Esercitazione 9 - Laboratorio

Informatica@DSS 2022/2023

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

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.

*http://massimolauria.net/informatica2022/

Per ogni esercizio ci sta un file di test indipendente, così da poter lavorare sugli esercizi uno alla volta con più agio. Controllate i parametri passati in input ed eventualmente sollevate le eccezioni `ValueError` o `TypeError`.

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 Sostituzione dei caratteri non alfabetici

Scrivere una funzione `rimuovi_nonalfabetici(frase)` che data una stringa di caratteri restituisce una stringa di caratteri. La stringa restituita contiene la stessa sequenza, ma ciascun carattere non alfabetico deve essere sostituito da uno spazio. Ad esempio, se il parametro è la stringa

```
"Buongiorno, vengo dall'Umbria."
```

deve essere restituita la stringa

```
"Buongiorno  vengo dall Umbria "
```

Notare la presenza di un elemento `" "` in corrispondenza dei caratteri virgola, apostrofo e punto.

Per verificare se un carattere contenuto in una variabile, per esempio di nome `pippo`, è una lettera si può usare il metodo `pippo.isalpha()`, che restituisce `True` oppure `False`.

La funzione deve sollevare una eccezione `TypeError` se il parametro ricevuto non è una stringa.

File della soluzione: `rimuovi_nonalfabetici.py`

File di test: `test_rimuovi_nonalfabetici.py`

2 Conversione in minuscolo

Scrivere una funzione `minuscolo(frase)` che, che data una stringa di caratteri che contiene esclusivamente lettere maiuscole, minuscole e spazi, restituisce una stringa nella quale per ciascun carattere maiuscolo viene inserito il corrispondente minuscolo. Ad esempio, se il parametro è la stringa

```
"Buongiorno  vengo dall Umbria "
```

deve essere restituire la stringa

```
"buongiorno  vengo dall umbria "
```

Per verificare se un carattere (contenuto in una variabile p.es. di nome pippo) è una lettera maiuscola si possono utilizzare condizioni del tipo

```
pippo >= "A"
```

e confrontando naturalmente anche con `"Z"`.

Per ottenere la versione minuscola di un carattere si può usare il metodo `.lower()`, che restituisce la versione minuscola di un carattere.

La funzione deve sollevare una eccezione `TypeError` se il parametro ricevuto non è una stringa.

File della soluzione: `minuscolo.py`

File di test: `test_minuscolo.py`

3 Convertire una stringa in una lista di parole

Scrivere una funzione `spezza_parole(frase)` che data una stringa restituisce una lista formata da un elemento per ciascuna parola nella stringa. Ad esempio, se il parametro è la stringa

```
"buongiorno  vengo dall umbria "
```

deve essere restituita la lista

```
["buongiorno", "vengo", "dall", "umbria"]
```

Si può assumere che la stringa fornita come parametro sia composta esclusivamente da caratteri minuscoli e spazi.

Le stringhe possiedono un metodo `.split()`. Verificarne il funzionamento dall'interprete Python provando ad eseguire ad esempio le istruzioni

```
s = "pippo  pluto    topolino"
print(s.split())
```

```
['pippo', 'pluto', 'topolino']
```

La funzione deve sollevare una eccezione `TypeError` se il parametro ricevuto non è una stringa.

File della soluzione: `spezza_parole.py`

File di test: `test_spezza_parole.py`

4 Ordinare una lista di parole

Scrivere una funzione `parole_ordinate(lista_parole)` che data una lista di parole ne restituisce una copia ordinata. Ad esempio, se il parametro è la lista

```
["buongiorno", "vengo", "dall", "umbria", "e", "in", "umbria", "torno"]
```

allora deve essere restituita la lista

```
["buongiorno", "dall", "e", "in", "torno", "umbria", "umbria", "vengo"]
```

Per ordinare la lista si può utilizzare uno qualunque degli algoritmi di ordinamento visti a lezione, oppure la funzione `sorted()`. Ad esempio, avendo una lista `pippo`, la lista `sorted(pippo)` sarà ordinata.

La funzione deve sollevare una eccezione `TypeError` se il parametro ricevuto non è una lista.

File della soluzione: `parole_ordinate.py`

File di test: `test_parole_ordinate.py`

5 Costruire una lista eliminando i duplicati

Scrivere una funzione `elimina_duplicati(lista)` che data una lista **ordinata** restituisce un'altra lista che contiene gli stessi elementi, ma senza

duplicati. Poiché la lista è ordinata, eventuali parole duplicate compaiono in posizioni consecutive. Ad esempio, se il parametro è la lista

```
["buongiorno", "dall", "e", "e", "in", "torno", "umbria", "umbria", "vengo"]
```

deve essere restituita la lista

```
["buongiorno", "dall", "e", "in", "torno", "umbria", "vengo"]
```

La funzione deve sollevare una eccezione `TypeError` se il parametro ricevuto non è una lista.

La funzione deve sollevare una eccezione `ValueError` se il parametro ricevuto non è una lista ordinata in senso crescente.

File della soluzione: `elimina_duplicati.py`

File di test: `test_elimina_duplicati.py`

6 Elenco di parole in un testo

Per risolvere questo esercizio è essenziale usare tutte le funzioni definite nei punti precedenti di questa sezione.

Scrivere una funzione `estrai_parole(frase)` che data una stringa costruisce la lista delle parole presenti, senza duplicati, in ordine alfabetico, considerando uguali caratteri minuscoli e maiuscoli.

Ad esempio, se il parametro è la stringa

```
"Pippo, Pluto e Topolino incontrano Minnie. E lei li ignora, come se
non li avesse visti!"
```

deve essere restituita la lista

```
["avesse", "come", "e", "ignora", "incontrano", "lei",
 "li", "minnie", "non", "pippo", "pluto", "se", "topolino", "visti"]
```

File della soluzione: `estrai_parole.py`

File di test: `test_estrai_parole.py`