Università degli Studi di Cagliari
Corsi di Laurea in Ing. Civile e Ing. per l'Ambiente e il Territorio

Fondamenti d'Informatica

A.A. 2025/2026
Docente: Giorgio Fumera

Esercizi sulla programmazione in linguaggio Python

Una possibile soluzione per ogni esercizio si trova nei *file* allegati. Una breve descrizione delle soluzioni proposte è riportata nella seconda parte di questo documento.

Gli esercizi contrassegnati da un asterisco presentano un livello di complessità maggiore.

Per ulteriori esercizi si faccia riferimento alla traccia delle lezioni disponibile nella pagina web del corso, alle esercitazioni di tutoraggio disponibili nel gruppo Teams del corso, e al testo di riferimento.

1. Determinare che cosa viene stampato dal seguente programma (eseguire il programma passo a passo, tenendo traccia del valore di ciascuna variabile).

```
x = ["Che", "cosa", "viene", "stampato", "da", "questo", "programma?"]
y = ""
z = 0
for w in x:
    y = y + w[z]
print(y)
```

2. Determinare che cosa viene stampato dal seguente programma.

```
a = [3, 1, 5, 2, 4]
b = 0
while b < len(a):
    c = a[b]
    d = b + 1
    while d < len(a):
        c = c + a[d]
        d = d + 1
    print(c)
    b = b + 1
```

3. Definire una funzione che riceva come argomenti due liste qualsiasi, e determini se *tutti* gli elementi della prima lista siano presenti anche nella seconda: in questo caso la funzione dovrà restituire il valore `True`, altrimenti il valore `False`.

4. Definire una funzione che riceva come argomento una stringa contenente una parola (si assuma che tutte le lettere siano minuscole), e determini se tale parola sia *palindromica*: in questo caso la funzione dovrà restituire il valore `True`, altrimenti il valore `False`. Si ricorda che una parola è palindromica se è identica a quella che si ottiene leggendola da destra verso sinistra. Esempi di parole palindromiche: `oro`, `ingegni`, `onorarono`.

5. Definire una funzione che riceva come argomento una lista L e restituisca una nuova lista contenente gli stessi elementi di L, ma in ordine inverso. Per esempio, se l'argomento fosse `[3, 1, 8, 5]`, la funzione dovrebbe restituire la lista `[5, 8, 1, 3]`.

6. Definire una funzione che riceva come argomenti due liste qualsiasi e restituisca una lista contenente *tutti* i loro elementi, ma *senza duplicati* (l'ordine degli elementi non è rilevante). Per esempio, se le due liste fossero `[4, -2, 7]` e `[8, 1, 8, 10, -2]`, la lista da restituire sarebbe `[4, -2, 7, 8, 1, 10]` (o una qualunque lista contenente gli stessi elementi); si noti che sia il valore 8 che il valore -2 compaiono più volte nelle due liste da elaborare, ma solo una volta nella nuova lista.

7. Definire una funzione che riceva come argomenti due liste che si assumono avere la stessa lunghezza, contenenti i voti (numeri interi compresi tra 0 e 30) conseguiti dagli studenti che hanno sostenuto le due prove intermedie di un esame. Posizioni corrispondenti nelle due liste si riferiscono allo stesso studente. La funzione dovrà restituire il valore `True` se *tutti* gli studenti hanno superato *entrambe* le prove; in caso contrario dovrà restituire il valore `False`. Per esempio, se gli argomenti della funzione fossero `[21, 28, 26]` e `[23, 27, 28]`, la funzione dovrebbe restituire `True`; se invece gli argomenti fossero `[21, 28, 26]` e `[23, 12, 28]`, la funzione dovrebbe restituire `False`.

8. Definire una funzione che riceva come argomenti tre stringhe f, p e s, contenenti rispettivamente una frase (che si assume per semplicità essere composta solo da lettere minuscole e spazi, senza caratteri di punteggiatura), e due singole parole; la funzione dovrà restituire una nuova stringa ottenuta sostituendo ciascuna occorrenza della parola p nella frase f con la parola s.

   Esempio: se $f =$ `"lanciò una palla contro il vetro e il vetro si ruppe"`, $p =$ `"vetro"` e $s =$ `"vaso"`, la nuova stringa sarà `"lanciò una palla contro il vaso e il vaso si ruppe"`.

9. Il valore $\log(1+x)$ può essere approssimato con una precisione arbitraria, per $x \to 0$, con i primi n termini della corrispondente serie di Taylor, dove il valore di n dipende dalla precisione desiderata:

$$\sum_{k=1}^{n}(-1)^{k-1}\frac{x^k}{k} \quad = \quad x - \frac{x^2}{2} + \frac{x^3}{3} + \ldots + (-1)^{n-1}\frac{x^n}{n} \ .$$

   Definire una funzione che riceva come argomenti i valori di x e di n, e restituisca la corrispondente approssimazione di $\log(1+x)$.

10. Un numero naturale è detto *triangolare* se è pari alla somma dei primi k numeri naturali, per un certo valore di k. Per esempio, il numero 10 è triangolare, poiché è pari alla somma dei primi $k = 4$ numeri naturali: $10 = 1+2+3+4$; il numero 8 non è invece triangolare, come si può osservare dal fatto che $1+2+3 = 6 < 8$, mentre $1+2+3+4 = 10 > 8$. Definire una funzione che riceva come argomento un numero naturale e determini se esso sia triangolare: in questo caso la funzione dovrà restituire il valore `True`, altrimenti il valore `False`.

11. * È noto che il valore di $\pi/4$ coincide con il valore della seguente serie:

$$\sum_{k=1}^{\infty}(-1)^{k-1}\frac{1}{2k-1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ \ldots$$

    Definire una funzione che riceva come argomento un numero reale positivo ϵ, e restituisca il numero *minimo* di termini di tale serie necessario per approssimare $\pi/4$ con un errore in valore assoluto non superiore a ϵ, ovvero il più piccolo valore di n tale che:

$$\left| \frac{\pi}{4} - \sum_{k=1}^{n}(-1)^{k-1}\frac{1}{2k-1} \right| \leq \epsilon \ .$$

    Si usi la variabile predefinita `pi` della libreria `math` come approssimazione della costante π.

12. Definire una funzione che riceva come argomento una lista composta da dizionari, ciascuno dei quali contiene le coordinate di un punto in uno spazio a tre dimensioni, associate alle chiavi `"x"`, `"y"` e `"z"`, come in questo esempio: `{"x":2, "y":-1, "z":3}`.

    La funzione dovrà restituire un dizionario contenente le coordinate del *baricentro* di tali punti, che dovrà essere calcolato assumendo identiche le loro masse. Si ricorda che il baricentro di un insieme di n punti $(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)$ aventi massa identica è il punto avente coordinate:

$$x = \frac{\sum_{k=1}^{n} x_k}{n}, \quad y = \frac{\sum_{k=1}^{n} y_k}{n}, \quad z = \frac{\sum_{k=1}^{n} z_k}{n} \ .$$

13. Definire una funzione che riceva come argomento una lista composta da dizionari, ciascuno dei quali contenente una coppia di numeri reali associati alle chiavi `"x"` e `"y"`. Un esempio: `{"x": 3, "y": -1}`. La funzione dovrà restituire il *coefficiente di correlazione* tra tutte le coppie di valori:

$$\frac{\sum_{k=1}^{n}(x_k-\overline{x})(y_k-\overline{y})}{\sqrt{\sum_{k=1}^{n}(x_k-\overline{x})^2}\sqrt{\sum_{k=1}^{n}(y_k-\overline{y})^2}} \in [-1,+1] \ ,$$

dove n indica il numero di coppie, e $\overline{x}$ e $\overline{y}$ indicano la media aritmetica rispettivamente di $x_1,\ldots,x_n$ e di $y_1,\ldots,y_n$. Si noti che il coefficiente di correlazione non è definito se il denominatore dell'espressione precedente è nullo: in questo caso la funzione dovrà restituire il valore convenzionale -2 (al di fuori dell'intervallo dei valori possibili per il coefficiente di correlazione).

14. * Un *quadrato magico* di ordine n, per un dato intero $n \geq 3$, è una matrice quadrata di $n \times n$ elementi contenente i valori $1, 2, 3, \ldots, n^2$, tale che le somme degli elementi di ogni riga, di ogni colonna e delle due diagonali abbiano valore identico. Per esempio, la matrice seguente è un quadrato magico di ordine $n = 3$, nel quale le somme di ogni riga, colonna e diagonale sono tutte pari a 15:

$$\begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

Una matrice può essere rappresentata mediante liste nidificate, come mostrato di seguito in riferimento all'esempio precedente (ognuna delle liste nidificate corrisponde a una riga della matrice): `[[8, 1, 6], [3, 5, 7], [4, 9, 2]]`

Definire una funzione che riceva come argomento una lista rappresentante una matrice quadrata di dimensione qualsiasi, e verifichi se tale matrice sia un quadrato magico: in questo caso la funzione dovrà restituire il valore `True`, altrimenti il valore `False`.

15. Si consideri una lista composta da dizionari, ciascuno dei quali contiene le seguenti informazioni sugli insegnamenti di un dato corso di laurea: nome, codice, crediti, anno di corso. Un esempio:

`{"nome": "Fisica 1", "codice": "70/0004-M", "crediti": 8, "anno": 1}`

Definire una funzione che riceva come argomenti una tale lista e un intero positivo n idicante un numero di crediti, e restituisca una lista contenente i nomi di tutti gli insegnamenti da n crediti.

16. Si assuma che alcune informazioni sugli esami superati da un certo studente (codice dell'esame, numero di crediti e voto) siano memorizzate in una lista composta da dizionari, ciascuno dei quali corrisponde a un esame; in particolare, il voto è codificato come un numero intero tra 18 e 31, dove 31 codifica il 30 con lode. Un esempio:

`[{"codice": "12A", "CFU": 5, "voto": 28}, {"codice": "23B", "CFU": 6, "voto": 30}]`

Definire una funzione che riceva come argomento una tale lista e restituisca la media dei voti pesata in base ai crediti. Gli eventuali 30 con lode devono essere considerati come 30. Si ricorda che la media di un insieme di valori $x_1,\ldots,x_n$, pesata con coefficienti $c_1,\ldots,c_n$, è definita da:

$$\frac{\sum_{k=1}^{n} c_k x_k}{\sum_{k=1}^{n} c_k} \ .$$

Nell'esempio precedente la media pesata dei voti dei due esami è data da $\frac{5\times28+6\times30}{5+6} \simeq 29.09$.

17. Si assuma che una lista contenga gli esiti di una sequenza di lanci di un dado a sei facce, codificati con i numeri da 1 a 6. Per esempio, se l'esito di dieci lanci fosse ⚂ ⚂ ⚂ ⚁ ⚅ ⚂ ⚄ ⚅ ⚄ ⚀, la lista corrispondente sarebbe `[3, 3, 3, 2, 6, 3, 5, 6, 5, 1]`.

Definire una funzione che riceva come argomento una tale lista, contenente l'esito di un numero *qualsiasi* di lanci, e restituisca una lista contenente il numero di occorrenze di ciascuno dei sei possibili esiti. Nell'esempio precedente la lista da restituire sarebbe `[1, 1, 4, 0, 2, 2]` (vi è una sola occorrenza degli esiti ⚀ e ⚁, quattro occorrenze di ⚂, ecc.).

18. Definire una funzione che riceva come argomento una stringa contenente una frase f (che si assume essere composta da lettere minuscole e spazi, senza segni di punteggiatura), e restituisca un dizionario nel quale ogni coppia chiave-valore sia composta da una delle parole di f (chiave) e dal numero di occorrenze di tale parola all'interno di f (valore). Per esempio, se l'argomento della funzione fosse la stringa `"il libro è sopra il tavolo"`, il dizionario da restituire sarebbe:

    ```
    {"il": 2, "libro": 1, "è": 1, "sopra": 1, "tavolo": 1}
    ```

    Nota: per verificare se un dizionario contiene una certa chiave è possibile usare l'operatore `in`. Per esempio, l'espressione `"a" in {"a": 1, "b": 2}` produce il valore `True`, mentre l'espressione `"c" in {"a": 1, "b": 2}` produce `False`.

19. Si consideri un *file* contenente le seguenti informazioni sugli insegnamenti di un dato corso di laurea: nome, codice, crediti, anno di corso; ogni riga corrisponde a un insegnamento distinto. Si consideri come esempio il *file* allegato `insegnamenti.txt`, del quale si riportano due righe:

    ```
    Fisica_1  70/0004-M  8  1
    Fondamenti_di_Informatica  IA/0220  8  1
    ```

    Definire una funzione che riceva come argomenti una stringa contenente il nome del *file* e un numero n (che si assume essere un intero positivo), e restituisca il numero totale di crediti di tutti gli insegnamenti dell'anno n-esimo.

20. Si assuma che un *file* di nome `voli.txt` contenga le seguenti informazioni sui voli effettuati in un certo giorno da una compagnia aerea: codice del volo, città di partenza e di arrivo (si assuma che il nome di ogni città sia composto da un unico termine), orari di partenza e di arrivo (ore e minuti). Ogni riga si riferisce a un volo distinto. Si consideri come esempio il *file* allegato `voli.txt`, del quale si riporta una riga corrispondente al volo `XY1234` con partenza da Cagliari alle 6:30 e arrivo a Roma alle 7:20:

    ```
    XY1234 Cagliari Roma 6 30 7 20
    ```

    Definire una funzione che riceva quattro argomenti: due stringhe contenenti il nome di una città di partenza (P) e di una città di destinazione (D), e due interi corrispondenti a un orario di partenza (ore e minuti); la funzione dovrà stampare sullo schermo il codice di tutti i voli in partenza da P con destinazione D, *non prima* dell'orario desiderato. Se nessuno dei voli presenti nel *file* corrispondesse a tali requisiti, la funzione dovrebbe stampare un messaggio opportuno.

21. Si assuma che un *file* di testo contenga le temperature rilevate in un insieme di località nelle ventiquattro ore di un certo giorno, espresse come numeri interi in gradi Celsius. Ogni riga corrisponde a una singola località, e contiene il suo nome (che si assume per semplicità essere composto da un singolo termine, senza spazi al suo interno) seguito dai valori delle temperature, tutti separati da caratteri di spaziatura. Si consideri come esempio la prima riga del *file* allegato `temperature.txt`:

    ```
    Cagliari 7 8 10 13 15 16 17 17 18 20 22 22 23 23 22 21 21 20 20 18 16 15 13 11
    ```

    (a) Scrivere un programma che acquisisca il nome del *file* attraverso la tastiera e stampi sullo schermo i nomi di ciascuna località e le temperature minima e massima in essa registrate, come nell'esempio seguente:

    ```
    Cagliari 7 23
    ```

    (b) Definire una funzione che riceva come argomento una stringa contenente il nome del *file*, ne acquisisca il contenuto e lo memorizzi in una lista di dizionari; ogni dizionario deve contenere due coppie chiave-valore corrispondenti al nome di una località e alle relative temperature, queste ultime memorizzate in una lista, come mostrato nell'esempio seguente:

    ```
    {"città": "Cagliari", "temperature": [7, 8, 10, ... ]}
    ```

22. Si consideri un *file* di nome `rubrica.txt` contenente una rubrica telefonica (si veda come esempio l'omonimo *file* allegato). Ogni riga contiene il nome, il cognome e il numero di telefono di una persona, come nell'esempio seguente (si assuma che ogni persona abbia un solo nome e un solo cognome, e che i numeri di telefono non contengano spazi):

    ```
    Francesca Verdi +390123456789
    ```

    Definire una funzione che riceva come argomenti due stringhe contenenti un nome e un cognome, e stampi sullo schermo il numero di telefono della persona corrispondente (si assuma che la rubrica non contenga omonimi). Se tale persona non fosse presente nella rubrica, la funzione dovrebbe stampare un messaggio opportuno. In alternativa, se la persona cercata non fosse presente, la funzione dovrebbe inserirla nella rubrica, aggiungendo una riga alla fine del *file*, dopo aver acquisito mediante la tastiera il suo numero di telefono (si ricordi che non è possibile eseguire operazioni di scrittura su un *file* aperto in modalità di lettura, e viceversa: in questo caso il *file* aperto inizialmente in lettura dovrà essere chiuso e riaperto in scrittura).

23. Ogni riga di un *file* di nome `campionato.txt` contiene il nome di una squadra che ha partecipato a un campionato di pallacanestro e il numero di punti da essa conseguiti (si assuma che il nome di ciascuna squadra sia composto da un solo termine). Le squadre sono elencate nel *file* in un ordine qualsiasi (*non* in base ai punti). Si veda come esempio il *file* allegato `campionato.txt`, del quale si riporta una riga:

    ```
    Milano 40
    ```

    Scrivere un programma che determini la squadra prima in classifica, o le squadre prime a pari merito, e ne scriva i nomi (ognuno in una riga distinta) in un nuovo file di nome `prime.txt`.

24. Si assuma che gli esiti di una gara podistica siano stati scritti in un *file* di testo. Ogni riga contiene i dati di una delle partecipanti: numero di gara, cognome, nome ed esito della gara. Se un'atleta ha concluso la gara, l'esito consiste nel tempo da essa riportato, rappresentato da tre numeri interi: minuti, secondi e centesimi di secondo; in caso contrario l'esito è codificato con la stringa `NC` (non classificata). Si assuma per semplicità che ogni atleta abbia un solo nome e un solo cognome. Si veda come esempio il *file* allegato `gara_podistica.txt`, del quale si riportano due righe:

    ```
    1 Neri Giovanna 15 39 10
    7 Rossi Francesca NC
    ```

    Definire una funzione che riceva come argomento una stringa contenente il nome del *file*, determini la vincitrice della gara o le vincitrici a pari merito, e stampi sullo schermo il numero di gara e il cognome di ciascuna di esse. Se nessun'atleta avesse concluso la gara, la funzione dovrebbe stampare un messaggio opportuno.

25. Si consideri un *file* di testo contenente gli esiti di una gara di salto in lungo. Ogni riga corrisponde a un atleta e contiene il numero di gara, il cognome (si assuma per semplicità che ogni atleta abbia un solo cognome) e le misure (in cm) dei suoi salti; i salti nulli vengono codificati con il valore 0. Si noti che atleti diversi possono aver eseguito un diverso numero di salti. Si veda come esempio il *file* allegato `gara_salto_in_lungo.txt`, del quale si riporta una riga corrispondente al caso di un atleta che abbia eseguito quattro salti, il terzo dei quali sia nullo:

    ```
    15 Rossi 783 792 0 779
    ```

    Definire una funzione che riceva come argomenti due stringhe contenenti il nome del *file* con gli esiti della gara e il nome di un nuovo *file* che dovrà essere creato dalla funzione, e nel quale dovranno essere scritti il numero di gara e il cognome dei soli atleti che hanno eseguito almeno un salto valido, e per ciascuno di essi la misura del salto più lungo (i dati di ogni atleta dovranno essere scritti in una riga distinta). La riga del nuovo *file* corrispondente all'atleta dell'esempio precedente sarebbe:

    ```
    15 Rossi 792
    ```

Soluzioni

Si riporta una descrizione sintetica delle soluzioni degli esercizi 3–25 proposte nei programmi allegati.

1. Il programma stampa le iniziali di ciascuna parola della lista `x`: `Ccvsdsp`.

2. Il programma stampa i seguenti valori (ciascuno in una riga distinta): `15 12 11 6 4`. Ogni valore corrisponde alla somma del corrispondente elemento della lista `a` con tutti quelli successivi: $15 = 3 + 1 + 5 + 2 + 4$, $12 = 1 + 5 + 2 + 4$, $11 = 5 + 2 + 4$, $6 = 2 + 4$, $4 = 4$.

3. Mediante un'istruzione iterativa si accede a ciascun elemento della prima lista, e si verifica se esso sia presente nella seconda lista usando l'operatore `not in`; non appena s'incontra un elemento che *non* è presente nella seconda lista, l'iterazione viene interrotta e la funzione restituisce `False`; in caso contrario l'iterazione procede fino a completare l'analisi degli elementi della prima lista, e la funzione restituisce `True`. Vengono proposte due versioni della soluzione: una con l'istruzione `while` e una con `for`.

4. Per verificare se una parola sia palindromica si deve confrontare la prima lettera con l'ultima, la seconda con la penultima, ecc.: la parola è palindromica solo se *tutte* le coppie di lettere sono identiche. Si noti che nel caso di parole composte da un numero *dispari* di lettere (per esempio, `ingegni`) la lettera in posizione centrale è ininfluente.

   Vengono proposte due versioni della funzione. Della prima versione si dànno due varianti. Nella prima variante si usano due variabili come indici delle due lettere da confrontare in ogni passo dell'iterazione, mentre nella seconda su usa solo l'indice della prima lettera, ricavando da questo e dalla lunghezza della parola l'indice della seconda lettera. Seguendo una logica analoga a quella dell'esercizio precedente, non appena due lettere risultano diverse s'interrompe l'esecuzione della funzione restituendo `False`; se invece tutte le coppie di lettere sono identiche l'iterazione viene completata, e la funzione termina restituendo `True`.

   Nella seconda versione si costruisce una nuova stringa contenente le lettere della parola in esame in ordine inverso; le due stringhe vengono poi confrontate.

5. La soluzione proposta segue un procedimento analogo a quello usato nell'esercizio precedente per costruire una stringa contenente le lettere di una parola in ordine inverso. Vengono proposte due versioni della funzione: una fa uso dell'istruzione `while`, l'altra di `for`. Poiché l'istruzione `for` accede agli elementi di una sequenza (lista o stringa) dal primo all'ultimo, ogni elemento della lista originale viene inserito (per concatenazione) *all'inizio* della nuova lista.

6. Nella soluzione proposta si esegue un'iterazione sugli elementi della *concatenazione* delle due liste (si ricordi che la concatenazione non elimina gli eventuali duplicati); ciascuno di essi viene inserito (sempre per concatenazione) in una nuova lista, purché non ne faccia già parte (la verifica viene eseguita mediante l'operatore `not in`). La concatenazione degli elementi delle due liste consente di evitare due iterazioni per costruire la nuova lista.

7. Se tutti gli studenti hanno superato entrambe le prove, tutti gli elementi delle due liste sono maggiori o uguali a 18. La verifica può allora essere eseguita analizzando mediante un'unica iterazione gli elementi della *concatenazione* delle liste originali, con una logica ancora una volta analoga a quella dell'esercizio 3: l'esecuzione della funzione viene interrotta mediante l'istruzione `return`, restituendo il valore `False`, non appena si trova un valore minore di 18; in caso contrario l'iterazione viene portata a termine, e la funzione si conclude restituendo il valore `True`.

8. La stringa contenente la frase da elaborare viene suddivisa nelle singole parole mediante la funzione `split`; la nuova stringa viene costruita per concatenazione, a partire da una stringa vuota, mediante un'iterazione che accede a ciascuna parola della frase originale: alla nuova stringa verrà aggiunta la parola in esame, se quest'ultima è diversa da p, altrimenti verrà aggiunta la parola s. In ogni caso, dopo l'aggiunta di una parola si inserisce anche un carattere di spaziatura.

   Viene proposta anche una seconda versione della funzione, nella quale si evita l'inserimento di uno spazio dopo l'ultima parola. Lo stesso risultato si potrebbe ottenere mediante la funzione predefinita `strip`, che elimina gli spazi (inclusi i *newline*) all'inizio e alla fine di una stringa. Per esempio, l'ultima istruzione della prima versione della funzione potrebbe essere:

   ```
   return nuova_frase.strip()
   ```

9. La soluzione viene proposta in due versioni, nelle quali si usa il noto procedimento iterativo per il calcolo della somma di una sequenza di valori. Nella prima versione il valore di ciascun elemento viene ottenuto mediante una singola espressione.

   Nella seconda si adottano due accorgimenti per rendere il calcolo più efficiente. Il primo tiene conto del fatto che i valori della potenza $(-1)^{k-1}$ per i diversi termini della serie sono un'alternanza dei valori $+1$ e -1. Il secondo è basato sul fatto che il valore della potenza x^k può essere ottenuto in modo incrementale partendo dal valore di x (ovvero x^1), e moltiplicando il valore attuale per x prima del calcolo di ciascun termine.

10. Nella soluzione proposta vengono calcolate mediante un'iterazione le somme dei primi k numeri naturali, per $k = 1, 2, 3, \ldots$ (ovvero i valori $1,\ 1+2,\ 1+2+3,\ \ldots$), fino a ottenere un valore uguale o superiore a n. Al termine dell'iterazione è sufficiente confrontare tale valore con n per determinare se quest'ultimo sia un numero triangolare o meno.

11. La soluzione proposta consiste nel calcolare le somme dei primi n termini della serie, per $n = 1, 2, 3, \ldots$, finché la differenza in valore assoluto tra l'ultima somma calcolata e $\pi/4$ (quest'ultimo approssimato mediente la variabile predefita `pi`) risulta *maggiore* di ϵ. Il risultato richiesto coincide quindi con il valore di n al termine dell'iterazione. Si noti che i valori del fattore $(-1)^{k-1}$ per ciascun termine della serie costituiscono un'alternanza di $+1$ e -1, e vengono pertanto calcolati in modo analogo all'esercizio 9.

12. Vengono proposte due versioni della funzione, nelle quali le coordinate del baricentro vengono calcolate sommando le coordinate corrispondenti di ciascun punto, mediante una *singola* iterazione, dividendo poi ciascuna somma per il numero di punti. Nella prima versione le somme parziali vengono assegnate a tre variabili, i cui valori finali vengono poi copiati in un dizionario. Nella seconda versione le somme parziali vengono memorizzate direttamente nel dizionario.

    Si noti che la divisione per il numero di punti potrebbe anche essere eseguita sulle coordinate di ogni *singolo* punto (questo corrisponderebbe alle espressioni equivalenti $x = \sum_{k=1}^{n}(x_k/n)$, ecc.). Tuttavia ciò richiederebbe, per ogni coordinata, n operazioni di divisione invece di una sola: tale procedimento sarebbe meno efficiente, e anche più soggetto a errori di approssimazione.

13. Vengono prima calcolate le medie aritmetiche $\overline{x}$ e $\overline{y}$, mediante una *singola* iterazione (come nella soluzione dell'esercizio 12), e successivamente i valori delle tre sommatorie presenti nel numeratore e nel denominatore, anche in questo caso mediante un'unica iterazione.

14. Per calcolare la somma degli elementi di ciascuna riga è sufficiente una sola iterazione, e l'uso della funzione predefinita `sum`. Per gli elementi di ciascuna colonna sono necessarie due iterazioni nidificate. Anche per gli elementi delle due diagonali è sufficiente una singola iterazione, tenendo conto del fatto che gli elementi della diagonale principale hanno indici di riga e colonna identici, mentre nel caso della diagonale opposta, detto k l'indice di colonna di un suo elemento e n la dimensione della matrice, l'indice di riga dello stesso elemento è pari a $n - k - 1$.

    Per verificare se le somme sono tutte identiche, nella prima versione della soluzione proposta i loro valori vengono memorizzati in una lista, che viene successivamente analizzata per verificare se i suoi elementi siano tutti identici (a questo scopo si confronta mediante un'iterazione il primo elemento con ciascuno degli altri).

    Nella seconda versione viene memorizzato solo il valore della prima somma calcolata (corrispondente alla prima riga della matrice); il confronto con tale valore viene poi eseguito subito dopo il calcolo di *ciascuna* delle altre somme: in questo modo l'esecuzione della funzione può essere interrotta (restituendo il valore `False`) non appena si trovi una somma diversa dalla prima.

15. La lista richiesta viene costruita in modo incrementale, a partire da un lista vuota, mediante un'iterazione sugli elementi (dizionari) della lista degli insegnamenti.

16. Il numeratore e il denominatore dell'espressione della media pesata vengono calcolati mediante un'unica iterazione; per ogni esame si verifica se il voto è pari a 31: in questo caso il valore da considerare al numeratore è 30.

17. Si propongono due versioni della funzione. Nella prima, la lista con il numero delle occorrenze viene costruita in modo incrementale (a partire da una lista vuota), considerando per mezzo di un'iterazione ciascuno dei possibili esiti, e calcolando tramite un'iterazione nidificata il corrispondente numero di occorrenze.

    Nella seconda versione si sfrutta la corrispondenza tra la codifica dell'esito di ciascun lancio (un numero da 1 a 6) e la posizione del corrispondente numero di occorrenze nella nuova lista: detto k il valore dell'esito, il numero delle sue occorrenze dovrà trovarsi nell'elemento di indice $k-1$. Tenendo conto di ciò, mediante una prima iterazione si costruisce la lista delle occorrenze assegnando il valore 0 a ciascuno dei suoi elementi; mediante una *successiva* iterazione sulla lista degli esiti si incrementa di un'unità l'elemento corrispodente della lista delle occorrenze.

    Come esempio di uso della libreria `random`, il *file* allegato contiene anche le istruzioni che consentono di simulare un numero qualsiasi di lanci mediate la funzione di libreria `randint`, calcolando poi le occorrenze dei vari esiti mediante una delle due versioni della funzione.

18. Vengono proposte due versioni della funzione. In entrambe, le stringhe corrispondenti alle singole parole della frase vengono memorizzate in una lista mediante la funzione `split`, e il dizionario delle occorrenze viene costruito in modo incrementale a partire da un dizionario vuoto.

    Nella prima versione, mediante un'iterazione sulla lista delle parole vengono aggiunte al dizionario le chiavi corrispondenti a ciascuna di esse, associate al valore 0. Mediante una successiva iterazione sulla stessa lista, il valore associato alla chiave corrispondente a ciascuna parola viene incrementato di un'unità.

    Nella seconda versione si esegue un'unica iterazione sulla lista delle parole. Se una parola non è ancora presente come chiave nel dizionario, si è incontrata la sua prima occorrenza: viene quindi aggiunta al dizionario la chiave corrispondente, associata al valore 1. Se invece tale parola è già presente come chiave nel dizionario, il valore a essa associato viene incrementato di un'unità.

19. Poiché è necessario analizzare separatamente tutte le righe del *file*, è conveniente acquisirne il contenuto mediante la funzione predefinita `readlines`, oppure `readline`, come mostrato nelle due versioni proposte della funzione. Mediante un'iterazione si elabora ciascuna delle righe, suddividendo i suoi elementi tramite la funzione `split`, e sommando il numero di crediti degli insegnamenti dell'anno richiesto (a questo scopo è necessario convertire in numeri interi le stringhe corrispondenti ai crediti e all'anno di corso, mediante la funzione predefinita `int`).

20. Anche in questo caso è necessario analizzare separatamente tutte le righe del *file*. Nella soluzione proposta il suo contenuto viene acquisito mediante la funzione `readlines`. Si lascia come esercizio l'uso in alternativa della funzione `readline`.

    Mediante un'iterazione si elabora ogni riga (ovvero ogni stringa nella lista restituita da `readlines`), suddividendone prima di tutto gli elementi tramite la funzione `split`; per determinare se un volo parta non prima dell'orario desiderato, si tiene conto che un orario $h_1 : m_1$ precede un altro orario $h_2 : m_2$ se $h_1 < h_2$, oppure se $h_1 = h_2$ e $m_1 < m_2$.

    Per stampare un messaggio specifico nel caso in cui non siano presenti voli con le caratteristiche desiderate, si usa una variabile alla quale si assegna inizialmente il valore convenzionale `False` (prima dell'analisi del contenuto del *file* non si è ancora trovato nessun volo); se si trova un volo con le caratteristiche desiderate, a tale variabile viene assegnato il valore `True`.

21. Anche in questo caso nella soluzione proposta si usa la funzione `readlines`, e si lascia come esercizio l'uso in alternativa della funzione `readline`. Per calcolare le temperature minima e massima di ciascuna località si usano le funzioni predefinite `max` e `min`. A questo scopo si costruisce prima una lista contenente i valori numerici delle temperature.

22. Nella soluzione proposta si usa la funzione `readline` per acquisire una riga del *file* alla volta, mediante un'iterazione; in questo modo si può interrompere l'intera funzione (tramite l'istruzione `return`) non appena si raggiunga la riga corrispondente alla persona richiesta, mentre l'iterazione potrà terminare solo se tale persona non è presente nella rubrica. Nella seconda versione della funzione, se la persona cercata non è presente nella rubrica il *file* viene chiuso e riaperto in modalità *append* (`"a"`), ovvero in modalità di scrittura con aggiunta dei nuovi dati al termine di quelli eventualmente già presenti.

23. Si propongono due versioni della soluzione. In entrambe il contenuto del *file* è acquisito mediante la funzione `readlines`. Nella prima versione si eseguono in successione due iterazioni sui dati di ciascuna squadra (ovvero sulle stringhe della lista restituita da `readlines`): la prima per trovare il punteggio più alto, la seconda per individuare tutte le squadre che hanno conseguito tale punteggio. Si noti che in entrambe le iterazioni si usa la funzione `split`, in quanto essa non modifica la stringa sulla quale opera.

    Nella seconda versione si usa un'unica iterazione per trovare le squadre prime in classifica, i cui nomi vengono memorizzati in una lista, inizialmente vuota. A questo scopo durante l'iterazione si tiene traccia del punteggio più alto tra le squadre già analizzate, e si procede come segue: se si trova una squadra con un punteggio ancora più alto, si costruisce una nuova lista contenente solo il nome di tale squadra; se si trova una squadra con un punteggio identico a quello più alto, alla lista attuale viene *aggiunto* il nome di tale squadra.

24. Nella soluzione proposta il contenuto del *file* viene acquisito mediante la funzione `readlines`. Per la ricerca del tempo migliore e delle atlete vincitrici si usa un procedimento analogo a quello della seconda versione proposta per l'esercizio precedente: mediante un'iterazione sui dati delle atlete (ovvero sulle righe del *file*) si tiene traccia del tempo migliore tra quelli già analizzati, e si memorizzano in una lista i nomi e i numeri di gara delle atlete corrispondenti (i dati di ogni atleta vengono inseriti in un dizionario).

    Per semplificare i confronti tra i tempi di gara, questi ultimi vengono espressi in centesimi di secondo mediante l'espressione $c + 100s + 6000m$, dove c indica i centesimi di secondo, s i secondi e m i minuti.

25. Nella soluzione proposta il contenuto del *file* viene acquisito mediante `readlines`. Mediante un'iterazione sui dati degli atleti (righe del *file*) si determina la misura del salto più lungo di ciascuno di essi; un valore pari a 0 significa che l'atleta non ha eseguito salti validi.