda che il formato dei codici fiscali è:

  ABC XYZ aaMgg WnnnV

  Dove ABC e XYZ sono sequenze di lettere MAIUSCOLE prese rispettivamente dal cognome e dal nome, aa denota le ultime due cifre dell'anno, M è una lettera maiuscola che specifica il mese secondo la seguente tabella riportata a lato, gg denota il giorno di nascita con la regola che se è maggiore di 40 allora il sesso è femminile e per calcolare la data corretta bisogna togliere 40. L'ultima parte indica il codice del comune (o stato estero) di nascita, composto da una lettera maiuscola e 3 cifre, mentre l'ultima lettera maiuscola è un carattere di controllo. I 4 campi possono essere separati da spazi bianchi oppure essere attaccati. Se la riga NON contiene un codice fiscale corretto dovete inserire nella lista la stringa 'Codice errato', se il codice del mese è inesistente allora inserite nella lista la stringa 'Mese errato', se il giorno è scorretto allora inserite nella lista la stringa 'Giorno errato'. Nella verifica dei giorni potete ignorare gli anni bisestili ed assumere che Febbraio abbia sempre 28 giorni. Ad esempio, se il file contiene

Lettera	Mese	Lettera	Mese	Lettera	Mese
A	gennaio	E	maggio	P	settembre
B	febbraio	H	giugno	R	ottobre
C	marzo	L	luglio	S	novembre
D	aprile	M	agosto	T	dicembre

  VXRTRR71C12H501W
  PSCTRS 21S33 P
  CVV PSX 11D55 H911T
  CVV PSX 11O55 H911T
  CVV PSX 11D79 H911T

  la funzione deve restituire ['12/03/1971', 'Codice errato', '15/04/2011', 'Mese errato', 'Giorno errato'].