Esercitazione 6 - Laboratorio

Informatica@DSS 2022/2023

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

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>
```

*http://massimolauria.net/informatica2022/

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. 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 Lista di numeri (prima versione)

Scrivere una funzione `crealista1(a, b)` che:

- si aspetti come argomenti due numeri `a` e `b`, con `b` naturale;
- sollevi un'eccezione `TypeError` se i parametri ricevuti non sono numeri;
- sollevi un'eccezione `ValueError` se il secondo parametro è un intero negativo;
- crei e restituisca una lista contenente i primi `b` interi a partire da `a`. Ad esempio, l'istruzione `s = crealista1(14.3, 5)` deve assegnare a `s` la lista `[15, 16, 17, 18, 19]`.

File della soluzione: `crealista1.py`

File di test: `test_crealista1.py`

2 Lista di numeri (seconda versione)

Scrivere una funzione `crealista2(a, b, passo=1)`, con un parametro opzionale, che si comporti come la funzione al punto precedente nel caso venga invocata con due argomenti, mentre nel caso di invocazione con tre argomenti crei e restituisca una lista di `b` interi a partire dal primo intero maggiore di `a` con passo uguale al terzo argomento. Ad esempio, l'istruzione `s = crealista2(14.6, 5, 3)` deve assegnare a `s` la lista `[15, 18, 21, 24, 27]`

Anche in questo caso si deve sollevare una eccezione `ValueError` se il secondo argomento ricevuto è un numero intero negativo. Il terzo parametro può essere un intero qualunque. Ad esempio `crealista2(14.6, 5, -3)` deve restituire la lista `[15, 12, 9, 6, 3]`.

File della soluzione: `crealista2.py`

File di test: `test_crealista2.py`

3 Azzerare elementi negativi

Scrivere una funzione `azzeranegativi(valori)` che:

- si aspetti come argomento una lista valori, in cui ciascun elemento è un numero (`int` oppure `float`);
- sollevi una eccezione `TypeError` se il parametro ricevuto non è una lista, o se gli elementi della lista non sono tutti numeri;
- modifichi la lista ricevuta, azzerando tutti gli elementi negativi.

La funzione non restituisce nulla (quindi non contiene alcuna `return` oppure contiene solo dei `return None` o senza alcuna espressione), ma si limita a lasciare la lista modificata.

File della soluzione: `azzeranegativi.py`

File di test: `test_azzeranegativi.py`

4 Posizione del minimo

Scrivere una funzione `minimo(valori, start=0)` che:

- si aspetti come argomento una lista valori;

- sollevi un'eccezione `TypeError` se il parametro ricevuto non è una lista;
- sollevi un'eccezione `ValueError` se il parametro ricevuto è una lista vuota;
- sollevi un'eccezione `TypeError` se il parametro ricevuto contiene elementi non numerici;
- sollevi un' eccezione `ValueError` se il parametro `start` non è un indice corretto per la sequenza valori;
- restituisca la posizione del valore minimo tra gli elementi della lista che compaiono a partire dalla posizione `start`. Il parametro `start` è opzionale, e vale 0 se non viene specificato.

File della soluzione: `minimo.py`

File di test: `test_minimo.py`

5 Massimi locali

Scrivere una funzione `massimilocali(L)` che:

- si aspetti come argomento una lista `L`;
- sollevi un'eccezione `TypeError` se il parametro ricevuto non è una lista;
- sollevi un'eccezione `TypeError` se il parametro ricevuto contiene elementi non numerici;
- restituisca una lista contenente i massimi locali presenti in sequenza.

Un elemento di una sequenza è un massimo locale se:

- non si trova ad un estremo della lista;
- è preceduto e seguito da elementi strettamente minori dell'elemento stesso.

File della soluzione: `massimilocali.py`

File di test: `test_massimilocali.py`

6 Verificare se una sequenza è bitonica

Scrivere una funzione `bitonica(L)` che:

- si aspetti come argomento una lista `L`;
- sollevi un'eccezione `TypeError` se il parametro ricevuto non è una lista;
- sollevi un'eccezione `TypeError` se il parametro ricevuto contiene elementi non numerici;
- restituisce `True` se la sequenza è **bitonica**, e `False` altrimenti.

Una sequenza è bitonica se è composta da una prima parte strettamente crescente, composta da almeno due elementi, seguita da una parte strettamente decrescente, anche questa composta da almeno due elementi. Ad esempio sono bitoniche le seguenti sequenze:

- `2 4 13 34 23`
- `3 4 1`
- `27 48 113 134 23 12`

File della soluzione: `bitonica.py`

File di test: `test_bitonica.py`