

# Prova di Informatica@DSS (2023/2024)

Donatella Firmani      Paolo Franciosa      Massimo Lauria

## Simulazione Esame

**Modalità di lavoro:** Il testo della prova, i file di test degli esercizi e una tavola sinottica sulle sintassi Python si trovano nella cartella **esame** sulla scrivania, nel sistema “Statistica-Laborario”. Le soluzioni degli esercizi devono essere messe nella stessa cartella **esame**. **Per prima cosa compilate il file `studente.txt`** con le informazioni di nome, cognome e matricola, secondo la struttura

```
nome: Giacomo  
cognome: Leopardi  
matricola: 123456789
```

Non compilare il file **`studente.txt`** è come non consegnare il compito. Per l'esame potete avere al tavolo solo

- un documento;
- una bevanda e uno snack;

e dovete lasciare borse e zaini in un angolo della sala, con cellulari e dispositivi elettronici spenti al loro interno.

**Il voto è la somma dei punti degli esercizi**, pertanto è possibile avere 30 anche senza completarli tutti.

**Uso dei file di test:** i test di questa simulazione controllano molto sommariamente l'impostazione delle soluzioni e passare i test non implica affatto che la soluzione sia corretta. Leggete bene il testo degli esercizi prima di iniziare a risolverli. Per portare a termine ogni esercizio è necessario

- scrivere un file di soluzione **col nome specificato**;
- avere dentro la funzione **col nome specificato**;
- porre il file di soluzione nella cartella **esame**;
- eseguire in quella cartella il comando `python3 <fileeditest>`

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 dei valori grandi in un dizionario (7 punti)

Scrivete una funzione `sommavalorigrandi(D,soglia)` che accetti come argomento un dizionario e un valore numerico, e che calcoli la somma di tutti i valori corrispondenti ad una chiave del dizionario, che siano strettamente maggiori della soglia indicata da secondo valore.

Ad esempio:

```
diz = {'xx': 44, 'yy': 33, 'zz': 22, 'qw': 22, 'das': 11}      1
print(sommavalorigrandi(diz,20))                             2
print(sommavalorigrandi(diz,25))                             3
print(sommavalorigrandi(diz,120))                             4
```

```
121
77
0
```

La funzione può **assumere** che il primo parametro `diz` sia un dizionario che fa corrispondere alle chiavi solamente valori numerici, e che il secondo parametro `soglia` sia un valore numerico. Pertanto non dovete preoccuparvi di controllare queste condizioni nella soluzione del vostro esercizio.

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

File di test: `test_sommavalorigrandi.py`

## 2 Numeri compresi in intervallo (7 punti)

Scrivere una funzione `seq_compresi(L,low,high)` che restituisca il numero di valori nella lista `L` che sono compresi tra `low` e `high` (estremi inclusi). Potete assumere che gli argomenti siano di tipo corretto, cioè una lista di numeri e due numeri.

Perciò **non è necessario** che verifichiate queste condizioni o che solleviate errori a riguardo.

```
seq = [3,5,1,8,4,2,5,7,6,3,4,1,9,8,4] 1
n = seq_compresi(seq,3,5) 2
print(n) 3
```

7

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

File di test: `test_seq_compresi.py`

### 3 Ciclo for e range (4 punti)

Quale sarà il valore della variabile `somma` dopo l'esecuzione del ciclo `for`

```
somma=0 1
L = [1, 1, 2, 2, 7, 7, -3, -3, 6, 6] 2
for i in range(3,9): 3
    somma += L[i] 4
5
6
```

Scrivete la vostra risposta sul foglio dedicato che vi è stato fornito.

## 4 Trova riga massima in file (5 punti)

Scrivere una funzione `riga_max_file(nomefile)` che riceva come argomento il nome di un file di testo contenente in ciascuna riga una serie di interi separati da spazio, e restituisca la posizione della riga con somma massima. Le posizioni di riga si devono contare a partire da 0. Nel caso di più righe con somma massima deve essere restituita la posizione della prima di tali righe.

Si assuma che il file contenga almeno una riga, e che tutte le righe contengano esclusivamente interi (positivi o negativi) separati da spazi. Il programma non deve necessariamente occuparsi di controllare queste condizioni.

Nel caso si voglia leggere tutto il file in una sola operazione `.read()` si consiglia di spezzare la stringa in singole righe con il metodo `.splitlines()`

Ad esempio sul file `riga_max_file2.txt` contenente

```
2 0 1 -4 5
6 17 10 -9 1
2 4 5 5 0
23 4 -2
```

si dovrà restituire il valore 1, poiché le due righe con somma massima, pari a 25, sono la riga 1 e la riga 3

```
print(riga_max_file('riga_max_file2.txt'))
```

1

```
1
```

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

File di test: `test_riga_max_file.py`

## 5 Bordo della Matrice (5 punti)

Scrivere una funzione `bordo_zero_mat(M)` che prenda come argomento una matrice rettangolare di dimensioni  $R \times C$  con  $R \geq 1$  e  $C \geq 1$ , che restituisca `True` se il bordo della matrice è costituito esclusivamente da zeri, e che restituisca `False` altrimenti.

Potete assumere che `M` sia una lista di liste, tutte della stessa lunghezza, e tutte contenenti numeri. Potete assumere che la matrice, così rappresentata, abbia almeno una riga e una colonna. Quindi non dovete verificare che l'argomento sia una matrice ben formata, e non dovete sollevare errori.

```
M1 = [[0]] 1
M2 = [[0,0,0,0], [0,7,8,0], [0,0,0,0]] 2
M3 = [[1,0,0], [0,8,0], [0,2,-1], [0,0,0]] 3
print(bordo_zero_mat(M1)) 4
print(bordo_zero_mat(M2)) 5
print(bordo_zero_mat(M3)) 6
```

```
True
True
False
```

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

File di test: `test_bordo_zero_mat.py`

## 6 Coppia con differenza minima (6 punti)

Scrivere una funzione `coppia_differenza_minima(L)` che riceva una lista `L` contenente solo numeri, e che restituisca una lista contenente le due posizioni `[i,j]` dove

- `i` è minore di `j`;
- `L[i]` e `L[j]` sono i due elementi della lista la cui differenza in valore assoluto è minima rispetto ad ogni altra coppia di elementi in `L`. La differenza minima può essere anche zero.

Potete assumere che la lista `L`

- contenga solo numeri;
- abbia almeno due elementi;
- la coppia a differenza minima sia unica.

Pertanto **non è necessario** che verifichiate queste condizioni e che solleviate errori a riguardo.

```
seq = [-5,4,8,-3,1,3,10] 1
print(coppia_differenza_minima(seq)) 2
```

[1, 5]

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

File di test: `test_coppia_differenza_minima.py`



## **7 Complessità della ricerca binaria (6 punti)**

Discutete l'algoritmo della ricerca binaria. In particolare - descrivete le condizioni necessarie per il suo utilizzo - discutete la complessità computazionale dell'algoritmo - descrivete gli input peggiori in termini di tempo di esecuzione

Completate la discussione dimostrando che la complessità computazionale è quella che avete indicato.

**Scrivete la vostra risposta sul foglio dedicato che vi è stato fornito.**