

Esercitazione di Laboratorio Informatica@DSS (2026/2027)

Massimo Lauria

Laboratorio 6 - 2/11/2026

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 <filedittest>`

dove `<filedittest>` 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 Somma di liste

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

- si aspetti come argomenti due liste `a`, `b` della stessa lunghezza, in cui ciascun elemento è un numero `int`;
- crei e restituisca una lista, della stessa lunghezza di quelle ricevute come argomenti, in cui l'elemento `i`-esimo della lista è uguale alla somma dell'elemento `i`-esimo di `a` e l'elemento `i`-esimo di `b`.

Ad esempio la somma delle liste `[3, 5, 1]` e `[1, -2, 4]` è `[4, 3, 5]`.

Il vostro programma può **assumere** che le due liste in input abbiano la stessa lunghezza e siano costituite da numeri interi

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

File di test: `test_sommaliste.py`

2 Prodotto Scalare tra due vettori

Il prodotto scalare tra due vettori $\langle a_0, a_1, \dots, a_{n-1} \rangle$ e $\langle b_0, b_1, \dots, b_{n-1} \rangle$ è il valore

$$\sum_{i=0}^{n-1} a_i b_i$$

Le liste sono ovviamente un modo naturale per rappresentare il concetto matematico di vettore. Scrivere una funzione `prodottoscalare(a, b)` che:

- riceva come parametri due liste non vuote `a`, `b` della stessa lunghezza, in cui ciascun elemento è un numero (`int` oppure `float`);
- restituisca il prodotto scalare delle due liste. Il prodotto

scalare tra due liste è quindi la somma dei prodotti degli elementi omologhi. Ad esempio, il prodotto scalare tra i vettori `[1, 4, 3]` e `[5, 5, 1]` è $1 \times 5 + 4 \times 5 + 3 \times 1 = 28$. Il prodotto scalare di due vettori è un valore numerico. Osservate che il prodotto scalare di due liste vuote è 0.

Potete **assumere** che le liste non siano vuote, che contengano solo numeri e che abbiano la stessa lunghezza.

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

File di test: `test_prodottoscalare.py`

3 Media aritmetica

Scrivere una funzione `media(L)` che,

- si aspetti come argomento una lista `L`, in cui ciascun elemento è un numero (`int` oppure `float`);
- restituisca la media aritmetica dei numeri contenuti nella lista `L`, scritta con al massimo 5 cifre decimali di approssimazione.

Nota: per troncare un numero a 5 cifre decimali potete usare la funzione `round`.

Potete assumere che la lista `L` contenga esclusivamente elementi numerici (`int` oppure `float`).

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

File di test: `test_media.py`

4 Separa elementi in base alla media

Scrivere una funzione `separaelementi(valori)` che:

- si aspetti come argomento una lista `valori`, in cui ciascun elemento è un numero (`int` oppure `float`);
- restituisca due liste (notare che è ammessa l'istruzione `return lista1, lista2` per restituire due liste), dove `lista1` deve contenere tutti gli elementi di `valori` che sono maggiori della media degli elementi in `valori`, mentre `lista2` deve contenere tutti gli elementi che sono minori o uguali alla media degli elementi in `valori`.

Ad esempio, se la lista fosse `[0.96, 0.2, 0.4, 0.1, 0.55, 0.03, 0.88]`, poiché la media degli elementi è `0.4457`, si dovrebbero restituire le liste `[0.96, 0.55, 0.88]` e `[0.2, 0.4, 0.1, 0.03]`.

All'interno di ognuna delle due liste gli elementi devono comparire nello stesso ordine in cui compaiono nella lista originale.

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

File di test: `test_separaelementi.py`

5 Intersezione

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

- si aspetti come argomenti due liste `a`, `b`;
- restituisca una lista **ordinata** che rappresenti l'intersezione tra le due liste, cioè gli elementi che appartengono sia alla lista `a` che alla lista `b`.
Si può assumere che `a` e `b` non contengano rispettivamente doppioni.

Ad esempio

```
L = intersezione([2,4,3,1],[4,1,5])  
print(L)
```

1
2

[1, 4]

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

File di test: `test_intersezione.py`

6 Unione

Scrivere una funzione `unione(a, b)` che si aspetti come argomenti due liste `a`, `b` e restituisca una lista **ordinata** che rappresenti l'unione tra le due liste `a` e `b`, cioè gli elementi che appartengono alla lista `a` e/o alla lista `b`.

Si può assumere che `a` e `b` non contengano rispettivamente doppioni. Tuttavia un elemento potrebbe apparire sia in `a` che in `b`, e deve apparire una volta sola nella loro unione. Per ordinare la risposta è sufficiente usare la funzione `sorted` o il metodo `sort`.

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

File di test: `test_unione.py`