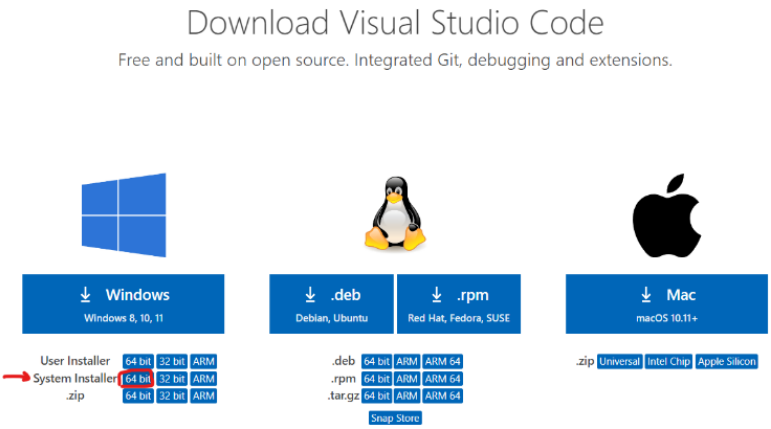
RACCOLTA DI MATERIALE ED ESERCIZI PER LA PROGRAMMAZIONE IN LINGUAGGIO PYTHON

Ambiente di sviluppo

1. Installazione dell'ambiente di lavoro:

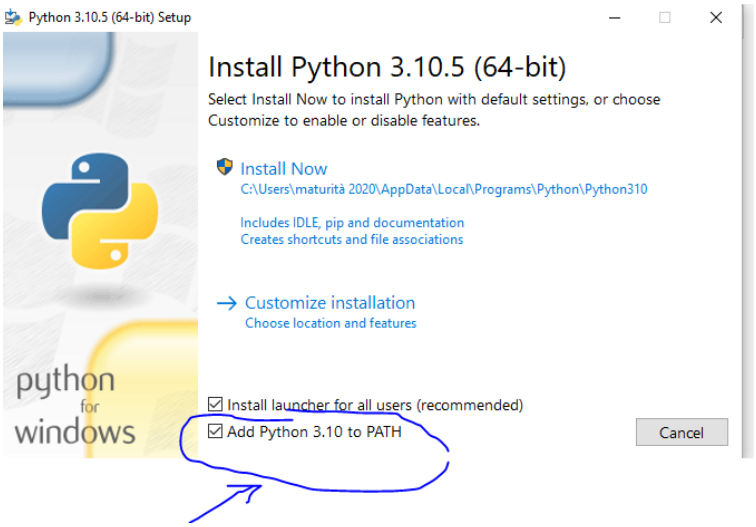
Installare Python che si può scaricare da:
https://www.python.org/downloads/



Consiglio, a meno di particolari necessità, di scaricare il system installer a 64bit.

Durante l'installazione:

- Flaggare l’opzione "Add Python x.xx to PATH" come mostrato nella seguente immagine;

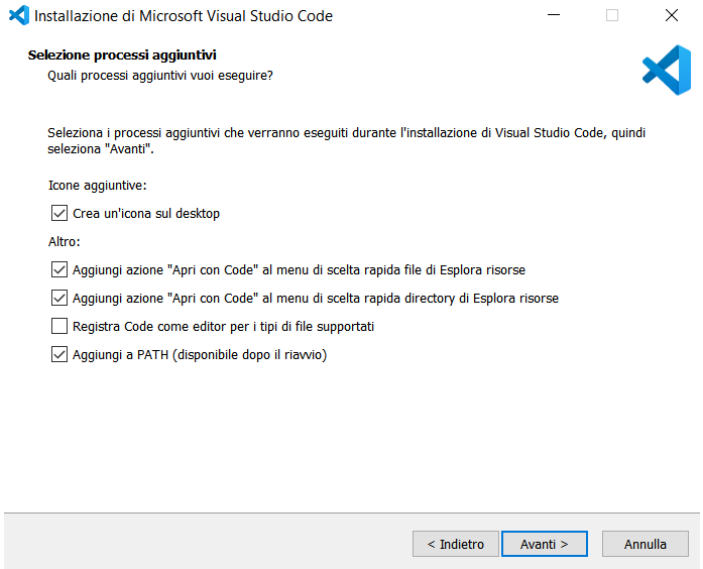


- Scegliere customize installation.
- Alla fase “optional features” lasciare tutto flaggato
- Alla fase “Advanced options” consiglierei, a meno di casi particolari, di aggiungere il flag alla prima voce, lasciare il flag agli altri 3 già flaggati e se si vuole flaggare anche la quinta voce.

Per verificare la corretta installazione basta eseguire in una qualsiasi console (cerca cmd su start) il comando “python” (python3 se su mac)

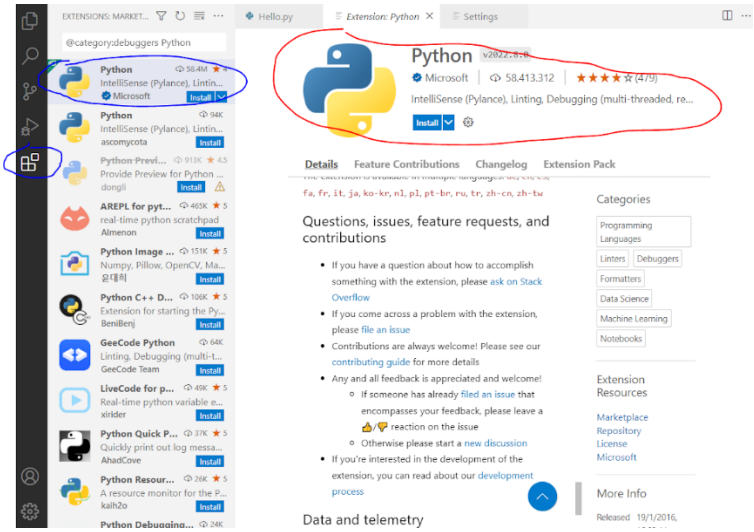
Per chi non lo avesse ancora fatto, installare visual studio code, consiglierei di usare, a meno di esigenze particolari, il system installer da 64bit che puoi trovare sul sito https://code.visualstudio.com/download .

Ricordarsi durante l’installazione di flaggare le voci corrette durante tutti i passaggi come mostrato dalla seguente immagine:

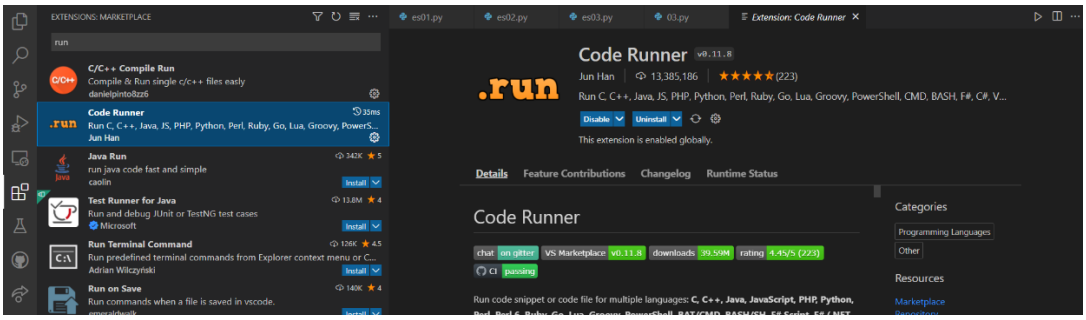


Per programmare comodamente in python con Visual studio code consiglio di installare le seguenti estensioni:

- Python.

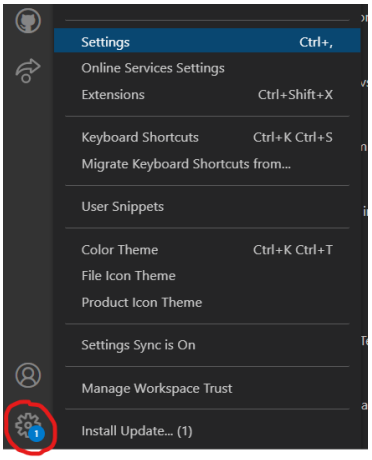


- ~~Code Runner~~



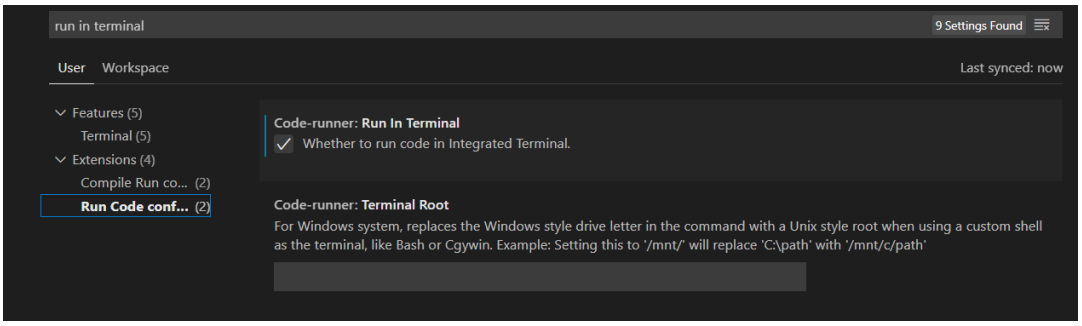
- Al posto di Code Runner ora consiglio C/C++ Compile Run

Una volta installate le estensioni vanno modificate alcune impostazioni. Il menù impostazioni è velocemente accessibile dal tasto in basso a sinistra:

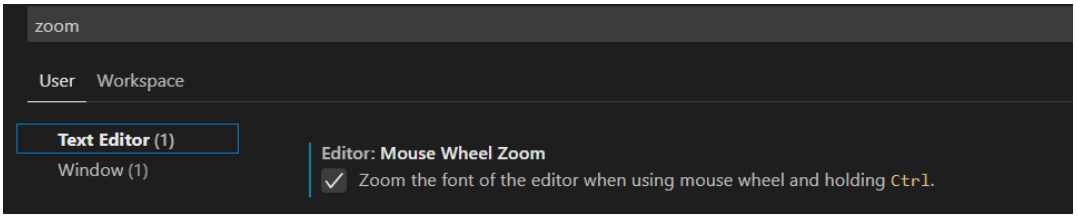


Le impostazioni da modificare sono:

- “run in terminal” da spuntare nelle impostazioni e trovabile velocemente inserendo la voce nella barra di ricerca e cliccando sull’estensione interessata come nella seguente immagine:



- "Mouse Wheel Zoom" da spuntare come fatto per l'impostazione precedente



È possibile modificare a piacere altre impostazioni come:

- Save file before run
- Save all files before run
- Color Theme
- Clear Previous Output

Indicazioni e materiale

Come manuale contenente esempi e spiegazioni di base su ogni aspetto del linguaggio consiglio di guardare il sito https://www.w3schools.com/python/ .
Online sono presenti molti altri tutorial anche in lingua italiana come ad esempio

1. https://www.programmareinpython.it/corsi-e-lezioni-python-dal-nostro-canale-youtube/
2. https://www.html.it/guide/guida-python/

Esercizi

La parte di eserciziario è stata inizialmente adattata dall'eserciziario per il linguaggio C, possono quindi essere presenti esercizi che fanno riferimento a elementi di tale linguaggio che devono ancora essere modificati.

Di seguito sono riportati gli esercizi normalmente svolti durante le lezioni del mio corso, voglio comunque riportare questo link (https://q2a.di.uniroma1.it/assets/eserciziario-python/it/script/) dove potete trovare un gran numero di esercizi, alcuni semplici altri anche piuttosto difficili, proposti per un corso universitario di informatica.

1. Variabili e operazioni elementari

1.1.1 Dati in input due numeri, scrivi il risultato di tutte le operazioni che è possibile fare con i due numeri (somma, sottrazione, moltiplicazione, divisione intera, resto della divisione, divisione con decimali, potenza)

Soluzione

```python
a = int(input("Scrivi il primo numero: "))
b = int(input("Scrivi il secondo numero: "))

print(f"{a} + {b} = {a+b}")
print(f"{a} - {b} = {a-b}")
print(f"{a} * {b} = {a*b}")
print(f"{a} ** {b} = {a**b}")       # elevato a
print(f"{a} / {b} = {a/b:.2f}")     # produce un float
print(f"{a} // {b} = {a//b}")       # produce un intero troncato
print(f"{a} % {b} = {a%b}")         # resto
```

1.1.2 Dato un numero decimale, ottieni e stampa:

1. Valore intero approssimato per difetto
2. Valore intero approssimato per eccesso
3. Valore intero approssimato in maniera intelligente
4. approssimazione del numero al secondo decimale
5. Valore assoluto del numero

Soluzione

```python
from math import ceil, floor, pi

n = -pi

print(f"a. {n} intero troncato: {int(n)}")
print(f"a. {n} intero approssimato per difetto: {floor(n)}") # devo importare math
print(f"b. {n} intero approssimato per eccesso: {ceil(n)}") # devo importare math
print(f"c. {n} intero approssimato in maniera intelligente: {round(n)}")
print(f"d. {n} intero approssimato al secondo decimale: {round(n, 2)}")
print(f"e. {n} valore assoluto del numero: {abs(n)}")
```

1.1.3 Data l'area del cerchio, stampare la circonferenza

Soluzione

```python
import math



area = float(input("inserisci l'area del cerchio: "))
```

```
circonferenza = 2*math.pi*(area/math.pi)**(1/2)

print("circonferenza: ", circonferenza)
```

1.1.4 Data la base e l'altezza di un triangolo, scrivere l'area

Soluzione

```
base = float(input("scrivi il valore della base: "))

altezza = float(input("scrivi il valore dell'altezza': "))

area = base*altezza/2

print("L'area è: ", area)
```

1.1.5 Somma di due numeri (v1 - usando una terza variabile)

2. Algoritmi e strutture di controllo

2.1.1 Dati due numeri calcolare il loro quoziente se il divisore è != 0, ritornare "impossibile" se il divisore = 0

Soluzione

```
num = float(input("scrivi il valore del numeratore: "))
den = float(input("scrivi il valore del denominatore': "))
if den == 0:
    print("Non è possibile dividere per 0")
else:
    print(num, "/", den, " = ", num/den)
```

2.1.2 Dati in input due numeri, stampa il più grande tra i due

2.1.3 Dato in input un numero n, stampa la prima potenza di 2 maggiore o uguale a n

Soluzione

```
# 2.1.3 Dato in input un numero n, stampa la prima potenza di 2 maggiore o uguale a n

n = int(input("Scrivi un numero: "))

ris = 1
while ris < n:
    ris = ris * 2

print(ris)
```

2.1.4 Moltiplicazione di due numeri senza usare l'operatore *

2.1.5 Divisione di due numeri interi senza usare l'operatore di divisione

Variante: Stampa anche il valore del resto

Soluzione

```
num = int(input("scrivi il valore del numeratore: "))
den = int(input("scrivi il valore del denominatore': "))

ris = 0
resto = num
while resto > den:
    resto -= den
    ris += 1

# print(num, "/", den, " = ", ris, " resto ", resto)
print(f"{num} / {den} = {ris}  resto {resto}")
```

2.1.6 dati dall'utente due numeri interi n1 e n2 il programma deve stampare:

1. tutti i numeri dal più piccolo dei due al più grande dei due
2. tutti i numeri dal più grande dei due al più piccolo dei due
3. tutti i numeri da n1 a n2 (comunque siano n1 e n2 quindi potresti dover andare in ordine crescente o decrescente)
4. tutti i numeri da n2 a n1 (comunque siano n1 e n2 quindi potresti dover andare in ordine crescente o decrescente)

Soluzione

```
# 2.1.6 dati dall'utente due numeri interi n1 e n2 il programma deve stampare:
# 1.    tutti i numeri dal più piccolo dei due al più grande dei due
# 2.    tutti i numeri dal più grande dei due al più piccolo dei due
# 3.    tutti i numeri da n1 a n2 (comunque siano n1 e n2 quindi potresti dover andare in ordine
crescente o decrescente)
# 4.    tutti i numeri da n2 a n1 (comunque siano n1 e n2 quindi potresti dover andare in ordine
crescente o decrescente)

n1 = int(input("Scrivi il primo numero: "))
n2 = int(input("Scrivi il secondo numero: "))

# 1.    tutti i numeri dal più piccolo dei due al più grande dei due
if n1 > n2:
    max = n1
    min = n2
else:
    max = n2
    min = n1

for i in range(min, max+1):
    print(i, end=" ")

print("\n")

# 2.    tutti i numeri dal più grande dei due al più piccolo dei due
for i in range(max, min-1, -1):
    print(i, end=" ")
```

```
print("\n")

# 3.    tutti i numeri da n1 a n2 (comunque siano n1 e n2 quindi potresti dover andare in ordine crescente o decrescente)
if n1 < n2:
    for i in range(n1, n2+1):
        print(i, end=" ")
else:
    for i in range(n1, n2-1, -1):
        print(i, end=" ")

print("\n")

# 4.    tutti i numeri da n2 a n1 (comunque siano n1 e n2 quindi potresti dover andare in ordine crescente o decrescente)
if n2 < n1:
    for i in range(n2, n1+1):
        print(i, end=" ")
else:
    for i in range(n2, n1-1, -1):
        print(i, end=" ")

print("\n")
```

2.1.7 Dato in input un numero, stampa tutti i suoi divisori

Soluzione

```
# 2.1.7 Dato in input un numero, stampa tutti i suoi divisori
# Variante: stampa la sua scomposizione in fattori primi

n = int(input("Scrivi un numero: "))

for div in range(2,n):
    if n % div == 0:
        print(div, end=" ")
```

2.1.8 Inserire n numeri != 0 (0 per finire), contare quanti sono i numeri inseriti

Soluzione

```
# VERSIONE 1
c = 0
num = -1
while num!=0:
  num=int(input("scrivi un numero, 0 per uscire: "))
  if num!=0:
    c+=1
print(c)

# VERSIONE 2
c = 0
num = -1
while True:
  num=int(input("scrivi un numero, 0 per uscire: "))
  if num == 0:
    break
  c+=1
print(c)
```

2.1.9 Variante avanzata: i numeri sono voti che devono essere validi e alla fine ne va calcolata la media.

2.1.10 Moltiplicazione di due numeri senza usare l'operatore di moltiplicazione

Soluzione

```
a = int(input("scrivi il primo valore: "))
b = int(input("scrivi il secondo valore: "))

ris=0
for i in range(b):
  ris += a

print(a, " x ", b, " = ", ris)
```

2.1.11 Dati tre numeri reali dire che tipo di triangolo essi formano (classificazione dei triangoli in base ai lati).

Soluzione

```
a = float(input("scrivi il primo valore: "))
b = float(input("scrivi il secondo valore: "))
c = float(input("scrivi il terzo valore: "))

if a==b and b==c:
  ris = "equilatero"
elif a==b or b==c or a==c:
  ris = "isoscele"
else:
  ris = "scaleno"

print(a, ", ", b, ", ", c, " formano un triangolo ", ris)
```

2.1.12 Data una sequenza di n numeri interi (valore di n dato dall'utente), calcolare la somma dei pari ed il prodotto dei dispari

Soluzione

```
# VERSIONE 1
n = int(input("Scrivi quanti numeri vuoi inserire: "))
somma = 0
prodotto = 1

for i in range(n):
    num = float(input("Scrivi il "+str(i+1)+"° numero: "))
    if num%2 == 0: #cioè se è pari
        somma+=num
    else:
        prodotto*=num

print("somma dei pari: ", somma)
print("prodotto dei dispari: ", prodotto)

# VERSIONE 2
n = int(input("Scrivi quanti numeri vuoi inserire: "))
somma = 0
prodotto = 1
```

```python
l = []

for i in range(n):
    num = int(input("Scrivi il "+str(i+1)+"° numero: "))
    l.append(num)

print(l)

for num in l:
    if num%2 == 0: #cioè se è pari
        somma+=num
    else:
        prodotto*=num

print("somma dei pari: ", somma)
print("prodotto dei dispari: ", prodotto)
```

2.1.13 Con un opportuno ciclo chiedere all'utente di inserire due numeri n1 e n2 compresi tra 1 e 100. Se i numeri non fossero corretti, rimanere nel ciclo e ripetere la richiesta. Stampare quanti sono i numeri pari compresi tra i due numeri

2.1.14 Data una sequenza di numeri terminata dal numero 0 (leggo numeri finchè non mi viene dato il numero 0), stampo il maggiore e il minore

2.1.15 Letto un numero intero positivo n stampare il fattoriale: n! = 1 · 2 · 3 · . . . ·n

2.1.16 Dati in input 3 numeri, stamparli in ordine crescente

2.1.17 Realizzare un programma che, presi in input 2 operandi reali e un operatore (+, -, *, /), esegue l'operazione stampandone il risultato

2.1.18 Progettare un algoritmo che legga da terminale una sequenza di interi positivi e negativi terminati dal valore 0 (uno su ogni linea) e stampi il prodotto degli interi positivi e la somma dei negativi.

2.1.19 Progettare un algoritmo che legga da terminale una sequenza di interi positivi e negativi fintanto che l'utente dice di volerne inserire ancora, e stampi il prodotto degli interi positivi e la somma dei negativi.

2.1.20 Dati due numeri, determinare il maggiore (verificare anche se sono uguali)

2.1.21 Data una sequenza di numeri terminata da 0, dire quanti sono i numeri inseriti (diversi da 0)

2.1.22 Dati due numeri in input b ed e, calcola e scrivi b^e in due modi diversi, usando l'operatore ** e senza poterlo usare.

Soluzione

```python
b = 2
e = 5
```

```
print(f"{b}^{e} = {b**e}")

ris = 1
for i in range(e):
    ris = ris*b

print(f"{b}^{e} = {ris}")
```

2.1.23 Dati in ingresso 4 numeri, che rappresentano gli orari in cui avvengono due diversi eventi della giornata, in modo che i primi 2 numeri siano ore e minuti del primo orario e gli altri 2 numeri siano ore e minuti del secondo orario, stabilire quale evento è avvenuto prima. Inserisci dei controlli durante l'input in modo che le ore possano andare da 0 a 23 e i minuti da 0 a 59.

2.1.24 Dato un numero di minuti e un numero di secondi rappresentare il conto alla rovescia.

2.1.25 Dato in input un numero intero n, stampa l'ennesimo numero della successione di Fibonacci. La successione di Fibonacci è quella successione di numeri in cui ogni numero è la somma dei due numeri precedenti. I primi due numeri sono 1 1. La successione inizia così: 1 1 2 3 5 8 13 21 …

2.1.26 Dato in input un numero intero, conta da quante cifre è composto.

2.1.27 Dato in input un numero intero n di 3 cifre (in questo caso sarebbe utile mettere un controllo sull'input, cioè continuare a richiedere il numero fintanto che il numero dato non è di 3 cifre), stampa separatamente le sue cifre. Consiglio: usa divisioni per 10 e resti della divisione per 10. Puoi provare anche a generalizzare il programma in modo che funzioni anche con numeri di dimensione diversa da 3 cifre.

2.1.28 Generalizza l'esercizio precedente in modo che possa funzionare con un numero qualsiasi.

2.1.29 Dato in input un numero intero n, stabilisci se è primo

Soluzione

```
n = int(input("Scrivi un numero maggiore di 1: "))
while n <= 1:
    n = int(input("Scrivi un numero maggiore di 1: "))

primo = True
for i in range(2, (n//2)+1):
    if n % i == 0:
        primo = False
        break

if primo:
    print(f"{n} è primo")
else:
    print(f"{n} non è primo")
```

2.1.30 Dato in input un numero intero n, stampa la sua scomposizione in fattori primi

Soluzione

```
# 2.1.30    Dato in input un numero, stampa la sua scomposizione in fattori primi

n = int(input("Scrivi un numero: "))

trovati = 0

div = 2
while div <= n:
    while n % div == 0:
        if trovati != 0:
            print(" * ", end="")
        print(div, end="")
        trovati += 1
        n /= div
    div += 1
```

2.1.31 Data una sequenza di n numeri (n dato dall'utente) stabilire qual è il numero più grande, qual è il più piccolo e calcolare la media.

2.1.32 Data una sequenza di n numeri (n dato dall'utente) stabilire qual è il secondo numero più grande, qual è il secondo più piccolo.

2.1.33 Data una sequenza di numeri positivi (fai il controllo sull'input) terminati da 0, scrivi qual è la maggior differenza tra due numeri dati consecutivamente

2.1.34 Data una sequenza di numeri positivi (fai il controllo sull'input) terminati da 0, scrivi la somma dei numeri divisibili per 3

2.1.35 Dato un numero n scrivi somma e prodotto di tutti i numeri minori o uguali a n.

2.1.36 Dato un numero n stabilire quante volte è possibile dividerlo per 2. (esempio 20 è divisibile per 2, 2 volte)

2.1.37 Dato un numero n, decidere se è primo

2.1.38 Data una sequenza di prezzi di prodotti, calcolare la spesa totale sapendo che se un prodotto costa meno di 100€ lo devo scontare del 10% altrimenti del 5%. Decidi tu il metodo per capire quando terminare la lettura della sequenza di prezzi.

2.1.39 Data una sequenza di 5 numeri che rappresentano i voti presi nelle diverse materie, stabilire se lo studente sarà promosso, bocciato o rimandato a settembre. Lo studente è promosso se non ha insufficienze, è bocciato se ha almeno 3 insufficienze, altrimenti è rimandato. Ricordati di controllare i valori dei voti in input che devono essere voti validi.

2.1.40 Leggi 3 parole e stampale tutte insieme in un'unica volta (solo un parallelogramma di scrittura).

2.1.41 Dato un numero n positivo stampa tutti i numeri da 1 a n, i primi n multipli di 2 (2 compreso che consideri il primo multiplo) e i primi n multipli di 3

2.1.42 Dati tre numeri positivi verificare che questi tre numeri siano una terna pitagorica (una terna pitagorica è un insieme di 3 numeri per cui la somma del quadrato di due numeri sia uguale al quadrato del terzo numero, in altre parole sono le lunghezze dei lati di un triangolo rettangolo)

2.1.43 Scrivere l'algoritmo per il pagamento della spesa che consiste nel chiedere inizialmente se si è in possesso della carta fedeltà, poi chiedere tutti i prezzi dei prodotti acquistati terminando la sequenza con un prezzo uguale a zero. Il programma deve sommare i prezzi, e se si è in possesso della carta, scontare del 10% i prezzi minori di 50 e del 5% i prezzi maggiori di 50

2.1.44 Un libro deve essere restituito in biblioteca dopo 15 giorni di prestito altrimenti si è multati di 0,80€ al giorno di ritardo. Ricevuto in ingresso il numero di giorni di un prestito, visualizza se il socio deve essere multato per il ritardo e a quanto ammonta la multa da pagare.

2.1.45 Calcolare il costo della bolletta telefonica sapendo che i primi 30 scatti costano 20 centesimi l'uno, gli scatti dal 31 al 100 costano 15 centesimi l'uno, mentre gli ulteriori scatti costano 10 centesimi l'uno. Aggiungere infine una tassa fissa di 2,50€ per le spese telefoniche. In input al programma è dato il numero di scatti effettuati.

2.1.46 Costruire uno schema di flusso che rappresenti l'algoritmo per il seguente gioco: il computer genera un numero segreto (usa la funzione random(0, 100) che genera un numero casuale tra 0 e 99) e il giocatore deve individuarlo seguendo le indicazioni fornite dal computer (ti dice se il numero da trovare è più grande o più piccolo di quello che hai provato); una volta trovato il numero segreto, il numero di tentativi effettuati è visualizzato a video.

2.1.47 Calcolare il quoziente e il resto della divisione intera di due numeri interi positivi forniti in ingresso chiamati dividendo e divisore, applicando il metodo delle sottrazioni successive. Per esempio, se dividendo=13 e divisore=5, il programma dovrà restituire Quoziente=2 e Resto=3, calcolati sottraendo successivamente il valore di divisore dal valore di dividendo

2.1.48 Visualizza i termini della successione di Fibonacci compresi nell'intervallo tra due interi positivi N1 e N2, entrambi forniti in ingresso, con N2>N1 (controlla bene gli input)

2.1.49 Simula il lancio di un dado per N volte (con N intero e positivo in ingresso e usando la funzione random(7) per il lancio) e verifica che effettivamente la probabilità che esca 6 è 1/6. Per farlo fai tanti tiri e vedi se esce 6 un sesto delle volte (più tiri fai più dovresti avere un risultato positivo).

2.1.50 Simula il lancio di due dadi e verifica le probabilità che escano: una coppia di 6, un valore totale uguale a 7 (due diverse probabilità)

2.1.51 Dati in ingresso due numeri positivi x e y, visualizza in ordine decrescente la sequenza di numeri interi compresi tra x e y che sono divisibili per il minore tra x e y. Ad esempio, se x = 7 e y = 35, la sequenza è 35 28 21 14 7.

Soluzione

```
x = int(input("Scrivi il primo numero: "))
while x <= 0:
    x = int(input("Scrivi il primo numero: "))

y = int(input("Scrivi il secondo numero: "))
while y <= 0:
    y = int(input("Scrivi il primo numero: "))

if x < y:
    min = x
    max = y
else:
    min = y
    max = x

for i in range(max, min-1, -1):
    if i % min == 0:
        print(i, end=" ")
```

2.1.52 Simula una serie di lanci di un dado a 6 facce (d6) e un dado da 20 facce (d20). Calcola il valore medio che si ottiene lanciando il d6 e quello ottenuto lanciando il d20.

2.1.53 Valuta se è più probabile ottenere 7 oppure ottenere 8 lanciando 2 dadi da 6 e sommando i 2 valori ottenuti.

2.1.54 Voglio valutare la probabilità che tirando due dadi da 6 esca il valore 2. Il valore calcolato deve essere abbastanza preciso e decido di calcolarlo misurando la media di tanti tiri di dado e fermandomi solo quando vedo che facendo uscire un nuovo 2 valore calcolato non varia di più di 0,01%

2.1.55 Vengono dati in input i valori delle altezze di n persone (n chiesto all'utente). Per ogni persona oltre all'altezza viene indicato anche il sesso. Deve essere stampato il valore medio di altezza separatamente per i maschi e per le femmine. L'utente può decidere di non indicare né maschio né femmina, in quel caso non contare quella persona.

2.1.56 Scrivi il codice di un gioco per 2 persone basato sul lancio di dadi. All'inizio del gioco si decide che tipo di dado usare, si possono scegliere i dadi da 4, 6, 8 o 20 facce. Il gioco consiste nel tirare uno per volta il dado e sommando ogni volta i numeri usciti finchè uno dei due giocatori non supera un valore limite che è 4 volte il valore massimo rappresentato sul dado scelto (ad esempio se si sceglie il dado da 6 il limite è 24). Per rendere il gioco equo si può fare in modo che i due giocatori debbano tirare per forza lo stesso numero di tiri e che entrambi i giocatori possano superare il limite facendo finire il gioco in parità

2.1.57 Scrivi una variante del gioco precedente in cui l'obiettivo è avvicinarsi il più possibile al valore limite senza superarlo. Chi si avvicina di più vince. In ogni momento un giocatore può decidere di accontentarsi del valore ottenuto e smettere di tirare.

2.1.58 Leggi un numero positivo e controlla se esso è divisibile per 2 o divide 3 e non è compreso tra 10 e 20

2.1.59 Leggi un numero positivo e controlla se esso è divisibile per 2 o è un multiplo di 3 ed è compreso tra 10 e 20

2.1.60 dato un numero n stampa tutti i numeri da 1 a n che sono divisibili da 2 o 3 o 5 ma non sono divisibili per 4

2.1.61 Dato un numero positivo n disegna le figure geometriche riportate di seguito:

per n = 5

```
xxxxx          xxxxx          x
xxxx           xxxxx          xx
xxx            xxxxx          xxx
xx             xxxxx          xxxx
x              xxxxx          xxxxx
```

Soluzione

```
def f1(n):
    for i in range(n):
        for j in range(n-i):
            print("x", end="")
        print()

def f2(n):
    for i in range(n):
        for j in range(n):
            print("x", end="")
        print()

def f3(n):
    for i in range(n):
        for j in range(1+i):
            print("x", end="")
        print()



f1(5)
f2(5)
f3(5)
```

2.1.62 Esegui un programma che abbia il seguente output (chiaramente usando dei cicli):

```
1 2 3 4 5 6
2 3 4 5 6 +
3 4 5 6 + +
4 5 6 + + +
5 6 + + + +
6 + + + + +
```

Soluzione

```
n = 6

for i in range(n):
    for j in range(n):
        x = j+1+i
        if x <= n:
            print(x, end=" ")
        else:
            print("+", end=" ")
    print()
```

2.1.63 Simula l'esecuzione di un gioco di dadi in cui due persone si scontrano lanciando ognuno un dado. Il gioco è diviso in round. Al primo round i due giocatori lanciano ognuno un dado da venti facce e vince chi fa il numero più alto, in caso di pareggio ritirano i dadi finchè uno dei due non vince. Chi perde il round deve affrontare i round successivi con un dado con una faccia in meno. Perde la partita chi arriva ad avere un dado con 0 facce. Mostra ad ogni round quante facce hanno i dadi dei due giocatori e mostra il risultato di ogni lancio dei dadi. Dichiara infine il vincitore.

2.1.64 Scrivi un programma che dato un numero n stampi una piramide come nel seguente esempio:

```
Con n = 3
  1
 121
12321
```

2.1.65 Scrivere un programma che legga due interi n e m con valori compresi tra 1 e 9, i cui prodotto sia inferiore a 35, e stampi m piramidi di altezza n. L'esempio si riferisce al caso n=3 e m=4.

```
  1     1     1     1
 121   121   121   121
12321123211232112321
```

2.1.66 Scrivi un programma che permetta di controllare i dati di input immessi dall'utente:

1. se l'utente inserisce un intero N compreso tra 1 e 10, il programma deve stampare a video il valore N^N,
2. se l'intero N e' compreso tra 11 e 20, il programma deve stampare a video la somma 1 + 2 + 3 +… + N
3. altrimenti deve dare un segnale di errore.

2.1.67 Si realizzi un programma che legga un intero N da tastiera, e stampi a video il risultato della seguente sommatoria:

$$\sum_{i=0}^{N} \left[(-1)^i \frac{4}{2 * i + 1} \right]$$

Una volta calcolato e stampato il valore a video, il programma deve chiedere un nuovo numero all'utente e ripetere il calcolo. Il programma deve terminare solo qualora l'utente inserisca un valore negativo.

2.1.68 Si progetti in C un programma che legge un float, rappresentante un ammontare di euro; di seguito il programma deve leggere un tasso d'interesse (in percentuale), ed un numero di anni. Il programma deve stampare, in uscita, per ogni anno, come l'ammontare cresce con gli interessi. Si ricordi che l'interesse si calcola con la seguente formula:

$C_{fin} = C_{in} (1 + (r/100))^N$

dove C_{fin} è il capitale finale, C_{in} è quello iniziale, r è l'interesse, e N rappresenta il numero di anni in cui si applicano gli interessi.

Ad esempio supponiamo che il capitale iniziale sia di 1000.0 €, con un tasso del 3%, per un periodo di 3 anni. L'output stampato deve avere all'incirca questo aspetto:

```
Capitale iniziale: 1000.00€
Dopo 1 anno: 1030.00 €
Dopo 2 anni: 1060.90 €
Dopo 3 anni: 1092.73 €
```

2.1.69 Realizzare un programma che legga da input un carattere dell'alfabeto e stampi a video il carattere stesso ed il suo valore ASCII. Il programma deve controllare che il carattere inserito sia compreso tra 'a' e 'z' o tra 'A' e 'Z' (in caso contrario si stampi un messaggio di errore). Dopo la stampa, il programma deve continuare a chiedere nuovi caratteri, finché l'utente non inserisce il carattere corrispondente al numero zero ('0'): in tal caso il programma termina.

2.1.70 Realizzare un programma che prende in input una sequenza di caratteri '0' e '1' e conta la lunghezza della più lunga sotto-sequenza di '0' di fila L'inserimento della sequenza termina quando si inserisce un carattere diverso da '0' e '1' A quel punto, si stampa a video il valore trovato

2.1.71 Realizzare un programma che prenda in input una sequenza di cifre (tra 1 e 9) e calcoli la somma massima fra le sottosequenze di cifre non decrescenti Il programma termina quando viene inserito lo 0

Esempio:

2	2	4	5	3	9	3	1	5	0
13				12		3	6		

2.1.72 Data in input una sequenza di n numeri (n dato dall'utente), stampare per ognuno il numero di quadrati perfetti minori di quel numero.

2.1.73 Conta quanti tiri di moneta devi fare prima di vedere uscire testa per 10 volte di fila.

2.1.74 Conta quanti tiri di un dado da 6 devi fare prima di vedere uscire lo stesso numero 5 volte di fila.

3. Strutture dati fondamentali: Liste Stringhe Matrici e Dizionari

3.1 Liste di soli numeri

3.1.1 Leggi n voti, caricali in una lista, calcolane la media, poi stampa tutti i numeri e la media.

Soluzione

```
numeri = []
n = int(input("Dimmi quanti numeri vuoi inserire:"))
media = 0
for i in range(n):
    numero = int(input(f"Scrivi il {i+1}° numero: "))
    while numero < 1 or numero > 10:
        numero = int(input("\nVoto non valido, reinseriscilo: "))
    numeri.append(numero)
    media += numero
media /= n
print("\nI numeri inseriti sono:")
for numero in numeri:
    print(numero, end=" ")
# print("\nLa media dei numeri è", round(media, 2))
print(f"\nLa media dei numeri è {media:.2f}")
```

3.1.2 Continua l'esercizio precedente calcolando e stampando la media dei soli voti sufficienti.

3.1.3 Carica una lista di numeri casuali, chiedi poi all'utente un numero e digli se questo numero è presente nella lista.

Soluzione

```
import random

# numeri = []
# for i in range(20):
#     numeri.append(random.randint(0,99))
numeri = [random.randint(0,99) for i in range(20)]

print(numeri)

numero = int(input("Dimmi un numero da cercare: "))

if numero in numeri:
    print("il numero è presente nella lista")
else:
    print("il numero non è presente nella lista")
```

3.1.4 Carica un array, di dimensione a piacere, di numeri casuali compresi in un range di numeri deciso dall'utente. Esegui poi le seguenti operazioni:

1. inverti l'ordine dei numeri contenuti nell'array,
2. Date due posizioni a e b dell'array (chiaramente controlla che a sia minore di b e che siano compresi tra 0 e N-1) stampa prima i numeri dell'array dalla posizione a alla posizione b poi i numeri dalla posizione b alla posizione a
3. Dato un numero positivo x qualsiasi stampa tutti i numeri dell'array dalla posizione 0 alla posizione x. Se x > N-1, dopo aver stampato tutto l'array ricomincia a stampare dalla posizione 0 e continua fino ad aver stampato x+1 numeri.

Soluzione

```
from random import randint


## Metodo C++ style
# lista = []
# for _ in range(10):
#     lista.append(randint(0,9))

## Metodo Python style
lista = [randint(0,9) for _ in range(10)]
print(lista)

# 1.    inverti l'ordine dei numeri contenuti nell'array,
lista = lista[::-1]
# lista.reverse()
print(lista)

# 2.    Date due posizioni a e b dell'array (chiaramente controlla che a sia minore di
# b e che siano compresi tra 0 e N-1) stampa prima i numeri dell'array dalla posizione a
# alla posizione b poi i numeri dalla posizione b alla posizione a
a = 2
b = 5
print(lista[a:b+1])
print(lista[b:a-1:-1])


# 3.    Dato un numero positivo x qualsiasi stampa tutti i numeri dell'array dalla
# posizione 0 alla posizione x. Se x > N-1, dopo aver stampato tutto l'array ricomincia
# a stampare dalla posizione 0 e continua fino ad aver stampato x+1 numeri.

x = 24
for i in range(x):
    pos = i % len(lista)
    print(lista[pos], end = " ")
print()
```

3.1.5 Scrivi una funzione a cui vengono passati come parametro un elemento e una lista di elementi, e che ti dica in output se l'elemento passato sia presente o meno nella lista.

Qualora l'elemento sia presente nella lista, la funzione dovrà inoltre comunicarci l'indice dell'elemento.

Soluzione

```
# 3.1.5 Scrivi una funzione a cui vengono passati come parametro un
# elemento e una lista di elementi, e che ti dica in output se l'elemento
```

```
# passato sia presente o meno nella lista.
# Qualora l'elemento sia presente nella lista, la funzione dovrà
# inoltre comunicarci l'indice dell'elemento.

from random import randint

# funzione che restituisce la posizione dell'elemento nella lista
# se non lo trova restituisce None cioè niente
def f1(cercare, lista):
    for i,el in enumerate(lista):
        if cercare == el:
            return i

# funzione che restituisce la lista delle posizioni dell'elemento nella lista
# se non lo trova mai restituisce una lista vuota
def f2(cercare, lista):
    posizioni = []
    for i,el in enumerate(lista):
        if cercare == el:
            posizioni.append(i)
    return posizioni

# a = []
# for _ in range(10):
#     a.append(randint(0,9))
a = [randint(0,9) for _ in range(10)]


el = 4

print(a)
print(f1(el, a))
print(f2(el, a))
```

3.1.6 Dato un array di numeri interi casuali, calcola e stampa il valore massimo e il valore minimo contenuti nell'array

Soluzione

```
import random

lista = [random.randint(0,9) for _ in range(10)]
print(lista)
max = lista[0]
min = lista[0]
for num in lista:
    if num > max:
        max = num
    if num < min:
        min = num

print(max, min)
```

Versioni un po' più elaborate che usano le funzioni

```
from random import randint

def miomin(a):
    min = None
    for el in a:
        if min == None or el < min:
            min = el
    return min
```

```python
def miomax(a):
    max = None
    for el in a:
        if max == None or el > max:
            max = el
    return max

def minmaxtupla(a):
    return (min(a), max(a))
def minmaxlista(a):
    return [min(a), max(a)]


a = [randint(0,9) for _ in range(10)]
print(a)
print("massimo:", max(a))
print("minimo:", min(a))
print("massimo:", miomax(a))
print("minimo:", miomin(a))
print(minmaxtupla(a))
print(minmaxlista(a))
```

3.1.7 Data un lista di numeri interi casuali, calcola e stampa media e mediana della lista

3.1.8 Riempi una lista di numeri casuali poi rappresenta la lista come un istogramma. Ad es.la lista [1, 0, 4, 6, 3, 9] viene rappresentata come segue

```
1: x
0:
4: xxxx
6: xxxxxx
3: xxx
9: xxxxxxxxx
```

Soluzione

```python
import random

def stampa_istogramma(lista):
    for num in lista:
        print(f"{num}: ", end="")
        for _ in range(num):
            print("x", end="")
        print()

def crea_istogramma(lista):
    ris = ""
    for num in lista:
        ris += f"{num}: "
        for _ in range(num):
            ris += "x"
        ris += "\n"
    return ris

lista = [random.randint(0,9) for _ in range(6)]
print("Stampa con la prima versione:")
stampa_istogramma(lista)
print("\nStampa con la seconda versione:")
print(crea_istogramma(lista))
```

3.1.9 Riempi una lista di numeri casuali tra 0 e 9 e calcola la somma massima fra le sottosequenze di cifre non decrescenti. Il valore deve essere calcolato da un'apposita funzione.

Esempio:

2	2	4	5	3	9	3	1	5	0
13				12		3	6		

la prima riga rappresenta la lista e la seconda i vari valori calcolati (viene restituito 13 che è il più grande)

Soluzione

```
import random

lista = [2, 2, 4, 5, 3, 9, 3, 1, 5]
lista = [random.randint(0,9) for _ in range(15)]
somma = lista[0]
max = somma
for i in range(1, len(lista)):
    if lista[i] >= lista[i-1]:
        somma += lista[i]
    else:
        somma = lista[i]

    if somma > max:
        max = somma

print(lista)
print(max)
```

3.1.10 Scrivi un programma che riempia un array di numeri casuali di una cifra in modo tale che ogni numero sia sempre maggiore o minore di entrambi i numeri adiacenti (ad esempio 1 8 3 5 2 7 4 5 3). L'array generato deve essere stampato poi ordinato e infine ristampato. Per implementare le funzionalità descritte realizza due funzioni: una che riceve un array e lo riempie di numeri casuali come descritto, una che riceve un array e un indicazione che dica se ordinare l'array in ordine crescente o decrescente (ad esempio una variabile booleana).

3.1.11 Dato un array di numeri interi casuali, stampa tutti i numeri superiori alla media partendo dal fondo

Soluzione

```
import random

A = [random.randint(0, 99) for _ in range(20)]
print(A)

media = sum(A) / len(A)

print(media)

# for num in reversed(A):
for num in A[::-1]:
    if num > media:
```

```
        print(num, end=" ")
```

3.1.12 Scrivere un paio di funzioni che calcolano perimetro e area di un triangolo rettangolo. Chiedere all'utente l'inserimento dei due cateti (i cateti vanno inseriti in una sola riga, in modo da forzare l'utilizzo della funzione split()). Richiamare le funzioni e stampare i risultati.

Variante: Scrivi un'unica funzione che calcola e restituisce i valori di perimetro e area

Soluzione

```
import os
os.system('cls')

import math

def perimetro(c1, c2):
    ipo = math.sqrt(c1**2 + c2**2)
    return (c1+c2+ipo)

def area(c1, c2):
    return c1*c2/2

def perimetro_area(c1, c2):
    ipo = math.sqrt(c1**2 + c2**2)
    p = (c1+c2+ipo)
    a = c1*c2/2
    return (p, a)


# strcateti = input("Inserisci le lunghezze dei cateti: ")
# # [scateto1, scateto2] = strcateti.split()
# strcateti = strcateti.strip()
# scateto1, scateto2 = strcateti.split()
# cateto1 = float(scateto1)
# cateto2 = float(scateto2)

cateto1, cateto2 = list(map(float, (input("Inserisci le lunghezze dei cateti: ").strip().split())))

p = perimetro(cateto1, cateto2)
a = area(cateto1, cateto2)

p, a = perimetro_area(cateto1, cateto2)
print("Perimetro=", p)
print("Area=", a)
```

3.1.13 Scrivi un programma che esegua le seguenti operazioni:

1. Chiedere con un ciclo all'utente di inserire i voti presi in matematica ed inserirli in una lista, uscire dal ciclo quando l'utente inserisce un valore negativo;
2. richiamare la funzione MediaVoti1() che restituisce la media dei voti presenti nella lista;
3. richiamare la funzione MediaVoti2() che restituisce la media di tutti i voti escludendo il più basso e il più alto. (si consiglia di usare il metodo sort() delle liste);

Variante: modificare l'esercizio in modo che i voti vengano inseriti in una sola riga con un solo comando input.

Soluzione

```python
def MediaVoti1(voti):
    media = 0
    for voto in voti:
        media += voto
    media /= len(voti)
    return media

def MediaVoti2(voti):
    # trovo massimo e minimo
    imin, imax = 0, 0
    for i, voto in enumerate(voti):
        if voto > voti[imax]:
            imax = i
        if voto < voti[imin]:
            imin = i

    #calcolo la media
    media = 0
    for i, voto in enumerate(voti):
        if i != imax and i != imin:
            media += voto
    media /= len(voti) - 2
    return media

def MediaVoti3(voti):
    voti.sort()
    media = 0
    for i in range(1, len(voti)-1):
        media += voti[i]
    media /= len(voti) - 2
    return media


voti = []
while True:
    voto = float(input("Inserisci un voto: "))
    if voto == 0 or voto > 10:
        print("Voto non valido")
        continue
    if voto < 0:
        break
    voti.append(voto)

media = MediaVoti1(voti)
print("Media normale: ", media)
media = MediaVoti2(voti)
print("Media escuso max e min versione 1: ", media)
media = MediaVoti3(voti)
print("Media escuso max e min versione 2: ", media)
```

3.1.14 Scrivere una funzione che con un ciclo while chiede all'utente di inserire i 4 voti delle ultime verifiche di matematica (ed inserirli in una lista). Se il voto è inferiore a 1 o seperiore a 10, non va considerato. Solo quando l'utente ha inserito 4 voti corretti, uscire dal ciclo. Stampare la media dello studente in Matematica

3.1.15 Supponendo di avere una classe di 20 alunni (numerati nel registro da 1 a 20), il docente di matematica decide di dedicare i 5 appuntamenti di una settimana ad interrogare 4 studenti casuali, ogni giorno. Sapendo che ogni studente può essere interrogato al massimo due volte, stampare in ognuno dei 5 giorni i quattro interrogati, stampare quali studenti non sono mai stati interrogati.

3.1.16 Scrivere una funzione guessMyAge() che deve indovinare la mia età e stampare in quanti tentativi ha indovinato la mia età.

Nel programma principale chiedere all'utente di inserire la sua età e poi invoca la funzione guessMyAge() per indovinare il valore inserito.

3.1.17 Dichiarare due liste:

Con opportuni cicli while, riempire le due liste con 5 valori CRESCENTI casuali compresi tra 1 e 100

(naturalmente il primo valore non può superare 96, il secondo 97 ... l'ultimo 100)

Stampare il contenuto delle due liste

Stampare il quinto valore (in ordine di grandezza) considerando i dieci valori presenti nelle due liste

Soluzione

```
# Dichiarare due liste:
# Con opportuni cicli while, riempire le due liste con 5 valori CRESCENTI casuali compresi tra 1
e 100
# (naturalmente il primo valore non può superare 96, il secondo 97 ... l'ultimo 100)
# Stampare il contenuto delle due liste
# Stampare il quinto valore (in ordine di grandezza) considerando i dieci valori presenti nelle
due liste

import random

min, max = 1, 100
nvalori = 5

# # primo metodo che però non dà una distribuzione casuale dei numeri
# lista1 = [0]*nvalori
# lista2 = [0]*nvalori
# lista1[0] = random.randint(min, max-nvalori+1)
# lista2[0] = random.randint(min, max-nvalori+1)
# for i in range(1, nvalori):
#     lista1[i] = random.randint(lista1[i-1]+1, max-nvalori+1+i)
#     lista2[i] = random.randint(lista2[i-1]+1, max-nvalori+1+i)

# secondo metodo che dà una distribuzione casuale
lista1 = []
lista2 = []
```

```python
for _ in range(nvalori):
    num = random.randint(min, max)
    while num in lista1:
        num = random.randint(min, max)
    lista1.append(num)

    num = random.randint(min, max)
    while num in lista2:
        num = random.randint(min, max)
    lista2.append(num)

lista1.sort()
lista2.sort()

print(lista1)
print(lista2)


## seconda parte in cui stampo il quinto valore in ordine tra le due liste

# metodo 1, più comodo ma più lento
lista3 = lista1 + lista2
lista3.sort()
print(lista3)
print(lista3[4])

# metodo 2, decisamente meno comodo ma più veloce
i, j = 0, 0
for _ in range(5):
    if lista1[i] > lista2[j]:
        ris = lista2[j]
        j += 1
    else:
        ris = lista1[i]
        i += 1
print(ris)
```

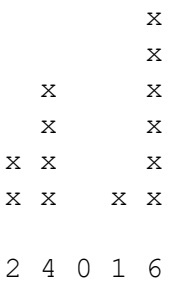
3.1.18 Letta in input una sequenza di numeri interi positivi memorizzarla in una lista. Costruire una seconda lista contenente soltanto gli elementi della prima lista che non siano numeri primi. Stampare la seconda lista.

3.1.19 Date due liste di numeri ordinate (ottienile come vuoi), costruire una terza lista di numeri interi ordinata, ottenuta mediante la “fusione” delle prime due. Stampare la lista.

3.1.20 Leggere in input una sequenza di numeri interi ordinati in ordine crescente. Dopo aver memorizzato la sequenza in una lista, inserire nella posizione corretta all’interno della lista, tutti i numeri mancanti. Stampare in output la lista. Non devono essere usate altre iste o array di appoggio.

Esempio: supponiamo che sia fornita in input la sequenza 4, 7, 8, 9,15,17,21. Dopo aver memorizzato gli elementi nella lista 4, 7, 8, . . . 21, vengono inseriti i numeri mancanti, ottenendo la lista composta dagli elementi 4, 5, 6, 7, 8, . . . 19, 20, 21.

3.1.21 Riempi una lista di 10 numeri casuali poi stampa un istogramma basato sui numeri inseriti come nel seguente esempio:



3.1.22 Riempi una lista di n numeri casuali (n scelto da te), poi verifica se gli elementi della lista sono disposti in ordine crescente.

Variante: scrivi e usa una funzione che restituisca la risposta alla domanda

Continua l'esercizio creando e stampando liste fino a che non ne viene stampata una che rispetta la condizione.

Continua l'esercizio calcolando mediamente quante liste vengono create prima di crearne una che rispetti la condizione; in questo caso puoi stampare solo il numero medio di volte e non ogni lista (dovresti stamparne un'enormità).

3.2 Stringhe

3.2.1 Scrivi una funzione che, dato un carattere in ingresso, restituisca in output il codice ASCII associato al carattere passato.

Soluzione

```
def trova_ascii():
    carattere = input("Inserisci il carattere che ti interessa convertire: ")
    valore = ord(carattere)
    output = f"Il valore ASCII associato a '{carattere}' è {valore}"
    return output
```

3.2.2 Stampa una stringa di una sola riga e una stringa formata da più righe

Soluzione

```
stringa1 = "Ciao a tutti!"

stringa2 = """
Ciao, come va?
Bene grazie
"""

stringa3 = "Ciao, come va?\nBene grazie"

print(stringa1)
```

```
print(stringa2)
print(stringa3)
```

3.2.3 Scrivi una funzione (puoi farlo anche senza la funzione se non le hai ancora imparate) a cui viene passato un carattere come parametro, e risponde dicendo se il carattere è o meno una vocale.

Soluzione

```
def f(c):
    if c in "aeiouAEIOU":
        return True
    else:
        return False

print(f("a"))
print(f("E"))
print(f("f"))
```

3.2.4 Leggi una stringa da input e poi comunica all'utente di quante lettere è composta la stringa. Il conteggio deve essere fatto in due modi diversi, usando la funzione len e contando i caratteri uno alla volta.

Soluzione

```
stringa = input("Scrivi qualcosa: ")

# print(stringa)

numero_lettere = 0
for lettera in stringa:
    numero_lettere += 1

print(f"Contando i caratteri uno alla volta ho contato {numero_lettere} lettere")

numero_lettere = len(stringa)
print(f"\nContando i caratteri con la funzione len ho contato {numero_lettere} lettere")
```

3.2.5 Data la seguente stringa stampa:

stringa = " Vediamo come me la CAVO " # attento che ci sono spazi extra alla fine e all'inizio

1. La stringa così com'è
2. La stringa rimuovendo eventuali spazi iniziali e finali
3. La stringa nella versione con tutte le lettere minuscole
4. La stringa nella versione con tutte le lettere maiuscole

5. La stringa con la sola prima lettera maiuscola
6. La stringa con le sole prime lettere di ogni parola maiuscole
7. La seconda parola della stringa
8. La stringa ottenuta sostituendo tutte le 'a' con delle 'e'

Soluzione

```
stringa = "   Vediamo come me la CAVO   "   # attento che ci sono spazi extra alla fine e
all'inizio


#     1. La stringa così com'è
print(stringa)


#     2. La stringa rimuovendo eventuali spazi iniziali e finali
# uso quindi le funzioni
# strip(): returns a new string after removing any leading and trailing whitespaces including
tabs (\t). - comprende le altre due
# rstrip(): returns a new string with trailing whitespace removed. It’s easier to remember as
removing white spaces from “right” side of the string.
# lstrip(): returns a new string with leading whitespace removed, or removing whitespaces from
the “left” side of the string.

# print(len(stringa))
stringa = stringa.strip()
# print(len(stringa))
print(stringa)


#     3. La stringa nella versione con tutte le lettere minuscole
print(stringa.lower())


#     4. La stringa nella versione con tutte le lettere maiuscole
print(stringa.upper())


#     5. La stringa con la sola prima lettera maiuscola
print(stringa.capitalize())

#     6. La stringa con le sole prime lettere di ogni parola maiuscole
lista = stringa.split()
lista = list(map(str.capitalize, lista))
print(" ".join(lista))


#     7. La seconda parola della stringa
print(stringa.split()[1])


#     8. La stringa ottenuta sostituendo tutte le 'a' con delle 'e'
print(stringa.replace('a', 'e')) # così funziona solo con le minuscole
tmp = stringa.replace('a', 'e')
tmp = tmp.replace('A', 'E')
print(tmp)
```

3.2.6 Leggi una stringa da input e poi stampala in due modi diversi, la prima volta tutta insieme poi stampando una lettera per volta separando le lettere con uno spazio.

Soluzione

```
import os
os.system('cls')

stringa = input("Scrivi qualcosa: ")
print(stringa)
for lettera in stringa:
    print(lettera, end=" ")
print("\n\n")
```

3.2.7 Leggi una stringa da input e poi stampala al contrario, cioè partendo dal fondo ma non modificandone il contenuto.

Soluzione

```
import os
os.system('cls')

stringa = input("Scrivi qualcosa: ")
print(stringa)
for lettera in reversed(stringa):
    print(lettera, end="")
print("\n\n")
```

3.2.8 Leggi una stringa da input e poi modificala in modo da ottenere la stringa inversa (ad esempio “ciao” diventa “oaic”). Stampa infine la stringa.

Soluzione

```
import os
os.system('cls')

stringa = input("Scrivi qualcosa: ")
print(stringa)
stringa = stringa[::-1]
print(stringa)
print("\n\n")
```

3.2.9 Dichiara e inizializza due stringhe con valori a piacere, stampale. Scambia il contenuto delle due stringhe in modo che la prima contenga il contenuto della seconda e viceversa, poi ristampale.

3.2.10 Leggi da input due stringhe, se le due stringhe sono uguali comunica che sono uguali altrimenti scrivi quella che viene prima in ordine alfabetico

Soluzione

```
import os
os.system('cls')

stringa1 = input("Scrivi la prima stringa: ")
stringa2 = input("Scrivi la seconda stringa: ")

if stringa1 == stringa2:
    print("Le stringhe sono uguali\n\n")
else:
    prima = (stringa1, stringa2)[stringa1 > stringa2]
    print(f"La stringa che viene prima in ordine alfabetico è: {prima}\n\n")
print("\n\n")
```

3.2.11 Leggi usando un'unica funzione input tre parole. Indica poi qual è la parola più lunga, la parola che viene prima in ordine alfabetico e la parola che contiene più vocali. (Questo esercizio è molto bello se si usa la funzione sort)

Soluzione

```
def conta_vocali(s):
    ris = 0
    for l in s:
        if l in "aeiou":
            ris += 1
    return ris

parole = input("Scrivi tre parole:")
parole = parole.split()
parole = list(map(str.strip, parole))

print(parole)

parole.sort(key=lambda el: len(el))
print("La parola più lunga:", parole[-1])

parole.sort(key=lambda el: str.lower(el))
print("La parola che viene prima in ordine alfabetico:", parole[0])

parole.sort(key=conta_vocali)
print("La parola che contiene più vocali:", parole[-1])
```

3.2.12 Data una lista di stringhe, crea e poi stampa un'unica stringa contenente tutte le stringhe della lista separate da "; " (puntoevirgola e spazio)

Ad esempio dalla lista

["casa", "sedia", "cavallo", "andare in bici", "uscire di casa", "che bello!"]

viene creata la stringa

"casa; sedia; cavallo; andare in bici; uscire di casa; che bello!"

Soluzione

```
lista = ["casa", "sedia", "cavallo", "andare in bici", "uscire di casa", "che bello!"]
stringa = "; ".join(lista)
print(stringa)
```

3.2.13 Scrivi e utilizza una funzione che riceve una stringa e un carattere e che stampi tutto il contenuto della stringa andando a capo ogni volta che trova il carattere dato (il carattere rappresenta un separatore della stringa in sottostringhe).

Variante: la funzione non stampa la stringa ma la restituisce e basta (la stampa falla fuori dalla funzione)

Soluzione

```
def fbase(stringa, carattere):
    for lettera in stringa:
        print(lettera, end = "")
        if lettera == carattere:
            print()

def fvar(stringa, carattere):
    ris = ""
    for lettera in stringa:
        ris += lettera
        if lettera == carattere:
            ris += "\n"
    return ris

fbase("Ciao a tutti, oggi facciamo esercizi sulle stringhe.", 'o')
print("\n")
print(fvar("Ciao a tutti, oggi facciamo esercizi sulle stringhe.", 'o'))
```

3.2.14 Scrivi un programma che contenga in un array la seguente stringa “Ciao a tutti, mi chiamo Francesco\nOggi vi voglio parlare delle stringhe, siete d’accordo?” e che la stampi in modo tale che quando trova una virgola va a capo(oltre a stampare la virgola) e quando trova un a capo, va a capo due volte.

3.2.15 data una stringa, trasforma la stringa in modo tale che le lettere in posizione pari siano maiuscole e le altre minuscole,

ad esempio “Ciao a tutti, come state?” diventa “CiAo a tUtTi, CoMe sTaTe?”

Soluzione

```
# data una stringa, trasforma la stringa in modo tale che
# le lettere in posizione pari siano maiuscole e le altre
# minuscole

frase = "Ciao a tutti, come state?"


# metodo 1

lista = list(frase)

for i in range(1,len(lista),2):
```

```
    lista[i] = lista[i].lower()

for i in range(0,len(lista),2):
    lista[i] = lista[i].upper()

# poi devo unire tutti gli elementi e ricostruire la frase

# 1.1 metodo farlocco di ricostruzione
modificata = ""
for el in lista:
    modificata += el
print("Metodo 1.1:", modificata)

# 1.2 metodo veloce di ricostruzione (funzione join)
modificata = "".join(lista) # non puoi fare str(lista)
print("Metodo 1.2:", modificata)



# metodo 2 più veloce
mod = ""
for i, lettera in enumerate(frase):
    if i % 2 == 0:
        mod += frase[i].upper()
    else:
        mod += frase[i].lower()

print("Metodo 2  :", mod)
```

3.2.16 Scrivi due funzioni che implementino il cifrario di Cesare cioè un cifrario che per cifrare un testo trasforma ogni lettera nella lettera che si trova tre posizioni più avanti nell'alfabeto. Le due funzioni sono le funzioni "cifra" e "decifra". Scrivi infine un programma che permetta di inserire un testo da cifrare o decifrare a scelta.

Soluzione

```
import string

minuscole = string.ascii_lowercase
maiuscole = string.ascii_uppercase

def cifra(testo, chiave):
    cifrato = ""
    for i in range(len(testo)):
        if testo[i] in minuscole:
            offset = ord('a')
        elif testo[i] in maiuscole:
            offset = ord('A')
        else:
            offset = 0
        if offset != 0:
            cifrato += chr(((ord(testo[i]) - offset + chiave) % len(minuscole)) + offset)
        else:
            cifrato += testo[i]
    return cifrato

def decifra(testo, chiave):
    decifrato = ""
    for i in range(len(testo)):
        if testo[i] in minuscole:
            offset = ord('a')
```

```
        elif testo[i] in maiuscole:
            offset = ord('A')
        else:
            offset = 0
        if offset != 0:
            decifrato += chr(((ord(testo[i]) - offset - chiave) % len(minuscole)) + offset)
        else:
            decifrato += testo[i]
    return decifrato

testo = input("Inserisci un testo: ")
chiave = int(input("Inserisci la chiave: "))
scelta = int(input("Vuoi cifrare o decifrare? scrivi 1 per cifrare, 2 per decifrare: "))
while scelta != 1 and scelta != 2:
    scelta = input("Risposta non prevista, scrivi 1 per cifrare, 2 per decifrare: ")
if scelta == 1:
    testo = cifra(testo, chiave)
    print("Il messaggio cifrato è:")
    print(testo)
else:
    testo = decifra(testo, chiave)
    print("Il messaggio decifrato è:")
    print(testo)
```

3.2.17 In Svezia, i bambini giocano spesso utilizzando un linguaggio un po' particolare detto "rövarspråket", che significa "linguaggio dei furfanti": consiste nel raddoppiare ogni consonante di una parola e inserire una "o" nel mezzo. Ad esempio la parola "mangiare" diventa "momanongogiarore".

Se sai usare le funzioni, scrivi una funzione in grado di tradurre una parola o frase passata tramite input in "rövarspråket" se no implementa il programma senza funzioni.

3.2.18 Scrivi una funzione generatrice di password.

La funzione deve generare una stringa alfanumerica di 8 caratteri qualora l'utente voglia una password semplice, o di 20 caratteri qualsiasi (anche simboli) qualora desideri una password più complicata.

Soluzione

```
import string, random

alfanum = string.ascii_letters + string.digits
tutti = alfanum + string.punctuation

# print(alfanum)

def gen_pass(diff = "facile"):
    if diff == "facile":
        caratteri = alfanum
        lung = 8
    elif diff == "difficile":
        caratteri = tutti
        lung = 20
    else:
        return None

    password = "".join([random.choice(caratteri) for _ in range(lung)])
```

```
    return password


print(gen_pass())
print(gen_pass(diff="difficile"))
print(gen_pass(diff="difficilesdfdsf"))


# alternativa con i booleani

insiemesemplice = string.ascii_letters + string.digits
insiemecomplicato = insiemesemplice + string.punctuation

def f(complicato = False):
    ris = ""
    if not complicato:
        for _ in range(8):
            ris += random.choice(insiemesemplice)
    else:
        for _ in range(20):
            ris += random.choice(insiemecomplicato)
    return ris

print(f())
print(f(True))
```

3.2.19 Un indirizzo MAC (Media Access Control address) è un indirizzo univoco associato dal produttore, a un chipset per comunicazioni wireless (es WiFi o Bluetooth), composto da 6 coppie di cifre esadecimali separate da due punti.

Un esempio di MAC è 02:FF:A5:F2:55:12.

Scrivi una funzione genera_mac che generi degli indirizzi MAC pseudo casuali.

Soluzione

```
from random import choice

def genera_mac_meno_bello():
    mac = ""
    caratteri = "0123456789ABCDEF"
    for i in range(6):
        mac += choice(caratteri)
        mac += choice(caratteri)
        if i < 5:
            mac += ':'
    return mac

def genera_mac_piu_bello():
    caratteri = "0123456789ABCDEF"
    mac = [choice(caratteri)+choice(caratteri) for _ in range(6)]
    print(mac)
    mac = ":".join(mac)
    return mac

print(genera_mac_piu_bello())
```

3.2.20 Scrivi e utilizza una funzione che riceve due stringhe e sia in grado di dire se la stringa più piccola è una sottostringa di quella più grande. Nel tuo programma il main deve creare

due stringhe di lunghezza a piacere contenenti lettere casuali (solo lettere maiuscole e minuscole senza altri simboli o cifre; la funzione non cambia)

Variante: invece di dire se la stringa è contenuta nell'altra stringa restituire la posizione di partenza della sottostringa trovata.

Ad esempio "mi c" è sottostringa di "Ciao a tutti mi chiamo Francesco"

Soluzione

```
# 3.2.20 - Scrivi e utilizza una funzione che riceve due stringhe e sia in grado di dire
# se la stringa più piccola è una sottostringa di quella più grande.
# Nel tuo programma il main deve creare due stringhe di lunghezza a piacere
# contenenti lettere casuali (solo lettere maiuscole e minuscole senza altri simboli
# o cifre; la funzione non cambia)

# Variante:
# invece di dire se la stringa è contenuta nell'altra stringa restituire la
# posizione di partenza della sottostringa trovata.

# Ad esempio "mi c" è sottostringa di "Ciao a tutti mi chiamo Francesco"

import os
os.system('cls')

def sottostringa(piccola, grande):
    if piccola in grande:
        return True
    else:
        return False

def sottostringa2(piccola, grande):
    for i in range(len(grande)-len(piccola)):
        if piccola == grande[i:i+len(piccola)]:
            return True

    return False

def posSottostringa(piccola, grande):
    for i in range(len(grande)-len(piccola)):
        if piccola == grande[i:i+len(piccola)]:
            return i

    return -1

def posSottostringaRicorsivo(piccola, grande):
    if len(grande) < len(piccola):
        return -1

    if piccola == grande[:len(piccola)]:
        return 0

    resto = posSottostringaRicorsivo(piccola, grande[1:])
    if resto != -1:
        return 1 + resto
    else:
        return -1

def posSottostringaRicorsivo2(piccola, grande, i = 0, f = -1):
    if f < 0:
        f = len(grande)+f

    if f-i+1 < len(piccola):
        return -1

    if piccola == grande[i:i+len(piccola)]:
```

```
        return i

    return posSottostringaRicorsivo2(piccola, grande, i+1, f)

def sottostringaRicorsivo(piccola, grande):
    if len(grande) < len(piccola):
        return False

    if piccola == grande[:len(piccola)]:
        return True

    return sottostringaRicorsivo(piccola, grande[1:])

print('sottostringa')
print(sottostringa('ia', 'Ciao'))
print(sottostringa('asd', 'Ciao'))
print()

print('posSottostringa')
print(posSottostringa('ia', 'Ciao'))
print(posSottostringa('asd', 'Ciao'))
print()

print('sottostringaRicorsivo')
print(sottostringaRicorsivo('ia', 'Ciao'))
print(sottostringaRicorsivo('asd', 'Ciao'))
print()

print('posSottostringaRicorsivo')
print(posSottostringaRicorsivo('ia', 'Ciao'))
print(posSottostringaRicorsivo('asd', 'Ciao'))
print()

print('posSottostringaRicorsivo2')
print(posSottostringaRicorsivo2('ia', 'Ciao'))
print(posSottostringaRicorsivo2('asd', 'Ciao'))
print()
```

3.2.21 Scrivi un programma che chieda all'utente nome e cognome e poi:

1. verifichi che il nome e il cognome siano stati scritti con le iniziali maiuscole (devi controllare se la prima lettera ha i valori corretti secondo la tabella ASCII, i caratteri sono come numeri);
2. crei una stringa che contenga le generalità dell'utente (nome e cognome insieme);
3. stampi le generalità al contrario;
4. chieda nuovamente nome e cognome e verifichi se sono state riscritte le stesse cose di prima.

3.2.22 Data la seguente lista che contiene alcuni paesi europei

eu = ["italia", "germania", "svizzera", "francia", "spagna", "regnoUnito", "belgio"]

e la seguente lista di stringhe (fatta da coppie città paese)

s = [

"madrid spagna istanbul turchia bruxelles belgio ostenda belgio hannover germania berlino germania aleppo siria",

```
    "roma italia milano italia berna svizzera anversa belgio toledo spagna londra regnoUnito",
    "damasco siria  charleroi belgio munchen germania homs siria preston regnoUnito teheran iran",
    "bilbao spagna brema germania zurigo svizzera torino italia beirut libano marsiglia francia",
    "gerusalemme israele ankara turchia kobane siria colonia germania francoforte germania lione francia",
    "lugano svizzera siviglia spagna mosul iraq dortmund germania bordeaux francia manchester regnoUnito"
]
```

stampare i nomi dei due paesi europei, con il maggior numero di città nell'elenco

3.2.23 Le seguenti stringhe della lista, contengono il nome di una regione e alcune delle sue province:

```
ligureLombardo =
[
    "Lombardia Monza Milano Como Lecco Varese",
    "Liguria Genova",
    "Lombardia Pavia Lodi Cremona Mantova",
    "Liguria LaSpezia Savona Imperia",
    "Lombardia Brescia Bergamo Sondrio"
]
```

1. Riconoscere e memorizzare opportunamente le 12 province lombarde.
2. Stampare le 12 province in ordine alfabetico

3.2.24 Scoprire le città presenti in entrambi i tour di Ligabue e di Vasco

```
tourLigabue =
[
    "Prato 10 giugno",
    "Lucca 12 giugno",
    "Arezzo 15 giugno",
    "Siena 16 giugno",
    "Firenze 17-19 giugno",
    "Viareggio 20 giugno",
    "LaSpezia 22 giugno",
    "Genova 23-25 giugno",
    "Savona 27 giugno"
]

tourVasco =
[
    "Modena 3-5 giugno",
    "Parma 7 giugno",
    "Pavia 9 giugno",
    "Genova 10-11 giugno",
```

```
    "Carrara 14 giugno",
    "Lucca 16 giugno",
    "Pisa 17 giugno",
    "Livorno 18 giugno",
    "Firenze 20-23 giugno",
    "Arezzo 25 giugno",
    "Perugia 27 giugno"
]
```

3.2.25 La seguente lista descrive i più celebri virologi Italiani nell'epoca covid

(Nome e Cognome - Ospedale - città - (O) Ottimista (P) Pessimista

```
covid= [
  "Roberto Burioni - San Raffaele - Milano (P)",
  "Matteo Bassetti - San Martino - Genova (O)",
  "Alberto Zangrillo - San Raffaele - Milano (O)",
  "Massimo Galli - Luigi Sacco - Milano (P)",
  "MariaRita Gismondo - Luigi Sacco - Milano (O)",
  "Andrea Crisanti - Università di Padova - Padova (P)",
  "Ilaria Capua - One Health - Florida (P)",
  "Antonella Viola - Città della Speranza - Padova (P)",
  "Fabrizio Pregliasco - Galeazzi - Milano (O)",
  "Walter Ricciardi - Policlino Gemelli - Roma (P)",
  "Franco Locatelli - Bambin Gesù - Roma (P)"
]
```

1. stampare nome e cognome dei virologi ordinati per cognome (si consiglia si usare la split() con il separatore "-";
2. stampare il numero di virologi ottimisti e di quelli pessimisti.

si ricorda che il metodo find() delle stringhe, restituisce -1 se l'elemento non è trovato.

3.2.26 Per ognuno dei seguenti numeri, stampare se è presente la cifra "tre", e nel caso quante volte è presente:

lista = [

"trentatre", "milletrecento","milequattrocentonovantadue", "trecentotredici"

]

3.2.27 Scrivi una semplice funzione rimario, a cui viene passato un elenco di parole come parametro e che riceva una parola inserita dall'utente tramite la funzione input.

La funzione rimario dovrà confrontare la parola inserita dall'utente con quelle presenti nell'elenco passato, alla ricerca di rime, intese come parole le cui ultime 3 lettere siano uguali alla parola inserita dall'utente.

Le rime dovranno essere quindi mostrate in output dall'utente.

3.2.28 Genera parole di 4 lettere minuscole a caso fino a che non ne ottieni una palindroma. Stampa tutte le parole ottenute. Il controllo sulla parola deve essere fatto con una funzione da te scritta che restituisca true se la parola è palindroma.

Soluzione

```
# 3.2.28 - Genera parole di 4 lettere minuscole a caso fino a che
# non ne ottieni una palindroma. Stampa tutte le parole ottenute. Il controllo
# sulla parola deve essere fatto con una funzione da te scritta che restituisca
# true se la parola è palindroma.

from random import choice
from string import ascii_lowercase
import os
os.system('cls')


def palindroma(parola):
    if len(parola) == 0:
        return None

    # for i in range(len(parola)/2):
    #     if parola[i] != parola[-1-i]:
    #         return False
    # return True

    if parola == parola[::-1]:
        return True
    else:
        return False

lettere = ascii_lowercase

# parola = ""
# while not palindroma(parola):
#     parola = "".join([choice(lettere) for _ in range(4)])
#     print(parola)

# oppure
while True:
    parola = "".join([choice(lettere) for _ in range(4)])
    print(parola)
    if palindroma(parola):
        break
```

3.3 Espressioni regolari

3.3.1 Per ognuno dei seguenti numeri, stampare se è presente la cifra "tre", e nel caso quante volte è presente:

lista = [

"trentatre", "milletrecento","milequattrocentonovantadue", "trecentotredici"

]

Soluzione

```
import re # iniziali di regular expressions

lista = ["trentatre", "milletrecento","milequattrocentonovantadue", "trecentotredici"]

# classico 1
trovare = "tre"
for el in lista:
    nmatches = 0
    for i in range(len(el)-len(trovare)+1):
        match = True
        for j in range(len(trovare)):
            if el[i+j] != trovare[j]:
                match = False
                break
        if match:
            nmatches += 1
    print(el, nmatches)

print("\n")

# classico 2
trovare = "tre"
for el in lista:
    match = 0
    for i in range(len(el)-len(trovare)+1):
        if trovare == el[i:i+len(trovare)]:
            match += 1
    print(el, match)

print("\n")

# con le espressioni regolari
esp = r"tre"
for el in lista:
    ris = re.findall(esp, el)
    print(el, len(ris))
```

3.3.2 Scrivi una semplice funzione rimario, a cui viene passato un elenco di parole come parametro e che riceva una parola inserita dall'utente tramite la funzione input.

La funzione rimario dovrà confrontare la parola inserita dall'utente con quelle presenti nell'elenco passato, alla ricerca di rime, intese come parole le cui ultime 3 lettere siano uguali alla parola inserita dall'utente.

Le rime dovranno essere quindi mostrate in output dall'utente.

3.3.3

3.4 Liste miste

3.4.1 La seguente lista, contiene i nomi di una materia e i voti conseguiti

(il numero di voti, è diverso a seconda della materia)

lista = ["matematica", 5.5, 6, 4, "storia", 8,9, "italiano", 7, 7.5, 7, "inglese", 6.5 ,6, 5.5, "fisica", 6, 5.5]

1) stampare per ogni materia la media dei voti

Si può usare la funzione type(x) per verificare il tipo della variabile x

Soluzione

```
lista = ["matematica", 5.5, 6, 4, "storia", 8, 9, "italiano",
         7, 7.5, 7, "inglese", 6.5, 6, 5.5, "fisica", 6, 5.5]

inizio = True
for el in lista:
    if type(el) == str:
        if (not inizio):
            media /= nvoti
            print(f"{materia}: {media:.2f}")

        inizio = False
        materia = el
        media = 0
        nvoti = 0

    else:
        media += el
        nvoti += 1

media /= nvoti
print(f"{materia}: {media:.2f}")
```

3.4.2 la seguente lista di stringhe descrive una serie di operazioni

listaOp = ["15.5 + 13.8" ,

"16.4 / 2",

"7 * 2", "6.5 - 4.1"],

stamparne i risultati di queste operazioni (dove sono presenti due operandi separati da un operatore)

Soluzione

```
listaOp = ["15.5 + 13.8",
           "16.4  /  2",
           "7  *  2",
           "6.5  - 4.1"]

for riga in listaOp:
    riga = riga.split()
    a = float(riga[0])
```

```
    op = riga[1]
    b = float(riga[2])
    if op == '+':
        print(f"{a} + {b} = {a+b:.2f}")
    elif op == '-':
        print(f"{a} - {b} = {a-b:.2f}")
    elif op == '*':
        print(f"{a} * {b} = {a*b:.2f}")
    elif op == '/':
        print(f"{a} / {b} = {a/b:.2f}")
    else:
        print("Operazione sconosciuta.")
```

3.4.3 5 alunni (denominati alunno_1, alunno_2 alunno_3 alunno_4 alunno_5)

hanno ricevuto in storia rispettivamente 2, 3, 4, 3, 4 voti

e questa informazione è contenuta nella listaN:

listaN = [2, 3, 4, 3, 4]

la seconda lista riporta i sedici voti

listaVoti = [8.5, 7, 6, 4.5, 7, 5, 5.5, 7, 6.5, 9, 8.5, 7, 7, 6.5, 5, 6.5]

1. Stampare il nome dell'alunno che ha preso il voto più alto (alunno_3 con voto 9)
2. stampare i nomi delli alunni e la loro media in storia, ordinati in base alla media

3.4.4 Scrivi un programma che simuli il gioco del Black Jack.

Una mano del gioco è gestita da una funzione che devi chiamare manoBlackJack() che riceve come parametro una lista contenente la puntata fatta da ogni giocatore partecipante (implicitamente quindi anche il numero di giocatori) e restituisce la quantità di denaro da dare alla fine ad ogni giocatore (0 se perde, il doppio di quanto puntato se vince o lo stesso di prima se pareggia)

La funzione deve simulare la gestione di un mazzo di carte con una opportuna struttura dati. Per semplificare il gioco la funzione manoBlackJack() utilizza un nuovo mazzo mischiato ad ogni mano (ben lontano dalla realtà).

La funzione shuffle consente di mescolare il mazzo (restituisce un mazzo di 52 carte pieno e mischiato)

Sia ai giocatori che al banco, vengono assegnate inizialmente 2 carte pescate dal mazzo, entrambe scoperte per i giocatori, una coperta per il banco.

I giocatori giocano a turno e viene chiesto loro inserire la propria decisione che può essere solo di due tipi "vedo" o "continuo". Il singolo giocatore continua a pescare carte finchè perde (somma più di 21) o decide di fermarsi. Finito il turno dei giocatori il banco gioca automaticamente secondo le regole standard del gioco (pesca fintanto che raggiunge meno di 17, da 17 in su si ferma):

Il programma principale chiede se un giocatore si vuole aggiungere alla partita e con quale puntata, finchè si aggiungono giocatori continua a chiederlo (massimo 7 giocatori), poi chiama la funzione per giocare una mano e alla fine chiede se i giocatori vogliono rimanere

(se hanno perso ne devono mettere altri) o vogliono ritirare soldi dal tavolo (tutti e se ne vanno, una parte e rimangono). Il programma poi chiede nuovamente se ci sono nuovi giocatori che vogliono partecipare (sempre massimo 7)

Il programma termina quanto tutti i giocatori lasciano il tavolo.

Varianti per completare il programma

1. Gestisci il mazzo in maniera realistica, queste sono le regole:

   "Il Blackjack classico (o francese) si gioca con sei mazzi di carte da poker (o francesi), per un totale di 312 carte. In mezzo ad esse vengono mescolate una carta di plastica e quando, durante il gioco, si arriva ad essa verranno rimescolate tutte le carte."

   "Normalmente una taglia o sabot che dir si voglia è composta da sei mazzi di carte,all'inizio del gioco vengono tolte le prima 5 carte e si mette una carta nera che normalmente viene messa verso la fine della taglia e tiene fuori gioco un trenta quaranta carte(questo nasce al fine di dare filo da torcere a chi usa la conta),si mescolano le carte con l'uscita della carta nera. oggi però è sempre più in uso l'utilizzo delle mescolatrici,delle macchinette nelle quali il croupier mette le carte della mano precedente e che automaticamente vengono mescolate e rimesse in gioco,sono in uso presso i casino svizzeri e quello di montecarlo,altri non saprei,mentre sia a saint vincent che a san remo vengono ancora oggi mescolate a mano,ovviamente con le mescolatrice la conta diventa impossibile.

   Tutto questo ovviamente per quello che riguarda i casino reali."

3.4.5 Scrivi un programma che chieda all'utente una serie di nomi che devono essere tutti memorizzati in una lista. Il numero di nomi da inserire deve essere chiesto all'utente. Quando viene inserito un nome bisogna controllare che inizi con una lettera e se questa è minuscola deve essere fatta diventare maiuscola. Alla fine stampa tutte le stringhe.

Aggiunte: Continua l'esercizio aggiungendo le seguenti funzionalità:

1. Cerca e stampa il nome più lungo
2. Cerca e stampa il nome che in ordine alfabetico è il primo
3. Chiedi all'utente un nome e digli se il nome è presente nell'elenco dei nomi.

3.4.6 Data una lista di stringhe (quindi una matrice), riempito come preferisci, stampa la stringa più lunga, la più corta e la lunghezza media delle stringhe. I tre risultati devono essere ottenuti per mezzo di apposite funzioni.

3.4.7 Scrivi e usa una funzione che data una lista di stringhe, restituisca il numero di parole palindrome contenute nell'array. Riempi l'array come preferisci.

3.4.8 Scrivi un programma che permetta di rappresentare il campo di gioco di "Campo minato". Il campo deve essere formato da una superficie suddivisa in caselle quadrate, quindi avrai una grande tabella (ad esempio di dimensioni 8x12 ma puoi scegliere diversamente). Sul campo devono essere disposte a caso una serie di mine (suggerisco 16 ma puoi cambiare).

Fai poi in modo che ogni casella che non sia una mina abbia un numero che rappresenti il numero di mine adiacenti (anche in diagonale quindi un numero da 1 a 8) lasciando vuote le celle che non hanno mine adiacenti (non scrivere 0)

Soluzione

```
# sviluppo questo programma senza creare classi

import random

nmine = 16
nrighe = 8
ncolonne = 12
vuoto = " "
mina = "X"


campo = [[vuoto for j in range(ncolonne)] for i in range(nrighe)]
for _ in range(nmine):
    i = random.randint(0, nrighe-1)
    j = random.randint(0, ncolonne-1)
    while (campo[i][j] != " "):
        i = random.randint(0, nrighe-1)
        j = random.randint(0, ncolonne-1)
    campo[i][j] = mina

for i in range(nrighe):
    for j in range(ncolonne):
        if campo[i][j] != mina:
            val = 0
            if i > 0        and j > 0           and campo[i-1][j-1] == mina:        # ↖
                val +=1
            if i > 0                            and campo[i-1][j] == mina:          # ↑
                val +=1
            if i > 0        and j < ncolonne-1  and campo[i-1][j+1] == mina:        # ↗
                val +=1
            if                  j < ncolonne-1  and campo[i][j+1] == mina:          # →
                val +=1
            if i < nrighe-1 and j < ncolonne-1  and campo[i+1][j+1] == mina:        # ↘
                val +=1
            if i < nrighe-1                     and campo[i+1][j] == mina:          # ↓
                val +=1
            if i < nrighe-1 and j > 0           and campo[i+1][j-1] == mina:        # ↙
                val +=1
            if                  j > 0           and campo[i][j-1] == mina:          # ←
                val +=1

            if val > 0:
                campo[i][j] = val

        print(campo[i][j], end = " ")
    print()

# ← ↑  →  ↓  ↔  ↕  ↖  ↗  ↘  ↙
```

3.4.9 Crea un programma di gestione di una agenda. L'agenda deve permettere di visualizzare e inserire i dati riguardanti i propri contatti. Per ogni persona devono essere memorizzati

nome cognome e numero di telefono. La visualizzazione dei dati deve avvenire in ordine alfabetico per nome o cognome (scegli tu). Aggiungi poi una funzione per cancellare un contatto in una specifica posizione dell'agenda. (Non avendo ancora fatto la gestione dei files ogni volta l'agenda partirà da uno stato predefinito che puoi decidere tu). Aggiungi anche un'interfaccia utilizzabile.

3.4.10 Scrivi un programma che chieda all'utente di scrivere una frase da leggere in un unico input. Le parole della frase devono poi essere stampate una per volta andando sempre a capo tra una parola e l'altra.

3.4.11 Scrivi un programma che chieda all'utente di scrivere una serie di parole lette una ad una. Queste parole devono essere unite a formare un'unica stringa che deve essere infine stampata.

3.4.12 Scrivi un programma che chieda all'utente di scrivere una frase. Le parole della frase devono poi essere ordinate e stampate in ordine alfabetico.

3.5 Matrici di soli numeri

3.5.1 Creare una matrice che contiene le tabelline dei numeri dall' 1 al 10. Alla fine stamparla.

Soluzione

```
# metodo classico
tabelline = []
for i in range(10):
    tabelline.append([])
    for j in range(10):
        tabelline[i].append((i+1)*(j+1))

# list comprehension
tabelline = [[(i+1)*(j+1) for j in range(10)] for i in range(10)]

for riga in tabelline:
    for num in riga:
        print(f"{num:3d}", end=" ")
    print("")
```

3.5.2 Scrivi un programma che istanzia una matrice n x m di interi casuali (n e m scelti da te). Oltre a stampare l'intera matrice, chiedi all'utente quale riga o quale colonna stampare e stampala sullo schermo

Soluzione

```
import random
import os
os.system('cls')
```

```python
# lettura delle dimensioni della matrice
n = int(input("Scrivi il numero di righe (min 1 max 20): "))
while n<1 or n>20:
    n = int(input("Input errato, inserisci un numero tra 1 e 20: "))

m = int(input("Scrivi il numero di colonne: "))
while m<1 or m>20:
    m = int(input("Input errato, inserisci un numero tra 1 e 20: "))

# creazione della matrice e inserimento dei numeri casuali

# metodo C++
# matrice = []
# for i in range(n):
#     riga = []
#     for j in range(m):
#         riga.append(random.randint(1, 99))
#     matrice.append(riga)

matrice = [[random.randint(1, 99) for i in range(m)] for j in range(n)]


# stampa della matrice
for riga in matrice:
    for numero in riga:
        print(f"{numero:3d}", end=" ")
    print("")

scelta_riga_colonna = int(input(
    "Scegli se stampare una riga o una colonna (scrivi 0 per riga, 1 per colonna) "))
while scelta_riga_colonna != 1 and scelta_riga_colonna != 0:
    scelta_riga_colonna = int(input("Input errato, reinserisci: "))

if scelta_riga_colonna == 0:
    numero_riga_colonna = int(input(f"Scegli la riga da stampare (le righe vanno da 0 a {n-1}):
"))
    while numero_riga_colonna < 0 or numero_riga_colonna >= n:
        numero_riga_colonna = int(input("Input errato, reinserisci: "))
    for numero in matrice[numero_riga_colonna]:
        print(numero, end=" ")

else:
    numero_riga_colonna = int(input(f"Scegli la colonna da stampare (le righe vanno da 0 a {m-
1}): "))
    while numero_riga_colonna < 0 or numero_riga_colonna >= m:
        numero_riga_colonna = int(input("Input errato, reinserisci: "))
    for riga in matrice:
        print(riga[numero_riga_colonna])


print("\n\n")
```

3.5.3 Scrivere un programma che riempia due matrici A e B di dimensione n x m e calcoli la matrice C = A + B e la matrice D = A x B (in realtà è un finto prodotto di matrici). Stampa le matrici ottenute. Per il calcolo della somma calcola normalmente C[i][j] = A[i][j] + B[i][j] mentre per la moltiplicazione hai due possibilità:

1. Versione semplice se non riesci proprio a fare la versione corretta: C[i][j] = A[i][j] x B[i][j].
2. Versione complicata: vai a guardare su internet come si calcola e scrivi l'algoritmo per farlo.

Soluzione

```
import random
import os
os.system('cls')

def stampa_matrice(M):
    for riga in M:
        for numero in riga:
            print(f"{numero:3d}", end=" ")
        print("")


# creazione e riempimento di A e B
n = 2
m = 2
A = []
B = []
for i in range(n):
    rigaA = []
    rigaB = []
    for j in range(m):
        rigaA.append(random.randint(1, 9))
        rigaB.append(random.randint(1, 9))
    A.append(rigaA)
    B.append(rigaB)

print("Matrice A:")
stampa_matrice(A)
print("\nMatrice B:")
stampa_matrice(B)

# creazione e calcolo di C
C = []
for i in range(n):
    riga = []
    for j in range(m):
        riga.append(A[i][j] + B[i][j])
```

```
    C.append(riga)
# print(f"C: {C}")
print("\nMatrice C:")
stampa_matrice(C)

# creazione e calcolo di D (semplice)
D = []
for i in range(n):
    riga = []
    for j in range(m):
        riga.append(A[i][j] * B[i][j])
    D.append(riga)
# print(f"D semplice: {D}")
print("\nMatrice D semplice:")
stampa_matrice(D)

# calcolo di D normale
if m != n:
    print("Il prodotto tra A e B non si può calcolare")
else:
    D = []
    for i in range(n):
        riga = []
        for j in range(m):
            somma = 0
            for k in range(m):
                somma += A[i][k] * B[k][j]
            riga.append(somma)
        D.append(riga)
    # non sono sicuro del risultato
    # print(f"D completo: {D}")
    print("\nMatrice D normale: (da rivedere il procedimento)")
    stampa_matrice(D)


print("\n\n")
```

3.5.4 Scrivi un programma che data una matrice n x m di interi, scambia le righe pari con quelle dispari
variante: le dimensioni della matrice vengono scelte dall'utente tra i valori massimi 10 x 20.

Soluzione

```
# 3.5.4 - Scrivi un programma che data una matrice n x m di
# interi, scambia le righe pari con quelle dispari
# variante: le dimensioni della matrice vengono scelte dall'utente tra i valori
```

```
# massimi 10 x 20.

import os;
os.system('cls')
from random import randint

def stampaMatrice(mat):
    for riga in mat:
        for num in riga:
            print(num, end = " ")
        print()

def scambiaRighe(mat):
    for i in range(0, len(mat)-1, 2):
        mat[i], mat[i+1] = mat[i+1], mat[i]

maxr = 10
maxc = 20

nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))
while nrig < 1 or nrig > maxr:
    nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))

ncol = int(input(f"Scrivi il numero di colonne (massimo {maxc}): "))
while ncol < 1 or ncol > maxc:
    ncol = int(input(f"Scrivi il numero di colonne (massimo {maxc}): "))

mat = [[randint(0,9) for j in range(ncol)] for i in range(nrig)]

# # in alternativa
# mat = []
# for i in range(nrig):
#     mat.append([])
#     for j in range(ncol):
#         mat[i].append(randint(0,9))

stampaMatrice(mat)
scambiaRighe(mat)
print()
stampaMatrice(mat)
```

3.5.5 Scrivi un programma che data una matrice n x n di interi, con n scelto dall'utente, scambia le righe con le colonne

Variante: La matrice non è quadrata ma è una generica matrice n x m

Soluzione base

```
# 3.5.5 - Scrivi un programma che data una matrice n x n di
# interi, con n scelto dall'utente, scambia le righe con le colonne


import os;
os.system('cls')
from random import randint

def stampaMatrice(mat):
    for riga in mat:
        for num in riga:
            print(num, end = " ")
        print()

def scambiaRighe(mat):
```

```
    for i in range(len(mat)):
        for j in range(i+1, len(mat[i])):
            mat[i][j], mat[j][i] = mat[j][i], mat[i][j]

maxr = 10

nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))
while nrig < 1 or nrig > maxr:
    nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))

mat = [[randint(0,9) for j in range(nrig)] for i in range(nrig)]

stampaMatrice(mat)
scambiaRighe(mat)
print()
stampaMatrice(mat)
```

Soluzione variante

```
import os;
os.system('cls')
from random import randint

def stampaMatrice(mat):
    for riga in mat:
        for num in riga:
            print(num, end = " ")
        print()

def scambiaRighe(mat):
    ris = [[0 for j in range(len(mat))] for i in range(len(mat[0]))]
    # len(mat) è il numero di righe di mat
    # len(mat[0]) è il numero di colonne di mat (considero che ogni riga sia lunga uguale)

    for i in range(len(mat)):
        for j in range(len(mat[i])):
            ris[j][i] = mat[i][j]

    return ris

def scambiaRighe2(mat):
    ris = []
    for j in range(len(mat[0])):
        ris.append([])
        for i in range(len(mat)):
            ris[j].append(mat[i][j])
    return ris


maxr = 10
maxc = 20

nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))
while nrig < 1 or nrig > maxr:
    nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))

ncol = int(input(f"Scrivi il numero di colonne (massimo {maxc}): "))
while ncol < 1 or ncol > maxc:
    ncol = int(input(f"Scrivi il numero di colonne (massimo {maxc}): "))

mat = [[randint(0,9) for j in range(ncol)] for i in range(nrig)]

stampaMatrice(mat)
```

```
ris = scambiaRighe2(mat)
print()
stampaMatrice(ris)
```

3.5.6 Scrivi un programma che riempia una matrice n x m di numeri interi a caso, n e m sono scelti dall'utente e il range dei numeri è da 0 a x scelto anch'esso dall'utente. Determina poi il numero più frequente generato nella matrice. Il riempimento delle matrici e la ricerca del numero più frequente devono essere svolti da due apposite funzioni, mentre l'interazione con l'utente dev'essere lasciata al main.

Soluzione

```
# 3.5.6 - Scrivi un programma che riempia una matrice n x m di
# numeri interi a caso, n e m sono scelti dall'utente e il range dei numeri è da
# 0 a x scelto anch'esso dall'utente. Determina poi il numero più frequente
# generato nella matrice. Il riempimento delle matrici e la ricerca del numero
# più frequente devono essere svolti da due apposite funzioni, mentre
# l'interazione con l'utente dev'essere lasciata al main.


import os;
os.system('cls')
from random import randint

def generaMatrice(nrig, ncol, maxint):
    return [[randint(0,maxint) for j in range(ncol)] for i in range(nrig)]

def stampaMatrice(mat):
    for riga in mat:
        for num in riga:
            print(num, end = " ")
        print()

def piuFrequente(mat):
    d = {}
    ris = None
    for riga in mat:
        for num in riga:
            if num in d:
                d[num] += 1
            else:
                d[num] = 1
            if ris == None or d[num] > d[ris]:
                ris = num
    return ris

maxr = 10
maxc = 20

nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))
while nrig < 1 or nrig > maxr:
    nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))

ncol = int(input(f"Scrivi il numero di colonne (massimo {maxc}): "))
while ncol < 1 or ncol > maxc:
    ncol = int(input(f"Scrivi il numero di colonne (massimo {maxc}): "))

x = int(input(f"Scrivi il massimo valore generabile (minimo 0): "))
while x < 0:
    x = int(input(f"Scrivi il massimo valore generabile  (minimo 0): "))
```

```
mat = generaMatrice(nrig, ncol, x)
stampaMatrice(mat)
print()
print("Il numero più frequente è", piuFrequente(mat))
```

3.5.7 Scrivi un programma che riempia una matrice n x m di numeri interi a caso, n e m sono scelti dall'utente e il range dei numeri è da 0 a x scelto anch'esso dall'utente. Determina poi il la riga o la colonna con la somma dei numeri più grande. Il riempimento delle matrici e la ricerca della riga o della colonna devono essere svolti da apposite funzioni, mentre l'interazione con l'utente dev'essere lasciata al main.

Soluzione

```
# 3.5.7 - Scrivi un programma che riempia una matrice n x m di
# numeri interi a caso, n e m sono scelti dall'utente e il range dei numeri è da
# 0 a x scelto anch'esso dall'utente. Determina poi il la riga o la colonna con
# la somma dei numeri più grande. Il riempimento delle matrici e la ricerca della
# riga o della colonna devono essere svolti da apposite funzioni, mentre
# l'interazione con l'utente dev'essere lasciata al main.

import os;
os.system('cls')
from random import randint

def generaMatrice(nrig, ncol, maxint):
    return [[randint(0,maxint) for j in range(ncol)] for i in range(nrig)]

def stampaMatrice(mat):
    for riga in mat:
        for num in riga:
            print(num, end = " ")
        print()

def rigaColonnaMax(mat):
    somme_righe = [0 for _ in range(len(mat))]
    somme_colonne = [0 for _ in range(len(mat[0]))] # considero tutte le righe lunghe uguali
    for i, riga in enumerate(mat):
        for j, num in enumerate(riga):
            somme_righe[i] += num
            somme_colonne[j] += num
    maxi = 0
    for i in range(len(somme_righe)):
        if somme_righe[i] > somme_righe[maxi]:
            maxi = i
    maxj = 0
    for j in range(len(somme_colonne)):
        if somme_colonne[j] > somme_colonne[maxj]:
            maxj = j

    if somme_righe[maxi] >= somme_colonne[maxj]:
        return ('r', maxi)
    else:
        return ('c', maxj)

maxr = 10
maxc = 20

nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))
while nrig < 1 or nrig > maxr:
    nrig = int(input(f"Scrivi il numero di righe (massimo {maxr}): "))

ncol = int(input(f"Scrivi il numero di colonne (massimo {maxc}): "))
```

```
while ncol < 1 or ncol > maxc:
    ncol = int(input(f"Scrivi il numero di colonne (massimo {maxc}): "))

x = int(input(f"Scrivi il massimo valore generabile (minimo 0): "))
while x < 0:
    x = int(input(f"Scrivi il massimo valore generabile  (minimo 0): "))

mat = generaMatrice(nrig, ncol, x)
stampaMatrice(mat)
print()
ris = rigaColonnaMax(mat)
if ris[0] == 'c':
    print("La somma più grande la trovo nella colonna", ris[1])
else:
    print("La somma più grande la trovo nella riga", ris[1])
```

3.5.8 Scrivi un programma che:

1. crei e riempia di valori casuali una matrice di dimensioni a tua scelta,
2. ne stampi il contenuto sullo schermo,
3. chieda all'utente un numero e cerchi se il numero è presente nella matrice
4. se il numero è presente si stampino le coordinate della posizione nella matrice

Soluzione

```
# 3.5.8 - Scrivi un programma che:
# 1.  crei e riempia di valori casuali una matrice di dimensioni a tua
# scelta,
# 2.  ne stampi il contenuto sullo schermo,
# 3.  chieda all'utente un numero e cerchi se il numero è presente nella
# matrice
# 4.  se il numero è presente si stampino le coordinate della posizione
# nella matrice

from random import randint

def stampaMatrice(mat):
    for riga in mat:
        for numero in riga:
            print(f"{numero:2d} ", end='')
        print()
    print()

# 1.  crei e riempia di valori casuali una matrice di dimensioni a tua
# scelta,
n = 4
m = 6
mat = [[randint(0,9) for _ in range(m)] for _ in range(n)]

# 2.  ne stampi il contenuto sullo schermo,
stampaMatrice(mat)

# 3.  chieda all'utente un numero e cerchi se il numero è presente nella
# matrice
cercare = int(input("Scrivi un numero da cercare: "))

def contiene(mat, num):
    for riga in mat:
        for numero in riga:
            if numero == num:
```

```
                return True
    return False

print(contiene(mat, cercare))

# 4.  se il numero è presente si stampino le coordinate della posizione
# nella matrice

def posizione(mat, num):
    for i,riga in enumerate(mat):
        for j,numero in enumerate(riga):
            if numero == num:
                return (i,j)

def posizioni(mat, num):
    ris = []
    for i,riga in enumerate(mat):
        for j,numero in enumerate(riga):
            if numero == num:
                ris.append((i,j))
    return ris

print(posizione(mat, cercare))
print(posizioni(mat, cercare))
```

3.5.9 Scrivi un programma che istanzia una matrice n x m di interi casuali (n e m scelti da te). Oltre a stampare l'intera matrice, chiedi all'utente quale riga o quale colonna stampare e stampala sullo schermo

Soluzione

```
# 3.5.9 - Scrivi un
# programma che istanzia una matrice n x m di interi casuali (n e m scelti da
# te). Oltre a stampare l'intera matrice, chiedi all'utente quale riga o quale
# colonna stampare e stampala sullo schermo

from random import randint

def stampaMatrice(mat):
    for riga in mat:
        for numero in riga:
            print(f"{numero:2d} ", end='')
        print()
    print()

def stampaRiga(mat, nriga):
    for numero in mat[nriga]:
        print(f"{numero:2d}", end='')

def stampaColonna(mat, ncol):
    for riga in mat:
        print(f"{riga[ncol]:2d}")

n = 4
m = 6
mat = [[randint(0,9) for _ in range(m)] for _ in range(n)]

stampaMatrice(mat)
n = int(input("Vuoi stampare una riga o una colonna? (1. riga, 2. colonna) "))
while n != 1 and n != 2:
    n = int(input("Vuoi stampare una riga o una colonna? (1. riga, 2. colonna) "))

if n == 1:
    n = int(input("Scrivi l'indice della riga da stampare: "))
```

```
    if n < 0 or n >= len(mat):
        print("La riga non esiste.")
    else:
        stampaRiga(mat, n)
        print()
else:
    n = int(input("Scrivi l'indice della colonna da stampare: "))
    if n < 0 or n >= len(mat[0]):
        print("La colonna non esiste.")
    else:
        stampaColonna(mat, n)
        print()
```

3.5.10 data una matrice n x m di numeri casuali (n e m scelti da te), stampa l'indice della riga con la massima somma dei numeri contenuti. Stampa anche l'indice della colonna con la massima somma dei numeri.

Soluzione

```
# 3.5.10 - data una
# matrice n x m di numeri casuali (n e m scelti da te), stampa l'indice della
# riga con la massima somma dei numeri contenuti. Stampa anche l'indice della
# colonna con la massima somma dei numeri.

from random import randint

def stampaMatrice(mat):
    for riga in mat:
        for numero in riga:
            print(f"{numero:2d} ", end='')
        print()
    print()

n = 4
m = 6
mat = [[randint(0,9) for _ in range(m)] for _ in range(n)]

stampaMatrice(mat)

def rigaMassima(mat):
    somme = [sum(riga) for riga in mat]
    # return somme.index(max(somme))
    posmax = 0
    for i,val in enumerate(somme):
        if val > somme[posmax]:
            posmax = i
    return posmax

print(rigaMassima(mat))

def colonnaMassima(mat):
    somme = []
    for j in range(len(mat[0])):
        somma = 0
        for i in range(len(mat)):
            somma += mat[i][j]
        somme.append(somma)
    # print(somme)
    return somme.index(max(somme))


print(colonnaMassima(mat))
```

3.5.11 Scrivi un programma che per mezzo di un'apposita funzione riempia una matrice di numeri casuali. Stampa poi il numero di numeri pari contenuti in ogni riga della matrice (riga per riga separatamente) e il numero di numeri dispari contenuti in ogni colonna della matrice. Anche in questo caso definisci una o due funzioni per farlo (a scelta).

3.5.12 Riempi una matrice di numeri casuali, poi scrivi una funzione che stampa i soli numeri pari della matrice in modo che la matrice compaia con la stessa forma ma senza i numeri dispari al posto dei quali compariranno spazi vuoti.

Soluzione

```
# 3.5.16 - Riempi una matrice di numeri casuali, poi scrivi una
# funzione che stampa i soli numeri pari della matrice in modo che la matrice
# compaia con la stessa forma ma senza i numeri dispari al posto dei quali
# compariranno spazi vuoti.

from random import randint

n = 4
m = 6

mat = [[randint(0,99) for _ in range(m)] for _ in range(n)]

def stampaMat(mat):
    for riga in mat:
        for num in riga:
            print(f'{num:2d} ', end='')
        print()

stampaMat(mat)

def stampaMatPari(mat):
    for riga in mat:
        for num in riga:
            if num % 2 == 0:
                print(f'{num:2d} ', end='')
            else:
                print('   ', end='')
        print()

print()
stampaMatPari(mat)
```

3.5.13 Riempi una matrice quadrata di numeri casuali, poi stampa la matrice in modo che siano visibili solo i numeri appartenenti alle diagonali

Soluzione

```
# 3.5.15    Riempi una matrice quadrata di numeri casuali,
# poi stampa la matrice in modo che siano visibili solo
# i numeri appartenenti alle diagonali


from random import randint

n = 4
m = 4

mat = [[randint(0,99) for _ in range(m)] for _ in range(n)]
```

```
for riga in mat:
    for num in riga:
        print(f'{num:2d} ', end='')
    print()

print()

# stampo solo le diagonali
for i, riga in enumerate(mat):
    for j, num in enumerate(riga):
        if i == j or i+j == n-1:
            print(f'{num:2d} ', end='')
        else:
            print('   ', end='')
    print()
```

3.5.14 Scrivi un programma che riempia una matrice quadrata di numeri casuali e poi stampi le due diagonali in entrambi i versi

Variante: scrivi una funzione che restituisca quattro liste che contengono i valori delle due diagonali nei due versi

Soluzione

```
# 3.5.14    Scrivi un programma che riempia una matrice quadrata
# di numeri casuali e poi stampi le due diagonali in entrambi i
# versi
# Variante: scrivi una funzione che restituisca quattro liste
# che contengono i valori delle due diagonali nei due versi

from random import randint

n = 4
m = 4

mat = [[randint(0,99) for _ in range(m)] for _ in range(n)]

def stampaMat(mat):
    for riga in mat:
        for num in riga:
            print(f'{num:2d} ', end='')
        print()

stampaMat(mat)
print()

# ← ↑   →   ↓   ↔   ↕   ↖   ↗   ↘   ↙

# diagonale ↘
for i in range(n):
    print(mat[i][i],end=' ')
print()

# diagonale ↖
for i in range(n-1,-1,-1):
    print(mat[i][i],end=' ')
print()

# diagonale ↙
for i in range(n):
    print(mat[i][n-1-i],end=' ')
print()
```

```
# diagonale ↗
for i in range(n):
    print(mat[n-1-i][i],end=' ')
print()

# diagonale ↘
def diag_SuSx_GiuDx(mat):
    n = len(mat)
    # ris = []
    # for i in range(n):
    #     ris.append(mat[i][i])
    # return ris
    return [mat[i][i] for i in range(n)]

# diagonale ↖
def diag_GiuDx_SuSx(mat):
    # n = len(mat)
    # ris = []
    # for i in range(n-1,-1,-1):
    #     ris.append(mat[i][i])
    # return ris

    # return [mat[i][i] for i in range(n-1,-1,-1)]

    return diag_SuSx_GiuDx(mat)[::-1]

# diagonale ↙
def diag_SuDx_GiuSx(mat):
    n = len(mat)
    return [mat[i][n-1-i] for i in range(n)]

# diagonale ↗
def diag_GiuSx_SuDx(mat):
    return diag_SuDx_GiuSx(mat)[::-1]

print('↘', diag_SuSx_GiuDx(mat))
print('↖', diag_GiuDx_SuSx(mat))
print('↙', diag_SuDx_GiuSx(mat))
print('↗', diag_GiuSx_SuDx(mat))
```

3.5.15 (Difficile) Scrivi un programma che riempia una matrice di numeri casuali e che stampi i numeri contenuti in ogni diagonale, prima delle diagonali che vanno da in basso a sinistra a in alto a destra, poi da in alto a sinistra a in basso a destra.

Variante: invece che stampare solamente i numeri, crea una lista di liste che contenga per ogni riga i valori contenuti in una diagonale. Devono essere create due matrici, una per ogni diversa direzione delle diagonali.

Soluzione

```
# 3.5.13    Scrivi un programma che riempia una matrice di numeri
# casuali e che stampi i numeri contenuti in ogni diagonale,
# prima delle diagonali che vanno da in basso a sinistra a in
# alto a destra, poi da in alto a sinistra a in basso a destra.

# Variante: invece che stampare solamente i numeri, crea una
# lista di liste che contenga per ogni riga i valori contenuti
# in una diagonale. Devono essere create due matrici, una per
# ogni diversa direzione delle diagonali.

from random import randint
```

```python
n = 4
m = 2
mat = [[randint(0,9) for _ in range(m)] for _ in range(n)]

# stampa matrice
for riga in mat:
    for num in riga:
        print(f"{num} ", end='')
    print()
print()

# Metodo 1 base che pensa alle diagonali che escono dalla matrice
# di cui stampo solo le parti dentro alla matrice
def diagonaliAltoBasso1(mat):
    ris = []
    for j in range(-n+1, m):
        diag = []
        for i in range(n):
            if j+i >= 0 and j+i < m:
                diag.append(mat[i][j+i])
        ris.append(diag)
    return ris

# Metodo 2 (difficile, più adatto alle altre diagonali)
# che non costruisce le diagonali una alla volta ma man mano
# che si scorre la matrice aggiunge i valori alle varie diagonali che all'inizio
# sono vuote, sapendo che per ogni diagonale la differenza tra
# gli indici i e j dei valori è costante
def diagonaliAltoBasso2(mat):
    ris = [[] for _ in range(n+m-1)]
    for j in range(m):
        for i in range(n):
            ris[(j-i)+(n-1)].append(mat[i][j])
    return ris

# Metodo 1: complicato ma che segue il ragionamento
# del metodo 1 per le diagonali alto basso
# qui i valori di j sono difficili
def diagonaliBassoAlto1(mat):
    ris = []
    for j in range(-n+1, m):
        diag = []
        for i in range(n-1,-1,-1):
            if j+(n-1-i) >= 0 and j+(n-1-i) < m:
                diag.append(mat[i][j+(n-1-i)])
        ris.append(diag)
    return ris

def diagonaliBassoAlto2(mat):
    ris = []
    for i in range(n + m - 1):
        diag = []
        for j in range(m):
            if i-j < n and i-j >= 0:
                diag.append(mat[i-j][j])
        ris.append(diag)
    return ris

# metodo 3 (astuto, ideato da Alessandro Beretta)
# che considera che gli elementi di una stessa diagonale
# basso-sinistra a in alto-destra,
# hanno la somma delle coordinate costante
def diagonaliBassoAlto3(mat):
    ris = [[] for _ in range(n+m-1)]
    for j in range(m):
        for i in range(n):
            ris[i+j].append(mat[i][j])
```

```
    return ris



print(diagonaliAltoBasso1(mat))
print(diagonaliAltoBasso2(mat))
print(diagonaliBassoAlto1(mat))
print(diagonaliBassoAlto2(mat))
print(diagonaliBassoAlto3(mat))
```

3.5.16 Riempi una matrice n x m di numeri casuali (n e m scelti da te), poi verifica se gli elementi della matrice sono disposti in modo tale che ogni numero sia maggiore di quelli che si trovano immediatamente alla sua sinistra e immediatamente sopra.

Variante: scrivi e usa una funzione che restituisca la risposta alla domanda

Continua l'esercizio creando e stampando matrici fino a che non ne viene stampata una che rispetta la condizione.

Continua l'esercizio calcolando mediamente quante matrici vengono create prima di crearne una che rispetti la condizione; in questo caso puoi stampare solo il numero medio di volte e non ogni lista (dovresti stamparne un'enormità).

3.6 Matrici miste

3.6.1 Un file di testo contiene i seguenti dati:

```
Marco Polo 1254 1324
Cristoforo Colombo 1451 1506
Amerigo Vespucci 1454 1512
Francisco Pizarro 1475 1541
Ferdinando Magellano 1480 1521
Hernan Cortez 1485 1547
Walter Raleigh 1552 1618
Henry Hudson 1570 1611
James Cook 1728 1779
Charles Darwin 1809 1882
Kit Carson 1809 1866
David Livingstone 1813 1873
Charles Foucauld 1858 1916
Ronald Amundsen 1872 1928
Ernest Shakleton 1874 1922
```

Scrivi un programma che legga i dati contenuti nel file, li memorizzi opportunamente, e poi scrivi e utilizza le seguenti funzioni:

1. una funzione che stampa tutti i dati.
2. una funzione che ordina i dati in ordine alfabetico secondo il nome degli esploratori
3. una funzione che ordina i dati in ordine crescente secondo le date di nascita degli esploratori
4. una funzione che calcola la durata della vita di ogni esploratore e aggiunge questo dato agli altri

5. una funzione che ordina gli esploratori in base alla durata della loro vita in ordine decrescente e in caso di parità in ordine crescente secondo la data di nascita
6. una funzione che riscrive tutti i dati in un secondo file

Soluzione

```
import os
os.system("cls")

# 1.    una funzione che stampa tutti i dati.
def f1(dati):
    for esp in dati:
        print(esp)

# 2.    una funzione che ordina i dati in ordine alfabetico secondo il nome degli esploratori
def f2(dati):
    dati.sort()

# 3.    una funzione che ordina i dati in ordine crescente secondo le date di nascita degli
esploratori
def f3(dati):
    dati.sort(key = lambda x: (x[2]))

# 4.    una funzione che calcola la durata della vita di ogni esploratore e agiunge questo dato
agli altri
def f4(dati):
    for esp in dati:
        esp.append(esp[3]-esp[2])

# 5.    una funzione che ordina gli esploratori in base alla durata della loro vita in ordine
decrescente e in caso di
#       parità in ordine crescente secondo la data di nascita
def f5(dati):
    dati.sort(key = lambda x: (-x[4], x[2]))

# 6.    una funzione che riscrive tutti i dati in un secondo file
def f6(dati, nomefile):
    with open(nomefile,'w',encoding = 'utf-8') as f:
        for esp in dati:
            for dato in esp:
                f.write(f"{dato} ")
            f.write("\n")

dati = []
with open("8-2-1.txt",'r',encoding = 'utf-8') as f:
    for riga in f:
        riga = riga.strip()
        riga = riga.split()
        riga[2] = int(riga[2])
        riga[3] = int(riga[3])
        dati.append(riga)

print("Dati letti dal file:")
f1(dati)

f2(dati)
print("Esploratori in ordine alfabetico:")
f1(dati)

f4(dati)
f5(dati)

print("\nClassifica per durata di vita:")
```

```
for el in dati:
    print(el)

# scrivo nel secondo file
f6(dati, "8-2-1-out.txt")
```

3.6.2 Scrivi un programma che legga i dati contenuti in una lista di stringhe (riportata di seguito) e li inserisca in opportune strutture. Di questi dati il programma deve fare le seguenti cose:

1. stamparli sullo schermo
2. ordinarli in ordine decrescente di voto
3. stamparli nuovamente in ordine

Dati:

```
dati = [
    'Amici 7.00',
    'Biella 9.00',
    'Brescia 6.00',
    'Carolla 8.00',
    'Cunegatti 7.00',
    'DeBella 4.00',
    'DeVecchi 9.00',
    'Fumagalli 7.00',
    'Galimberti 9.00',
    'Germanò 9.00',
    'Gubellini 9.00',
    'Lepore 5.00',
    'Maconi 6.00',
    'Mariani 8.00',
    'Mattavelli 9.00',
    'Oggiano 4.00',
    'Passoni 5.00',
    'Pastori 4.00',
    'Pirovano 7.00',
    'Rudi 10.00',
    'Russell 6.00',
    'Sogos 4.00',
    'Tezza 6.00',
    'Varisco 8.00',
]
```

Soluzione

3.6.3 Scrivi un programma che legga i dati contenuti in una stringa di testo (riportata di seguito) e li inserisca in opportune strutture dati che rappresentino il registro dei voti per una materia. Il programma deve poi calcolare la media dei voti per ogni singolo studente e aggiungerla alla struttura dati. Il programma deve mostrare tutti i dati raccolti e calcolati sullo schermo.

```
dati = '''Amici 7.00 8.00 7.00
Biella 8.00 9.50 7.00
Brescia 5.00 7.50 9.00
Carolla 7.00 8.50 8.50
Cunegatti 7.00 7.00 5.50
DeBella 4.00 4.00 4.50
DeVecchi 8.00 10.00 6.00
```

```
Fumagalli 8.50 6.00 4.50
Galimberti 9.00 9.00 9.00
Germanò 8.50 9.00 7.50
Gubellini 8.00 10.00 7.00
Lepore 5.50 4.00 3.00
Maconi 7.50 5.00 7.50
Mariani 8.00 8.00 7.00
Mattavelli 9.00 9.50 8.50
Oggiano 5.00 3.00 3.00
Passoni 5.00 6.00 6.00
Pastori 3.50 5.00 7.00
Pirovano 6.50 7.50 7.50
Rudi 9.50 10.00 10.00
Russell 7.50 4.50 5.50
Sogos 4.00 3.50 3.50
Tezza 7.00 5.50 5.50
Varisco 8.00 9.00 8.50'''
```

Soluzione

3.6.4 Scrivi un programma che legga i dati contenuti in una stringa di testo (riportata di seguito) e li inserisca in opportune strutture dati che rappresentino il registro dei voti per una materia. Il programma deve poi calcolare la media dei voti per ogni singolo studente e aggiungerla alla struttura dati. Il programma deve mostrare tutti i dati raccolti e calcolati sullo schermo. Questa versione dell'esercizio è diversa e più difficile della precedente perché si può notare che i cognomi possono essere formati da più parole e sono separati dai voti da un ":", inoltre il numero di voti varia per ogni studente.

```
dati = '''Amici: 7.00 8.00 7.00
Biella: 8.00 9.50 7.00
Brescia: 5.00 7.50 9.00
Carolla: 7.00 8.50 8.50
Cunegatti: 7.00 7.00
De Bella: 4.00 4.00 4.50 5.00
De Vecchi: 8.00 10.00 6.00
Fumagalli: 8.50 6.00 4.50
Galimberti: 9.00 9.00 9.00
Germanò: 8.50 9.00 7.50
Gubellini: 8.00 10.00 7.00
Lepore: 5.50 4.00 3.00 5.00
Maconi: 7.50 5.00
Mariani: 8.00 8.00 7.00
Mattavelli: 9.00 9.50 8.50
Oggiano: 5.00 3.00 3.00 2.00
Passoni: 5.00 6.00 6.00 7.50
Pastori: 3.50 5.00 7.00 2.00
Pirovano: 6.50 7.50 7.50
Rudi: 9.50 10.00 10.00
Russell: 7.50 4.50
Sogos: 4.00 3.50 3.50 2.00
Tezza: 7.00 5.50 5.50
Varisco: 8.00 9.00 8.50
Pirovano: 6.50 7.50 7.50
Rudi: 9.50 10.00 10.00
Russell: 7.50 4.50 5.50
Sogos: 4.00 3.50 3.50
```

```
Tezza: 7.00 5.50 5.50
Varisco: 8.00 9.00 8.50'''
```

Soluzione

3.6.5 La seguente lista descrive i più celebri virologi Italiani nell'epoca covid

(Nome e Cognome - Ospedale - città - (O) Ottimista (P) Pessimista

```
lista = [
    'Roberto Burioni - San Raffaele - Milano (P)',
    'Matteo Bassetti - San Martino - Genova (O)',
    'Alberto Zangrillo - San Raffaele - Milano (O)',
    'Massimo Galli - Luigi Sacco - Milano (P)',
    'MariaRita Gismondo - Luigi Sacco - Milano (O)',
    'Andrea Crisanti - Università di Padova - Padova (P)',
    'Ilaria Capua - One Health - Florida (P)',
    'Antonella Viola - Città della Speranza - Padova (P)',
    'Fabrizio Pregliasco - Galeazzi - Milano (O)',
    'Walter Ricciardi - Policlino Gemelli - Roma (P)',
    'Franco Locatelli - Bambin Gesù - Roma (P)'
]
```

1. stampare nome e cognome dei virologi ordinati per cognome (si consiglia si usare la split() con il separatore "-";
2. stampare il numero di virologi ottimisti e di quelli pessimisti.

3.6.6 Data la seguente lista di liste

```
highways2 = [
    ["A1", "Milano-Napoli", ["Lombardia", "Emilia Romagna", "Toscana", "Umbria", "Lazio", "Campania"], 759],
    ["A2","Salerno-ReggioCalabria", ["Campania", "Basilicata", "Calabria"], 442],
    ["A3","Napoli-Salerno", ["Campania"], 52],
    ["A4","Torino-Trieste", ["Piemonte", "Lombardia", "Veneto", "Friuli"], 524],
    ["A6","Torino-Savona", ["Piemonte", "Liguria"], 124],
    ["A7","Milano-Genova", ["Lombardia", "Liguria"], 133],
    ["A8","Milano-Varese", ["Lombardia"], 43],
    ["A10","Savona-Ventimigla", ["Liguria"], 113],
    ["A11","Firenze-Pisa", ["Toscana"], 81],
    ["A12","Genova-Cecina", ["Liguria", "Toscana"], 210],
    ["A13","Bologna-Padova", ["Emilia Romagna", "Veneto"], 116],
    ["A14","Bologna-Taranto", ["Emilia Romagna", "Marche", "Abruzzo", "Molise", "Puglia"], 743],
    ["A15","Parma-LaSpezia", ["Emilia Romagna", "Liguria"], 108],
    ["A16","Napoli-Canosa", ["Campania", "Puglia"], 172],
    ["A19","Palermo-Catania", ["Sicilia"], 191],
    ["A21","Torino-Piacenza-Brescia", ["Piemonte", "Emilia Romagna", "Lombardia"], 238],
    ["A22","Brennero-Modena", ["Alto Adige", "Trentino", "Veneto", "Emilia Romagna"], 315],
    ["A23","Palmanova-Tarvisio", ["Friuli"], 119],
```

```
    ["A24","Roma-Teramo", ["Lazio", "Abruzzo"], 159],
    ["A25","Torano-Pescara", ["Lazio", "Abruzzo"], 115],
    ["A26","Genova-Gravellona", ["Liguria", "Piemonte"], 197]
]
```

1. stampare tutte le autostrade che attraversano la Lombardia
2. stampare l'autostrada che attraversa più regioni
3. stampare, tutte le autostrade, ordinate in base al numero di regioni attraversate, e a parità, in base alla loro lunghezza
4. stampare per ogni regione, il numero di autostrade che la attraversano
5. stampare per ogni regione tutte le autostrade che la attraversano
6. stampare per ogni regione tutte le autostrade che la attraversano, ordinandole in base al numero di regioni attraversate

Soluzione

```python
# Data la seguente lista di liste

highways2 = [
    ["A1", "Milano-Napoli",  ["Lombardia", "EmiliaRomagna", "Toscana", "Umbria", "Lazio",
"Campania"],759],
    ["A2","Salerno-ReggioCalabria",["Campania", "Basilicata","Calabria"], 442],
    ["A3","Napoli-Salerno",["Campania"], 52],
    ["A4","Torino-Trieste",["Piemonte", "Lombardia", "Veneto", "Friuli"], 524],
    ["A6","Torino-Savona",["Piemonte", "Liguria"], 124],
    ["A7","Milano-Genova",["Lombardia", "Liguria"], 133],
    ["A8","Milano-Varese",["Lombardia"], 43],
    ["A10","Savona-Ventimigla",["Liguria"], 113],
    ["A11","Firenze-Pisa",["Toscana"], 81],
    ["A12","Genova-Cecina",["Liguria", "Toscana"], 210],
    ["A13","Bologna-Padova",["EmiliaRomagna", "Veneto"], 116],
    ["A14","Bologna-Taranto",["EmiliaRomagna", "Marche", "Abruzzo", "Molise", "Puglia"], 743],
    ["A15","Parma-LaSpezia",["EmiliaRomagna", "Liguria"], 108],
    ["A16","Napoli-Canosa",["Campania", "Puglia"], 172],
    ["A19","Palermo-Catania",["Sicilia"], 191],
    ["A21","Torino-Piacenza-Brescia",["Piemonte","EmiliaRomagna","Lombardia"], 238],
    ["A22","Brennero-Modena",["AltoAdige", "Trentino", "Veneto","EmiliaRomagna"], 315],
    ["A23","Palmanova-Tarvisio",["Friuli"], 119],
    ["A24","Roma-Teramo",["Lazio", "Abruzzo"], 159],
    ["A25","Torano-Pescara",["Lazio", "Abruzzo"], 115],
    ["A26","Genova-Gravellona",["Liguria", "Piemonte"], 197]
]

# 1) stampare tutte le autostrade che attraversano la Lombardia
print("Autostrade che attraversano la Lombardia:")
for autostrada in highways2:
    if "Lombardia" in autostrada[2]:
        print(autostrada)

# 2) stampare l'autostra che attraversa più regioni
print("\nAutostrada che attraversa più regioni:")
maxi = 0
max = len(highways2[0][2])
for i, el in enumerate(highways2):
    lung = len(el[2])
    if lung > max:
        max = lung
        maxi = i
print(highways2[maxi])
```

```
# 3) stampare, tutte le autostrade, ordinate in base al numero di regioni attraversate, e a
parità,
# in base alla loro lunghezza
print("\nAutostrade ordinate in base al numero di regioni attraversate, e a parità, in base alla
loro lunghezza:")
highways2.sort(key = lambda el: (len(el[2]), el[3]))
for el in highways2:
    print(el)

# 4) stampare per ogni regione, il numero di autostrade che la attraversano
print("\nPer ogni regione, il numero di autostrade che la attraversano:")
regioni = {}
for aut in highways2:
    for reg in aut[2]:
        if reg not in regioni:
            regioni[reg] = 1
        else:
            regioni[reg] += 1

for key, val in regioni.items():
    print(key, val)

# 5) stampare per ogni regione tutte le autostrade che la attraversano
print("\nPer ogni regione, tutte le autostrade che la attraversano:")
regioni = {}
for aut in highways2:
    for reg in aut[2]:
        if reg not in regioni:
            regioni[reg] = [aut]
        else:
            regioni[reg].append(aut)

for key, val in regioni.items():
    print (f"{key}:")
    for aut in val:
        print(" -", aut)

# 6) stampare per ogni regione tutte le autostrade che la attraversano, ordinandole in base al
numero di
# regioni attraversate
print("\n\nPer ogni regione, tutte le autostrade che la attraversano, ordinate in base al numero
di regioni attraversate:")
regioni = {}
for aut in highways2:
    for reg in aut[2]:
        if reg not in regioni:
            regioni[reg] = [aut]
        else:
            regioni[reg].append(aut)

for key, val in regioni.items():
    print (f"{key}:")
    val.sort(key = lambda el: -len(el[2]))
    for aut in val:
        print(" -", aut)
```

Senza dizionario

```
# 3.5.12    Data la seguente lista di liste
highways2 = [
    ["A1", "Milano-Napoli", ["Lombardia", "Emilia Romagna", "Toscana", "Umbria", "Lazio",
"Campania"], 759],
    ["A2","Salerno-ReggioCalabria", ["Campania", "Basilicata", "Calabria"], 442],
    ["A3","Napoli-Salerno", ["Campania"], 52],
    ["A4","Torino-Trieste", ["Piemonte", "Lombardia", "Veneto", "Friuli"], 524],
    ["A6","Torino-Savona", ["Piemonte", "Liguria"], 124],
    ["A7","Milano-Genova", ["Lombardia", "Liguria"], 133],
```

```
    ["A8","Milano-Varese", ["Lombardia"], 43],
    ["A10","Savona-Ventimigla", ["Liguria"], 113],
    ["A11","Firenze-Pisa", ["Toscana"], 81],
    ["A12","Genova-Cecina", ["Liguria", "Toscana"], 210],
    ["A13","Bologna-Padova", ["Emilia Romagna", "Veneto"], 116],
    ["A14","Bologna-Taranto", ["Emilia Romagna", "Marche", "Abruzzo", "Molise", "Puglia"], 743],
    ["A15","Parma-LaSpezia", ["Emilia Romagna", "Liguria"], 108],
    ["A16","Napoli-Canosa", ["Campania", "Puglia"], 172],
    ["A19","Palermo-Catania", ["Sicilia"], 191],
    ["A21","Torino-Piacenza-Brescia", ["Piemonte", "Emilia Romagna", "Lombardia"], 238],
    ["A22","Brennero-Modena", ["Alto Adige", "Trentino", "Veneto", "Emilia Romagna"], 315],
    ["A23","Palmanova-Tarvisio", ["Friuli"], 119],
    ["A24","Roma-Teramo", ["Lazio", "Abruzzo"], 159],
    ["A25","Torano-Pescara", ["Lazio", "Abruzzo"], 115],
    ["A26","Genova-Gravellona", ["Liguria", "Piemonte"], 197]
]
# 1.    stampare tutte le autostrade che attraversano la Lombardia
for el in highways2:
    if 'Lombardia' in el[2]:
        print(el)
print()

# 2.    stampare l'autostrada che attraversa più regioni
# lunghezze = [len(el[2]) for el in highways2]
# # print(lunghezze)
# imax = 0
# for i, num in enumerate(lunghezze):
#     if num > lunghezze[imax]:
#         imax = i
# print(highways2[imax])

imax = 0
for i, el in enumerate(highways2):
    if len(el[2]) > len(highways2[imax][2]):
        imax = i
print(highways2[imax])
print()

# 3.    stampare, tutte le autostrade, ordinate in base al numero di regioni attraversate,
# e a parità, in base alla loro lunghezza
highways2.sort(key= lambda x: (-len(x[2]), -x[-1]))
for el in highways2:
    print(el)
print()

# 4.    stampare per ogni regione, il numero di autostrade che la attraversano
regioni = []
for el in highways2:
    for regione in el[2]:
        if regione not in regioni:
            regioni.append(regione)
regioni = [[regione, 0] for regione in regioni]
for regione in regioni:
    for el in highways2:
        if regione[0] in el[2]:
            regione[1] += 1
for regione in regioni:
    print(regione)
print()

# 5.    stampare per ogni regione tutte le autostrade che la attraversano
for regione in regioni:
    nome = regione[0]
    print(f'{nome}: ', end='')

    # con list comprehension
    # autostrade = [el[0] for el in highways2 if nome in el[2]]
```

```
    # senza list comprehension
    autostrade = []
    for el in highways2:
        if nome in el[2]:
            autostrade.append(el[0])

    print(', '.join(autostrade))

# 6.    stampare per ogni regione tutte le autostrade che la attraversano, ordinandole
# in base al numero di regioni attraversate

for regione in regioni:
    nome = regione[0]

    # con list comprehension
    autostrade = [el for el in highways2 if nome in el[2]]
    autostrade.sort(key= lambda x: -len(x[2]))
    print(f'{nome}:')
    for el in autostrade:
        print(' -', el)
```

3.6.7

3.7 Algoritmi di ordinamento base

3.7.1 Carica un array di numeri casuali e utilizza il selection sort per ordinarlo. Deve essere fatta una stampa dell'array prima e dopo l'ordinamento

3.7.2 Carica un array di numeri casuali e utilizza il bubble sort per ordinarlo. Deve essere fatta una stampa dell'array prima e dopo l'ordinamento

3.7.3 Carica un array di numeri casuali e utilizza la funzione sort della libreria standard per ordinarlo. Deve essere fatta una stampa dell'array prima e dopo l'ordinamento

3.7.4 Scrivi un programma che dato un array di numeri interi, sia in grado di:

1. controllare se l'array è palindromo,
2. invertire l'ordine dei numeri dell'array,
3. ordinare in ordine decrescente l'array.

Soluzione

```
numeri = [10,13,6,13,7]

# controllare se l'array è palindromo
if numeri == numeri[::-1]:
    print(f"{numeri} è palindromo")
else:
```

```
    print(f"{numeri} non è palindromo")

# invertire l'ordine dei numeri dell'array,
numeri = numeri[::-1]
print(numeri)

# ordinare in ordine decrescente l'array.
numeri.sort(reverse=True)
print(numeri)
```

3.7.5 Scrivi un programma che ordini un array con l'algoritmo selection sort, prima in ordine crescente poi in ordine decrescente.

3.7.6 Scrivi un programma che ordini un array con l'algoritmo bubble sort, prima in ordine crescente poi in ordine decrescente.

3.7.7 Scrivi un programma che dato un array di dimensione n e due valori x y tali che x<=y<=n, ordini l'array dalla posizione x alla posizione y comprese.

3.7.8 Dato un array di stringhe, che puoi riempire come preferisci, ordina l'array in ordine alfabetico prima crescente e poi decrescente. Devono essere inserite le stampe dell'intero array prima e dopo ogni ordinamento.

Soluzione

```
elenco = ["ciao belli", "come state?", "Io bene"]
print(elenco)

elenco.sort()
print(elenco)

elenco.sort(reverse=True)
print(elenco)
```

3.7.9 Crea un programma di gestione di una agenda. L'agenda deve permettere di visualizzare e inserire i dati riguardanti i propri contatti. Per ogni persona devono essere memorizzati nome cognome e numero di telefono. La visualizzazione dei dati deve avvenire in ordine alfabetico per nome o cognome (scegli tu). (Non avendo ancora fatto la gestione dei files ogni volta l'agenda partirà da uno stato predefinito che puoi decidere tu).

3.7.10 Con la tecnica del Bubble Sort (2 cicli FOR e una IF):

1. ordinare la seguente lista di numeri in ordine crescente
   lista1 = [8, 5, 7, 12, 18, 17, 3, 5]

2. ordinare la seguente lista di capitali in ordine alfabetico
   lista2 = ["Tallinn", "Vilnius", "Riga", "Copenaghen", "Stoccolma", "Helsinki"]
3. ordinare la lista delle capitali in base alla lunghezza del nome della Capitale (e a parità di lunghezza in base al nome della capitale)

3.7.11 La seguente lista di stringhe contiene una serie di stringhe che descrivono alcuni celebri immunologi:

imLista = [

"Pasteur 1822 1895 Francia", "Koch 1843 1910 Germania", "Fleming 1881 1955 RegnoUnito",

"Dulbecco 1914 2012 Italia", "Montagnier 1932 2022 Francia", "Gallo 1937 1982 Usa",

"Gram 1853 1938 Danimarca", "Jenner 1749 1823 RegnoUnito", "Sabin 1906 1993 Usa",

]

1. stampare gli immunologi ordinati in base a quanti anni sono vissuti
2. stampare gli immunologi ordinati in base alla nazione, e a parità di nazione in base al nome

Soluzione

```
# La seguente lista di stringhe contiene una serie di stringhe
# che descrivono alcuni celebri immunologi:

# 1) stampare gli immunologi ordinati in base a quanti anni sono vissuti
# 2) stampare gli immunologi ordinati in base alla nazione, e a parità di nazione in base al nome

imLista = [
    "Pasteur 1822 1895 Francia", "Koch 1843 1910 Germania", "Fleming 1881 1955 RegnoUnito",
    "Dulbecco 1914 2012 Italia", "Montagnier 1932 2022 Francia", "Gallo 1937 1982 Usa",
    "Gram 1853 1938 Danimarca", "Jenner 1749 1823 RegnoUnito", "Sabin 1906 1993 Usa",
]

lista = [el.split(" ") for el in imLista]
lista = [[el[0], int(el[2])-int(el[1]), el[3]] for el in lista]
# ordino in base all'età
lista.sort(key=lambda el: el[1])
print(lista)
#ordino in base a nazione e poi nome
lista.sort(key=lambda el: (el[2], el[0]))
print(lista)
```

3.7.12 Data la seguente lista:

highways = [

"A1 Milano-Napoli 759",

"A2 Salerno-ReggioCalabria 442",

```
    "A3 Napoli-Salerno 52",
    "A4 Torino-Trieste 524",
    "A6 Torino-Savona 124",
    "A7 Milano-Genova 133",
    "A8 Milano-Varese 43",
    "A10 Savona-Ventimigla 113",
    "A11 Firenze-Pisa 81",
    "A12 Genova-Cecina 210",
    "A13 Bologna-Padova 116",
    "A14 Bologna-Taranto 743",
    "A15 Parma-LaSpezia 108",
    "A16 Napoli-Canosa 172",
    "A19 Palermo-Catania 191",
    "A21 Torino-Piacenza-Brescia 238",
    "A22 Brennero-Modena 315",
    "A23 Palmanova-Tarvisio 119",
    "A24 Roma-Teramo 159",
    "A25 Torano-Pescara 115",
    "A26 Genova-Gravellona 197"
]
```

1. Stampare l'autostrada più corta e quella più lunga (Risultato A1 Milano-Napoli 759 A8 Milano-Varese 43)
2. stampare la classifica delle autostrade ordinate per lunghezza
3. stampare le prime tre città che sono capolinea di una autostrada

3.7.13 La seguente lista, contiene alternati i nomi di una materia e il voto finale nella materia

lista = ["matematica", 5.5, "storia", 8, "italiano", 7, "inglese", 6.5 , "fisica", 6]

1. stampare la materia che ha preso il voto più alto
2. utilizzando la tecnica del bubble sort - stampare le materie ordinate in base al voto
3. stampare le materie ordinate in base al voto

3.8 Dizionari

3.8.1 Scrivi una funzione a cui passare una stringa come parametro, e che restituisca un dizionario rappresentante la "frequenza di comparsa" di ciscun carattere componente la stringa.

Semplicemente, data una stringa "ababcc", otterremo in risultato {"a": 2, "b": 2, "c": 2}

Soluzione

```
def f(stringa):
    d = {}
```

```
    for lettera in stringa:
        if lettera in d:
            d[lettera] += 1
        else:
            d[lettera] = 1
    return d


d = f("Ciao a tutti, come state?")
for key in d:
    print(f"{key}: {d[key]}")
```

3.8.2 Scrivi un programma che utilizzi una funzione che, ricevuto un array di numeri interi, restituisce il numero che compare con più frequenza (in caso di parità restituisce il primo numero trovato con la frequenza maggiore). Puoi risolvilo in due modi:

1. Con sole liste (più lungo e complicato)
2. Con un dizionario

Soluzione

```
def numero_con_frequenza_massima_lista(numeri):
    frequenze = [0 for _ in range(len(numeri))]
    for i in range(len(numeri)):
        if frequenze[i] == 0:
            frequenze[i] = 1
            for j in range(i+1, len(numeri)):
                if(numeri[i] == numeri[j]):
                    frequenze[i] += 1
                    frequenze[j] += 1

    val_max = frequenze[0]
    pos_max = 0
    for i in range(1, len(numeri)):
        if frequenze[i] > val_max:
            val_max = frequenze[i]
            pos_max = i

    return numeri[pos_max]


def numero_con_frequenza_massima_dizionario(lista):
   d = {}
    for num in lista:
        if num in d:
            d[num] += 1
        else:
            d[num] = 1

    max = None
    freqmax = 0
    for key in d:
        if d[key] > freqmax:
            max = key
            freqmax = d[key]

    return max, freqmax

numeri = [1,1,2,2,2,3,4,8,6,4,4]
print(numeri)
print("Primo numero con frequenza massima (liste):", numero_con_frequenza_massima_lista(numeri))
```

```
print("Primo numero con frequenza massima (dizionario):",
numero_con_frequenza_massima_dizionario(numeri))
```

3.8.3 Scrivi e usa una funzione che presa in input una lista contenente interi e stringhe, modifica la lista distruttivamente e restituisce un dizionario. Al termine della funzione dalla lista devono risultare cancellate tutte le stringhe e il dizionario restituito deve contenere come chiavi le stringhe cancellate ciascuna con attributo il numero di volte in cui occorrevano nella lista.

Ad esempio per lista=[1,'a',2,'b','a',8,'d',8] la funzione al termine restituisce il dizionario {'a':2,'b':1,'d':1} e la lista diventa [1,2,8,8]

Soluzione

```
def f(lista):
    ris = {}
    i = 0
    while i < len(lista):
        if type(lista[i]) == str:
            if lista[i] in ris:
                ris[lista[i]] += 1
            else:
                ris[lista[i]] = 1

            lista.pop(i)
        else:
            i += 1

    return ris



print(f([1,'a',2,'b','a',8,'d',8]))
```

3.8.4 Data la seguente lista di stringhe che rappresenta un girone di Champions:

```
champions= [
    "Sheriff Tiraspol - Shaktar Donetsk;  2 0",
    "Inter - Real Madrid; 0 1",
    "Shaktar Donetsk  - Inter; 0 0",
    "Real Madrid - Sheriff Tiraspol; 1 2",
    "Shaktar Donetsk  - Real Madrid;  0 5",
    "Inter - Sheriff Tiraspol;  3 1",
    "Real Madrid - Shaktar Donetsk;  2 1",
    "Sheriff Tiraspol – Inter;  1 3",
    "Inter - Shaktar Donetsk;  2 0",
    "Sheriff Tiraspol - Real Madrid;  0 3",
    "Shaktar Donetsk  -  Sheriff Tiraspol;  1 1",
```

```
    "Real Madrid – Inter;   2 0"
]
```

Sapendo che la vittoria vale 3 punti, il pareggio 1 punto e la sconfitta 0 punti, calcola e stampa per ogni squadra, quanti punti ha fatto

Variante: togli dalla stringa in input i punti e virgola che separano i nomi delle squadre dai goal fatti.

Soluzione

```
# 3.8.4 - Data la seguente lista di stringhe che rappresenta un girone di Champions:

champions= [
    "Sheriff Tiraspol - Shaktar Donetsk;  2 0",
    "Inter - Real Madrid; 0 1",
    "Shaktar Donetsk  - Inter; 0 0",
    "Real Madrid - Sheriff Tiraspol; 1 2",
    "Shaktar Donetsk  - Real Madrid;  0 5",
    "Inter - Sheriff Tiraspol;  3 1",
    "Real Madrid - Shaktar Donetsk;  2 1",
    "Sheriff Tiraspol - Inter;  1 3",
    "Inter - Shaktar Donetsk;  2 0",
    "Sheriff Tiraspol - Real Madrid;  0 3",
    "Shaktar Donetsk  -  Sheriff Tiraspol;  1 1",
    "Real Madrid - Inter;   2 0"
]

# Sapendo che la vittoria vale 3 punti, il pareggio 1 punto e la sconfitta 0 punti,
# calcola e stampa per ogni squadra, quanti punti ha fatto
# Variante: togli dalla stringa in input i punti e virgola che
# separano i nomi delle squadre dai goal fatti.

d = {}
for riga in champions:
    riga = riga.strip().split(';')
    squadre = riga[0].strip().split('-')
    sq1 = squadre[0].strip()
    sq2 = squadre[1].strip()
    goals = riga[1].strip().split()
    g1 = int(goals[0].strip())
    g2 = int(goals[1].strip())

    if g1 > g2:
        p1 = 3
        p2 = 0
    elif g2 > g1:
        p2 = 3
        p1 = 0
    else:
        p1 = 1
        p2 = 1

    if sq1 not in d:
        d[sq1] = p1
    else:
        d[sq1] += p1

    if sq2 not in d:
        d[sq2] = p2
    else:
        d[sq2] += p2

classifica = list(d.items())
```

```
classifica.sort(key = lambda x: -x[1])
for el in classifica:
    print(el)
```

3.8.5 Dato il seguente dizionario:

d = {

"senegal" : ("africa", "francese"), "quebec" : ("nordAmerica", "francese"),

"stati uniti" : ("nordAmerica", "inglese"), "canada" : ("nordAmerica", "inglese"),

"brasile" : ("sudAmerica", "portoghese"), "messico" : ("nordAmerica", "spagnolo"),

"australia" : ("oceania", "inglese"), "haiti" : ("centroAmerica", "francese"),

"angola" : ("africa", "portoghese"), "ciad" : ("africa", "francese"),

"filippine" : ("asia", "spagnolo"), "argentina" : ("sudAmerica", "spagnolo"),

"neoZelanda" : ("oceania", "inglese"), "cuba" : ("centroAmerica", "spagnolo"),

"mozambico" : ("africa", "portoghese"), "mali" : ("africa", "francese"),

"nigeria" : ("africa", "inglese"), "martinica" : ("centroAmerica", "francese"),

"uruguay" : ("sudAmerica", "spagnolo"), "cile" : ("sudAmerica", "spagnolo"),

"caienna" : ("sudAmerica", "francese"), "perù" : ("sudAmerica", "spagnolo")

}

1. stampare tutti i paesi di lingua francese
2. per ogni lingua, stampare il numero di paesi
3. per ogni lingua stampare la lista ordinata dei paesi che parlano appunto quella lingua

Soluzione

```
import os
os.system("cls")

d = {
    "senegal": ("africa", "francese"), "quebec": ("nordAmerica", "francese"),
    "stati uniti": ("nordAmerica", "inglese"), "canada": ("nordAmerica", "inglese"),
    "brasile": ("sudAmerica", "portoghese"), "messico": ("nordAmerica", "spagnolo"),
    "australia": ("oceania", "inglese"), "haiti": ("centroAmerica", "francese"),
    "angola": ("africa", "portoghese"), "ciad": ("africa", "francese"),
    "filippine": ("asia", "spagnolo"), "argentina": ("sudAmerica", "spagnolo"),
    "neoZelanda": ("oceania", "inglese"), "cuba": ("centroAmerica", "spagnolo"),
    "mozambico": ("africa", "portoghese"), "mali": ("africa", "francese"),
    "nigeria": ("africa", "inglese"), "martinica": ("centroAmerica", "francese"),
    "uruguay": ("sudAmerica", "spagnolo"), "cile": ("sudAmerica", "spagnolo"),
    "caienna": ("sudAmerica", "francese"), "perù": ("sudAmerica", "spagnolo")
}

# 1.    stampare tutti i paesi di lingua francese
print("Paesi in cui si parla francese:")
for key, val, in d.items():
    if val[1] == "francese":
        print(key)
print("")

print("Dizionario dopo l'inserimento:")
```

```
for key, val, in d.items():
    print(f"{key}: {val}")
print("")

# 2.    per ogni lingua, stampare il numero di paesi
lingue = {}
for nazione in d:
    lingua = d[nazione][1]
    if lingua in lingue:
        lingue[lingua] += 1
    else:
        lingue[lingua] = 1

print("Numero di paesi per ogni lingua:")
for tupla in lingue.items():
    print(tupla)
print("")

# 3.    per ogni lingua stampare la lista ordinata dei paesi che parlano appunto quella lingua
lingue = {}
for nazione in d:
    lingua = d[nazione][1]
    if lingua in lingue:
        lingue[lingua].append(nazione)
    else:
        lingue[lingua] = [nazione]

print("Lista ordinata di paesi per ogni lingua:")
for lingua in lingue:
    lingue[lingua].sort()
    print(f"{lingua}: {lingue[lingua]}")
```

3.8.6 Data la seguente lista di numeri

lista = [882, 1024, 917, 777, 89, 7, 274, 112, 898, 666, 76, 75737, 6767, 39]

stampare la cifra più presente

(Risultato: la cifra 7 è presente 12 volte)

Soluzione

```
import os
os.system("cls")

lista = [882, 1024, 917, 777, 89, 7, 274, 112, 898,  666, 76, 75737, 6767, 39]

d = {}
for numero in lista:
    s = str(numero)
    for cifra in s:
        cifra = int(cifra)
        if cifra in d:
            d[cifra] += 1
        else:
            d[cifra] = 1

print(d)

classifica = list(d.items())
print(classifica)
classifica.sort(key = lambda x: (-x[1], x[0]))
```

```
for el in classifica:
    print(f"{el[0]}: {el[1]}")
print()

classifica = dict(classifica)
for key in classifica:
    print(f"{key}: {classifica[key]}")
```

3.8.7 (dizionario di dizionario --> il secondo dizionario è in pratica usato come fosse una struct)

```
d = {
    "Atalanta": {"city": "Bergamo", "country": "Italia"},
    "Hellas": {"city": "Verona", "country": "Italia"},
    "Sampdoria": {"city": "Genova", "country": "Italia"},
    "Genoa": {"city": "Genova", "country": "Italia"},
    "Liverpool": {"city": "Liverpool", "country": "RegnoUnito},
    "Everton": {"city": "Liverpool", "country": "RegnoUnito"},
    "Espanol": {"city": "Barcellona", "country": "Spagna"},
    "Barcellona": {"city": "Barcellona", "country": "Spagna"},
    "Siviglia": {"city": "Siviglia", "country": "Spagna"},
    "Betis": {"city": "Siviglia", "country": "Spagna"},
    "Juventus": {"city": "Torino", "country": "Italia"},
    "YoungBoys": {"city": "Berna", "country": "Svizzera"},
    "Chievo": {"city": "Verona", "country": "Italia"}
}
```

1. Trovare la città dell'Hellas, e stampare il nome dell'altra squadra della stessa città
2. stampare in ordine alfabetico tutte le squadre italiane
3. Stampare le sole squadre italiane, ordinate in base alla città, e a parità di città in ordine alfabetico

Soluzione

```
# Riprendendo il dizionario dell'esercizio precedente:
# 1) Trovare la città dell'Hellas, e stampare il nome dell'altra squadra della stessa città
# 2) stampare in ordine alfabetico tutte le squadre italiane
# 3)Stampare le sole squadre italiane , ordinate in base alla città, e a parita di città in
ordine alfabetico

d = {
    "Atalanta": {"city": "Bergamo", "country": "Italia"},
    "Hellas":    {"city": "Verona", "country": "Italia"},
    "Sampdoria": {"city": "Genova", "country": "Italia"},
    "Genoa":    {"city": "Genova", "country": "Italia"},
    "Liverpool": {"city": "Liverpool", "country": "RegnoUnito"},
    "Everton": {"city": "Liverpool", "country": "RegnoUnito"},
    "Espanol": {"city": "Barcellona", "country": "Spagna"},
    "Barcellona": {"city": "Barcellona", "country": "Spagna"},
```

```
    "Siviglia": {"city": "Siviglia", "country": "Spagna"},
    "Betis":    {"city": "Siviglia", "country": "Spagna"},
    "Juventus":    {"city": "Torino", "country": "Italia"},
    "YoungBoys":    {"city": "Berna", "country": "Svizzera"},
    "Chievo":    {"city": "Verona", "country": "Italia"}
}

# 1) Trovare la città dell'Hellas, e stampare il nome dell'altra squadra della stessa città
print("1) Trovare la città dell'Hellas, e stampare il nome dell'altra squadra della stessa
città")
citta = d["Hellas"]["city"]
for key, value in d.items():
    if key != "Hellas" and value["city"] == citta:
        print(key)

# 2) stampare in ordine alfabetico tutte le squadre italiane
print("\n2) stampare in ordine alfabetico tutte le squadre italiane")
italiane = [key for key, value in d.items() if value["country"] == "Italia"]
print(sorted(italiane))

# 3)Stampare le sole squadre italiane , ordinate in base alla città, e a parita di città in
ordine alfabetico
print("\n3)Stampare le sole squadre italiane , ordinate in base alla città, e a parita di città
in ordine alfabetico")
## metodo con le liste
# italiane = [(key, list(value.values())[0], list(value.values())[1]) for key, value in d.items()
if d[key]["country"] == "Italia"]
italiane = [(key, list(value.values())) for key, value in d.items() if d[key]["country"] ==
"Italia"]
italiane.sort(key=lambda el: (el[1], el[0]))
print(italiane)
```

3.8.8 Dato il seguente dizionario di dizionari:

```
d = {
    "A1" : {"percorso":"Milano-Napoli", "gestore":"Autostrade per l'Italia", "km":759},
    "A2" : {"percorso":"Salerno-ReggioCalabria", "gestore":"Anas", "km":442},
    "A3" : {"percorso":"Napoli-Salerno", "gestore":"Autostrade per l'Italia", "km":52},
    "A4" : {"percorso":"Torino-Trieste", "gestore":"Autostrade per l'Italia", "km":524},
    "A6" : {"percorso":"Torino-Savona", "gestore":"Gruppo Gavio", "km":124},
    "A7" : {"percorso":"Milano-Genova", "gestore":"Enti Pubblici", "km":133},
    "A8" : {"percorso":"Milano-Varese", "gestore":"Autostrade per l'Italia", "km":43},
    "A10" : {"percorso":"Savona-Ventimiglia", "gestore":"Gruppo Gavio", "km":113},
    "A11" : {"percorso":"Firenze-Pisa", "gestore":"Autostrade per l'Italia", "km":81},
    "A12" : {"percorso":"Genova-Cecina", "gestore":"Gruppo Gavio", "km":210},
    "A13" : {"percorso":"Bologna-Padova", "gestore":"Autostrade per l'Italia", "km":116},
    "A14" : {"percorso":"Bologna-Taranto", "gestore":"Autostrade per l'Italia", "km":743},
    "A15" : {"percorso":"Parma-LaSpezia", "gestore":"Gruppo Gavio", "km":108},
    "A16" : {"percorso":"Napoli-Canosa", "gestore":"Autostrade per l'Italia", "km":172},
    "A19" : {"percorso":"Palermo-Catania", "gestore":"Anas", "km":191},
    "A21" : {"percorso":"Torino-Piacenza-Brescia", "gestore":"Gruppo Gavio", "km":238},
    "A22" : {"percorso":"Brennero-Modena", "gestore":"Enti Pubblici", "km":315},
    "A23" : {"percorso":"Palmanova-Tarvisio", "gestore":"Enti Pubblici", "km":119},
```

```
    "A24" : {"percorso":"Roma-Teramo", "gestore":"Gruppo Toto", "km":159},
    "A25" : {"percorso":"Torano-Pescara", "gestore":"Gruppo Toto", "km":115},
    "A26" : {"percorso":"Genova-Gravellona", "gestore":"Autostrade per l'Italia", "km":197},
}
```

1. Stampare il gestore dell'autostrada A8
2. stampare il gestore dell'autostrada "Savona-Ventimiglia"
3. stampare l'autostrada più lunga, con il suo percorso e il suo gestore
4. Stampare quanti differenti gestori sono presenti nel dizionario (per la soluzione utilizzare una semplice lista)
5. Per ognuno dei gestori presenti stampare il numero di autostrade gestite, e il numero di km gestiti

Soluzione

```
# Dato il seguente dizionario di dizionario:

d = {
    "A1" : {"percorso":"Milano-Napoli", "gestore":"Autostrade per l'Italia", "km":759},
    "A2" : {"percorso":"Salerno-ReggioCalabria", "gestore":"Anas", "km":442},
    "A3" : {"percorso":"Napoli-Salerno", "gestore":"Autostrade per l'Italia", "km":52},
    "A4" : {"percorso":"Torino-Trieste", "gestore":"Autostrade per l'Italia",
"km":524},
    "A6" : {"percorso":"Torino-Savona", "gestore":"Gruppo Gavio", "km":124},
    "A7" : {"percorso":"Milano-Genova", "gestore":"Enti Pubblici", "km":133},
    "A8" : {"percorso":"Milano-Varese", "gestore":"Autostrade per l'Italia", "km":43},
    "A10" : {"percorso":"Savona-Ventimiglia", "gestore":"Gruppo Gavio", "km":113},
    "A11" : {"percorso":"Firenze-Pisa", "gestore":"Autostrade per l'Italia", "km":81},
    "A12" : {"percorso":"Genova-Cecina", "gestore":"Gruppo Gavio", "km":210},
    "A13" : {"percorso":"Bologna-Padova", "gestore":"Autostrade per l'Italia",
"km":116},
    "A14" : {"percorso":"Bologna-Taranto", "gestore":"Autostrade per l'Italia",
"km":743},
    "A15" : {"percorso":"Parma-LaSpezia", "gestore":"Gruppo Gavio", "km":108},
    "A16" : {"percorso":"Napoli-Canosa", "gestore":"Autostrade per l'Italia",
"km":172},
    "A19" : {"percorso":"Palermo-Catania", "gestore":"Anas", "km":191},
    "A21" : {"percorso":"Torino-Piacenza-Brescia", "gestore":"Gruppo Gavio", "km":238},
    "A22" : {"percorso":"Brennero-Modena", "gestore":"Enti Pubblici", "km":315},
    "A23" : {"percorso":"Palmanova-Tarvisio", "gestore":"Enti Pubblici", "km":119},
    "A24" : {"percorso":"Roma-Teramo", "gestore":"Gruppo Toto", "km":159},
    "A25" : {"percorso":"Torano-Pescara", "gestore":"Gruppo Toto", "km":115},
    "A26" : {"percorso":"Genova-Gravellona", "gestore":"Autostrade per l'Italia",
"km":197}
}

# 1) Stampare il gestore dell'autostrada A8
```

```
print("\n\nGestore dell'autostrada A8:")
print(" -", d["A8"]["gestore"])

# 2) stampare il gestore dell'autostrada "Savona-Ventimiglia"
print("\n\nGestore dell'autostrada \"Savona-Ventimiglia\":")
for aut, dat in d.items():
    if dat["percorso"] == "Savona-Ventimiglia":
        print(" -", dat["gestore"])

# 3) stampare l'autostrada più lunga, con il suo percorso e il suo gestore
print("\n\nL'autostrada più lunga, con il suo percorso e il suo gestore:")
kmax = None
lungmax = None
for aut, dat in d.items():
    if lungmax == None or lungmax < dat["km"]:
        kmax = aut
        lungmax = dat["km"]
print(" -", kmax, d[kmax])

# 4) Stampare quanti differenti gestori sono presenti nel dizionario (per la soluzione
utilizzare una semplice lista)
print("\n\nNumero di gestori:")
gestori = []
for aut, dat in d.items():
    if dat["gestore"] not in gestori:
        gestori.append(dat["gestore"])
print(" -", len(gestori))

# 5) Per ognuno dei gestori presenti stampare il numero di autostrade gestite, e il
numero di km gestiti
print("\n\nPer ognuno dei gestori presenti: il numero di autostrade gestite, e il
numero di km gestiti:")
gestori = {}
for aut, dat in d.items():
    if dat["gestore"] not in gestori:
        gestori[dat["gestore"]] = [1, dat["km"]]
    else:
        gestori[dat["gestore"]][0] += 1
        gestori[dat["gestore"]][1] += dat["km"]

for gestore, dati in gestori.items():
    print(" -", gestore, dati[0], dati[1])
```

3.8.9 Il seguente dizionario descrive quante ore hanno lavorato in una settimana, una serie di dipendenti: ad esempio Mr White ha lavorato 8 ore il lunedì, 9 ore il martedì, 8 ore il mercoledì, 9 ore il giovedì e 4 ore il venerdì

```
d2 = {
    "Mr White": [8, 9, 8, 9, 4],
    "Mr Brown": [0, 8, 7, 10, 8],
    "Mr Blonde": [8, 8, 8, 9, 9],
    "Mr Black": [6, 9, 9, 8, 8],
    "Mr Red": [ 8, 7, 8, 8, 8],
    "Mr Green": [ 4, 8, 8, 8, 4]
}
```

1. Stampare la classifica dei lavoratori, in base al numero di ore lavorate
2. Stampare il giorno con più ore lavorate, e quante ore sono state lavorate in quel giorno

Soluzione

```python
giorni = ['Lunedì', 'Martedì', 'Mercoledì', 'Giovedì', 'Venerdì']

d = {
    "Mr White": [8, 9, 8, 9, 4],
    "Mr Brown": [0, 8, 7, 10, 8],
    "Mr Blonde": [8, 8, 8, 9, 9],
    "Mr Black": [6, 9, 9, 8, 8],
    "Mr Red": [ 8, 7, 8, 8, 8],
    "Mr Green": [ 4, 8, 8, 8, 4]
}

# 1.  Stampare la classifica dei lavoratori, in base al numero di ore lavorate
mat = [[key, val, sum(val)] for key, val in d.items()]
mat.sort(key = lambda x: -x[2])
for riga in mat:
    print(f"{riga[0]}: {riga[2]}")

# 2.  Stampare il giorno con più ore lavorate, e quante ore sono state lavorate
# in quel giorno
jmax = None
max = None
for j in range(len(mat[0][1])):
    somma = 0
    for i in range(len(mat)):
        somma += mat[i][1][j]
    if max == None or somma > max:
        jmax = j
        max = somma

print(f"Il giorno in cui si è lavorato di più è {giorni[jmax]} in cui si è lavorato in torale per
{max} ore.")
```

3.8.10 progettare la funzione f(ftesto,k) che, presi in input il nome (o l'indirizzo) di un file di testo ed un intero k, restituisce una stringa di caratteri lunga k. Il file di testo contiene stringhe di diversa lunghezza (una per riga ed ogni riga termina con '\n'), un possibile testo da inserire nel file di input è riportato di seguito.

I k caratteri della stringa restituita dalla funzione si ottiengono considerando le stringhe lunghe k presenti nel file di testo. L'i-mo carattere della stringa sara' il carattere che

compare con maggior frequenza come i-mo carattere delle stringhe lunghe k nel file di testo (in caso di parita' di occorrenze viene scelto il carattere che precede gli altri lessicograficamente).

Nel caso il file di testo non contenga parole lunghe k allora viene restituita

la stringa vuota.

Ad Esempio, per il file di testo:

```
porta
rigore
ora
giacca
follia
gara
finale
salute
fiore
lampada
cane
erba
palo
tre
due
fionda
limite
polo
soglia
amo
```

e k=3 la funzione restituisce la stringa 'are' a seguito della presenza delle seguenti 4 stringhe lunghe 3:

```
tre
due
amo
ora
```

Soluzione

```python
def f(ftesto,k):

    with open(ftesto) as f:
        parole = f.read().split()

    parole = [parola for parola in parole if len(parola) == k]

    ris = ""
    for j in range(k):
        freq = {}
        for parola in parole:
            if parola[j] in freq:
                freq[parola[j]] += 1
            else:
                freq[parola[j]] = 1

        lista = list(freq.items())
        lista.sort(key=lambda el: (-el[1], el[0]))
```

```
        ris += lista[0][0]

    return ris


    ## alternativa

    # with open(ftesto) as f:
    #     parole = f.read().split()

    # parole = [parola for parola in parole if len(parola) == k]
    # if len(parole) == 0:
    #     return ''
    # # print(parole)
    # lettere_per_posizione = [[parola[i] for parola in parole] for i in range(k)]
    # # print(lettere_per_posizione)
    # frequenze = [{c: 0 for c in posizione} for posizione in lettere_per_posizione]
    # for i in range(k):
    #     for lettera in lettere_per_posizione[i]:
    #         frequenze[i][lettera] += 1
    # # print(frequenze)
    # frequenze = [[(c, posizione[c]) for c in posizione] for posizione in frequenze]
    # # print(frequenze)
    # for posizione in frequenze:
    #     posizione.sort(key=lambda x: (-x[1], x[0]))
    # # print(frequenze)
    # parola = [posizione[0][0] for posizione in frequenze]
    # parola = ''.join(parola)
    # # print(parola)
    # return parola



ftesto, k = 'nomefile.txt', 3
print(f(ftesto,k))
```

4. Funzioni

4.1 Funzioni semplici

4.1.1 Scrivi e utilizza una funzione per calcolare e restituire la somma tra due numeri

4.1.2 Scrivi una funzione che prende due numeri come parametro e manda in print il più grande tra i due.

4.1.3 Scrivi e usa una funzione che, passata come parametro una lista di interi, restituisce il maggiore tra i numeri contenuti nella lista.

4.1.4 Scrivi e utilizza 4 funzioni che effettuano le 4 operazioni e restituiscono il risultato.

4.1.5 Scrivi e utilizza una funzione che dati due valori e un simbolo di una delle 4 operazioni effettua quella operazione e restituisce il risultato

Soluzione

```
def calcola(a,b,simbolo):
    if simbolo == '+':
        return a+b
    elif simbolo == '-':
        return a-b
    elif simbolo == '*':
        return a*b
    elif simbolo == '/':
        return a/b
    else:
        print(f"Operatore \"{simbolo}\" errato")
        return 0

a = float(input("Inserisci il primo numero: "))
b = float(input("Inserisci il secondo numero: "))
simbolo = (input("Inserisci l'operatore: "))

print(calcola(a,b,simbolo))
```

4.1.6 Quando possibile, scrivi e utilizza le seguenti funzioni:

1. raddoppia un numero passando parametri per valore,
2. raddoppia un numero passando parametri per indirizzo,
3. raddoppia due numeri contemporaneamente.

Soluzione

```
# 1. raddoppia un numero passando parametri per valore,

# il passaggio è sempre per riferimento ma se riassegno la variabile perdo il riferimento alla
variabile originale
# è quindi possibile solo restituire il doppio
def raddoppia(x):
    x *= 2
    return x

a = 5
```

```
a = raddoppia(a)
print("1.", a)


# 2. raddoppia un numero passando parametri per indirizzo,

# il passaggio è sempre per riferimento ma se riassegno la variabile perdo il riferimento alla variabile originale
# vediamo che cosa succede se provo a farlo
def raddoppia2(x):
    x *= 2

a = 5
raddoppia2(a)
print("2.", a)

# 3. raddoppia due numeri contemporaneamente.
# quello che posso fare è passare una struttura dati iterabile e raddoppiare tutti i valori in essa contenuti,
# in questo caso non modifico il riferimento alla struttura originale ma solo il contenuto
def raddoppia3(v):
    for i in range(len(v)):
        v[i] *=2


# a = (2, 3) # attento con una tupla non lo puoi fare, la tupla è immodificabile
a = [2, 3]
raddoppia3(a)
print("3.", a)
```

4.1.7 Scrivi e utilizza una funzione che restituisca il prodotto di due numeri e alla fine dell'esecuzione lasci modificati i due numeri in modo che valgano il doppio di prima.

4.1.8 Funzione che date le coordinate di due punti nel piano, calcolino e restituiscano le coordinate del punto medio tra i due (senza approssimazioni).

Soluzione

```
def punto_medio(a, b):
    medio = ((a[0]+b[0])/2, (a[1]+b[1])/2)
    return medio

a=(1,2)
b=(2,2)
print(punto_medio(a,b))
```

4.1.9 Funzione che dato il tempo di caduta di un oggetto, calcola l'altezza da cui è caduto l'oggetto e la velocità di impatto. Le formule che ti servono sono: h = 1/2 * g * t^2, con g=9,81 e v=g*t.

Soluzione

```
def altezza_e_velocita(t):
    return (1/2*9.81*t**2, 9.81*t)


t = 1
h, v = altezza_e_velocita(t)
print(
    f"Se il tempo di caduta è {t} secondi, allora l'oggetto è caduto di {h} metri e ha urtato il
suolo ad una velocità di {v} m/s")
```

4.1.10 Funzione che prevede se un oggetto si rompe cadendo da una certa altezza. Quello che ti serve sapere è quindi l'altezza da cui cade l'oggetto e la velocità massima di impatto che l'oggetto può sopportare. Le formule che ti servono sono: h = 1/2 * g * t^2, con g=9,81 e v=g*t.

Soluzione

```
def altezza_e_velocita(t):
    return (1/2*9.81*t**2, 9.81*t)


t = 2
max_v = 10
h, v = altezza_e_velocita(t)
print(
    f"Se il tempo di caduta è {t} secondi, allora l'oggetto è caduto di {h} metri e ha urtato il
suolo ad una velocità di {v} m/s")

if v > max_v:
    print("L'oggetto si è rotto all'impatto")
else:
    print("Dopo l'impatto l'oggetto è ancora intero")
```

4.1.11 Scrivi una funzione che data in ingresso una lista A contenente n parole, restituisca in output una lista B di interi che rappresentano la lunghezza delle parole contenute in A.

4.1.12 Scrivi ed utilizza una funzione che dati due numeri li aumenta entrambi di un certo valore passato come parametro.

4.1.13 Scrivi e utilizza una funzione che dato un array di numeri, trova e restituisce il massimo e il minimo numero presenti.

4.1.14 Scrivi e utilizza una funzione che, data una stringa, restituisce true se la stringa inizia con una lettera maiuscola.

4.1.15 Funzione che data una stringa, la stampa separando ogni carattere della stringa con uno spazio.

4.1.16 Scrivi e utilizza una funzione che riceve come parametri due stringhe e restituisce una stringa che è la concatenazione delle altre due.

4.1.17 Scrivi un programma che, dati n voti in un array, li passi ad una funzione che restituisca la media dei voti escludendo il voto più alto e il voto più basso.

4.1.18 Scrivi e utilizza una funzione che stampa una stringa ricevuta come parametro.

4.1.19 Scrivi una funzione che scambia il contenuto di due variabili intere (i valori devono rimanere scambiati per chi chiama la funzione).

4.1.20 Scrivi un programma che sappia riordinare un array. Per farlo dividi le seguenti funzionalità in diverse funzioni:

1. funzione che riceve un array e lo riempie di numeri casuali
2. funzione che stampa un array
3. funzione che riceve un array e lo ordina; puoi usare l'algoritmo che preferisci
4. main che dichiara un array e poi usando le funzioni descritte sopra lo riempie di numeri casuali, lo stampa, lo ordina e poi lo ristampa.

Soluzione

```
import random


def stampa_lista(lista):
    for elemento in lista:
        print(elemento, end=" ")
    print()


def ordina(lista):
    if(type(lista) == list):
        # lista.sort()

        for i in range(len(lista)):
            for j in range(len(lista)-1):
                if(lista[j] > lista[j+1]):
```

```
                    lista[j], lista[j+1] = lista[j+1], lista[j]
        # print("\nlista ordinata")


lista = [random.randrange(10) for i in range(5)]
stampa_lista(lista)
ordina(lista)
stampa_lista(lista)
```

4.1.21 Scrivi e utilizza una funzione che dato un array restituisce i valori massimi e minimi dell'array

4.1.22 Scrivi e utilizza una funzione che calcola il quoziente tra due numeri interi e restituisce, oltre al quoziente anche il resto della divisione.

Soluzione

```
def divisione_intera(a,b):
    quoziente = a // b
    resto = a % b
    return (quoziente, resto)

a = 22
b = 3
q, r = divisione_intera(a,b)
print(f"{a}/{b} = {q} resto {r}")
```

4.1.23 Scrivi e utilizza una funzione che data una stringa, fa in modo che la prima lettera sia maiuscola e tutte le altre minuscole. i caratteri che non sono lettere non devono essere modificati.

4.1.24 Scrivi una funzione che calcola il massimo comune divisore tra due numeri.
Scrivi ed utilizza poi una funzione che dati due valori numeratore e denominatore, semplifica la frazione, usando la funzione mcd appena definita.

4.1.25 Scrivi ed utilizza una funzione che riceva come parametri tre numeri e che restituisca true se questi numeri possono essere le misure di un triangolo. In questo caso la funzione deve anche restituire una stringa in cui c'è scritto il tipo di triangolo (equilatero, isoscele, scaleno).

4.1.26 Scrivi ed utilizza una funzione che riceve una stringa e restituisce un'altra stringa ottenuta mischiando in ordine casuale le parole della stringa ricevuta. Attento che la stringa originale non va persa.

4.1.27 Scrivi ed utilizza una funzione che ti permetta di scegliere a caso quali studenti interrogare da una lista di nomi di studenti ricevuta in input. Oltre alla lista dei nomi la funzione riceve anche il numero di studenti da interrogare. È possibile interrogare tante volte uno studente prima di interrogarne altri. La lista infine restituisce la lista dei nomi degli studenti da interrogare.

4.1.28 Scrivi ed utilizza una funzione che ti permetta di scegliere a caso quali studenti interrogare da una lista di nomi di studenti ricevuta in input. Oltre alla lista dei nomi la funzione riceve anche il numero di studenti da interrogare. Se il numero di studenti da interrogare supera il numero di studenti, è possibile riestrarre studenti già interrogati, ma non prima di averli interrogati tutti (con questo metodo si possono anche fare tanti "giri" di interrogazioni). La lista infine restituisce la lista dei nomi degli studenti da interrogare.

4.2 Funzioni ricorsive

4.2.1 Scrivi e utilizza tre diverse versioni di una funzione che stampa una lista elemento per elemento. Le tre versioni sono:

1. Versione iterativa, cioè che usa un ciclo for
2. Versione ricorsiva che riceve ogni volta una lista più piccola della precedente (prima quella intera poi quella senza il primo elemento…)
3. Versione ricorsiva che riceve ogni volta la stessa lista ma sono espressi anche gli estremi della lista (inizio e fine) su cui lavorare.

Soluzione

```
# 6.1.1 - Scrivi e utilizza tre diverse versioni di una funzione
# che stampa una lista elemento per elemento. Le tre versioni sono:

#    1. Versione iterativa, cioè che usa un ciclo for
def stampaIterativa(lista):
    for el in lista:
        print(el, end = " ")

#    2. Versione ricorsiva che riceve ogni volta una lista più
# piccola della precedente (prima quella intera poi quella senza il primo
# elemento…)
def stampaRicorsiva(lista):
    if len(lista) < 1:
        return
    else:
        print(lista[0], end = " ")
        stampaRicorsiva(lista[1:])

#    3. Versione ricorsiva che riceve ogni volta la stessa
# lista ma sono espressi anche gli estremi della lista (inizio e fine) su cui
# lavorare.
def stampaRicorsiva2(lista, inizio, fine):
    if len(lista) < 1 or inizio > fine:
```

```
        return
    else:
        print(lista[inizio], end = " ")
        stampaRicorsiva2(lista, inizio+1, fine)

numeri = [1,2,3,4,5]
stampaIterativa(numeri)
print()
stampaRicorsiva(numeri)
print()
stampaRicorsiva2(numeri, 0, len(numeri)-1)
print()
```

4.2.2 Fai la stessa cosa dell'esercizio 4.2.1 ma stampa la lista al contrario.

Soluzione

```
# 6.1.2 - Fai la stessa cosa dell'esercizio 6.1.1
# ma stampa la lista al contrario.

#    1. Versione iterativa, cioè che usa un ciclo for
def stampaIterativa(lista, reverse = False):
    if not reverse:
        for el in lista:
            print(el, end = " ")
    else:
        for el in lista[::-1]:
            print(el, end = " ")

#    2. Versione ricorsiva che riceve ogni volta una lista più
# piccola della precedente (prima quella intera poi quella senza il primo
# elemento…)
def stampaRicorsiva(lista, reverse = False):
    if len(lista) < 1:
        return
    else:
        if not reverse:
            print(lista[0], end = " ")
            stampaRicorsiva(lista[1:], reverse)
        else:
            print(lista[-1], end = " ")
            stampaRicorsiva(lista[:-1], reverse)

#    3. Versione ricorsiva che riceve ogni volta la stessa
# lista ma sono espressi anche gli estremi della lista (inizio e fine) su cui
# lavorare.
def stampaRicorsiva2(lista, inizio = 0, fine = -1, reverse = False):
    # se fine è negativo va trasformato nella versione equivalente positiva
    if fine < 0:
        fine = len(lista)+fine # + perchè il numero è già negativo

    # se no qui fa subito return
    if inizio > fine:
        return
    elif inizio == fine:
        print(lista[inizio], end = " ")
    else:
        if reverse:
            print(lista[fine], end = " ")
            stampaRicorsiva2(lista, inizio, fine-1, reverse)
        else:
            print(lista[inizio], end = " ")
            stampaRicorsiva2(lista, inizio+1, fine, reverse)
```

```
numeri = [1,2,3,4,5]
print("Iterativa")
stampaIterativa(numeri)
print()
stampaIterativa(numeri, True)
print()
stampaIterativa(numeri, reverse = True)
print()
print()
print("Ricorsiva 1")
stampaRicorsiva(numeri)
print()
stampaRicorsiva(numeri, True)
print()
print()

print("Ricorsiva 2")
stampaRicorsiva2(numeri)
print()
stampaRicorsiva2(numeri, reverse = True)
print()
stampaRicorsiva2(numeri, inizio = 2, reverse = True)
print()
```

4.2.3 Scrivi ed utilizza una funzione che calcola il fattoriale di un numero. Scrivi due versioni della funzione, una iterativa e una ricorsiva.

4.2.4 Scrivi e utilizza le versioni iterativa e ricorsiva di una funzione che dato un array di interi, calcola e restituisce la massima differenza tra due numeri consecutivi.

Soluzione

```
# 6.1.4 Scrivi e utilizza le versioni iterativa e ricorsiva di
# una funzione che dato un array di interi, calcola e restituisce
# la massima differenza tra due numeri consecutivi.

from random import randint
import os;
os.system('cls')

def maxdiffIterativa(lista):
    if len(lista) < 2:
        return 0

    ris = abs(lista[1] - lista[0])
    for i in range(1, len(lista)-1):
        diff = abs(lista[i+1] - lista[i])
        if diff > ris:
            ris = diff
    return ris

def maxdiffRicorsiva(lista):
    if len(lista) < 2:
        return 0
    diff = abs(lista[1]-lista[0])
    return max(diff, maxdiffRicorsiva(lista[1:]))

lista = [randint(0,9) for _ in range(10)]

print(lista)
print(maxdiffIterativa(lista))
print(maxdiffRicorsiva(lista))
```

4.2.5 Scrivi ed utilizza una funzione che calcola l'ennesimo numero della successione di Fibonacci. Scrivi due versioni della funzione, una iterativa e una ricorsiva.

4.2.6 Scrivi una funzione ricorsiva che effettui la moltiplicazione tra due numeri

Soluzione

```
# 6.1.6 - Scrivi una funzione ricorsiva che effettui la
# moltiplicazione tra due numeri

def moltiIterativa(a, b):
    # controllo dominio
    if b < 0:
        b = -b
        a = -a
    ris = 0
    for i in range(b):
        ris += a
    return ris

def moltiRicorsiva(a, b):
    # controllo dominio
    if b < 0:
        b = -b
        a = -a

    # caso finale
    if a == 0 or b == 0:
        return 0

    # ricorsione
    return a + moltiRicorsiva(a, b-1)

print(moltiIterativa(2, 5))
print(moltiRicorsiva(2, -5))
```

4.2.7 Scrivi una funzione che calcola ricorsivamente il quoziente tra due numeri utilizzando le sottrazioni.

Soluzione

```
# 6.1.7 - Scrivi una funzione che calcola ricorsivamente il
# quoziente tra due numeri utilizzando le sottrazioni.

def divIterativa(a, b):
    ris = 0
    while a > b:
        ris += 1
        a -= b
    return (ris, a)

def divRicorsiva(a, b):
    a = abs(a)
    b = abs(b)

    if a < b:
        return (0, a)

    q, r = divRicorsiva(a-b, b)
```

```
    return (1 + q, r)

a = 10
b = 3
q, r = divIterativa(a, b)
print(f"{a}/{b} = {q}  resto {r}")

q, r = divRicorsiva(a, b)
print(f"{a}/{b} = {q}  resto {r}")
```

4.2.8 Scrivi e utilizza le versioni iterativa e ricorsiva di una funzione che riceve un array e inverte l'ordine dei numeri in esso contenuti.

Soluzione

```
# 6.1.8 - Scrivi e utilizza le versioni iterativa e ricorsiva di
# una funzione che riceve un array e inverte l'ordine dei numeri
# in esso contenuti.

import os;
os.system('cls')

def invertiListaI(lista, inizio = 0, fine = -1):
    if fine < 0:
        fine = len(lista)+fine

    for i in range(inizio, fine//2 + 1):
        lista[i], lista[fine-i] = lista[fine-i], lista[i]

def invertiListaR(lista, inizio = 0, fine = -1):
    if fine < 0:
        fine = len(lista)+fine

    if inizio >= fine or len(lista) < 1:
        return

    lista[inizio], lista[fine] = lista[fine], lista[inizio]
    invertiListaR(lista, inizio+1, fine-1)

lista = [1,2,3,4,5,6]
print(lista)
invertiListaI(lista)
print(lista)
invertiListaR(lista, 1, -2)
print(lista)

print()

lista = [1,2,3,4,5,6]
print(lista)
invertiListaR(lista)
print(lista)
invertiListaR(lista, 1, -2)
print(lista)
```

4.2.9 Funzione ricorsiva per trovare i valori massimi e minimi di un vettore.

Soluzione

```
# 6.1.9 - Funzione ricorsiva per trovare i valori massimi e
# minimi di un vettore.
```

```python
def massimo(lista):
    if len(lista) < 1:
        return None
    if len(lista) == 1:
        return lista[0]
    else:
        resto = massimo(lista[1:])
        if lista[0] > resto:
            return lista[0]
        else:
            return resto

print(massimo([1,6,3,8,3]))
```

4.2.10 Funzione ricorsiva per cercare un valore all'interno di un vettore non ordinato (ricerca lineare). Risultato: None se non lo trova, altrimenti l'indice della posizione dove è stato trovato.

4.2.11 Scrivere una funzione ricorsiva per cercare un valore all'interno di un vettore ordinato (ricerca dicotomica). Risultato: None o l'indice.
Indicazioni: se il vettore è ordinato, iniziate a controllare il numero a metà del vettore, se il numero controllato è più piccolo di quello da trovare, cercherò nella parte del vettore a sinistra (richiamo la funzione indicando come intervallo in cui cercare solo la parte a sinistra), se è più piccolo dovrò cercare a destra, se è uguale l'ho trovato.

Soluzione

```python
# 6.1.11    Scrivere una funzione ricorsiva per cercare un valore all'interno di un
# vettore ordinato (ricerca dicotomica). Risultato: None o l'indice.
# Indicazioni: se il vettore è ordinato, iniziate a controllare il numero a metà del
# vettore, se il numero controllato è più piccolo di quello da trovare, cercherò
# nella parte del vettore a sinistra (richiamo la funzione indicando come intervallo
# in cui cercare solo la parte a sinistra), se è più piccolo dovrò cercare a destra,
# se è uguale l'ho trovato.

from random import randint

def contieneRic(lista, el):
    if len(lista) < 1:
        return False

    imid = len(lista) // 2
    if lista[imid] == el:
        return True

    if el < lista[imid]:
        return contieneRic(lista[:imid], el)
    else:
        return contieneRic(lista[imid+1:], el)


def cercaPosRic(lista, el, i = 0, f = -1):
    if f < 0:
        f = len(lista)+f

    if i >= f:
        if lista[i] == el:
```

```
            return i
        else:
            return -1

    imid = (i+f) // 2
    if lista[imid] == el:
        return imid

    if el < lista[imid]:
        return cercaPosRic(lista, el, i, imid-1)
    else:
        return cercaPosRic(lista, el, imid+1, f)


lista = [randint(0,99) for _ in range(20)]
lista.sort()
print(lista)
print(contieneRic(lista, 10))
print(cercaPosRic(lista, 10))
```

4.2.12 Scrivere una funzione ricorsiva che riceve un array di interi e restituisce true se gli elementi sono tutti dispari e false se non lo sono.

Soluzione

```
# Scrivere una funzione ricorsiva che riceve un array di
# interi e restituisce true se gli elementi sono tutti dispari e false se non lo
# sono.

def tuttiDispari(lista):
    if len(lista) < 1:
        return False
    if len(lista) == 1:
        return lista[0] % 2 != 0
    else:
         return lista[0] % 2 != 0 and tuttiDispari(lista[1:])

def tuttiDispariIterativa(lista):
    for num in lista:
        if num % 2 == 0:
            return False
    return True

print(tuttiDispari([1,2,7]))
```

4.2.13 Scrivere una funzione ricorsiva che riceve un array di interi e restituisce il prodotto degli elementi negativi.

Soluzione

```
# 6.1.13 - Scrivere una funzione ricorsiva che riceve un array di
# interi e restituisce il prodotto degli elementi negativi.

def prodottoNegativi(lista):
    if len(lista) < 1:
        return None
    elif len(lista) == 1:
        if lista[0] < 0:
```

```
            return lista[0]
        else:
            return 1 # se non metto questo uno ma metto 0 moltiplica per 0 e va sempre a 0, se
metto None cerca di moltiplicare sempre per None e da errore
    else:
        if lista[0] < 0:
            return lista[0] * prodottoNegativi(lista[1:])
        else:
            return prodottoNegativi(lista[1:])

print(prodottoNegativi([1,-4,3,-2,7]))
```

4.2.14 Scrivi le versioni iterativa e ricorsiva del selection sort.

Soluzione

```
import os;
os.system('cls')

def posmin(lista, inizio, fine):
    imin = inizio
    for i in range(inizio, fine+1):
        if lista[i] < lista[imin]:
            imin = i
    return imin

def selectionSortR(lista, inizio = 0, fine = -1):
    if fine < 0:
        fine = len(lista) + fine

    if inizio >= fine:
        return

    # imin = inizio
    # for i in range(inizio+1, fine+1):
    #     if lista[i] < lista[imin]:
    #         imin = i

    imin = posmin(lista, inizio, fine)
    lista[imin], lista[inizio] = lista[inizio], lista[imin]
    selectionSortR(lista, inizio+1, fine)


lista = [3,7,4,9,1]
selectionSortR(lista)
print(lista)
```

4.2.15 Scrivi le versioni iterativa e ricorsiva del bubble sort.

Soluzione

```
# 6.1.15 - Scrivi le versioni iterativa e ricorsiva del bubble
# sort.

def bubbleSortIterativo(lista):
    for i in range(len(lista)-1):
        for j in range(len(lista)-1):
            if lista[j] > lista[j+1]:
                lista[j], lista[j+1] = lista[j+1], lista[j]
    return

def bubbleSortRicorsivo(lista, inizio = 0, fine = -1):
```

```
    if fine < 0:
        fine = len(lista)+fine

    if fine <= inizio:
        return

    for j in range(inizio, fine):
         if lista[j] > lista[j+1]:
             lista[j], lista[j+1] = lista[j+1], lista[j]

    return bubbleSortRicorsivo(lista, inizio, fine-1)

def bubbleSortRicorsivo2(lista):
    if len(lista) < 1:
        return []
    for j in range(len(lista)-1):
        if lista[j] > lista[j+1]:
            lista[j], lista[j+1] = lista[j+1], lista[j]

    return bubbleSortRicorsivo2(lista[:-1]) + [lista[-1]]


lista = [2,6,4,8,3]
bubbleSortRicorsivo(lista)
print(lista)
# print(bubbleSortRicorsivo2(lista))
```

4.2.16 Scrivi una funzione iterativa e una ricorsiva per la stampa di una matrice.

Soluzione

```
from random import randint
import os;
os.system('cls')


def stampaMatriceIterativo(mat):
    for riga in mat:
        for num in riga:
            print(num, end = " ")
        print()

def stampaRigaRicorsiva(riga, inizio = 0, fine = -1):
    if fine < 0:
        fine = len(riga) + fine

    if inizio > fine or len(riga) < 1:
        return

    print(riga[inizio], end = " ")
    stampaRigaRicorsiva(riga, inizio+1, fine)

def stampaMatriceRicorsiva(mat, inizio = 0, fine = -1):
    if fine < 0:
        fine = len(mat) + fine

    if inizio > fine or len(mat) < 1:
        return
```

```
    stampaRigaRicorsiva(mat[inizio])
    print()
    stampaMatriceRicorsiva(mat, inizio+1, fine)



n = randint(3,5)
m = randint(4,6)
mat = [[randint(0,9) for j in range(m)] for i in range(n)]

stampaMatriceIterativo(mat)
print()
stampaMatriceRicorsiva(mat)
```

4.2.17 Scrivi e usa una funzione ricorsiva per il calcolo dell'ennesimo numero della successione di fibonacci che faccia uso della memoria per evitare che il tempo di calcolo sia esponenziale

4.2.18 Si definisca la funzione ricorsiva (o che usa una vostra funzione ricorsiva) che presa in input una stringa di caratteri restituisce la lista delle diverse "sottoparole crescenti" di tale stringa. Le sottoparole devono comparire nella lista in ordine lessicografico (alfabetico).

Si ricorda che una sottoparola e' quello che si ottiene da una parola concellandone 0 o piu' caratteri (in testa, in coda o nel mezzo). Inoltre una sottoparola si dice crescente se i caratteri che la compongono letti da sinistra a destra risultano ordinati lessicograficamente.

Ad esempio con l'input "zanzara" si ottiene:

['a', 'aa', 'aaa', 'aar', 'an', 'anr', 'anz', 'ar', 'az', 'n', 'nr', 'nz', 'r', 'z', 'zz']

Soluzione

Questa soluzione prevede l'uso degli insiemi (set) poiché è comodo usare il loro metodo union

```
def f(parola):
    # inserisci qui il tuo codice
    return sorted(list(ricorsiva(parola)))

def ricorsiva(parola):
    if len(parola) == 0:
        return set()

    if len(parola) == 1:
        return {parola}

    tmp = "".join(sorted(parola))
    if tmp == parola:
        ris = {parola}
    else:
        ris = set()
```

```
    for i in range(len(parola)):
        sottoparola = parola[:i]+parola[i+1:]
        ris = ris.union(ricorsiva(sottoparola))

    return ris


print(f("zanzara"))
```

4.3 Algoritmi di ordinamento avanzati

4.3.1 prova

4.3.2

5. Files

5.1.1 Scrivi un programma che scriva su un file un elenco di nomi, chiuda il file, lo riapra in lettura lo legga tutto e stampi ciò che ha letto.

5.1.2 Scrivi e usa una funzione che conta il numero di caratteri contenuti in un file

5.1.3 Scrivi una funzione che conta e restituisce il numero di righe di un file.
Scrivi poi un'altra funzione che calcola per ogni riga il numero di caratteri contenuti nella riga. I valori vanno inseriti in una lista o una tupla da restituire al main.

1. Variante: invece dei caratteri conta il numero di parole per riga

Soluzione

```
# 7.1.3 Scrivi una funzione che conta e restituisce il numero
# di righe di un file. Scrivi poi un'altra funzione che calcola
# per ogni riga il numero di caratteri contenuti nella riga.
# I valori vanno inseriti in una lista o una tupla da restituire
# al main.
# Variante: invece dei caratteri conta il numero di
# parole per riga

def contaRighe(nomefile):
    with open(nomefile, 'r', encoding='utf-8') as f:
        ris = 0
        for riga in f:
            # if len(riga) > 1 or (len(riga) == 1 and riga[-1] != '\n'):
            if riga != '\n':
                ris += 1
    return ris

def contaRighe2(nomefile):
    with open(nomefile, 'r', encoding='utf-8') as f:
```

```
        testo = f.read()
        testo = testo.split('\n')
        testo = [el for el in testo if el != '']
        return len(testo)

def caratteriPerRiga(nomefile):
    with open(nomefile, 'r', encoding='utf-8') as f:
        return [len(riga) for riga in f if riga != '\n']

def parolePerRiga(nomefile):
    with open(nomefile, 'r', encoding='utf-8') as f:
        return [len(riga.split()) for riga in f if riga != '\n']

def parolePerRigaFacile(nomefile):
    with open(nomefile, 'r', encoding='utf-8') as f:
        ris = []
        for riga in f:
            if riga != '\n':
                riga = riga.split()
                ris.append(len(riga))
        return ris



print("numero di righe:", contaRighe("7-1-3.txt"))
print("numero di righe:", contaRighe2("7-1-3.txt"))
print("numero di caratteri per riga:", caratteriPerRiga("7-1-3.txt"))
print("numero di caratteri per riga:", parolePerRiga("7-1-3.txt"))
print("numero di caratteri per riga:", parolePerRigaFacile("7-1-3.txt"))
```

5.1.4 Scrivi e usa una funzione che legga un file e scriva in un altro file tutte le parole del primo file in ordine alfabetico

Soluzione

```
def f(nomefile, nomefileout):
    with open(nomefile, 'r', encoding='utf-8') as f:
        testo = f.read()
        parole = testo.split()

        # per mettere tutto minuscolo
        for i in range(len(parole)):
            parole[i] = parole[i].lower()

        parole.sort()
        with open(nomefileout, 'w', encoding='utf-8') as f:
            for parola in parole:
                f.write(parola)
                f.write("\n")

f("8-1-4.txt", "8-1-4-out.txt")
```

5.1.5 Scrivi un programma che legga un file di testo e che scriva in un secondo file, lo stesso testo modificato in modo che tutte le lettere siano minuscole. Il file di testo iniziale generalo tu come vuoi.

5.1.6 Scrivi un programma che legga un file di testo e che aggiunga alla fine di tale file, dopo un paio di righe vuote, lo stesso testo modificato in modo che tutte le lettere siano

minuscole. Il file di testo iniziale generalo tu come vuoi. Attento a fare in modo che se il programma viene eseguito più volte, le righe da considerare sono solo quelle iniziali.

5.1.7 Scrivi una funzione parametrica in grado di modificare un file di testo di nome "miofile.txt" letto da disco in modo tale che, se l'ultimo elemento della linea è una virgola, la linea successiva venga eliminata

Soluzione

```
# 7.1.7 - Scrivi una funzione parametrica in grado di modificare
# un file di testo di nome "miofile.txt" letto da disco in modo tale
# che, se l'ultimo elemento della linea è una virgola, la linea successiva venga
# eliminata

def modificaFile(nomefile):
    with open(nomefile, 'r', encoding='utf-8') as fin:
        righe = fin.readlines()

    righe = [riga.strip() for riga in righe]
    print(righe)
    print()


    nuove = [righe[0]]
    for i in range(1, len(righe)):
        # print(righe[i])
        if righe[i-1][-1:] != ',':    # [-1:] perchè la riga prima può essere vuota
            nuove.append(righe[i])
        # print(nuove)

    for riga in nuove:
        print(riga)

    with open(nomefile[:-4]+'-modificato.txt', 'w', encoding='utf-8') as fout:
        fout.write('\n'.join(nuove))


modificaFile("7-1-7.txt")
```

5.1.8 Scrivi una funzione parametrica in grado di modificare un file di testo di nome "miofile.txt" letto da disco in modo tale che, se l'ultimo elemento della linea è una virgola, la linea successiva venga eliminata

5.1.9 Scrivi un programma che scriva su un file un elenco di nomi (o numeri se preferisci), chieda poi all'utente un nome da cercare nel file e stampi sullo schermo l'esito della ricerca (trovato o no). Le operazioni di scrittura su file devono essere eseguite da una funzione che riceve come parametro la stringa contenente il nome del file. Le operazioni di ricerca devono essere svolte da una funzione che riceve come parametro la stringa contenente il nome del file e la stringa da cercare.

1. Continua l'esercizio aggiungendo una funzione che dato il nome del file e il numero di riga, cerca, se esiste, la riga e la restituisce.
2. Aggiungi anche la possibilità di chiedere all'utente se vuole aggiungere dei nomi a quelli già presenti (aprendo poi il file in modalità append e aggiungendo i nomi dati dall'utente)

5.1.10 Il seguente testo (che devi inserire in un file di testo) descrive i più celebri virologi Italiani nell'epoca covid

(Nome e Cognome - Ospedale - città - (O) Ottimista (P) Pessimista

Roberto Burioni - San Raffaele - Milano (P)
Matteo Bassetti - San Martino - Genova (O)
Alberto Zangrillo - San Raffaele - Milano (O)
Massimo Galli - Luigi Sacco - Milano (P)
MariaRita Gismondo - Luigi Sacco - Milano (O)
Andrea Crisanti - Università di Padova - Padova (P)
Ilaria Capua - One Health - Florida (P)
Antonella Viola - Città della Speranza - Padova (P)
Fabrizio Pregliasco - Galeazzi - Milano (O)
Walter Ricciardi - Policlino Gemelli - Roma (P)
Franco Locatelli - Bambin Gesù - Roma (P)

1. stampare nome e cognome dei virologi ordinati per cognome (si consiglia si usare la split() con il separatore "-";
2. stampare il numero di virologi ottimisti e di quelli pessimisti.

Soluzione

```
# 8.1.11 -  Il seguente testo (che devi inserire in un file di
# testo) descrive i più celebri virologi Italiani nell'epoca covid

# (Nome e Cognome - Ospedale - città - (O) Ottimista (P) Pessimista

with open("8-1-11.txt", 'r', encoding='utf-8') as f:
    lista = []
    for riga in f:
        riga = riga.split('-')
        riga = [dato.strip() for dato in riga]
        nome, cognome = riga[0].split()
        ospedale = riga[1].strip()
        # citta = riga[2].split()[0]
        citta = riga[2][:-4]
        orientamento = riga[2][-2]
        lista.append({'nome': nome, 'cognome': cognome, 'ospedale': ospedale,
                      'citta': citta, 'orientamento': orientamento})


# 1. stampare nome e cognome dei virologi ordinati per
# cognome (si consiglia si usare la split() con il separatore "-";
lista.sort(key = lambda x: x['cognome'])
for riga in lista:
    print(riga)

# 2. stampare il numero di virologi ottimisti e di quelli
# pessimisti.
nottimisti = 0
npessimisti = 0
for riga in lista:
    if riga['orientamento'] == 'O':
        nottimisti += 1
    if riga['orientamento'] == 'P':
```

```
        npessimisti += 1
print(f"Ottimisti: {nottimisti}")
print(f"Pessimisti: {npessimisti}")
```

5.1.11 Un file di testo contiene i seguenti dati:

```
Marco Polo 1254 1324
Cristoforo Colombo 1451 1506
Amerigo Vespucci 1454 1512
Francisco Pizarro 1475 1541
Ferdinando Magellano 1480 1521
Hernan Cortez 1485 1547
Walter Raleigh 1552 1618
Henry Hudson 1570 1611
James Cook 1728 1779
Charles Darwin 1809 1882
Kit Carson 1809 1866
David Livingstone 1813 1873
Charles Foucauld 1858 1916
Ronald Amundsen 1872 1928
Ernest Shakleton 1874 1922
```

Scrivi un programma che legga i dati contenuti nel file, li memorizzi opportunamente, e poi scrivi e utilizza le seguenti funzioni:

1. una funzione che stampa tutti i dati.
2. una funzione che ordina i dati in ordine alfabetico secondo il nome degli esploratori
3. una funzione che ordina i dati in ordine crescente secondo le date di nascita degli esploratori
4. una funzione che calcola la durata della vita di ogni esploratore e aggiunge questo dato agli altri
5. una funzione che ordina gli esploratori in base alla durata della loro vita in ordine decrescente e in caso di parità in ordine crescente secondo la data di nascita
6. una funzione che riscrive tutti i dati in un secondo file

Soluzione

```
import os
os.system("cls")

# 1.    una funzione che stampa tutti i dati.
def f1(dati):
    for esp in dati:
        print(esp)

# 2.    una funzione che ordina i dati in ordine alfabetico secondo il nome degli esploratori
def f2(dati):
    dati.sort()

# 3.    una funzione che ordina i dati in ordine crescente secondo le date di nascita degli
esploratori
def f3(dati):
    dati.sort(key = lambda x: (x[2]))
```

```python
# 4.    una funzione che calcola la durata della vita di ogni esploratore e agiunge questo dato agli altri
def f4(dati):
    for esp in dati:
        esp.append(esp[3]-esp[2])

# 5.    una funzione che ordina gli esploratori in base alla durata della loro vita in ordine decrescente e in caso di
#       parità in ordine crescente secondo la data di nascita
def f5(dati):
    dati.sort(key = lambda x: (-x[4], x[2]))

# 6.    una funzione che riscrive tutti i dati in un secondo file
def f6(dati, nomefile):
    with open(nomefile,'w',encoding = 'utf-8') as f:
        for esp in dati:
            for dato in esp:
                f.write(f"{dato} ")
            f.write("\n")

dati = []
with open("8-2-1.txt",'r',encoding = 'utf-8') as f:
    for riga in f:
        riga = riga.strip()
        riga = riga.split()
        riga[2] = int(riga[2])
        riga[3] = int(riga[3])
        dati.append(riga)

print("Dati letti dal file:")
f1(dati)

f2(dati)
print("Esploratori in ordine alfabetico:")
f1(dati)

f4(dati)
f5(dati)

print("\nClassifica per durata di vita:")
for el in dati:
    print(el)

# scrivo nel secondo file
f6(dati, "8-2-1-out.txt")
```

5.1.12 Scrivi un programma che legga i dati contenuti in un file di testo (testo riportato di seguito) e li inserisca in opportune strutture. Di questi dati il programma deve fare le seguenti cose:

1. stamparli sullo schermo
2. ordinarli in ordine decrescente di voto
3. stamparli nuovamente in ordine
4. scriverli in ordine in un secondo file che puoi nominare come vuoi

Dati da mettere nel file di testo:

```
Amici  7.00
Biella 9.00
Brescia    6.00
Carolla    8.00
```

```
Cunegatti  7.00
DeBella    4.00
DeVecchi   9.00
Fumagalli  7.00
Galimberti 9.00
Germanò    9.00
Gubellini  9.00
Lepore 5.00
Maconi 6.00
Mariani    8.00
Mattavelli 9.00
Oggiano    4.00
Passoni    5.00
Pastori    4.00
Pirovano   7.00
Rudi   10.00
Russell    6.00
Sogos  4.00
Tezza  6.00
Varisco    8.00
```

Soluzione

```python
with open("8-2-2.txt", 'r', encoding='utf-8') as f:
    lista = []
    for riga in f:
        riga = riga.strip()
        riga = riga.split("\t")
        riga[1] = float(riga[1])
        lista.append(riga)

    # 1.    stamparli sullo schermo
    print("Stampa")
    for riga in lista:
        print(riga[0], riga[1])
    print()


    # 2.    ordinarli in ordine decrescente di voto
    lista.sort(key = lambda x: (-x[1], x[0]))
    print("Stampa ordinata")
    for riga in lista:
        print(riga[0], riga[1])
    print()
```

5.1.13 Scrivi un programma che legga i dati contenuti in un file di testo (testo riportato di seguito) e li inserisca in opportune strutture dati che rappresentino il registro dei voti per una materia. Il programma deve poi calcolare la media dei voti per ogni singolo studente e aggiungerla alla struttura dati. Il programma deve mostrare tutti i dati raccolti e calcolati sullo schermo e salvarli in un secondo file di testo.

```
Amici 7.00 8.00 7.00
Biella 8.00 9.50 7.00
Brescia 5.00 7.50 9.00
Carolla 7.00 8.50 8.50
Cunegatti 7.00 7.00 5.50
DeBella 4.00 4.00 4.50
DeVecchi 8.00 10.00 6.00
Fumagalli 8.50 6.00 4.50
Galimberti 9.00 9.00 9.00
Germanò 8.50 9.00 7.50
Gubellini 8.00 10.00 7.00
```

```
Lepore 5.50 4.00 3.00
Maconi 7.50 5.00 7.50
Mariani 8.00 8.00 7.00
Mattavelli 9.00 9.50 8.50
Oggiano 5.00 3.00 3.00
Passoni 5.00 6.00 6.00
Pastori 3.50 5.00 7.00
Pirovano 6.50 7.50 7.50
Rudi 9.50 10.00 10.00
Russell 7.50 4.50 5.50
Sogos 4.00 3.50 3.50
Tezza 7.00 5.50 5.50
Varisco 8.00 9.00 8.50
```

Soluzione

```python
lista = []
with open("8-2-3.txt", 'r', encoding='utf-8') as f:
    for riga in f:
        riga = riga.strip()
        riga = riga.split()
        studente = riga[0]
        voti = list(map(float, riga[1:]))
        media = sum(voti)/len(voti)
        lista.append([studente, voti, media])

# Stampa
print("Stampa")
for riga in lista:
    print(riga)
print()

# scrittura su file
with open("8-2-3-out.txt", 'w', encoding='utf-8') as f:
    for riga in lista:
        f.write(f"{riga[0]}: ")
        for voto in riga[1]:
            f.write(f"{voto} ")
        f.write(f"- media: {riga[2]:.2f}\n")
```

5.1.14 Scrivi un programma che legga i dati contenuti in un file di testo (testo riportato di seguito) e li inserisca in opportune strutture dati che rappresentino il registro dei voti per una materia. Il programma deve poi calcolare la media dei voti per ogni singolo studente e aggiungerla alla struttura dati (che quindi conterrà anche una variabile media). Il programma deve mostrare tutti i dati raccolti e calcolati sullo schermo e salvarli in un secondo file di testo. Questa versione del programma è più difficile della precedente perché si può notare che i cognomi possono essere formati da più parole e sono separati dai voti da un ":", inoltre il numero di voti varia per ogni studente.

```
Amici: 7.00 8.00 7.00
Biella: 8.00 9.50 7.00
Brescia: 5.00 7.50 9.00
Carolla: 7.00 8.50 8.50
Cunegatti: 7.00 7.00
De Bella: 4.00 4.00 4.50 5.00
De Vecchi: 8.00 10.00 6.00
Fumagalli: 8.50 6.00 4.50
Galimberti: 9.00 9.00 9.00
Germanò: 8.50 9.00 7.50
Gubellini: 8.00 10.00 7.00
```

```
Lepore: 5.50 4.00 3.00 5.00
Maconi: 7.50 5.00
Mariani: 8.00 8.00 7.00
Mattavelli: 9.00 9.50 8.50
Oggiano: 5.00 3.00 3.00 2.00
Passoni: 5.00 6.00 6.00 7.50
Pastori: 3.50 5.00 7.00 2.00
Pirovano: 6.50 7.50 7.50
Rudi: 9.50 10.00 10.00
Russell: 7.50 4.50
Sogos: 4.00 3.50 3.50 2.00
Tezza: 7.00 5.50 5.50
Varisco: 8.00 9.00 8.50
Pirovano: 6.50 7.50 7.50
Rudi: 9.50 10.00 10.00
Russell: 7.50 4.50 5.50
Sogos: 4.00 3.50 3.50
Tezza: 7.00 5.50 5.50
Varisco: 8.00 9.00 8.50
```

Soluzione con sole liste

```python
# 7.1.14    Scrivi un programma che legga i dati contenuti in un file di testo
# (testo riportato di seguito) e li inserisca in opportune strutture dati che
# rappresentino il registro dei voti per una materia. Il programma deve poi
# calcolare la media dei voti per ogni singolo studente e aggiungerla alla
# struttura dati (che quindi conterrà anche una variabile media). Il programma
# deve mostrare tutti i dati raccolti e calcolati sullo schermo e salvarli in
# un secondo file di testo. Questa versione del programma è più difficile della
# precedente perché si può notare che i cognomi possono essere formati da più
# parole e sono separati dai voti da un ":", inoltre il numero di voti varia
# per ogni studente.

with open('7-1-14.txt', 'r', encoding='utf-8') as f:
    dati = []
    for riga in f:
        riga = riga.strip().split(':')
        nome = riga[0]
        voti = riga[1].strip().split()
        voti = [float(voto) for voto in voti]
        # voti = list(map(float, voti))
        media = sum(voti)/len(voti)
        dati.append([nome, voti, media])

for dato in dati:
    print(dato)

with open('7-1-14-ris.txt', 'w', encoding='utf-8') as f:
    for dato in dati:
        f.write(f'{dato[0]:12s}')
        voti = list(map(str, dato[1]))
        strvoti = ' '.join(voti)
        f.write(f'{strvoti:20s}')
        f.write(f' Media: {dato[2]:.2f}\n')
```

Soluzione con le classi

```python
from statistics import mean

class Studente:
    def __init__(self, cognome = "", voti = []):
        self.cognome = cognome
```

```
        self.voti = voti
        self.media = mean(voti) # se no te la calcoli tu con il solito ciclo o con sum(voti) /
len(voti) per cui non serve importare niente

    def __repr__(self) -> str:
        ris = f"{self.cognome}: "
        for voto in self.voti:
            ris += f"{voto} "
        ris += f"- Media: {self.media:.2f}"
        return ris


with open("6-13-input.txt", encoding = 'utf-8') as fi:
    studenti = []
    for riga in fi:
        dati = list(map(str.strip, riga.split(":")))
        cognome = dati[0]
        voti = list(map(str.strip, dati[1].split()))
        voti = list(map(float, voti))
        studenti.append(Studente(cognome, voti))
        # print(studenti[-1])

    for studente in studenti:
        print(studente)

    with open("6-13-output.txt", "w", encoding = 'utf-8') as fo:
        # due alternative

        # 1
        # for studente in studenti:
            # fo.write(str(studente))
            # fo.write("\n")

        # 2 (qui map non lo userei perchè va aggiunto il \n)
        fo.writelines(str(s) + '\n' for s in studenti)
```

5.1.15 Leggi le informazioni riguardanti una serie di libri da un file (il testo è riportato di seguito) e:

1. carica i dati in una lista di libri (usa i dizionari per costruire una sorta di struttura con le chiavi: titolo, autore, anno)
2. ordina i libri per data di scrittura e riscrivili in un altro file nello stesso formato del file originale
3. scrivi e usa una funzione che riceve i dati già caricati e che restituisce il libro più recente
4. scrivi e usa una funzione che restituisca il secolo in cui sono stati scritti più libri e il numero di libri scritti in quel secolo

```
Divina Commedia;Dante Alighieri;1321
Promessi Sposi;Alessandro Manzoni;1840
Il Decamerone;Alessandro Boccaccio;1350
L'Orlando Furioso;Ludovico Ariosto;1516
Il fu Mattia Pascal;Luigi Pirandello;1904
Ultime lettere di Jacopo Ortis;Ugo Foscolo;1802
Se questo è un uomo;Primo Levi;1947
```

```
Il barone rampante;Italo Calvino;1957
I Malavoglia;Giovanni Verga;1881
Il Principe; Niccolò Macchiavelli;1532
La Gerusalemme Liberata;Torquato Tasso;1581
Cuore;Edmondo De Amicis;1886
Il deserto dei Tartari;Dino Buzzati;1940
La coscienza di Zeno;Italo Svevo;1923
Pinocchio;Carlo Collodi;1883
Il piacere;Gabriele D'Annunzio;1889
Myricae;Giovanni Pascoli;1891
Canti;Giacomo Leopardi;1831
Uno, nessuno e centomila;Luigi Pirandello;1926
Il nome della rosa;Umberto Eco;1980
La casa in collina;Cesare Pavese;1948
Il gattopardo;Giuseppe Tomasi di Lampedusa;1958
I Vicerè;Federico De Roberto;1894
Mastro don Gesualdo;Giovanni Verga;1889
```

Soluzione

```python
class libro:
    def __init__(self, titolo = "", autore = "", anno = 0):
        self.titolo = titolo
        self.autore = autore
        self.anno = anno

    def __repr__(self) -> str:
        return f"{self.titolo};{self.autore};{self.anno}\n"


def piu_recente(libri):
    imin = 0
    min = libri[0].anno
    for i, libro in enumerate(libri):
        if libro.anno < min:
            imin = i
            min = libro.anno
    return libri[i]


def secolo_con_piu_libri(libri):
    secoli = {}
    for libro in libri:
        secolo = (libro.anno // 100) * 100
        if secolo in secoli:
            secoli[secolo] += 1
        else:
            secoli[secolo] = 1

    # print (secoli)

    smax = None
    for secolo, n in secoli.items():
        if smax == None or n > nmax:
            smax = secolo
            nmax = n
    return (smax, nmax)



with open("6-14-input.txt", encoding = 'utf-8') as fi:
```

```
    libri = []
    for riga in fi:
        # dati = [dato.strip() for dato in riga.split(";")]
        dati = list(map(str.strip, riga.split(";")))
        # print(dati)
        libri.append(libro(dati[0], dati[1], int(dati[2])))
        # print(libri[-1])

    libri.sort(key=lambda el: el.anno)
    print(libri)

    with open("6-14-output.txt", "w", encoding = 'utf-8') as fo:
        # ci sono 3 alternative

        # 1
        # for libro in libri:
        #     fo.write(str(libro))
        #     fo.write("\n")

        # 2
        # fo.writelines(str(libro) for libro in libri)

        # 3
        fo.writelines(map(str, libri))

    print(f"\n\nIl libro più recente è {str(piu_recente(libri))}")

    x = secolo_con_piu_libri(libri)
    print(f"\n\nIl secolo con più libri: {x[0]} con {x[1]} libri")
    # secolo_con_piu_libri(libri)
```

5.1.16 Scrivi un programma che sia in grado di leggere da file i dati riportati di seguito e salvarli in opportune strutture dati (punto e triangolo). Il programma deve poi stampare sia su schermo che in un altro file i dati riguardanti ogni triangolo e se esso è scaleno, isoscele o equilatero. Per decidere il tipo di triangolo devono essere usate tre diverse funzioni che controllano ognuna se il triangolo passato come parametro è di uno dei tre tipi richiesti.

```
Input
(1, 0) (3, 3) (-0.6, 3.23)
(1, 0) (1, 4) (-2.46, 2)
(2, 0) (5, 3) (2, 6)
(3, 0) (4.4, 3.3) (2.35, 4.8)
(2, 2) (4, 2) (4, 6)

Output
(1.000000, 0.000000) (3.000000, 3.000000) (-0.600000, 3.230000) - equilatero
(1.000000, 0.000000) (1.000000, 4.000000) (-2.460000, 2.000000) - equilatero
(2.000000, 0.000000) (5.000000, 3.000000) (2.000000, 6.000000) - isoscele
(3.000000, 0.000000) (4.400000, 3.300000) (2.350000, 4.800000) - scaleno
(2.000000, 2.000000) (4.000000, 2.000000) (4.000000, 6.000000) - scaleno
```

Soluzione

```
from math import sqrt
import re
import os

os.system("cls")

triangoli = []
```

```python
with open("8-2-6.txt", 'r', encoding='utf-8') as f:
    for riga in f:
        riga = riga.strip()


        # modalità 1
        # con lo split classico
        riga = riga.split(") (")
        triangolo = []
        for p in riga:
            if p[0] == '(':
                p = p[1:]
            if p[-1] == ')':
                p = p[:-1]
            p = p.split(", ")
            p = tuple(map(float, p))
            triangolo.append(tuple(p))
        triangolo = tuple(triangolo)
        # fine modalità 1


        # # modalità 2
        # # alternativa con lo split che usa le espressioni regolari
        # riga = re.split(r"\(|, |\) \(|\)", riga)
        # riga = list(filter(None, riga))
        # riga = list(map(float, riga))
        # # print(riga)
        # triangolo = []
        # for i in range(0, len(riga), 2):
        #     triangolo.append((riga[i], riga[i+1]))
        # triangolo = tuple(triangolo)
        # # fine alternativa

        triangoli.append(triangolo)


def distanza(p1, p2):
    return sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)


def circa(x, y):
    if x-y < 0.01 and x-y > -0.01:
        return True
    else:
        return False


def equilatero(triangolo):
    A = triangolo[0]
    B = triangolo[1]
    C = triangolo[2]
    a = distanza(C, B)
    b = distanza(A, C)
    c = distanza(A, B)
    if circa(a, b) and circa(b, c) and circa(c, a):
        return True
    else:
        return False


def isoscele(triangolo):
    A = triangolo[0]
    B = triangolo[1]
    C = triangolo[2]
    a = distanza(C,B)
    b = distanza(A,C)
    c = distanza(A,B)
```

```
    if circa(a, b) or circa(b, c) or circa(c, a):
        return True
    else:
        return False


def scaleno(triangolo):
    A = triangolo[0]
    B = triangolo[1]
    C = triangolo[2]
    a = distanza(C,B)
    b = distanza(A,C)
    c = distanza(A,B)
    if not circa(a, b) and not circa(b, c) and not circa(c, a):
        return True
    else:
        return False

with open("8-2-6-out.txt", 'w', encoding='utf-8') as f:
    for triangolo in triangoli:
        print(triangolo, end = " - ")
        f.write(f"{triangolo} - ")
        if equilatero(triangolo):
            print("equilatero")
            f.write("equilatero\n")
        elif isoscele(triangolo):
            print("isoscele")
            f.write("isoscele\n")
        elif scaleno(triangolo):
            print("scaleno")
            f.write("scaleno\n")
```

5.1.17 Leggi da un file (il testo è riportato di seguito) i dati che descrivono i risultati delle partite di un girone di Champions:

Sapendo che la vittoria vale 3 punti, il pareggio 1 punto e la sconfitta 0 punti, calcola e stampa la classifica del girone comprendente il nome della squadra e i punti fatti.

Contenuto del file:

```
Sheriff Tiraspol - Shaktar Donetsk 2 0
Inter - Real Madrid 0 1
Shaktar Donetsk - Inter 0 0
Real Madrid - Sheriff Tiraspol 1 2
Shaktar Donetsk - Real Madrid 0 5
Inter - Sheriff Tiraspol 3 1
Real Madrid - Shaktar Donetsk 2 1
Sheriff Tiraspol - Inter 1 3
Inter - Shaktar Donetsk 2 0
Sheriff Tiraspol - Real Madrid 0 3
Shaktar Donetsk - Sheriff Tiraspol 1 1
Real Madrid - Inter 2 0
```

Variante: invece che stampare solo i punti fatti, stampa anche il numero di vittorie, il numero di pareggi, il numero di sconfitte, il goal fatti, i goal subiti e la differenza reti.

Soluzione

```
d = {}
with open("8-2-7-in.txt", 'r', encoding = 'utf-8') as f:
    for riga in f:
```

```
        dati = riga.split()
        squadre = dati[:-2]
        squadre = " ".join(squadre)
        squadre = squadre.split("-")
        squadre = [squadra.strip() for squadra in squadre]
        goal = list(map(int, dati[-2:]))

        if goal[0] > goal[1]:
            if squadre[0] in d:
                d[squadre[0]] += 3
            else:
                d[squadre[0]] = 3

        elif goal[1] > goal[0]:
            if squadre[1] in d:
                d[squadre[1]] += 3
            else:
                d[squadre[1]] = 3

        else:
            if squadre[0] in d:
                d[squadre[0]] += 1
            else:
                d[squadre[0]] = 1

            if squadre[1] in d:
                d[squadre[1]] += 1
            else:
                d[squadre[1]] = 1

for key, val in d.items():
    print(f"{key}: {val}")

for tupla in d.items():
    print(f"{tupla}")

lista = list(d.items())
lista.sort(key = lambda x: (-x[1], x[0]))

print("\nClassifica:")
for s, g in lista:
    print(f"{s}: {g}")

# d = dict(lista)
# print(d)
```

Soluzione variante:

```
# 5.1.17 - Leggi da un file (il testo è riportato di seguito) i dati
# che descrivono i risultati delle partite di un girone di Champions:
# Sapendo che la vittoria vale 3 punti, il pareggio 1 punto e la
# sconfitta 0 punti, calcola e stampa la classifica del girone
# comprendente: il nome della squadra, i punti fatti, il numero
# di vittorie, il numero di pareggi, il numero di sconfitte, il goal
# fatti, i goal subiti e la differenza reti.

with open('5-1-17.txt', 'r', encoding='utf-8') as f:
    d = {}
    for riga in f:
        tmp = riga.strip().split()
        g1 = int(tmp[-2])
        g2 = int(tmp[-1])
        squadre = ' '.join(tmp[:-2])
        squadre = squadre.split('-')
```

```
        sq1 = squadre[0].strip()
        sq2 = squadre[1].strip()

        if g1 > g2:
            v1 = 1
            p1 = 0
            s1 = 0
            punti1 = 3
            v2 = 0
            p2 = 0
            s2 = 1
            punti2 = 0
        elif g2 > g1:
            v1 = 0
            p1 = 0
            s1 = 1
            punti1 = 0
            v2 = 1
            p2 = 0
            s2 = 0
            punti2 = 3
        else:
            v1 = 0
            p1 = 1
            s1 = 0
            punti1 = 1
            v2 = 0
            p2 = 1
            s2 = 0
            punti2 = 1

       # il nome della squadra, i punti fatti, il numero
        # di vittorie, il numero di pareggi, il numero di sconfitte, il goal
        # fatti, i goal subiti e la differenza reti.
        if sq1 not in d:
            d[sq1] = [punti1, v1, p1, s1, g1, g2, g1-g2]
        else:
            d[sq1] = [d[sq1][0]+punti1, d[sq1][1]+v1, d[sq1][2]+p1, d[sq1]
                      [3]+s1, d[sq1][4]+g1, d[sq1][5]+g2, d[sq1][6]+g1-g2]
        if sq2 not in d:
            d[sq2] = [punti2, v2, p2, s2, g2, g1, g2-g1]
        else:
            d[sq2] = [d[sq2][0]+punti2, d[sq2][1]+v2, d[sq2][2]+p2, d[sq2]
                      [3]+s2, d[sq2][4]+g2, d[sq2][5]+g1, d[sq2][6]+g2-g1]


classifica = list(d.items())
classifica.sort(key = lambda x: -x[1][0])

for el in classifica:
    print(el)
```

5.1.18 Dato il seguente file di dati in input:

A1: Milano - Napoli: Autostrade per l'Italia: 759

A2: Salerno - Reggio Calabria: Anas: 442

A3: Napoli - Salerno: Autostrade per l'Italia: 52

A4: Torino - Trieste: Autostrade per l'Italia: 524

A6: Torino - Savona: Gruppo Gavio: 124

A7: Milano - Genova: Enti Pubblici: 133

A8: Milano - Varese: Autostrade per l'Italia: 43
A10: Savona - Ventimiglia: Gruppo Gavio: 113
A11: Firenze - Pisa: Autostrade per l'Italia: 81
A12: Genova - Cecina: Gruppo Gavio: 210
A13: Bologna - Padova: Autostrade per l'Italia: 116
A14: Bologna - Taranto: Autostrade per l'Italia: 743
A15: Parma - La Spezia: Gruppo Gavio: 108
A16: Napoli - Canosa: Autostrade per l'Italia: 172
A19: Palermo - Catania: Anas: 191
A21: Torino - Piacenza - Brescia: Gruppo Gavio: 238
A22: Brennero - Modena: Enti Pubblici: 315
A23: Palmanova - Tarvisio: Enti Pubblici: 119
A24: Roma - Teramo: Gruppo Toto: 159
A25: Torano - Pescara: Gruppo Toto: 115
A26: Genova - Gravellona: Autostrade per l'Italia: 197

1. Stampare il gestore dell'autostrada A8
2. stampare il gestore dell'autostrada "Savona - Ventimiglia"
3. stampare l'autostrada più lunga, con il suo percorso e il suo gestore
4. Stampare quanti differenti gestori sono presenti nell'elenco
5. Per ognuno dei gestori presenti stampare il numero di autostrade gestite, e il numero di km gestiti

Soluzione con sole liste

```
import os
os.system("cls")

with open("8-2-8.txt", 'r', encoding = 'utf-8') as f:
    dati = []
    for riga in f:
        riga = riga.split(":")
        riga = [dato.strip() for dato in riga]
        riga[3] = int(riga[3])
        print(riga)
        dati.append(riga)

# 1. Stampare il gestore dell'autostrada A8
print("\nGestore autostrada A8: ", end="")
for dato in dati:
    if dato[0] == 'A8':
        print(dato[2])

# 2. stampare il gestore dell'autostrada "Savona - Ventimiglia"
print("\nGestore autostrada Savona - Ventimiglia: ", end="")
for dato in dati:
    if dato[1] == 'Savona - Ventimiglia':
        print(dato[2])

# 3. stampare l'autostrada più lunga, con il suo percorso e
# il suo gestore
print("\nAutostrada più lunga: ", end="")
dati.sort(key = lambda x: -x[-1])
print(dati[0])
```

```
# 4. Stampare quanti differenti gestori sono presenti nell'elenco
# 5. Per ognuno dei gestori presenti stampare il numero di
# autostrade gestite, e il numero di km gestiti

d = {}
for dato in dati:
    gestore = dato[2]
    if gestore in d:
        d[gestore][0] += 1
        d[gestore][1] += dato[3]
    else:
        d[gestore] = [1, dato[3]]

print("\nDati sui gruppi:")
for key, val in d.items():
    print(f"{key}: {val}")
# equivale a:
# for key in d:
#     print(f"{key}: {d[key]}")

# aggiunta: ordino i gestori per km gestiti
lista = list(d.items())
lista.sort(key = lambda x: -x[1][1])
print("\nOrdinato per km:")
for el in lista:
    print(el)
```

Soluzione con (quasi) solo dizionari

```
import os
os.system("cls")


with open("8-2-8.txt", 'r', encoding = 'utf-8') as f:
    dati = {}
    for riga in f:
        riga = riga.split(":")
        riga = [dato.strip() for dato in riga]
        riga[3] = int(riga[3])
        dati[riga[0]] = {'tratta': riga[1], 'gestore': riga[2], 'km': riga[3]}

for key, val in dati.items():
    print(f"{key}: {val}")

# 1. Stampare il gestore dell'autostrada A8
print("\nGestore autostrada A8: ", end="")
print(dati['A8'])

# 2. stampare il gestore dell'autostrada "Savona - Ventimiglia"
print("\nGestore autostrada Savona - Ventimiglia: ", end="")
for key, val in dati.items():
    if val['tratta'] == 'Savona - Ventimiglia':
        print(val['gestore'])

# 3. stampare l'autostrada più lunga, con il suo percorso e
# il suo gestore
print("\nAutostrada più lunga: ", end="")
max = None
for key, val in dati.items():
    if max == None or val['km'] > dati[max]['km']:
        max = key
print(f"{max}: {dati[max]}")

# 4. Stampare quanti differenti gestori sono presenti nell'elenco
# 5. Per ognuno dei gestori presenti stampare il numero di
# autostrade gestite, e il numero di km gestiti
```

```
d = {}
for key, val in dati.items():
    gestore = val['gestore']
    if gestore in d:
        d[gestore][0] += 1
        d[gestore][1] += val['km']
    else:
        d[gestore] = [1, val['km']]

print("\nDati sui gruppi:")
for key, val in d.items():
    print(f"{key}: {val}")
# equivale a:
# for key in d:
#     print(f"{key}: {d[key]}")
```

Soluzione mista

```
import os
os.system("cls")

with open("8-2-8.txt", 'r', encoding = 'utf-8') as f:
    dati = []
    for riga in f:
        riga = riga.split(":")
        riga = [dato.strip() for dato in riga]
        riga[3] = int(riga[3])
        dati.append({'nome': riga[0], 'tratta': riga[1], 'gestore': riga[2], 'km': riga[3]})

for dato in dati:
    print(dato)

# 1. Stampare il gestore dell'autostrada A8
print("\nGestore autostrada A8: ", end="")
for dato in dati:
    if dato['nome'] == 'A8':
        print(dato['gestore'])

# 2. stampare il gestore dell'autostrada "Savona - Ventimiglia"
print("\nGestore autostrada Savona - Ventimiglia: ", end="")
for dato in dati:
    if dato['tratta'] == 'Savona - Ventimiglia':
        print(dato['gestore'])

# 3. stampare l'autostrada più lunga, con il suo percorso e
# il suo gestore
print("\nAutostrada più lunga: ", end="")
max = None
for dato in dati:
    if max == None or dato['km'] > max['km']:
        max = dato
print(max)

# 4. Stampare quanti differenti gestori sono presenti nell'elenco
# 5. Per ognuno dei gestori presenti stampare il numero di
# autostrade gestite, e il numero di km gestiti

d = {}
for dato in dati:
    gestore = dato['gestore']
    if gestore in d:
        d[gestore][0] += 1
        d[gestore][1] += dato['km']
    else:
        d[gestore] = [1, dato['km']]
```

```
print("\nDati sui gruppi:")
for key, val in d.items():
    print(f"{key}: {val}")
```

5.1.19 Dato il file contenente le seguenti informazioni:

Atalanta: Bergamo, Italia
Hellas: Verona, Italia
Sampdoria: Genova, Italia
Genoa: Genova, Italia
Liverpool: Liverpool, Regno Unito
Everton: Liverpool, Regno Unito
Espanol: Barcellona, Spagna
Barcellona: Barcellona, Spagna
Siviglia: Siviglia, Spagna
Betis: Siviglia, Spagna
Juventus: Torino, Italia
Young Boys: Berna, Svizzera
Chievo: Verona, Italia

1. Leggi i dati e salvali in una struttura formata esclusivamente da dizionari (per i passaggi successivi puoi usare quello che vuoi)
2. Trovare la città dell'Hellas, e stampare il nome dell'altra squadra della stessa città
3. stampare in ordine alfabetico tutte le squadre italiane
4. Stampare le sole squadre italiane, ordinate in base alla città, e a parità di città in ordine alfabetico

Soluzione

```
# 7.1.19 - Dato il file contenente le seguenti informazioni:
# Atalanta: Bergamo, Italia
# Hellas: Verona, Italia
# Sampdoria: Genova, Italia
# Genoa: Genova, Italia
# Liverpool: Liverpool, Regno Unito
# Everton: Liverpool, Regno Unito
# Espanol: Barcellona, Spagna
# Barcellona: Barcellona, Spagna
# Siviglia: Siviglia, Spagna
# Betis: Siviglia, Spagna
# Juventus: Torino, Italia
# Young Boys: Berna, Svizzera
# Chievo: Verona, Italia

# 1. Leggi i dati e salvali in una struttura formata
# esclusivamente da dizionari (per i passaggi successivi puoi usare quello che
# vuoi)
# 2. Trovare la città dell'Hellas, e stampare il nome
# dell'altra squadra della stessa città
# 3. stampare in ordine alfabetico tutte le squadre italiane
# 4.  Stampare le sole squadre italiane, ordinate in base
# alla città, e a parità di città in ordine alfabetico
```

```python
import os
os.system('cls')

# 1. Leggi i dati e salvali in una struttura formata
# esclusivamente da dizionari (per i passaggi successivi puoi usare quello che
# vuoi)
with open('7-2-9.txt', 'r', encoding='utf-8') as f:
    dati = {}
    for riga in f:
        dato = riga.split(':')
        nome = dato[0]
        dato = ''.join(dato[1])
        dato = dato.strip()
        dato = dato.split(', ')
        citta = dato[0]
        nazione = dato[1]
        dati[nome] = {'citta': citta, 'nazione': nazione}

# for key in dati:
#     print(f'{key}: {dati[key]}')
# print()

# 2. Trovare la città dell'Hellas, e stampare il nome
# dell'altra squadra della stessa città
print("Squadra della stessa città dell'Hellas: ", end='')
citta = dati['Hellas']['citta']
for key, value in dati.items():
    if value['citta'] == citta and key != 'Hellas':
        print(key)
print()


# 3. stampare in ordine alfabetico tutte le squadre italiane
print('Squadre in ordine alfabetico:')
lista = list(dati.items())
lista.sort()
for dato in lista:
    print(f'- {dato[0]}')
print()

# 4.  Stampare le sole squadre italiane, ordinate in base
# alla città, e a parità di città in ordine alfabetico
italiane = [el for el in lista if el[1]['nazione'] == 'Italia']
italiane.sort(key = lambda x: (x[1]['citta'], x[0]))
print('Squadre italiane in ordine alfabetico per città e poi per nome della squadra')
for squadra in italiane:
    # print(squadra)
    print(f'- {squadra[0]}: {squadra[1]["citta"]}')
```

5.1.20 Devi assegnare ad ogni persona della classe un identificativo univoco formato dalle prime tre consonanti del nome seguite dalle prime tre consonanti del cognome seguito da una serie di 3 caratteri alfanumerici casuali. Se i nomi o i cognomi non contengono 3 consonanti aggiungi vocali a caso per arrivare alle 3 lettere richieste sia per il nome che per il cognome. Nel seguente file di testo è riportato l'elenco dei nomi. In un secondo file riscrivi i nomi seguiti dal codice identificativo.

```
Banani Fahd
Brambilla Carlo
Brambilla Federico
Carrera Aurora
Crespi Ryan
Germanò Riccardo
```

```
Grassi Arianna
Hoxha Visar
Kuka Blerina
La Rosa Abraham
Livieri Lorenzo
Meterangelo Chiara
Miele Serena
Puertas Moreno Alessandra
Pulvirenti Giorgia
Roncoroni Luca Davide
Rossetti Gabriele
Sambo Luigi
Sfondrini Diego
Tarboush Andrea
Tieni Ilaria
Tornaghi Francesca
```

Ad esempio Tieni Ilaria è associata a lrotnaf5u, lr sono scelti dal nome, o è a caso, tn presi dal cognome, a a caso, f5u a caso.

Soluzione

```python
# 8.2.10 - Devi assegnare ad ogni persona della classe un
# identificativo univoco formato dalle prime tre consonanti del nome seguite
# dalle prime tre consonanti del cognome seguito da una serie di 3 caratteri
# alfanumerici casuali. Se i nomi o i cognomi non contengono 3 consonanti
# aggiungi vocali a caso per arrivare alle 3 lettere richieste sia per il nome
# che per il cognome. Nel seguente file di testo è riportato l'elenco dei nomi.
# In un secondo file riscrivi i nomi seguiti dal codice identificativo.

# Ad esempio Tieni Ilaria è associata a lrotnaf5u, lr sono
# scelti dal nome, o è a caso, tn presi dal cognome, a a caso, f5u a caso.

from random import choice
from string import ascii_lowercase, digits

def generaCodice(nome, cognome):
    vocali = 'aeiou'
    consonanti = 'bcdfghjklmnpqrstvwxyz'
    alfanum = ascii_lowercase + digits
    codice = []
    nome = nome.lower()
    cognome = cognome.lower()
    i = 0
    for lettera in nome:
        if lettera in consonanti:
            codice.append(lettera)
            i += 1
            if i == 3:
                break
    while i < 3:
        codice.append(choice(vocali))
        i += 1

    i = 0
    for lettera in cognome:
        if lettera in consonanti:
            codice.append(lettera)
            i += 1
            if i == 3:
                break
    while i < 3:
        codice.append(choice(vocali))
        i += 1
```

```
    for i in range(3):
        codice.append(choice(alfanum))

    return "".join(codice)


with open("8-2-10.txt", 'r', encoding='utf-8') as f:
    persone = []
    for riga in f:
        dati = riga.split()
        dati = [dato.strip() for dato in dati]
        dati.append(generaCodice(dati[1],dati[0]))
        persone.append(dati)

for dati in persone:
    print(dati)


with open("8-2-10-out.txt", 'w', encoding='utf-8') as f:
    for dati in persone:
        f.write(f"{dati[1]} {dati[0]} {dati[2]}\n")
```

5.1.21 Sei un giocatore di giochi di ruolo e ti serve avere delle tabelle che ti indichino le probabilità di ottenere i vari punteggi con dei tiri di diversi dadi. Non conosci molto la matematica e il calcolo combinatorio (e oltretutto calcolare tutto a mano è lungo) ma sai programmare bene in Python, pensi quindi di stimare le probabilità con un programma e di scrivere dei file ognuno che idica le diverse probabilità ottenibili con un diverso dado. I dadi che usi sono: d4 (dado a quattro facce), d6, d8, d10, d12, d20. Per ogni dado vuoi sapere le probabilità di ottenere ogni diverso risultato possibile tirando il dado una volta, o 2, o 3… fino a 10 volte.

Alla fine devi generare 6 file (uno per dado) e ogni file deve avere una prima riga che indica il dado (ad esempio “d6”) e poi diverse righe, ognuna inizia indicando il numero di dadi tirati (da 1 a 10 quindi 10 righe) seguito da un “:” e poi un dizionario che associa ad ogni valore ottenibile la probabilità associata.

Soluzione

```
# 8.2.11 - Sei un giocatore di giochi di ruolo e ti serve avere delle tabelle che ti
# indichino le probabilità di ottenere i vari punteggi con dei tiri di diversi dadi.
# Non conosci molto la matematica e il calcolo combinatorio (e oltretutto calcolare
# tutto a mano è lungo) ma sai programmare bene in Python, pensi quindi di stimare
# le probabilità con un programma e di scrivere dei file ognuno che indica le
# diverse probabilità ottenibili con un diverso dado. I dadi che usi sono: d4 (dado
# a quattro facce), d6, d8, d10, d12, d20. Per ogni dado vuoi sapere le probabilità
# di ottenere ogni diverso risultato possibile tirando il dado una volta, o 2, o
# 3… fino a 10 volte.
# Alla fine devi generare 6 file (uno per dado) e ogni file
# deve avere una prima riga che indica il dado (ad esempio “d6”) e poi diverse
# righe, ognuna inizia indicando il numero di dadi tirati (da 1 a 10 quindi 10
# righe) seguito da un “:” e poi un dizionario
# che associa ad ogni valore ottenibile la probabilità associata.

import os
from random import randint
os.system('cls')

dadi = [4,6,8,10,12,20]
```

```
ntiri = 10000
ndadimax = 10

lista = []
for dado in dadi:
    probs = [{} for _ in range(ndadimax+1)]
    for ndadi in range(1, ndadimax+1):
        for caso in range(ndadi, dado*ndadi+1):
            probs[ndadi][caso] = 0
        for lanci in range(ntiri):
            ris = 0
            for tiro in range(ndadi):
                ris += randint(1,dado)

            # print(ris)
            probs[ndadi][ris] += 1

        for key in probs[ndadi]:
            probs[ndadi][key] /= ntiri
            probs[ndadi][key] *= 100

    lista.append(probs)
    print(f"d{dado} fatto")

# stampa delle probabilità con uno specifico tipo di dado
# for i, ndadi in enumerate(lista[0]):
#     print(f"Con {i} dadi: ",end="")
#     for val in ndadi:
#         print(f"{val}: {ndadi[val]:.3f}   ", end="")
#         # prob = format(ndadi[val], "10.2E")
#         # print(f"{val}: {prob:5f}   ", end="")
#     print()

# scrivo risultati nei files
for i in range(len(lista)):
    with open(f"8-2-11-outputs/d{dadi[i]}.txt", 'w', encoding='utf-8') as f:
        for j, ndadi in enumerate(lista[i]):
            if j > 0:
                if j == 1:
                    f.write(f"Con {j:2d} dado: ")
                else:
                    f.write(f"Con {j:2d} dadi: ")
                for val in ndadi:
                    f.write(f"{val:2d}: {ndadi[val]:7.4f}   ")
                    # prob = format(ndadi[val], "10.2E")
                    # print(f"{val}: {prob:5f}   ", end="")
                f.write("\n")
```

5.1.22 Crea un programma di gestione di una agenda. L'agenda deve permettere di visualizzare e inserire i dati riguardanti i propri contatti. Per ogni persona devono essere memorizzati nome cognome e numero di telefono. La visualizzazione dei dati deve avvenire in ordine alfabetico per nome o cognome (scegli tu). Tutti i dati devono essere salvati in un file in modo da poterli caricare ogni volta che si apre il programma.

Rendi il programma più modulare possibile dividendo le funzionalità in funzioni separate. Scrivi anche una funzione che ti permetta di cercare un contatto in base a nome o cognome o entrambi.

Variante: L'agenda è ordinata per cogome e la funzione di ricerca cerca il contatto a partire dal cognome con una ricerca dicotomica. Se hai fatto le funzioni ricorsive la funzione deve essere fatta sia nella versione iterativa che ricorsiva

5.1.23 Il seguenti dati, che dovrai inserire in un file di testo, descrivono quante ore hanno lavorato in una settimana, una serie di dipendenti: ad esempio Mr White ha lavorato 8 ore il lunedì, 9 ore il martedì, 8 ore il mercoledì, 9 ore il giovedì e 4 ore il venerdì

```
Mr White: 8, 9, 8, 9, 4
Mr Brown: 0, 8, 7, 10, 8
Mr Blonde: 8, 8, 8, 9, 9
Mr Black: 6, 9, 9, 8, 8
Mr Red: 8, 7, 8, 8, 8
Mr Green: 4, 8, 8, 8, 4
```

1. Stampare la classifica dei lavoratori, in base al numero di ore lavorate
2. Stampare il giorno con più ore lavorate, e quante ore sono state lavorate in quel giorno

Soluzione

```python
# 7.1.23 - l seguenti dati, che dovrai inserire in un file di
# testo, descrivono quante ore hanno lavorato in una settimana, una serie di
# dipendenti: ad esempio Me White ha lavorato 8 ore il lunedì, 9 ore il martedì,
# 8 ore il mercoledì, 9 ore il giovedì e 4 ore il venerdì

# Mr White: 8, 9, 8, 9, 4
# Mr Brown: 0, 8, 7, 10, 8
# Mr Blonde: 8, 8, 8, 9, 9
# Mr Black: 6, 9, 9, 8, 8
# Mr Red: 8, 7, 8, 8, 8
# Mr Green: 4, 8, 8, 8, 4

# 1. Stampare la classifica dei lavoratori, in base al
# numero di ore lavorate

# 2. Stampare il giorno con più ore lavorate, e quante ore
# sono state lavorate in quel giorno


import os
os.system('cls')

# leggo i dati dal file
with open('8-2-13.txt', 'r', encoding='utf-8') as f:
    dati = []
    for riga in f:
        riga = riga.split(':')
        nome = riga[0]
        riga = riga[1]
        riga = riga.strip()
        riga = riga.split(', ')
        ore = list(map(int, riga))
        # print(nome, riga)
        somma = sum(ore)
        dati.append([nome, ore, somma])

for riga in dati:
    print(riga)

# 1. Stampare la classifica dei lavoratori, in base al
# numero di ore lavorate
dati.sort(key = lambda x: -x[-1])
print()
for riga in dati:
    print(riga)

# 2. Stampare il giorno con più ore lavorate, e quante ore
```

```
# sono state lavorate in quel giorno

somme = [0 for _ in range(len(dati[0][1]))]
for dato in dati:
    for j, num in enumerate(dato[1]):
        somme[j] += num

print(somme)
imax = 0
for i, num in enumerate(somme):
    if somme[i] > somme[imax]:
        imax = i

giorni = ['lunedì', 'martedì', 'mercoledì', 'giovedì', 'venerdì']
print('il giorno in cui si è lavorato di più è', giorni[imax])
```

5.1.24 progettare la funzione f(ftesto,k) che, presi in input il nome (o l'indirizzo) di un file di testo ed un intero k, restituisce una stringa di caratteri lunga k. Il file di testo contiene stringhe di diversa lunghezza (una per riga ed ogni riga termina con '\n'), un possibile testo da inserire nel file di input è riportato di seguito.

I k caratteri della stringa restituita dalla funzione si ottiengono considerando le stringhe lunghe k presenti nel file di testo. L'i-mo carattere della stringa sara' il carattere che compare con maggior frequenza come i-mo carattere delle stringhe lunghe k nel file di testo (in caso di parita' di occorrenze viene scelto il carattere che precede gli altri lessicograficamente).

Nel caso il file di testo non contenga parole lunghe k allora viene restituita

la stringa vuota.

Ad Esempio, per il file di testo:

```
porta
rigore
ora
giacca
follia
gara
finale
salute
fiore
lampada
cane
erba
palo
tre
due
fionda
limite
polo
soglia
amo
```

Se k=3 la funzione restituisce la stringa 'are' a seguito della presenza delle seguenti 4 stringhe lunghe 3:

```
tre
due
amo
ora
```

Soluzione

```python
def f(ftesto,k):

    with open(ftesto) as f:
        parole = f.read().split()

    parole = [parola for parola in parole if len(parola) == k]

    ris = ""
    for j in range(k):
        freq = {}
        for parola in parole:
            if parola[j] in freq:
                freq[parola[j]] += 1
            else:
                freq[parola[j]] = 1

        lista = list(freq.items())
        lista.sort(key=lambda el: (-el[1], el[0]))
        ris += lista[0][0]

    return ris



    ## alternativa

    # with open(ftesto) as f:
    #     parole = f.read().split()

    # parole = [parola for parola in parole if len(parola) == k]
    # if len(parole) == 0:
    #     return ''
    # # print(parole)
    # lettere_per_posizione = [[parola[i] for parola in parole] for i in range(k)]
    # # print(lettere_per_posizione)
    # frequenze = [{c: 0 for c in posizione} for posizione in lettere_per_posizione]
    # for i in range(k):
    #     for lettera in lettere_per_posizione[i]:
    #         frequenze[i][lettera] += 1
    # # print(frequenze)
    # frequenze = [[(c, posizione[c]) for c in posizione] for posizione in frequenze]
    # # print(frequenze)
    # for posizione in frequenze:
    #     posizione.sort(key=lambda x: (-x[1], x[0]))
    # # print(frequenze)
    # parola = [posizione[0][0] for posizione in frequenze]
    # parola = ''.join(parola)
    # # print(parola)
    # return parola
```

```
ftesto, k = 'nomefile.txt', 3
print(f(ftesto,k))
```

6. Classi

6.1 Base

6.1.1 Scrivi un programma che definisca un punto e poi dati due punti scelti da te sia in grado, per mezzo di due apposite funzioni, di calcolare la distanza fra due punti e trovare il punto medio.

Soluzione

```
class punto:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __repr__(self):
        return f'({self.x}, {self.y})'

def distanza(a, b):
    return ((a.x-b.x)**2 + (a.y-b.y)**2)**(1/2)

def puntoMedio(a, b):
    return punto((a.x+b.x)/2, (a.y+b.y)/2)

a = punto()
b = punto(2, 1)

print(f"Distanza tra A{a} e B{b} = {distanza(a,b)}")
print(f"Punto medio tra A{a} e B{b} = C{puntoMedio(a,b)}")
```

6.1.2 Definisci una classe punto. Scrivi ed utilizza poi le funzioni:

1. Funzione per stampare una rappresentazione del punto (es. A(1,2))
2. Funzione per calcolare il punto medio
3. Funzione per calcolare la retta passante tra due punti. Dovrai anche definire la classe retta che contiene le informazioni necessarie ad identificare una retta e definire una funzione per stamparla in modo standard (y=mx+c)
4. Funzione che verifica se tre punti sono allineati.

Soluzione

```
class Punto:
```

```python
    def __init__(self, x = 0, y = 0) -> None:
        self.x = x
        self.y = y

    def __repr__(self) -> str:
        return f'({self.x}, {self.y})'

    def distanza(self, b):
        return ((self.x-b.x)**2 + (self.y-b.y)**2)**(1/2)

    @staticmethod
    def distanzaStatico(a, b):
        return ((a.x-b.x)**2 + (a.y-b.y)**2)**(1/2)

def distanzaFuori(a, b):
    return ((a.x-b.x)**2 + (a.y-b.y)**2)**(1/2)

def puntoMedio(a, b):
    return Punto((a.x+b.x)/2, (a.y+b.y)/2)

class Retta:
    def __init__(self, m = 1, q = 0) -> None:
        self.m = m
        self.q = q

    def __repr__(self) -> str:
        ris = 'y = '

        if self.m == 0:
            ris += f'{self.q}'
            return ris

        if self.m == 1:
            ris += 'x'
        else:
            ris += f'{self.m}x'

        if self.q > 0:
            ris += f' + {self.q}'
        if self.q < 0:
            ris += f' - {abs(self.q)}'

        return ris

    def __eq__(self, r) -> bool:
        return self.m == r.m and self.q == r.q

def rettaPassante(A, B):
    if type(A) is not Punto or type(B) is not Punto:
        return
    m = (A.y - B.y) / (A.x - B.x)
    q = A.y - m*A.x

    return Retta(m ,q)


def puntiAllineati(A, B, C):
    if type(A) is not Punto or type(B) is not Punto or type(C) is not Punto:
        return False
    return rettaPassante(A, B) == rettaPassante(B, C)



A = Punto(1, 1)
B = Punto(2, 2)
print(f'A{A}')
```

```
print(f'B{B}')
print(A.distanza(B))
print(Punto.distanzaStatico(A,B))
print(distanzaFuori(A,B))
C = puntoMedio(A,B)
print(f'C{C}')

r1 = Retta(-3, -2)
print(r1)
print()

print(rettaPassante(A, B))
print(puntiAllineati(A, B, Punto(0,5)))
```

6.1.3 Definisci la classe vettore. Scrivi e utilizza le funzioni necessarie a:

1. stampare su console il vettore,
2. Calcolare il modulo del vettore
3. Calcolare la somma di due vettori (con l'operatore +)
4. Calcolare la differenza tra due vettore (con l'operatore -)

Soluzione

```
# 8.1.3 Definisci la classe vettore. Scrivi e utilizza le funzioni necessarie a:
# 1.     stampare su console il vettore,
# 2.    Calcolare il modulo del vettore
# 3.    Calcolare la somma di due vettori (con l'operatore +)
# 4.    Calcolare la differenza tra due vettore (con l'operatore -)

class Vettore:
    def __init__(self, x, y) -> None:
        self.x = x
        self.y = y

    def __repr__(self) -> str:
        return f'({self.x}, {self.y})'

    def modulo(self):
        return (self.x**2 + self.y**2)**(1/2)

    def __add__(self, v2):
        return Vettore(self.x+v2.x, self.y+v2.y)

    def __sub__(self, v2):
        return Vettore(self.x-v2.x, self.y-v2.y)



v1 = Vettore(1, 1)
print(v1)
print(v1.modulo())

v2 = Vettore(1,2)
print(v1+v2)
print(v1-v2)
```

6.1.4 Scrivi un programma per gestire un corso. Al corso sono iscritte delle persone. Per ogni persona dovete sapere nome e cognome e ad ognuno alla fine del corso viene assegnato

un voto da 1 a 10. Se il corso non è finito il voto ha valore None. Crea a parte un file di testo che contenga i nomi degli iscritti (all'inizio senza voti) da cui il programma legge i nomi di tutti gli iscritti. Il programma legge i nomi, li visualizza, chiede se si vuole inserire un voto e gestisce l'inserimento. Alla fine il programma salva tutti i dati nel file di prima. Dalla seconda esecuzione il programma troverà già dei voti nel file.

Soluzione

```
import os
os.system('cls')

class persona:
    def __init__(self, nome, cognome, voto = None) -> None:
        self.nome = nome
        self.cognome = cognome
        self.voto = voto

    def __repr__(self) -> str:
        ris = f'{self.cognome} {self.nome}'
        if self.voto != None:
            ris += f': {self.voto:g}'
        return ris


def caricaDati(nomefile):
    dati = []
    with open(nomefile, 'r', encoding='utf-8') as f:
        for riga in f:
            riga = riga.split(',')
            nome = riga[1].strip()
            cognome = riga[0].strip()
            if len(riga) > 2 :
                voto = float(riga[2])
            else:
                voto = None
            dati.append(persona(nome, cognome, voto))
    return dati

def aggiornaFile(nomefile, dati):
    with open(nomefile, 'w', encoding='utf-8') as f:
        for dato in dati:
            f.write(f'{dato.cognome}, {dato.nome}')
            if dato.voto != None:
                f.write(f', {dato.voto}')
            f.write('\n')

def stampaDati(dati):
    for i, dato in enumerate(dati):
        print(f'{i+1} - {dato}')

def inserisciVoto(dati, pos, voto):
    if pos < 0 or pos >= len(dati):
        return False
    else:
        dati[pos].voto = voto
        return True

nomefile = '8-1-4.txt'
dati = caricaDati(nomefile)

si = ['si', 'sì', 'Si', 'Sì', 'SI']
no = ['no', 'NO', 'No']

while True:
```

```
    print('Elenco degli studenti con i rispettivi voti:')
    stampaDati(dati)
    scelta = input('\n\nVuoi inserire un voto? ')
    while scelta not in (si+no):
        print("Non capisco la risposta.")
        scelta = input('\nVuoi inserire un voto? ')
    if scelta in no:
        break

    pos = int(input("Inserisci il numero dello studente: "))
    if pos < 1 or pos > len(dati):
        print('Lo studente non esiste.')
    else:
        voto = float(input('Scrivi il voto: '))
        if inserisciVoto(dati, pos-1, voto):
            aggiornaFile(nomefile, dati)
    print('-----------------------------------------------------')
```

6.1.5 Scrivi un programma che gestisca una lista di dati riguardanti esploratori famosi. Devi definire le strutture dati adeguate a contenere queste informazioni: una classe esploratore e una lista di esploratori. Scrivi una serie di funzioni che lavorino sulle tue strutture dati:

1. una funzione che stampa la lista di esploratori.
2. una funzione che ordina la lista in ordine alfabetico secondo il nome degli esploratori
3. una funzione che ordina la lista in ordine crescente secondo le date di nascita degli esploratori
4. una funzione che ordina la lista in ordine decrescente secondo le durate delle vite degli esploratori

I dati sugli esploratori vanno inseriti in un file e sono

```
Marco Polo 1254 1324
Cristoforo Colombo 1451 1506
Amerigo Vespucci 1454 1512
Francisco Pizarro 1475 1541
Ferdinando Magellano 1480 1521
Hernan Cortez 1485 1547
Walter Raleigh 1552 1618
Henry Hudson 1570 1611
James Cook 1728 1779
Charles Darwin 1809 1882
Kit Carson 1809 1866
David Livingstone 1813 1873
Charles Foucauld 1858 1916
Ronald Amundsen 1872 1928
Ernest Shakleton 1874 1922
```

Soluzione

```
# 6.1.5 Scrivi un programma che gestisca una lista di dati riguardanti esploratori famosi.
# Devi definire le strutture dati adeguate a contenere queste informazioni:
# una classe esploratore e una lista di esploratori.
# Scrivi una serie di funzioni che lavorino sulle tue strutture dati:
# 1.    una funzione che stampa la lista di esploratori.
# 2.    una funzione che ordina la lista in ordine alfabetico secondo il nome degli esploratori
# 3.    una funzione che ordina la lista in ordine crescente secondo le date di nascita degli
esploratori
```

```python
# 4.    una funzione che ordina la lista in ordine decrescente secondo le durate delle vite degli esploratori
# I dati sugli esploratori vanno inseriti in un file e sono

testo = '''
    Marco Polo 1254 1324
    Cristoforo Colombo 1451 1506
    Amerigo Vespucci 1454 1512
    Francisco Pizarro 1475 1541
    Ferdinando Magellano 1480 1521
    Hernan Cortez 1485 1547
    Walter Raleigh 1552 1618
    Henry Hudson 1570 1611
    James Cook 1728 1779
    Charles Darwin 1809 1882
    Kit Carson 1809 1866
    David Livingstone 1813 1873
    Charles Foucauld 1858 1916
    Ronald Amundsen 1872 1928
    Ernest Shakleton 1874 1922
'''

class Esploratore:
    def __init__(self, nome, cognome, nascita, morte) -> None:
        self.nome = nome
        self.cognome = cognome
        if type(nascita) is str:
            nascita = int(nascita)
        if type(morte) is str:
            morte = int(morte)
        self.nascita = nascita
        self.morte = morte

        if type(nascita) is int and type(morte) is int:
            self.eta = morte-nascita
        else:
            self.eta = None

    def __repr__(self) -> str:
        return f'{self.nome} {self.cognome} {self.nascita} {self.morte} - {self.eta}'

esploratori = []
righe = testo.strip().split('\n')
for riga in righe:
    riga = riga.strip()
    # print(f'"{riga}"')
    nome, cognome, nascita, morte = riga.split()
    esploratori.append(Esploratore(nome, cognome, nascita, morte))

for esploratore in esploratori:
    print(esploratore)
```

6.1.6 Scrivi e utilizza un insieme di funzioni da utilizzare per fare calcoli con le frazioni. In pratica devono essere definite le funzioni per fare almeno le 4 operazioni (magari anche altre operazioni). Deve anche essere fatta una funzione per effettuare la semplificazione delle frazioni (puoi decidere se creare e usare anche un'altra funzione mcd o utilizzare un altro metodo). Scrivi la definizione della classe frazione e tutte le eventuali altre funzioni in un file separato e poi utilizza quanto definito per fare dei calcoli con le frazioni in un programma.

Soluzione

Separo in due file:

nel file libfrazione metto la classe frazione

```python
class frazione:
    def __init__(self, num, den) -> None:
        self.num = num
        self.den = den
        self.semplifica()

    def semplifica(self):
        segno = 1
        if self.num < 0:
            segno *= -1
            self.num = -self.num
        if self.den < 0:
            segno *= -1
            self.den = -self.den

        n = 2
        while n <= self.num and n <= self.den:
            while self.num % n == 0 and self.den % n == 0:
                self.num /= n
                self.den /= n
            n += 1

        self.num *= segno

    def __repr__(self) -> str:
        return f'{self.num} / {self.den}'

    def __add__(self, b):
        num = self.num*b.den + b.num*self.den
        den = self.den*b.den
        return frazione(num, den)

    def __sub__(self, b):
        num = self.num*b.den - b.num*self.den
        den = self.den*b.den
        return frazione(num, den)

    def __mul__(self, b):
        num = self.num*b.num
        den = self.den*b.den
        return frazione(num, den)

    def __truediv__(self, b):
        num = self.num*b.den
        den = self.den*b.num
        return frazione(num, den)
```

nel file main uso la classe

```python
from libfrazione import frazione

a = frazione(-1, 2)
b = frazione(3, 4)
print(f"{a} + {b} =", a+b)
print(f"{a} - {b} =", a-b)
print(f"{a} * {b} =", a*b)
print(f"{a} / {b} =", a/b)
```

6.1.7 Scrivi un programma in cui è definita una struttura studente. Dello studente ci interessano il nome, i voti presi e la media dei voti. Il programma deve istanziare uno o più studenti e aggiungere una serie di voti casuali (anche mezzi voti ma non tutti i decimali possibili). Voglio creare uno studente e poi usare delle funzioni per aggiungere voti e calcolare la media. Infine stampare lo studente con tutti i suoi dati con una funzione apposita.

6.1.8 Definire e utilizzare tutte le funzionalità delle la classe TriangoloRettangolo di seguito descritta:

1. Il costruttore riceve le misure dei due cateti e autonomamente calcola anche la misura dell'ipotenusa
2. definire gli ulteriori metodi che servono a:
   1 calcolare l'area
   2 calcolare il perimetro
   3 stampare il triangolo (solo i valori dei lati chiamandoli cateto1 cateto2 e ipotenusa)

Variante: il costruttore deve essere in grado di costruire il triangolo dati due valori qualsiasi tra cateti e ipotenusa

Soluzione

```
class TriangoloRettangolo:
    def __init__(self, c1, c2) -> None:
        self.c1 = c1
        self.c2 = c2
        self.ipo = ((c1**2)+(c2**2))**(1/2)

    def area(self):
        return self.c1 * self.c2 / 2

    def perimetro(self):
        return self.c1 + self.c2 + self.ipo

    def __repr__(self) -> str:
        return f'cateto1: {self.c1:.2f}, cateto2: {self.c2:.2f}, ipotenusa: {self.ipo:.2f}'

t1 = TriangoloRettangolo(1,1)
t2 = TriangoloRettangolo(3,4)

print(t1, "- Area:", t1.area(), "- Perimetro:", t1.perimetro())
print(t2, "- Area:", t2.area(), "- Perimetro:", t2.perimetro())
```

Soluzione variante

```
class TriangoloRettangolo:
    def __init__(self, c1 = None, c2 = None, ipo = None) -> None:
        if c1 == None:
            if c2 != None and ipo != None:
                self.c1 = ((ipo**2)-(c2**2))**(1/2)
                self.c2 = c2
                self.ipo = ipo
            else:
                raise Exception("Non posso creare un triangolo con meno di 2 lati.")

        elif c2 == None:
            if c1 != None and ipo != None:
```

```
                self.c1 = c1
                self.c2 = ((ipo**2)-(c1**2))**(1/2)
                self.ipo = ipo
            else:
                raise Exception("Non posso creare un triangolo con meno di 2 lati.")

        elif ipo == None:
            if c1 != None and c2 != None:
                self.c1 = c1
                self.c2 = c2
                self.ipo = ((c1**2)+(c2**2))**(1/2)
            else:
                raise Exception("Non posso creare un triangolo con meno di 2 lati.")

        else:
            # if ((c1**2)+(c2**2)) == ipo**2: # potrebbe non funzionare se do valori approssimati
            if abs(((c1**2)+(c2**2)) - ipo**2) < 0.0001:
                self.c1 = c1
                self.c2 = c2
                self.ipo = ipo
            else:
                raise Exception("Questi tre lati non formano un triangolo rettangolo")


    def area(self):
        return self.c1 * self.c2 / 2

    def perimetro(self):
        return self.c1 + self.c2 + self.ipo

    def __repr__(self) -> str:
        return f'cateto1: {self.c1}, cateto2: {self.c2}, ipotenusa: {self.ipo}'

t1 = TriangoloRettangolo(1,1)
t2 = TriangoloRettangolo(3,4,5)
t3 = TriangoloRettangolo(c2=4,ipo=5)
t4 = TriangoloRettangolo(c1=3,ipo=5)
t5 = TriangoloRettangolo(1,1,1.414213562373)

print(t1, "- Area:", t1.area(), "- Perimetro:", t1.perimetro())
print(t2, "- Area:", t2.area(), "- Perimetro:", t2.perimetro())
print(t3, "- Area:", t3.area(), "- Perimetro:", t3.perimetro())
print(t4, "- Area:", t4.area(), "- Perimetro:", t4.perimetro())
print(t5, "- Area:", t5.area(), "- Perimetro:", t5.perimetro())
```

6.1.9 Definite le classi:

1. TriangoloRettangolo
2. Rettangolo
3. Quadrato
4. Rombo

Svolgi le seguenti operazioni:

5. dichiara una serie di oggetti di queste classi e inseriscile in una lista,
6. ordina la lista in base all'Area(),

7. stampa ogni oggetto della lista.

Soluzione

```
# Definite le classi:
# TriangoloRettangolo
# Rettangolo
# Quadrato
# Rombo

# nella funzione principale, dichiarare una serie di oggetti di queste classi, e
inserirle in una lista
# Ordinare la lista in base all'Area()
# Invocare il metodo Stampa() su ogni oggetto della lista

import math
import random

class TriangoloRettangolo:
    def __init__(self, params) -> None:
        self.c1 = params[0]
        self.c2 = params[1]
        self.i = math.sqrt(self.c1**2 + self.c2**2)
        self.area = self.Area()

    def Area(self):
        return self.c1*self.c2/2.0

    def Perimetro(self):
        return self.c1 + self.c2 + self.i

    def __repr__(self) -> str:
        return f"Cateto 1: {self.c1:.2f} - Cateto 2: {self.c2:.2f} - Ipotenusa:
{self.i:.2f}"


class Rettangolo:
    def __init__(self, params) -> None:
        self.base = params[0]
        self.altezza = params[1]
        self.area = self.Area()

    def Area(self):
        return self.base * self.altezza

    def __repr__(self) -> str:
        return f"Base: {self.base:.2f} - Altezza: {self.altezza:.2f} - Area:
{self.area:.2f}"
```

```python
class Quadrato:
    def __init__(self, params) -> None:
        self.lato = params[0]
        self.area = self.Area()

    def Area(self):
        return self.lato ** 2

    def __repr__(self) -> str:
        return f"Lato: {self.lato:.2f} - Area: {self.area:.2f}"


class Rombo:
    def __init__(self, params) -> None:
        self.dmaggiore = params[0]
        self.dminore = params[1]
        self.area = self.Area()

    def Area(self):
        return self.dmaggiore * self.dminore / 2.0

    def __repr__(self) -> str:
        return f"Diagonale maggiore: {self.dmaggiore:.2f} - Diagonale minore:
{self.dminore:.2f} - Area: {self.area:.2f}"


class FiguraGeometrica:
    tipologie = [(TriangoloRettangolo, 2), (Rettangolo, 2), (Quadrato, 1), (Rombo, 2)]


# elenco = [Quadrato(3), Rettangolo(2,6), Rombo(2, 3), TriangoloRettangolo(2, 4),
Quadrato(4)]
elenco = []
for i in range(5):
    figura, parametri = random.choice(FiguraGeometrica.tipologie)
    parametri = [random.random() * 10 for _ in range(parametri)]
    elenco.append(figura(parametri))

for el in elenco:
    print(el)
print("\n")

print(type(elenco[0].area))

elenco.sort(key = lambda el: el.area)
for el in elenco:
    print(el)
print("\n")

print("ciao"+ str(elenco[0]))
```

6.1.10 Definisci e utilizza una classe orario che permetta di:

1. Istanziare orari (ad esempio 8:25)
2. Stampare orari
3. Confrontare orari (<, <=, >, >=, ==)
4. Calcolare differenze di tempo tra un orario ed un altro in due modi:
   1 Numero di minuti (solo minuti ad esempio 122 minuti)
   2 Numero di ore e minuti (misto ad esempio 2 h 2 m che equivale a 122 minuti)

Soluzione

```
# 6.1.10  Definisci e utilizza una classe orario che permetta di:
# 1.  Istanziare orari (ad esempio 8:25)
# 2.  Stampare orari
# 3.  Confrontare orari (<, <=, >, >=, ==)
# 4.  Calcolare differenze di tempo tra un orario ed un altro in due modi:
#   1  Numero di minuti (solo minuti ad esempio 122 minuti)
#   2  Numero di ore e minuti (misto ad esempio 2 h 2 m che equivale a 122 minuti)

class Orario:
    def __init__(self, h = 0, m = 0) -> None:
        self.h = h % 24
        self.m = m % 60

    def __repr__(self) -> str:
        return f"{self.h:02d}:{self.m:02d}"

    def __lt__(self, b):
        return (self.h < b.h) or (self.h == b.h and self.m < b.m)

    def __le__(self, b):
        return (self.h < b.h) or (self.h == b.h and self.m <= b.m)

    def __gt__(self, b):
        return (self.h > b.h) or (self.h == b.h and self.m > b.m)

    def __ge__(self, b):
        return (self.h > b.h) or (self.h == b.h and self.m >= b.m)

    def __eq__(self, b):
        return self.h == b.h and self.m == b.m

    def __sub__(self, b):
        return self.h * 60 + self.m - b.h * 60 - b.m

    def meno(self, b):
        h = self.h-b.h
        m = self.m-b.m
        if m < 0:
            h -= 1
        return Orario(h, m)



o1 = Orario()
o2 = Orario(24,15)
```

```
print(o1)
print(o2)

print(f"{o1} <  {o2}: {o1<o2}")
print(f"{o1} <= {o2}: {o1<=o2}")
print(f"{o1} <= {o1}: {o1<=o1}")
print(f"{o1} >  {o2}: {o1>o2}")
print(f"{o1} >= {o2}: {o1>=o2}")
print(f"{o1} == {o2}: {o1==o2}")
print(f"{o1} == {o1}: {o1==o1}")
print(f"{o1} - {o2} = {o1-o2}")
print(f"{o2} - {o1} = {o2-o1}")

print(f"{o1} meno {o2} = {o1.meno(o2)}")
```

Da qui in poi gli esercizi vanno sistemati

6.1.11 Dato il seguente file:

```
Milano-Centrale Bologna 7:30 8:34
Milano-Centrale Genova-Principe 8:05 9:44
Milano-Centrale Torino 7:30 8:30
Milano-Centrale Piacenza 8:01 8:53
Milano-Centrale Chiasso 8:10 9:10
Milano-Centrale Brescia 8:15 8:51
Milano-Centrale Verona 8:45 9:58
Milano-Centrale Varese 8:05 9:00
Brescia Venezia 8:13 9:52
Voghera Torino 7:57 9:25
```

O, se non sai leggere da file, la seguente lista di stringhe

```
listaTreni = [
    "Milano-Centrale Bologna 7:30 8:34",
    "Milano-Centrale Genova-Principe 8:05 9:44",
    "Milano-Centrale Torino 7:30 8:30",
    "Milano-Centrale Piacenza 8:01 8:53",
    "Milano-Centrale Chiasso 8:10 9:10",
    "Milano-Centrale Brescia 8:15 8:51",
    "Milano-Centrale Verona 8:45 9:58",
    "Milano-Centrale Varese 8:05 9:00",
    "Brescia Venezia 8:13 9:52",
    "Voghera Torino 7:57 9:25"
]
```

Definire una classe ViaggioTreno il cui costruttore (init) riceve una delle stringhe contenuta in listaTreni. Definisci anche una classe orario per gestire meglio gli orari.

Esegui poi le seguenti operazioni:

1. Stampa tutti i treni, in base alla durata del tragitto
2. Stampa tutte le stazioni raggiungibili da Milano entro un'ora
3. Stampa il numero di stazioni presenti nella lista dei treni, ed infine stampare tutte le stazioni ordinate in base al nome

Soluzione

```
# data la seguente lista di stringhe

listaTreni = [
    "Milano-Centrale Bologna  7:30  8:34", "Milano-Centrale Genova-
Principe  8:05  9:44",
    "Milano-Centrale Torino  7:30  8:30", "Milano-Centrale Piacenza  8:01  8:53",
    "Milano-Centrale Chiasso  8:10  9:10", "Milano-Centrale Brescia  8:15  8:51",
    "Milano-Centrale Verona  8:45  9:58", "Milano-Centrale Varese  8:05  9:00",
    "Brescia Venezia 8:13  9:52", "Voghera Torino 7:57  9:25"]

# Definire una classe ViaggioTreno che riceve nella __init__  una stringa di
listaTreni

class Orario:
    def __init__(self, ore, min) -> None:
        self.ore = ore
        self.min = min
        self.totmin = ore*60+min

    def __le__(self, altro):
        return self.ore < altro.ore or (self.ore == altro.ore and self.min <=
altro.min)

    def __repr__(self) -> str:
        return f"{self.ore}:{self.min:02d}"

    def __sub__(self, altro):
        return self.totmin - altro.totmin

class ViaggioTreno:
    def __init__(self, info) -> None:
        info = info.split()
        # print(info)
        self.partenza = info[0]
        self.arrivo = info[1]
        ore, min = map(int, info[2].split(":"))
        self.orario_partenza = Orario(ore, min)
        ore, min = map(int, info[3].split(":"))
        self.orario_arrivo = Orario(ore, min)
        self.durata = self.orario_arrivo - self.orario_partenza

    def __repr__(self) -> str:
        return f"{self.partenza} {self.arrivo} {self.orario_partenza}
{self.orario_arrivo}"


viaggi = [ViaggioTreno(info) for info in listaTreni]
# print(viaggi[0])
```

```
# 1) Stampare tutti i treni, in base alla durata del tragitto
viaggi.sort(key = lambda el: el.durata)
for viaggio in viaggi:
    print(viaggio)

# 2) Stampare tutte le stazioni raggiungibili da Milano entro un'ora
print("\n\nTutte le stazioni raggiungibili da Milano entro un'ora")
ris = [(el.arrivo, el.durata) for el in viaggi if el.durata <= 60 and "Milano" in
el.partenza]
for viaggio in ris:
    print(viaggio)


# 3) Stampare il numero di stazioni presenti in listaTreni, ed infine stampare
tutte le stazioni ordinate in base al nome
print("\n\nElenco di stazioni ordinate per nome e numero di stazioni presenti in
listaTreni")
elenco_stazioni = []
for viaggio in viaggi:
    if viaggio.partenza not in elenco_stazioni:
        elenco_stazioni.append(viaggio.partenza)
    if viaggio.arrivo not in elenco_stazioni:
        elenco_stazioni.append(viaggio.arrivo)

print(sorted(elenco_stazioni))
print(len(elenco_stazioni))
```

6.1.12 La seguente lista, rappresenta i principali goleador del campionato Italiano

```
listaBomber = [
    "Dusan Vlahovic Serbia 24 gol (5) rigori  Fiorentina",
    "Gianluca Scamacca Italia 16 gol (1) rigore Sassuolo",
    "Rafael Leao Portogallo 11 gol  Milan",
    "Dries Mertens Belgio 11 gol Napoli",
    "Norberto Beto Portogallo 11 gol Udinese",
    "Andrea Pinamonti Italia 13 gol (4) rigori Empoli",
    "Ciro Immobile  Italia 27 gol (7) rigori  Lazio",
    "Mario Arnautovic Austria 14 gol (2) rigori Bologna",
    "Lautaro Martinez Argentina 21 gol (3) rigori  Inter",
    "Tammy Abraham Inghilterra 17 gol (3) rigori Roma",
    "Domenico Berardi Italia 15 gol (5) rigori Sassuolo",
    "Victor Osimhen Nigeria 14 gol Napoli",
    "Gerard Deulofeu Spagna 13 gol (1) rigore Udinese",
    "Edin Dzeko Bosnia 13 gol Inter",
    "Antonin Barak RepCeca 11 gol (4) rigori Verona",
```

```
    "Joao Pedro Brasile 13 gol (2) rigori Cagliari",
    "Mario Pasalic Croazia 13 gol Atalanta",
    "Giovanni Simeone Argentina 17 gol Verona",
    "Gianluca Caprari Italia 12 gol (2) rigori Verona",
    "Lorenzo Insigne Italia 11 gol (9) rigori Napoli"
]

team = "Milan Inter Napoli Juventus Lazio Roma Fiorentina Atalanta Verona Torino Sassuolo Udinese Bologna Empoli Sampdoria Spezia Salernitana Cagliari Genoa Venezia"
```

Esegui le seguenti operazioni:

1. Definisci una classe giocatore il cui costruttore riceve una stringa tratta da ListaBomber,
2. Crea una lista di giocatori leggendo tutte le stringhe contenute in ListaBomber
3. Stampa la classifica dei goleador in base al numero di gol segnati
4. stampa i nomi dei primi 3 rigoristi
5. stampa quali squadre hanno almeno due giocatori tra i goleador
6. stampa la classifica dei goleador italiani
7. stampa tutte le squadre che non hanno neanche un goleador nell'elenco
8. stampa la classifica dei goleador contando solo i gol fatti su azione (e non quelli su rigore)

6.1.13 Data la seguente lista:

```
sceltaUniversitaria2018 = [
    "Banfi", "Vimercate", "LSA", 48,
    ["Ingegneria", 40.7, "Scientifica", 23, "Economica", 10.4, "Sociale", 7.4, "Giuridico-Politica", 5.2, "Umanistica", 5.2, "Sanitaria", 4.4, "Medica", 3, "Scienze Motorie", 0.3],
    "Frisi", "Monza", "LSA", 58,
    ["Ingegneria", 53, "Scientifica", 17.9, "Economica", 12.5, "Sociale", 3, "Giuridico-Politica", 2.4, "Umanistica", 1, "Sanitaria", 4.8, "Medica", 5.4, "Scienze Motorie", 0],
    "Agnesi", "Merate", "LSA", 41,
    ["Ingegneria", 46.5, "Scientifica", 28.9, "Economica", 10.5, "Sociale", 2.8, "Giuridico-Politica", 2.4, "Umanistica", 3.5, "Sanitaria", 2.6, "Medica", 2.6, "Scienze Motorie", 2.6],
    "Maiorana", "Desio", "LSA", 56,
    ["Ingegneria", 41.8, "Scientifica", 35.3, "Economica", 7.2, "Sociale", 3.9, "Giuridico-Politica", 1.3, "Umanistica", 1.3, "Sanitaria", 7.2, "Medica", 1.3, "Scienze Motorie", 0.7],
    "Banfi", "Vimercate", "LS", 97,
    ["Ingegneria", 30.7, "Scientifica", 18.7, "Economica", 18.4, "Sociale", 7.4, "Giuridico-Politica", 5.3, "Umanistica", 7.1, "Sanitaria", 6.4, "Medica", 4.2, "Scienze Motorie", 0.7],
    "Frisi", "Monza", "LS", 123,
    ["Ingegneria", 39, "Scientifica", 13.1, "Economica", 20.3, "Sociale", 4.2, "Giuridico-Politica", 6.1, "Umanistica", 4.2, "Sanitaria", 1.7, "Medica", 10.9, "Scienze Motorie", 0.5],
```

"Agnesi", "Merate", "LS", 41,

["Ingegneria", 25.4, "Scientifica", 22.3, "Economica", 18, "Sociale", 3.4, "Giuridico-Politica", 5.9, "Umanistica", 6.8, "Sanitaria", 8.4, "Medica", 8.7, "Scienze Motorie", 1.1],

"Maiorana", "Desio", "LS", 118,

["Ingegneria", 32.2, "Scientifica", 21.9, "Economica", 15.4, "Sociale", 5.6, "Giuridico-Politica", 3.6, "Umanistica", 7.7, "Sanitaria", 7.1, "Medica", 5.9, "Scienze Motorie", 0.6],

"Banfi", "Vimercate", "Classico", 24,

["Ingegneria", 8.6, "Scientifica", 7.1, "Economica", 11.4, "Sociale", 12.9, "Giuridico-Politica", 25.7, "Umanistica", 30, "Sanitaria", 1.4, "Medica", 2.9, "Scienze Motorie", 0],

"Zucchi", "Monza", "Classico", 121,

["Ingegneria", 13.8, "Scientifica", 16.1, "Economica", 16.7, "Sociale", 8.5, "Giuridico-Politica", 17.9, "Umanistica", 21.7, "Sanitaria", 1.5, "Medica", 3.8, "Scienze Motorie", 0.5],

]

Definire una classe CorsoStudi e una listaCorsi[] dove travasare le informazioni presenti in sceltaUniversitaria2018

Esegui poi le seguenti operazioni:

1. Stampare il numero totale di studenti per ognuno degli istituti presenti;
2. Costruire un nuovo oggetto della classe CorsoStudi, chiamandolo corsoLsa dove riversare le medie pesate (tenendo conto delle percentuali, ma anche del numero di diplomati) dei vari corsi di scienze applicate, ripetere la stessa cosa per classico e tradizionale;
3. Stampare la classifica dei corsi, in base alla somma delle percentuali di "Ingegneria" più "Scientifica";
4. Stampare la classifica delle scuole, valutando soltanto la somma degli iscritti tra scientifico e scienze applicate;
5. Stampare per ogni indirizzo universitario "Ingegneria Scientifica Economica ...), il corso scolastico con la più alta percentuale.

6.1.14 La seguente lista, associa a ogni tennista la sua nazionalità

listaPlayer= [

"Roger Federer (Svizzera)", "Mark Philippoussis (Australia)", "Andy Roddick (Usa)", "Rafael Nadal (Spagna)",

"Tomas Berdich (RepCeca)", "Novak Djokovic (Serbia)", "Andy Murray (RegnoUnito)", "Milos Raonic (Canada)",

"Marin Cilic (Croazia)", "Kevin Anderson (Sudafrica)", "Matteo Berrettini (Italia)", "Nick Kyrgyos (Australia)"

]

La seguente Lista, descrive le finali di Wimbledon:

wimbledon = [

"2003 Roger Federer - Mark Philippoussis", "7-6 6-2 7-6",

"2004 Roger Federer - Andy Roddick", "4-6 7-5 7-6 6-4",

"2005 Roger Federer - Andy Roddick", "6-2 7-6 6-4",

```
    "2006 Roger Federer - Rafael Nadal",  "6-0 7-6 6-7 6-3",
    "2007 Roger Federer - Rafael Nadal",  "7-6 4-6 7-6 2-6 6-2",
    "2008 Rafael Nadal - Roger Federer",  "7-6 4-6 7-6 2-6 6-2",
    "2009 Roger Federer - Andy Roddick",  "6-2 7-6 6-4",
    "2010 Rafael Nadal - Tomas Berdich",  "6-3 7-5 6-4",
    "2011 Novak Djokovic  -  Rafael Nadal",  "6-4 6-1 1-6 6-3",
    "2012 Roger Federer - Andy Murray",  "4-6 7-5 6-3 6-4",
    "2013 Andy Murray - Novak Djokovic",  "6-4 7-5 6-4",
    "2014 Novak Djokovic - Roger Federer",  "6-7 6-4 7-6 5-7 6-4",
    "2015 Novak Djokovic - Roger Federer",  "7-6 6-7 6-4 6-3",
    "2016 Andy Murray - Milos Raonic",  "6-4 7-6 7-6",
    "2017 Roger Federer - Marin Cilic",  "6-3 6-1 6-4",
    "2018 Novak Djokovic - Kevin Anderson",  "6-2 6-2 7-6",
    "2019 Novak Djokovic - Roger Federer",  "7-6 1-6 7-6 4-6 13-12",
    "2021 Novak Djokovic - Matteo Berrettini",  "6-7 6-4 6-4 6-3",
    "2022 Novak Djokovic - Nick Kyrgios",  "4-6 6-3 6-4 7-6",
]
```

1. Riversare le informazioni di listaPlayer in un dizionario: giocatore -nazione
2. Quale nazione ha almeno due player che hanno disputato una finale di wimbledon
3. In quale anno non è stato disputato il torneo di Wimbledon
4. L'anno del quinto torneo vinto, dai giocatori che hanno vinto Wimbledon almeno 5 volte
5. Definire una classe Finale, che descrive il match di una finale
6. Stampare la classifica dei tre match più combattuti
7. Stampare la classifica dei giocatori che hanno vinto più volte wimbledon
8. Stampare la classifica dei giocatori che hanno giocato più finali di wimbledon)
9. Stampare il nome del giocatore che ha vinto più finali consecutive di Wimbledon e quante ne ha vinte, e da che anno a che anno

6.2 Ereditarietà

6.2.1 Definisci le seguenti classi:

1. Persona che ha gli attributi nome, cognome
2. Studente, che è una persona e ha anche la matricola e la classe di appartenenza
3. Docente che è una persona e ha anche la lista delle materie insegnate in ogni classe

Genera una lista di studenti e di docenti, poi scrivi e utilizza una funzione che data la matricola di uno studente restituisce la lista dei suoi docenti.

7. Strutture dati avanzate

7.1 Pile / Stack

7.1.1 Scrivi un programma che legga da input una frase e che utilizzando una pila stampi le lettere della frase al contrario

Soluzione 1

Soluzione usando una lista come pila

```
frase = "Ciao Ryan come stai?"

pila = []
for lettera in frase:
    pila.append(lettera)
invertita = ''
while len(pila) > 0:
    invertita += pila[-1]
    pila.pop()

print(invertita)
```

Soluzione 2

Soluzione che prevede l'utilizzo della classe pila

```
class Pila:
    def __init__(self) -> None:
        self.dati = []

    def push(self, dato):
        self.dati.append(dato)

    def pop(self):
        self.dati.pop()

    def top(self):
        if len(self.dati) > 0:
            return self.dati[-1]
        else:
            return None

    def empty(self):
        return len(self.dati) == 0
```

```
from pila import Pila

frase = "Ciao Ryan come stai?"

pila = Pila()
for lettera in frase:
    pila.push(lettera)
invertita = ''
while not pila.empty():
    invertita += pila.top()
    pila.pop()

print(invertita)
```

7.1.2 Scrivi un programma che legga in input un'espressione matematica contenente parentesi e che sia in grado di dire se le parentesi contenute nell'espressione siano state aperte e chiuse correttamente. Puoi implementare due versioni:

1. Versione semplificata che usa solo parentesi tonde
2. Versione completa che usa anche parentesi quadre e se vuoi anche graffe

Soluzione

```
from pila import Pila

def controllaParentesi(esp):
    p = Pila()
    for sim in esp:
        if sim in "[({":
            if (sim in "{[" and p.top() == "(") or (sim in "{" and p.top() == "["):
                return False
            p.push(sim)
        if sim in "}])":
            if p.empty():
                return False
            if sim == "}" and p.top() != "{":
                return False
            if sim == "]" and p.top() != "[":
                return False
            if sim == ")" and p.top() != "(":
                return False
            p.pop()
    if p.empty():
        return True
    else:
        return False


esp = "16{2x+[3y-7]}+5"
print(esp, controllaParentesi(esp))
```

7.1.3 Scrivi un programma che dato in input un numero intero positivo qualsiasi sia in grado, usando una pila, di stampare la sua rappresentazione binaria. (vai a rivedere l'algoritmo per la conversione da decimale a binario)

Soluzione

```
import os
os.system('cls')

from pila import Pila


def convertiABinario(decimale):
    cifre = Pila()
    while decimale > 0:
        cifre.push(decimale%2)
        decimale //= 2

    ris = ''
    while not cifre.empty():
        ris += str(cifre.top())
        cifre.pop()

    return ris
```

```
print(convertiABinario(2))
```

7.1.4 Implementa una coda utilizzando due stack

Soluzione

```
from pila import Pila

class CodaConPile:
    def __init__(self) -> None:
        self.pila1 = Pila() # da qui posso prendere dalla cima l'elemento front della coda
        self.pila2 = Pila() # qui metto in cima l'elemento back della coda

    def push(self, dato):
        while not self.pila1.empty():
            self.pila2.push(self.pila1.top())
            self.pila1.pop()
        self.pila2.push(dato)

    def front(self):
        while not self.pila2.empty():
            self.pila1.push(self.pila2.top())
            self.pila2.pop()
        return self.pila1.top()

    def pop(self):
        while not self.pila2.empty():
            self.pila1.push(self.pila2.top())
            self.pila2.pop()
        self.pila1.pop()

    def empty(self):
        return self.pila1.empty() and self.pila2.empty()


coda = CodaConPile()
coda.push(1)
coda.pop()
coda.push(2)
coda.push(3)
coda.push(4)
print(coda.front())
coda.pop()
print(coda.front())
coda.pop()
print(coda.front())
coda.pop()
print(coda.front())
# coda.pop() # questo mi da errore perchè è vuoto (giustamente)
```

7.1.5 Dopo aver implementato la classe ‘frazione’ con tutte le quattro operazioni, scrivi una funzione che utilizzando anche uno stack riesce a risolvere un’espressione matematica contenente frazioni, operazioni aritmetiche (quelle che hai implementato per le frazioni) e parentesi.

https://www.cs.princeton.edu/courses/archive/spr01/cs126/exercises/adt.html

https://www.geeksforgeeks.org/c-programs-gq/stack-queue-cc-programs-gq/

7.2 Code / Queue

7.2.1 Definisci opportunamente una classe coda, poi riempila in modo che quando poi vengono letti i numeri in essa contenuti venga stampato il seguente oputput:

1 2 3 4 5 6

7.2.2 Definisci opportunamente una classe coda, poi istanzia una coda e riempila inserendo i numeri in ordine da 1 a 10, infine usa opportunamente i metodi della classe per stampare il seguente oputput:

1 4 5 7 9

7.3 Grafi

7.3.1 Leggi da un file di testo i dati riguardanti un grafo e istanzia un grafo da te definito.

Contenuto del file in cui la prima riga contiene i valori di ogni nodo e le altre righe le liste di adiacenza di ogni nodo:

```
0,1,2,3,4,5
1
2 3
3 4
4
0 2
0 1
```

Soluzione

```python
# 7.3.1 Leggi da un file di testo i dati riguardanti un grafo e istanzia un grafo da te definito.
# Contenuto del file in cui la prima riga contiene i valori di ogni nodo e le altre righe le
liste di adiacenza di ogni nodo:

class Nodo:
    def __init__(self, val = None, archi = []) -> None:
        self.val = val
        self.archi = archi

    def __repr__(self) -> str:
        return f'{self.val}'

class Grafo:
    def __init__(self, nodi = []) -> None:
        self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
        for i in range(len(self.nodi)):
            ris += f'{self.nodi[i].val}: {self.nodi[i].archi}\n'
        return ris

with open("7-3-1.txt", 'r', encoding='utf-8') as f:
    vals = list(map(int, f.readline().strip().split()))
    adiacenze = [list(map(int, riga.split())) for riga in f]
```

```
print(vals)
print(adiacenze)
g = Grafo()
for i in range(len(vals)):
    g.nodi.append(Nodo(vals[i], adiacenze[i]))

print(g)
```

7.3.2 Letto in input un grafo G = (N,A) non orientato, costruire il grafo H = (N,A') complementare, tale che l'arco (u, v) ∈ A se e solo se (u, v) ∉ A'. Testa il programma sul grafo:

```
0 1 2 3
0 1, 0 2, 2 3
```

Dove la prima riga rappresenta i valori dei nodi, la seconda riga gli archi.

```
class Grafo:
    def __init__(self, vals = None, archi = None) -> None:
        if vals == None:
            self.vals = []
        else:
            self.vals = vals
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def __repr__(self) -> str:
        ris = f'Nodi: {self.vals}\n'
        ris += f'Archi: {self.archi}\n'
        return ris

    def complementare(self):
        h = Grafo(vals = self.vals)
        for i in range(len(self.vals)):
            for j in range(i+1, len(self.vals)):
                if (i, j) not in self.archi and (j, i) not in self.archi:
                    h.archi.append((i,j))
        return h



with open('9-3-2.txt', 'r', encoding='utf-8') as f:
    vals = list(map(int, f.readline().strip().split()))
    archi = f.readline().strip().split(',')
    archi = [tuple(map(int, arco.strip().split())) for arco in archi]

g = Grafo(vals, archi)
print(g)
print()
h = g.complementare()
print(h)
```

7.3.3 Letti in input due interi n e k (0 < k < n), costruire un grafo G = (N,A) in modo casuale che abbia n nodi e ogni vertice abbia al massimo k archi uscenti (non bidirezionali).

Soluzione

```
from random import randint
```

```
class Nodo:
    def __init__(self, val = None, archi = []) -> None:
        self.val = val
        self.archi = archi

    def __repr__(self) -> str:
        return f'{self.val}'

class Grafo:
    def __init__(self, nodi = []) -> None:
        self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
        for i in range(len(self.nodi)):
            ris += f'{i}. Val: {self.nodi[i].val}, Archi: {self.nodi[i].archi}\n'
        return ris



def creaGrafoCasuale(n, k):
    g = Grafo()
    for i in range(n):
        val = randint(0,9)
        archi = []
        for j in range(randint(0,k)):
            dest = randint(0, n-1)
            while dest == i or dest in archi:
                dest = randint(0, n-1)
            archi.append(dest)
        g.nodi.append(Nodo(val, archi))
    return g


n = 4
k = 3
g = creaGrafoCasuale(n, k)
print(g)
```

7.3.4 Letto in input un grafo non orientato G = (N,A) con n nodi e una serie di liste di numeri interi compresi tra 0 e n-1, verificare per ogni lista se la lista rappresenta un cammino sul grafo.

I dati in input sono:

```
0 1, 0 4, 1 2, 2 3, 2 4, 3 4
1 2 4 3 2 0
4 0 1 2 4
0 1 3 4
2 1 0 4 4 2 1
```

La prima riga rappresenta gli archi del grafo (i valori dei nodi non sono riportati perché inutili, se vuoi per semplificare l'esercizio aggiungili tu nel file di testo, se no generali tu a partire dagli archi) e le altre righe sono i cammini da verificare

Per esercitarti genera l'intero grafo e lavora su di esso, in realtà è possibile lavorare solamente sulla lista degli archi (ed è anche più veloce).

```
class Grafo:
    def __init__(self, vals=None, archi=None):
        self.vals = vals
        if vals == None:
```

```
            self.vals = []
        self.archi = archi
        if archi == None:
            self.archi = []

    def __repr__(self) -> str:
        ris = "Valori: "
        for val in self.vals:
            ris = ris + f"{val}, "
        ris += "\nArchi: "
        for arco in self.archi:
            ris = ris + f"{arco}"
        return ris

archi = []
percorsi = []
with open("7-3-4.txt", "r", encoding="utf-8") as f:
    riga = f.readline()
    riga = riga.strip().split(",")
    riga = [el.strip().split() for el in riga]
    for arco in riga:
        arco = [int(el) for el in arco]
        archi.append(arco)
    for riga in f:
        riga = riga.strip().split()
        riga = [int(el) for el in riga]
        percorsi.append(riga)

print(archi)
# print(percorsi)

for percorso in percorsi:
    valido = True
    for i in range(len(percorso)-1):
        arco = [percorso[i], percorso[i+1]]
        if arco not in archi and arco[::-1] not in archi:
            valido = False
    print(percorso, end = '')
    if valido:
        print(" valido")
    else:
        print(" non valido")
```

7.3.5 Leggere in input k liste di numeri interi e costruire un grafo non orientato G = (N,A) per cui le k liste rappresentino dei cammini validi.

Soluzione

```
from random import randint

class Grafo:
    def __init__(self, vals=None, archi=None):
        self.vals = vals
        if vals == None:
            self.vals = []
        self.archi = archi
        if archi == None:
            self.archi = []

    def __repr__(self) -> str:
        ris = "Valori: "
        for val in self.vals:
            ris = ris + f"{val}, "
```

```
        ris += "\nArchi: "
        for arco in self.archi:
            ris = ris + f"{arco}"
        return ris



# invece che leggere i cammini da file li creo random
k = 4
maxlung = 6
numnodi = 5
cammini = []
# for _ in range(k):
#     cammino = []
#     for _ in range(randint(1,maxlung)):
#         cammino.append(randint(0, numnodi-1))
#     cammini.append(cammino)
cammini = [[randint(0, numnodi-1) for _ in range(randint(1,maxlung))] for _ in range(k)]
for cammino in cammini:
    print(cammino)

archi = []
for cammino in cammini:
    for i in range(len(cammino)-1):
        arco = [cammino[i], cammino[i+1]]
        arco.sort()
        if arco not in archi:
            archi.append(arco)
archi.sort()
print(archi)
```

7.3.6 Dato un grafo orientato G e dato in input un vertice v, stampare tutti i vertici di G a cui v è adiacente (che hanno archi che puntano a v)

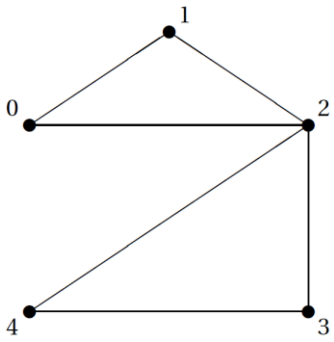
7.3.7 Dato un grafo G e in input un nodo v con i suo archi uscenti, aggiungere v a G.

7.3.8 Dato un grafo G e un vertice v di G, eliminare v da G.

7.3.9 Dato un grafo G, verifica se esso è completo cioè se tutti i nodi di G sono raggiungibili da ogni altro nodo di G con un solo passo.

7.3.10 Dato un grafo non orientato G = (V,E), letto in input un sottoinsieme L di vertici di V, verificare se L è una copertura di vertici di G, ossia se per ogni (u, v) ∈ E risulta u ∈ L o v ∈ L (o entrambi).

Esempio: consideriamo il grafo G = (V,E) rappresentato in figura, in cui l'insieme dei vertici è V = {0,1,2, 3,4} e l'insieme degli spigoli è E = {(0,1),(0, 2), (1, 2), (2, 3), (2, 4), (3,4)}:



La lista L = {1,2,3} è una copertura di G, mentre L' = {0,1,3} non è una copertura, infatti gli estremi dello spigolo (4,2) non appartengono a L'.

Soluzione

```
class Grafo:
    def __init__(self, vals=[], archi=[]) -> None:
        self.vals = vals
        self.archi = archi

    def __repr__(self) -> str:
        ris = f'Nodi: {self.vals}\n'
        ris += f'Archi: {self.archi}\n'
        return ris

    def copertura(self, nodi):
        # prima magari controllo se i nodi esistono
        for nodo in nodi:
            if nodo < 0 or nodo >= len(self.vals):
                return False

        # poi controllo se ho almeno un arco che non tocca nessuno dei nodi passati
        for arco in self.archi:
            if arco[0] not in nodi and arco[1] not in nodi:
                return False
        # se non ne ho trovati allora restituisco true
        return True


g = Grafo([0, 1, 2, 3, 4], [(0, 1), (0, 2), (1, 2), (2, 3), (2, 4), (3, 4)])
print(g)
print(g.copertura([1,2,3]))
print(g.copertura([0,1,3]))
```

7.3.11 Letto in input un grafo G = (V,E) non orientato, costruire e stampare un grafo G' = (V,E') orientato, tale che (u, v) ∈ E' se e solo se g(u) > g(v), dove con g(v) si è indicato il grado del vertice v (il numero di spigoli entranti o uscenti: in un grafo non orientato non c'è distinzione).

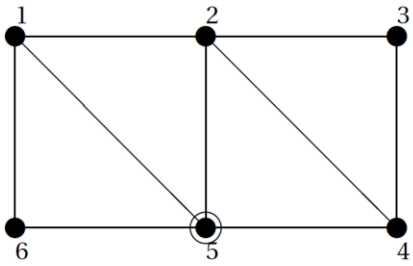
Soluzione

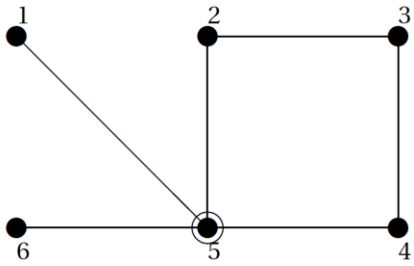
```
# 9.3.11 - Letto in input un grafo G = (V,E) non orientato,
# costruire e stampare un grafo G' = (V,E') orientato, tale che (u, v) ∈
# E' se e solo se g(u) > g(v), dove con g(v) si è indicato il grado del
# vertice v (il numero di spigoli entranti o uscenti: in un grafo non orientato
# non c'è distinzione).

class Nodo:
    def __init__(self, val, archi = None) -> None:
        self.val = val
```

```
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def grado(self):
        return len(self.archi)


class Grafo:
    def __init__(self, nodi = None) -> None:
        if nodi == None:
            self.nodi = []
        else:
            self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
        for i, nodo in enumerate(self.nodi):
            ris += f'{i}. {nodo.val} : '
            for arco in nodo.archi:
                ris += f'{arco} '
            ris += '\n'
        return ris


def creaGrafoOrientato(g):
    ris = Grafo()
    for i, nodo in enumerate(g.nodi):
        archi = []
        for dest in nodo.archi:
            if nodo.grado() > g.nodi[dest].grado():
                archi.append(dest)
        ris.nodi.append(Nodo(nodo.val, archi))
    return ris

g = Grafo()
g.nodi.append(Nodo(0, [1,2,3]))
g.nodi.append(Nodo(1, [0,3]))
g.nodi.append(Nodo(2, [0,4]))
g.nodi.append(Nodo(3, [0,1,4]))
g.nodi.append(Nodo(4, [3,2]))

g2 = creaGrafoOrientato(g)
print(g2)
```

7.3.12 Leggere in input un grafo G = (V,E) non orientato e memorizzarlo mediante liste di adiacenza. Scelto arbitrariamente uno dei vertici v ∈ V di grado massimo, eliminare dal grafo tutti gli spigoli (u,w) ∈ E per ogni u e w adiacenti a v. Stampare le liste di adiacenza del grafo così modificato.

Esempio Sia G = (V,E) il grafo letto in input rappresentato in figura (a sinistra), con V = {1, 2, 3, 4, 5, 6} ed E = {(1,2),(2, 3), (3, 4), (4, 5), (5, 6), (6,1),(1,5),(2,5),(2, 4)}. I vertici di grado massimo sono 2 e 5 (entrambi di grado 4). Scegliendo il vertice 5, devono essere eliminati gli spigoli (1,2) (perché 1,2 ∈ N(5)), (1,6) (perché 1,6 ∈ N(5)) e (2,4) (perché 4,2 ∈ N(5)). si ottiene così il grafo rappresentato a destra nella figura.





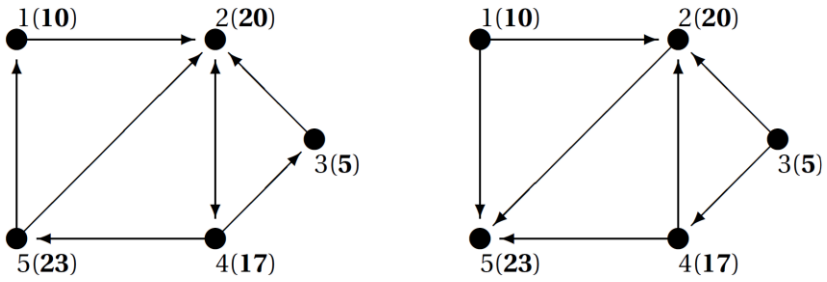
Soluzione

```python
# 9.3.12 - Leggere in input un grafo G = (V,E) non orientato e memorizzarlo
# mediante liste di adiacenza. Scelto arbitrariamente uno dei vertici v ∈
# V di grado massimo, eliminare dal grafo tutti gli spigoli (u,w) ∈
# E per ogni u e w adiacenti a v. Stampare le liste di adiacenza del grafo così
# modificato.

class Nodo:
    def __init__(self, val, archi = None) -> None:
        self.val = val
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def grado(self):
        return len(self.archi)


class Grafo:
    def __init__(self, nodi = None) -> None:
        if nodi == None:
            self.nodi = []
        else:
            self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
        for i, nodo in enumerate(self.nodi):
            ris += f'{i}. {nodo.val} : '
            for arco in nodo.archi:
                ris += f'{arco} '
            ris += '\n'
        return ris

    def scegliVGradoMassimo(self):
        gradi = [(i, nodo.grado()) for i, nodo in enumerate(self.nodi)]
        gradi.sort(key = lambda x: -x[1])
        # print(gradi)
        return gradi[0][0]

    def modificaGrafo(self, posNodo):
        for nodo in self.nodi:
            if posNodo in nodo.archi: # quindi adiacente
                eliminare = []
                for posAltro in nodo.archi:
                    if posNodo in self.nodi[posAltro].archi: # quindi anche questo è adiacente
                        eliminare.append(posAltro)
                for el in eliminare:
                    nodo.archi.remove(el)



g = Grafo()
g.nodi.append(Nodo(0, [1,2,3]))
```

```
g.nodi.append(Nodo(1, [0,3]))
g.nodi.append(Nodo(2, [0,4,3]))
g.nodi.append(Nodo(3, [0,1,2,4]))
g.nodi.append(Nodo(4, [3,2]))

g.modificaGrafo(g.scegliVGradoMassimo())
print(g.scegliVGradoMassimo())
print(g)
```

7.3.13 Leggere in input un grafo orientato G = (V,E) e rappresentarlo mediante liste di adiacenza. Leggere in input un insieme di pesi (interi) associati ai vertici del grafo: $\{w_1, ...,w_n\}$. Modificando le liste di adiacenza con cui è stato rappresentato il grafo G, variare l'orientamento degli spigoli in modo tale che per ogni spigolo (u, v) risulti $w_u \leq w_v$.

Esempio Sia G = (V,E) il grafo orientato letto in input rappresentato in figura, con V = {1,2,3, 4,5} ed E = {(1,2),(2, 4), (3, 2), (4, 2), (4, 3), (4,5),(5,1),(5,2)}. Sia W l'insieme dei pesi associati ai vertici del grafo: W = {10, 30, 5, 17, 23}. Sulla sinistra è rappresentato il grafo letto in input e sulla destra il grafo prodotto dalla rielaborazione richiesta dall'esercizio.



7.3.14 Le dieci città principali del regno di Banania sono così collegate da strade:

```
Strada - Città collegate - Lunghezza (km)
SS1 Acerorosso - Bananopoli 50
SS2 Bananopoli - Lamponeggia 80
SS3 Bananopoli - Cedrosa 30
SS4 Lamponeggia - Cedrosa 50
SS5 Acerorosso - Lamponeggia 25
SS6 Acerorosso - Dolcemela 60
SS7 Lamponeggia - Dolcemela 20
SS8 Cedrosa - Erbavoglio 35
SS9 Dolcemela - Erbavoglio 40
SS10 Acerorosso - Finoporto 45
SS11 Finoporto - Giuggiolo 60
SS12 Dolcemela - Giuggiolo 80
SS13 Dolcemela - Indacovia 90
SS14 Erbavoglio - Hoppavilla 70
SS15 Hoppavilla - Indacovia 45
SS16 Giuggiolo - Indacovia 55
```

Aggiungi i dati ad un file di testo e poi scrivi un programma che legga tali dati e istanzi un grafo che rappresenti le strade.

Il programma deve poi essere in grado di:

1. Scrivere il percorso più breve (in km) fra Bananopoli e Hoppavilla
2. Scrivere il percorso più breve (in km) fra Giuggiolo e Cedrosa
3. Scrivere la strada più diretta (meno città intermedie) fra Bananopoli e Hoppavilla

4. Scrivere la strada più diretta (meno città intermedie) fra Giuggiolo e Cedrosa

Soluzione 1

Qui ho usato solo un dizionario per rappresentare il grafo

```
def distanza(nodi, orig, dest = None):
    if orig not in nodi:
        return None
    if dest != None and dest not in nodi:
        return None

    # lista delle distanze di ogni nodo da orig
    # all'inizio sono tutte sconosciute (None)
    dist = dict([(nodo, None) for nodo in nodi])
    # tranne l'origine che ha distanza 0 da se stessa
    dist[orig] = 0

    # lista in cui segno per ogni nodo da quale altro nodo
    # ci arrivo nel percorso ottimo (all'inizio è tutto sconosciuto)
    prec = dict([(nodo, None) for nodo in nodi])

    # nodi da controllare
    controllare = [orig]
    controllati = []

    # # nodo da cui partire
    # corrente = orig

    while len(controllare) > 0:
        # scelgo il prossimo nodo da controllare (corrente)
        # trovo il nodo con distanza minima in controllare
        corrente = controllare[0]
        for nodo in controllare:
            if dist[nodo] < dist[corrente]:
                corrente = nodo

        # controllo tutti i nodi raggiungibili da corrente con i suoi archi
        # print(nodi[corrente])
        for nodo, peso in nodi[corrente]:
            # ogni nodo che raggiungo da corrente con il minor costo visto finora
            # per quel nodo va aggiornato (distanza ottima e nodo precedente nel
            # percorso ottimo) e aggiunto a controllare
            if dist[nodo] == None or dist[corrente] + peso < dist[nodo]:
                dist[nodo] = dist[corrente] + peso
                prec[nodo] = corrente
                if nodo not in controllare and nodo not in controllati:
                    controllare.append(nodo)

        # rimuovo da controllare il nodo corrente, qui non ci potrò mai più
        # tornare perchè è impossibile che possa tornarci con distanza minore
        # di ora perchè quando scelgo il nodo da controllare prendo sempre
        # quello con distanza minima
        controllare.remove(corrente)
        controllati.append(corrente)

    # caso in cui non specifico una destinazione
    # restituisco tutte le distanze e i nodi precedenti per ogni nodo
    if dest == None:
        return (dist, prec)
    # caso in cui specifico un nodo destinazione
    # restituisco la singola distanza e tutto il percorso da orig a dest
    else:

        # se il nodo dest non è raggiungibile
        if dist[dest] == None:
            percorso = []
```

```
        # altrimenti ricostruisco il percorso a ritroso da dest a orig
        else:
            percorso = [dest]
            pos = dest
            while prec[pos] != None:
                pos = prec[pos]
                percorso.append(pos)
            percorso = percorso[::-1]

        return (dist[dest], percorso)


with open('9-3-14.txt', 'r', encoding='utf-8') as f:
    nodi = {}
    for riga in f:
        riga = riga.strip().split()
        n1 = riga[1]
        n2 = riga[3]
        peso = int(riga[4])
        if n1 in nodi:
            nodi[n1].append((n2, peso))
        else:
            nodi[n1] = [(n2, peso)]
        if n2 in nodi:
            nodi[n2].append((n1, peso))
        else:
            nodi[n2] = [(n1, peso)]

for nodo in nodi:
    print(nodo, nodi[nodo])

# 1.    Scrivere il percorso più breve (in km) fra Bananopoli e Hoppavilla
print(distanza(nodi, 'Bananopoli', 'Hoppavilla'))
# 2.    Scrivere il percorso più breve (in km) fra Giuggiolo e Cedrosa
print(distanza(nodi, 'Giuggiolo', 'Cedrosa'))
# 3.    Scrivere la strada più diretta (meno città intermedie) fra Bananopoli e Hoppavilla
# 4.    Scrivere la strada più diretta (meno città intermedie) fra Giuggiolo e Cedrosa
```

Soluzione 2

Qui ho usato le classi nodo e grafo. In grafo ho usato un dizionario per contenere i nodi e accedervi tramite nome della città

```
class Nodo:
    def __init__(self, val = None, archi = None) -> None:
        self.val = val
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

class Grafo:
    def __init__(self, nodi = None) -> None:
        if nodi == None:
            self.nodi = {}
        else:
            self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
        for key, nodo in self.nodi.items():
            ris += f'{key}: '
            for arco in nodo.archi:
                ris += f'{arco} '
            ris += '\n'
        return ris
```

```python
def distanza(self, orig, dest = None):
    if orig not in self.nodi:
        return None
    if dest != None and dest not in self.nodi:
        return None

    # lista delle distanze di ogni nodo da orig
    # all'inizio sono tutte sconosciute (None)
    dist = dict([(nodo, None) for nodo in self.nodi])
    # tranne l'origine che ha distanza 0 da se stessa
    dist[orig] = 0

    # lista in cui segno per ogni nodo da quale altro nodo
    # ci arrivo nel percorso ottimo (all'inizio è tutto sconosciuto)
    prec = dict([(nodo, None) for nodo in self.nodi])

    # nodi da controllare
    controllare = [orig]
    controllati = []

    # # nodo da cui partire
    # corrente = orig

    while len(controllare) > 0:
        # scelgo il prossimo nodo da controllare (corrente)
        # trovo il nodo con distanza minima in controllare
        corrente = controllare[0]
        for nodo in controllare:
            if dist[nodo] < dist[corrente]:
                corrente = nodo

        # controllo tutti i nodi raggiungibili da corrente con i suoi archi
        # print(nodi[corrente])
        for nodo, peso in self.nodi[corrente].archi:
            # ogni nodo che raggiungo da corrente con il minor costo visto finora
            # per quel nodo va aggiornato (distanza ottima e nodo precedente nel
            # percorso ottimo) e aggiunto a controllare
            if dist[nodo] == None or dist[corrente] + peso < dist[nodo]:
                dist[nodo] = dist[corrente] + peso
                prec[nodo] = corrente
                if nodo not in controllare and nodo not in controllati:
                    controllare.append(nodo)

        # rimuovo da controllare il nodo corrente, qui non ci potrò mai più
        # tornare perchè è impossibile che possa tornarci con distanza minore
        # di ora perchè quando scelgo il nodo da controllare prendo sempre
        # quello con distanza minima
        controllare.remove(corrente)
        controllati.append(corrente)

    # caso in cui non specifico una destinazione
    # restituisco tutte le distanze e i nodi precedenti per ogni nodo
    if dest == None:
        return (dist, prec)
    # caso in cui specifico un nodo destinazione
    # restituisco la singola distanza e tutto il percorso da orig a dest
    else:

        # se il nodo dest non è raggiungibile
        if dist[dest] == None:
            percorso = []
        # altrimenti ricostruisco il percorso a ritroso da dest a orig
        else:
            percorso = [dest]
            pos = dest
```

```
                while prec[pos] != None:
                    pos = prec[pos]
                    percorso.append(pos)
                percorso = percorso[::-1]

            return (dist[dest], percorso)

    def passi(self, orig, dest = None):
        if orig not in self.nodi:
            return None
        if dest != None and dest not in self.nodi:
            return None

        # lista delle distanze di ogni nodo da orig
        # all'inizio sono tutte sconosciute (None)
        dist = dict([(nodo, None) for nodo in self.nodi])
        # tranne l'origine che ha distanza 0 da se stessa
        dist[orig] = 0

        # lista in cui segno per ogni nodo da quale altro nodo
        # ci arrivo nel percorso ottimo (all'inizio è tutto sconosciuto)
        prec = dict([(nodo, None) for nodo in self.nodi])

        # nodi da controllare
        controllare = [orig]
        controllati = []

        # # nodo da cui partire
        # corrente = orig

        while len(controllare) > 0:
            # scelgo il prossimo nodo da controllare (corrente)
            # trovo il nodo con distanza minima in controllare
            corrente = controllare[0]
            for nodo in controllare:
                if dist[nodo] < dist[corrente]:
                    corrente = nodo

            # controllo tutti i nodi raggiungibili da corrente con i suoi archi
            # print(nodi[corrente])
            for nodo, peso in self.nodi[corrente].archi:
                # ogni nodo che raggiungo da corrente con il minor costo visto finora
                # per quel nodo va aggiornato (distanza ottima e nodo precedente nel
                # percorso ottimo) e aggiunto a controllare
                if dist[nodo] == None or dist[corrente] + 1 < dist[nodo]:
                    dist[nodo] = dist[corrente] + 1
                    prec[nodo] = corrente
                    if nodo not in controllare and nodo not in controllati:
                        controllare.append(nodo)

            # rimuovo da controllare il nodo corrente, qui non ci potrò mai più
            # tornare perchè è impossibile che possa tornarci con distanza minore
            # di ora perchè quando scelgo il nodo da controllare prendo sempre
            # quello con distanza minima
            controllare.remove(corrente)
            controllati.append(corrente)

        # caso in cui non specifico una destinazione
        # restituisco tutte le distanze e i nodi precedenti per ogni nodo
        if dest == None:
            return (dist, prec)
        # caso in cui specifico un nodo destinazione
        # restituisco la singola distanza e tutto il percorso da orig a dest
        else:

            # se il nodo dest non è raggiungibile
            if dist[dest] == None:
```

```
                percorso = []
            # altrimenti ricostruisco il percorso a ritroso da dest a orig
            else:
                percorso = [dest]
                pos = dest
                while prec[pos] != None:
                    pos = prec[pos]
                    percorso.append(pos)
                percorso = percorso[::-1]

            return (dist[dest], percorso)


with open('9-3-14.txt', 'r', encoding='utf-8') as f:
    g = Grafo()
    for riga in f:
        riga = riga.strip().split()
        n1 = riga[1]
        n2 = riga[3]
        peso = int(riga[4])
        if n1 in g.nodi:
            g.nodi[n1].archi.append((n2, peso))
        else:
            g.nodi[n1] = Nodo(n1, [(n2, peso)])
        if n2 in g.nodi:
            g.nodi[n2].archi.append((n1, peso))
        else:
            g.nodi[n2] = Nodo(n2, [(n1, peso)])

print(g)

# 1.    Scrivere il percorso più breve (in km) fra Bananopoli e Hoppavilla
print(g.distanza('Bananopoli', 'Hoppavilla'))
# 2.    Scrivere il percorso più breve (in km) fra Giuggiolo e Cedrosa
print(g.distanza('Giuggiolo', 'Cedrosa'))
# 3.    Scrivere la strada più diretta (meno città intermedie) fra Bananopoli e Hoppavilla
print(g.passi('Bananopoli', 'Hoppavilla'))
# 4.    Scrivere la strada più diretta (meno città intermedie) fra Giuggiolo e Cedrosa
print(g.passi('Giuggiolo', 'Cedrosa'))
```

Soluzione 3

Qui ho adattato l'input per avere un grafo classico in cui i nodi sono indicati dalla posizione nella lista dei nodi

```
class Nodo:
    def __init__(self, val = None, archi = None) -> None:
        self.val = val
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def __repr__(self) -> str:
        return f'{self.val}'

class Grafo:
    def __init__(self, nodi = []) -> None:
        self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
        for i in range(len(self.nodi)):
            ris += f'{i}. Val: {self.nodi[i].val}, Archi: {self.nodi[i].archi}\n'
        return ris
```

```python
    def distanza(self, orig, dest = None):
        if orig < 0 or orig >= len(self.nodi):
            return None
        if dest != None and (dest < 0 or dest >= len(self.nodi)):
            return None

        # lista delle distanze di ogni nodo da orig
        # all'inizio sono tutte sconosciute (None)
        dist = [None for _ in range(len(self.nodi))]
        # tranne l'origine che ha distanza 0 da se stessa
        dist[orig] = 0

        # lista in cui segno per ogni nodo da quale altro nodo
        # ci arrivo nel percorso ottimo (all'inizio è tutto sconosciuto)
        prec = [None for _ in range(len(self.nodi))]

        # nodi da controllare
        controllare = [orig]
        controllati = []

        # # nodo da cui partire
        # corrente = orig

        while len(controllare) > 0:
            # scelgo il prossimo nodo da controllare (corrente)
            # trovo il nodo con distanza minima in controllare
            corrente = controllare[0]
            for nodo in controllare:
                if dist[nodo] < dist[corrente]:
                    corrente = nodo

            # controllo tutti i nodi raggiungibili da corrente con i suoi archi
            for pos, peso in self.nodi[corrente].archi:
                # ogni nodo che raggiungo da corrente con il minor costo visto finora
                # per quel nodo va aggiornato (distanza ottima e nodo precedente nel
                # percorso ottimo) e aggiunto a controllare
                if dist[pos] == None or dist[corrente] + peso < dist[pos]:
                    dist[pos] = dist[corrente] + peso
                    prec[pos] = corrente
                    if pos not in controllare and pos not in controllati:
                        controllare.append(pos)

            # rimuovo da controllare il nodo corrente, qui non ci potrò mai più
            # tornare perchè è impossibile che possa tornarci con distanza minore
            # di ora perchè quando scelgo il nodo da controllare prendo sempre
            # quello con distanza minima
            controllare.remove(corrente)
            controllati.append(corrente)

        # caso in cui non specifico una destinazione
        # restituisco tutte le distanze e i nodi precedenti per ogni nodo
        if dest == None:
            return (dist, prec)
        # caso in cui specifico un nodo destinazione
        # restituisco la singola distanza e tutto il percorso da orig a dest
        else:

            # se il nodo dest non è raggiungibile
            if dist[dest] == None:
                percorso = []
            # altrimenti ricostruisco il percorso a ritroso da dest a orig
            else:
                percorso = [dest]
                pos = dest
                while prec[pos] != None:
                    pos = prec[pos]
                    percorso.append(pos)
```

```
            percorso = percorso[::-1]

        return (dist[dest], percorso)


def passi(self, orig, dest = None):
    if orig < 0 or orig >= len(self.nodi):
        return None
    if dest != None and (dest < 0 or dest >= len(self.nodi)):
        return None

    # lista delle distanze di ogni nodo da orig
    # all'inizio sono tutte sconosciute (None)
    dist = [None for _ in range(len(self.nodi))]
    # tranne l'origine che ha distanza 0 da se stessa
    dist[orig] = 0

    # lista in cui segno per ogni nodo da quale altro nodo
    # ci arrivo nel percorso ottimo (all'inizio è tutto sconosciuto)
    prec = [None for _ in range(len(self.nodi))]

    # nodi da controllare
    controllare = [orig]
    controllati = []

    # # nodo da cui partire
    # corrente = orig

    while len(controllare) > 0:
        # scelgo il prossimo nodo da controllare (corrente)
        # trovo il nodo con distanza minima in controllare
        corrente = controllare[0]
        for nodo in controllare:
            if dist[nodo] < dist[corrente]:
                corrente = nodo

        # controllo tutti i nodi raggiungibili da corrente con i suoi archi
        for pos, peso in self.nodi[corrente].archi:
            # ogni nodo che raggiungo da corrente con il minor costo visto finora
            # per quel nodo va aggiornato (distanza ottima e nodo precedente nel
            # percorso ottimo) e aggiunto a controllare
            if dist[pos] == None or dist[corrente] + 1 < dist[pos]:
                dist[pos] = dist[corrente] + 1
                prec[pos] = corrente
                if pos not in controllare and pos not in controllati:
                    controllare.append(pos)

        # rimuovo da controllare il nodo corrente, qui non ci potrò mai più
        # tornare perchè è impossibile che possa tornarci con distanza minore
        # di ora perchè quando scelgo il nodo da controllare prendo sempre
        # quello con distanza minima
        controllare.remove(corrente)
        controllati.append(corrente)

    # caso in cui non specifico una destinazione
    # restituisco tutte le distanze e i nodi precedenti per ogni nodo
    if dest == None:
        return (dist, prec)
    # caso in cui specifico un nodo destinazione
    # restituisco la singola distanza e tutto il percorso da orig a dest
    else:

        # se il nodo dest non è raggiungibile
        if dist[dest] == None:
            percorso = []
        # altrimenti ricostruisco il percorso a ritroso da dest a orig
        else:
```

```
                percorso = [dest]
                pos = dest
                while prec[pos] != None:
                    pos = prec[pos]
                    percorso.append(pos)
                percorso = percorso[::-1]

            return (dist[dest], percorso)


with open('9-3-14.txt', 'r', encoding='utf-8') as f:
    posnodi = {}
    val = 0
    g = Grafo()
    for riga in f:
        riga = riga.strip().split()
        n1 = riga[1]
        n2 = riga[3]
        peso = int(riga[4])
        if n1 not in posnodi:
            posnodi[n1] = val
            val += 1
            g.nodi.append(Nodo(n1))
        if n2 not in posnodi:
            posnodi[n2] = val
            val += 1
            g.nodi.append(Nodo(n2))
        g.nodi[posnodi[n1]].archi.append((posnodi[n2], peso))
        g.nodi[posnodi[n2]].archi.append((posnodi[n1], peso))
        # print(posnodi)
        # print(g)

print(g)

# 1.    Scrivere il percorso più breve (in km) fra Bananopoli e Hoppavilla
print(g.distanza(posnodi['Bananopoli'], posnodi['Hoppavilla']))
# 2.    Scrivere il percorso più breve (in km) fra Giuggiolo e Cedrosa
print(g.distanza(posnodi['Giuggiolo'], posnodi['Cedrosa']))
# 3.    Scrivere la strada più diretta (meno città intermedie) fra Bananopoli e Hoppavilla
print(g.passi(posnodi['Bananopoli'], posnodi['Hoppavilla']))
# 4.    Scrivere la strada più diretta (meno città intermedie) fra Giuggiolo e Cedrosa
print(g.passi(posnodi['Giuggiolo'], posnodi['Cedrosa']))
```

7.3.15 Valerio, Teresa, Sergio, Nadia, Marco, Luca, Francesca e Beatrice sono compagni di classe. Capita che uno vada a casa dell'altro con il pullman; la seguente tabella riporta quanto tempo ci vuole in media perché uno arrivi dall'altro, in minuti (nelle colonne per brevità ciascuno è indicato dall'iniziale del suo nome). Quando non c'è scritto niente è perché quello spostamento non avviene mai: ad esempio Beatrice non va mai a casa di Teresa.

Una piccola difficoltà in più: non è detto che se uno va a casa dell'altro valga anche il viceversa… anzi, in molti casi questo non succede: Marco va da Francesca, ma Francesca non va da Marco.

Quando però c'è uno scambio (ad esempio Luca va da Teresa, e anche Teresa va da Luca) il tempo che ci mettono potrebbe essere diverso.

Fino a casa di → / **Da** casa di ↓	B	F	L	M	N	S	T	V
Beatrice			25	16	18	20		
Francesca					15			
Luca							32	
Marco	13	22						
Nadia			16					
Sergio			28		23			
Teresa			28		22			
Valerio	35			48		42		

Crea un grafo che rispetti i dati contenuti nella tabella e poi rispondi alle seguenti domande:

1. Stampa i ragazzi in ordine secondo il numero di persone che vanno a trovarle
2. Stampa i ragazzi in ordine secondo il numero di persone cha vanno a trovare
3. Qual è il ragazzo che vive più lontano rispetto a tutti gli altri?

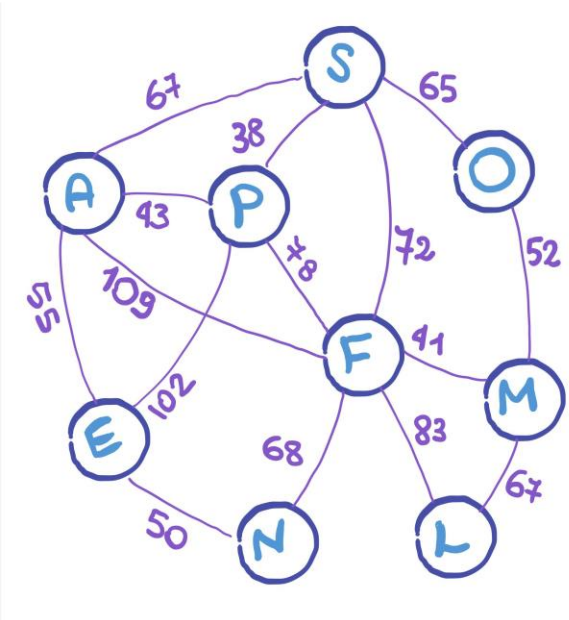
Soluzione

In questa soluzione ho inserito manualmente i dati via codice (non leggendo da file)

```
class Nodo:
    def __init__(self, val = None, archi = None) -> None:
        self.val = val
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def __repr__(self) -> str:
        return f'{self.archi}'

class Grafo:
    def __init__(self, nodi = {}) -> None:
        self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
        for key, nodo in self.nodi.items():
            ris += f'{key}: '
            for arco in nodo.archi:
                ris += f'{arco} '
            ris += '\n'
        return ris

# Crea un grafo che rispetti i dati contenuti nella tabella e poi rispondi alle seguenti domande:
g = Grafo()
g.nodi['Beatrice'] = Nodo('Beatrice', [('Luca', 25), ('Marco', 16), ('Nadia', 18), ('Sergio',
20)])
g.nodi['Francesca'] = Nodo('Francesca', [('Nadia', 15)])
g.nodi['Luca'] = Nodo('Luca', [('Teresa', 32)])
g.nodi['Marco'] = Nodo('Marco', [('Beatrice', 13), ('Francesca', 22)])
```

```python
g.nodi['Nadia'] = Nodo('Nadia', [('Luca', 16)])
g.nodi['Sergio'] = Nodo('Sergio', [('Luca', 28), ('Nadia', 23)])
g.nodi['Teresa'] = Nodo('Teresa', [('Luca', 28), ('Nadia', 22)])
g.nodi['Valerio'] = Nodo('Valerio', [('Beatrice', 35), ('Marco', 48), ('Sergio', 42)])

print(g)

# 1.    Stampa i ragazzi in ordine secondo il numero di persone che vanno a trovarle
persone = dict([(persona, 0) for persona in g.nodi])
print(persone)
for key in g.nodi:
    for persona, peso in g.nodi[key].archi:
        persone[persona] += 1

print(persone)
lista = list(persone.items())
lista.sort(key = lambda x: -x[1])
print(lista[0])

# 2.    Stampa i ragazzi in ordine secondo il numero di persone cha vanno a trovare
persone = dict([(persona, 0) for persona in g.nodi])
for persona in g.nodi:
    persone[persona] = len(g.nodi[persona].archi)
lista = list(persone.items())
lista.sort(key = lambda x: -x[1])
print(lista[0])

# 3.    Qual è il ragazzo che vive più lontano rispetto a tutti gli altri?
persone = dict([(persona, 0) for persona in g.nodi])
for persona in g.nodi:
    distanze = [arco[1] for arco in g.nodi[persona].archi]
    persone[persona] = sum(distanze) / len(distanze)
lista = list(persone.items())
lista.sort(key = lambda x: -x[1])
print(lista[0])
```

7.3.16 La Repubblica di Micimao (confinante col regno di Banania) ha nove città principali: Angoraville, East Meowia, Felinopolis, Lynxia, Manxester, Newcat, Ocelocity, Purrington e St. Cougar. Il seguente grafo rappresenta la rete stradale dello stato di Micimao (ogni città è rappresentata dalla sua iniziale):



Scrvi un programma che sia in grado di stampare una tabella come quella seguente che indica per ogni coppia di città la distanza minima da percorrere per andare da una all'altra:

km	A	E	F	L	M	N	O	P	S
Angoraville		55	109	192	150	105	132	43	67
East Meowia	55		118	126	159	50	187	98	122
Felinopolis	109	118		83	41	68	93	78	72
Lynxia	192	126	83		67	151	119	161	155
Manxester	150	159	41	67		109	52	119	113
Newcat	105	50	68	151	109		161	146	140
Ocelocity	132	187	93	119	52	161		103	65
Purrington	43	98	78	161	119	146	103		38
St. Cougar	67	122	72	155	113	140	65	38	

7.3.17 La seguente tabella mostra le case di una città fangosa (prendono il nome dalle famiglie che ci vivono) e quanto costerebbe (in migliaia di euro) asfaltare il percorso da una all'altra. Quando non c'è scritto niente è perché non è possibile asfaltare quel percorso. Inoltre, il percorso da A a B e quello da B ad A prevedono lo stesso costo, per cui la tabella riporta solo uno dei due.

	A	B	C	D	E	F	G	I	L	M
Alberti		12	6						9	
Brambilla			3	6	9			6	5	7
Coletti				4						
Donato					7					11
Erdani						6				5
Fumicola							9			8
Gianone								5		3
Illirio									11	10
Luzzati										8
Mura										

Scrivi un programma che sia in grado di dire:

1. data una coppia di case quali strade sia meglio asfaltare (minimizza il percorso) per collegare le due case con sole strade asfaltate.

2. Quali strade asfaltare per collegare tutte le case, asfaltando il meno possibile (algoritmo di kruskal)

7.3.18 Fai la stessa cosa che hai fatto per l'esercizio precedente ma con questo grafo:

	M	N	O	P	R	S	T	U	V	Z
Mariani		13	10	11	12					
Nedda			10	6	11					
Ortani				13	9					
Pozzo					10		8			
Rugato						5				
Santini							16	6	8	8
Terna								9	5	7
Uscari									10	6
Viola										5
Zante										

7.3.19 Leggi dal seguente file di testo i dati riguardanti un grafo che indica un insieme di città collegate da strade. Scrivi ed utilizza poi una funzione che sia in grado di dire se da ogni città è possibile raggiungere qualsiasi altra città percorrendo meno di 10km.

Nel file la prima riga indica il numero di nodi, la seconda le strade (archi) che li collegano alle altre città

```
5

1 25, 2 30, 4 60, 2 20, 3 70, 5 10
```

Soluzione

```python
class Nodo:
    def __init__(self, val = None, archi = None) -> None:
        self.val = val
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def __repr__(self) -> str:
        return f'{self.val}'

class Grafo:
    def __init__(self, nodi) -> None:
        self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
        for i in range(len(self.nodi)):
            ris += f'{i}. Val: {self.nodi[i].val}, Archi: {self.nodi[i].archi}\n'
        return ris


    def distanza(self, orig, dest = None):
```

```
        if orig < 0 or orig >= len(self.nodi):
            return None
        if dest != None and (dest < 0 or dest >= len(self.nodi)):
            return None

        # lista delle distanze di ogni nodo da orig
        # all'inizio sono tutte sconosciute (None)
        dist = [None for _ in range(len(self.nodi))]
        # tranne l'origine che ha distanza 0 da se stessa
        dist[orig] = 0

        # lista in cui segno per ogni nodo da quale altro nodo
        # ci arrivo nel percorso ottimo (all'inizio è tutto sconosciuto)
        prec = [None for _ in range(len(self.nodi))]

        # nodi da controllare
        controllare = [orig]
        controllati = []

        # # nodo da cui partire
        # corrente = orig

        while len(controllare) > 0:
            # scelgo il prossimo nodo da controllare (corrente)
            # trovo il nodo con distanza minima in controllare
            corrente = controllare[0]
            for nodo in controllare:
                if dist[nodo] < dist[corrente]:
                    corrente = nodo

            # controllo tutti i nodi raggiungibili da corrente con i suoi archi
            for pos, peso in self.nodi[corrente].archi:
                # ogni nodo che raggiungo da corrente con il minor costo visto finora
                # per quel nodo va aggiornato (distanza ottima e nodo precedente nel
                # percorso ottimo) e aggiunto a controllare
                if dist[pos] == None or dist[corrente] + peso < dist[pos]:
                    dist[pos] = dist[corrente] + peso
                    prec[pos] = corrente
                    if pos not in controllare and pos not in controllati:
                        controllare.append(pos)

            # rimuovo da controllare il nodo corrente, qui non ci potrò mai più
            # tornare perchè è impossibile che possa tornarci con distanza minore
            # di ora perchè quando scelgo il nodo da controllare prendo sempre
            # quello con distanza minima
            controllare.remove(corrente)
            controllati.append(corrente)

        # caso in cui non specifico una destinazione
        # restituisco tutte le distanze e i nodi precedenti per ogni nodo
        if dest == None:
            return (dist, prec)
        # caso in cui specifico un nodo destinazione
        # restituisco la singola distanza e tutto il percorso da orig a dest
        else:

            # se il nodo dest non è raggiungibile
            if dist[dest] == None:
                percorso = []
            # altrimenti ricostruisco il percorso a ritroso da dest a orig
            else:
                percorso = [dest]
                pos = dest
                while prec[pos] != None:
                    pos = prec[pos]
                    percorso.append(pos)
                percorso = percorso[::-1]
```

```
            return (dist[dest], percorso)

def funz(g, x):
    for nodo in range(len(g.nodi)):
        distanze = g.distanza(nodo)[0]
        # print(distanze)
        for distanza in distanze:
            if distanza == None or distanza > x:
                return False
    return True


with open('9-3-19.txt', 'r', encoding='utf-8') as f:
    nnodi = int(f.readline().strip())
    archi = f.readline().strip().split(',')
    archi = [tuple(map(int, arco.split())) for arco in archi]
    g = Grafo([Nodo() for i in range(nnodi)])
    for arco in archi:
        g.nodi[arco[0]].archi.append((arco[1], arco[2]))
        g.nodi[arco[1]].archi.append((arco[0], arco[2]))

print(g)
print(funz(g, 100))
print(funz(g, 200))
```

7.3.20 Leggi dal seguente file di testo i dati riguardanti un grafo che rappresentano le informazioni note riguardo al traffico in una rete. Scrivi e utilizza poi una funzione che dica al nodo 0 a quale nodo conviene spedire i dati perché arrivino nel più breve tempo possibile al nodo 3

Nel file la prima riga indica il numero dei nodi, le righe seguenti i tempi noti di trasmissione dei dati tra un nodo e un altro (in ms). Notare che siccome i canali di comunicazione possono essere gestiti in maniera non simmetrica o monodirezionale il grafo risulta orientato.

```
5
1 2, 2 10, 4 8
2 3
3 7, 4 2
4 3
0 3, 3 2
```

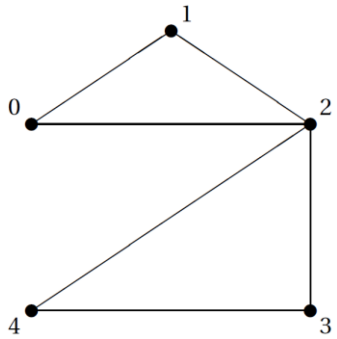
Soluzione

```
class Nodo:
    def __init__(self, val = None, archi = None) -> None:
        self.val = val
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def __repr__(self) -> str:
        return f'{self.val}'

class Grafo:
    def __init__(self, nodi = []) -> None:
        self.nodi = nodi

    def __repr__(self) -> str:
        ris = ''
```

```
        for i in range(len(self.nodi)):
            ris += f'{i}. Val: {self.nodi[i].val}, Archi: {self.nodi[i].archi}\n'
        return ris


    def distanza(self, orig, dest = None):
        if orig < 0 or orig >= len(self.nodi):
            return None
        if dest != None and (dest < 0 or dest >= len(self.nodi)):
            return None

        # lista delle distanze di ogni nodo da orig
        # all'inizio sono tutte sconosciute (None)
        dist = [None for _ in range(len(self.nodi))]
        # tranne l'origine che ha distanza 0 da se stessa
        dist[orig] = 0

        # lista in cui segno per ogni nodo da quale altro nodo
        # ci arrivo nel percorso ottimo (all'inizio è tutto sconosciuto)
        prec = [None for _ in range(len(self.nodi))]

        # nodi da controllare
        controllare = [orig]
        controllati = []

        # # nodo da cui partire
        # corrente = orig

        while len(controllare) > 0:
            # scelgo il prossimo nodo da controllare (corrente)
            # trovo il nodo con distanza minima in controllare
            corrente = controllare[0]
            for nodo in controllare:
                if dist[nodo] < dist[corrente]:
                    corrente = nodo

            # controllo tutti i nodi raggiungibili da corrente con i suoi archi
            for pos, peso in self.nodi[corrente].archi:
                # ogni nodo che raggiungo da corrente con il minor costo visto finora
                # per quel nodo va aggiornato (distanza ottima e nodo precedente nel
                # percorso ottimo) e aggiunto a controllare
                if dist[pos] == None or dist[corrente] + peso < dist[pos]:
                    dist[pos] = dist[corrente] + peso
                    prec[pos] = corrente
                    if pos not in controllare and pos not in controllati:
                        controllare.append(pos)

            # rimuovo da controllare il nodo corrente, qui non ci potrò mai più
            # tornare perchè è impossibile che possa tornarci con distanza minore
            # di ora perchè quando scelgo il nodo da controllare prendo sempre
            # quello con distanza minima
            controllare.remove(corrente)
            controllati.append(corrente)

        # caso in cui non specifico una destinazione
        # restituisco tutte le distanze e i nodi precedenti per ogni nodo
        if dest == None:
            return (dist, prec)
        # caso in cui specifico un nodo destinazione
        # restituisco la singola distanza e tutto il percorso da orig a dest
        else:

            # se il nodo dest non è raggiungibile
            if dist[dest] == None:
                percorso = []
            # altrimenti ricostruisco il percorso a ritroso da dest a orig
            else:
```

```
                percorso = [dest]
                pos = dest
                while prec[pos] != None:
                    pos = prec[pos]
                    percorso.append(pos)
                percorso = percorso[::-1]

            return (dist[dest], percorso)


with open('9-3-20.txt', 'r', encoding='utf-8') as f:
    nnodi = int(f.readline().strip()) # in realtà poi non lo uso per come ho fatto il resto
    g = Grafo()
    for riga in f:
        a = riga.strip().split(',')
        a = [tuple(map(int, arco.split())) for arco in a]
        g.nodi.append(Nodo(archi = a))

print(g)
percorso = g.distanza(0, 3)[1]
print(percorso)
print("Conviene spedire al nodo", percorso[1])
```

7.3.21 Istanzia il seguente grafo:



Scrivi:

1. una funzione che somma i valori di tutti i nodi del grafo.
2. Una funzione che determina se il grafo è connesso.

Soluzione 1 con grafo che contiene direttamente l'elenco di archi (creazione grafo più semplice, funzione un po più complessa)

```
# 9.3.21    Istanzia il seguente grafo:
# Scrivi:
# 1.    una funzione che somma i valori di tutti i nodi del grafo.
# 2.    Una funzione che determina se il grafo è connesso.

class Nodo:
    def __init__(self, val) -> None:
        self.val = val

    def __repr__(self) -> str:
        return f'{self.val}'


class Grafo:
    def __init__(self, nodi=None, archi=None) -> None:
        if nodi == None:
            self.nodi = []
        else:
```

```
            self.nodi = nodi
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def sommaValori(self):
        somma = 0
        for nodo in self.nodi:
            somma += nodo.val
        return somma

    def connesso(self):
        if len(self.nodi) == 0:
            return True

        rimasti = [i for i in range(len(self.nodi))]
        rimasti.pop(0)
        print(self.nodi)
        coda = [self.nodi[0].val]
        while len(coda) > 0:
            corr = coda[0]
            coda.pop(0)
            for arco in self.archi:
                if arco[0] == corr:
                    if arco[1] in rimasti:
                        coda.append(arco[1])
                        rimasti.remove(arco[1])
                if arco[1] == corr:
                    if arco[0] in rimasti:
                        coda.append(arco[0])
                        rimasti.remove(arco[0])

        return len(rimasti) == 0


g = Grafo([Nodo(0), Nodo(1), Nodo(2), Nodo(3), Nodo(4)], [
          (0, 1), (0, 2), (1, 2), (2, 4), (2, 3), (3, 4)])
print(g.connesso())

g = Grafo([Nodo(0), Nodo(1), Nodo(2), Nodo(3), Nodo(4), Nodo(5)], [
          (0, 1), (0, 2), (1, 2), (2, 4), (2, 3), (3, 4)])
print(g.connesso())
```

Soluzione 2 con nodi che contengono la lista di adiacenza (funzione più semplice)

```
class Nodo:
    def __init__(self, val, archi = None) -> None:
        self.val = val
        if archi == None:
            self.archi = []
        else:
            self.archi = archi

    def __repr__(self) -> str:
        return f'{self.val}'



class Grafo:
    def __init__(self, nodi=None) -> None:
        if nodi == None:
            self.nodi = []
        else:
```

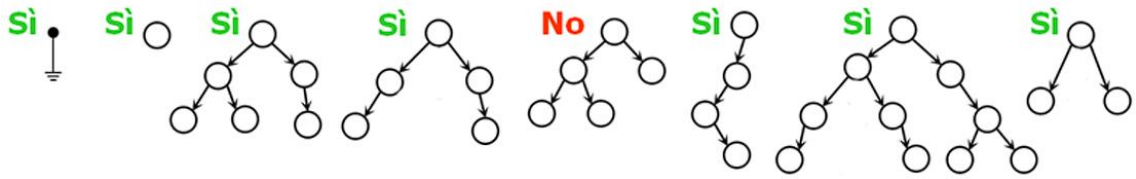
```
            self.nodi = nodi


    def sommaValori(self):
        somma = 0
        for nodo in self.nodi:
            somma += nodo.val
        return somma

    def connesso(self):
        if len(self.nodi) == 0:
            return True

        rimasti = [i for i in range(len(self.nodi))]
        rimasti.pop(0)
        print(self.nodi)
        coda = [self.nodi[0].val]
        while len(coda) > 0:
            corr = coda[0]
            coda.pop(0)
            for dest in self.nodi[corr].archi:
                if dest in rimasti:
                    coda.append(dest)
                    rimasti.remove(dest)


        return len(rimasti) == 0



g = Grafo()
g.nodi.append(Nodo(0, [1, 2]))
g.nodi.append(Nodo(1, [0, 2]))
g.nodi.append(Nodo(2, [0, 1, 3, 4]))
g.nodi.append(Nodo(3, [2, 4]))
g.nodi.append(Nodo(4, [2, 3]))
print(g.connesso())

g = Grafo()
g.nodi.append(Nodo(0, [1, 2]))
g.nodi.append(Nodo(1, [0, 2]))
g.nodi.append(Nodo(2, [0, 1, 3, 4]))
g.nodi.append(Nodo(3, [2, 4]))
g.nodi.append(Nodo(4, [2, 3]))
g.nodi.append(Nodo(5))
print(g.connesso())
```

7.3.22

7.4 Alberi

7.4.1 Un albero binario si dice isobato se tutti i cammini dalla radice alle foglie hanno la stessa lunghezza. Dopo aver definito opportunamente la struttura dati, scrivi una funzione che determina se un albero è isobato.

Esempi:



Variante: Usa un albero generico e non uno binario.

Soluzione

```
# Un albero binario si dice isobato se tutti i cammini
# dalla radice alle foglie hanno la stessa lunghezza. Dopo aver definito
# opportunamente la struttura dati, scrivi una funzione che determina se un
# albero è isobato.
# Vedi gli esempi nell'immagine allegata

class Nodo:
    def __init__(self, val, sx = None, dx = None) -> None:
        self.val = val
        self.sx = sx
        self.dx = dx

    def __repr__(self) -> str:
        return f'{self.val}'

    def isobato(self, prof = [], curr = 0):
        if self.sx == None and self.dx == None:
            prof.append(curr)
        if self.sx != None:
            self.sx.isobato(prof, curr+1)
        if self.dx != None:
            self.dx.isobato(prof, curr+1)
        # if prof == []:
        #     return True
        p = prof[0]
        for num in prof:
            if num != p:
                return False
        return True


radice = Nodo(1, Nodo(2, Nodo(10)), Nodo(9, Nodo(36), Nodo(44)))
print(radice.isobato())
radice = Nodo(1, Nodo(2), Nodo(9, Nodo(36), Nodo(44)))
print(radice.isobato())
```

Soluzione con albero generico

```
class Nodo:
    def __init__(self, val, figli=None) -> None:
        self.val = val
        if figli == None:
            self.figli = []
        else:
            self.figli = figli

    def __repr__(self) -> str:
        return f'{self.val}'

    def isobato(self, prof=[], curr=0):
        if self.figli == []:
            prof.append(curr)
        for figlio in self.figli:
            figlio.isobato(prof, curr+1)
```

```
        if prof == []:
            return True
        p = prof[0]
        for num in prof:
            if num != p:
                return False
        return True


radice = Nodo(1, [Nodo(2, [Nodo(10)]), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.isobato())
radice = Nodo(1, [Nodo(2), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.isobato())
```

Soluzione alternativa

```
from mytree import Tree

def f(nodo):
    if nodo.figli == []:
        return 0
    else:
        profFigli = [f(figlio) for figlio in nodo.figli]
        prof = profFigli[0]
        if prof == -1:
            return -1
        for profFiglio in profFigli:
            if profFiglio != prof:
                return -1
        return prof + 1

def isobato(nodo):
    if f(nodo) == -1:
        return False
    else:
        return True

t1 = Tree(5, [
        Tree(3, [
            Tree(1, [])
        ]),
        Tree(7, [
            Tree(5, [
                Tree(4, [])
            ]),
            Tree(9, [])
        ])
    ])

t1.stampa()
print()
print(isobato(t1))



t2 = Tree(5, [
        Tree(3, [
            Tree(1, [])
        ]),
        Tree(7, [
            Tree(5, []),
            Tree(9, [])
        ])
    ])

t2.stampa()
```

```
print()
print(isobato(t2))
```

7.4.2 Scrivi una funzione in grado di restituire il numero di nodi di un albero. L'albero può essere binario o no, in base a ciò la funzione cambia.

Soluzione implementando la funzione come metodo della classe

```
# 9.4.2 - Scrivi una funzione in grado di restituire il numero di
# nodi di un albero. L'albero può essere binario o no, in base a
# ciò la funzione cambia.

class Nodo:
    def __init__(self, val, figli = None) -> None:
        self.val = val
        if figli == None:
            self.figli = []
        else:
            self.figli = figli

    def __repr__(self) -> str:
        return f'{self.val}'

    def numNodi(self):
        ris = 1
        for figlio in self.figli:
            ris += figlio.numNodi()
        return ris

radice = Nodo(1, [Nodo(2, [Nodo(10)]), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.numNodi())
radice = Nodo(1, [Nodo(2), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.numNodi())
```

Soluzione fatta importando la classe e implementando la funzione come funzione esterna alla classe

```
from mytree import Tree

def contaNodi(nodo):
    ris = 1
    for figlio in nodo.figli:
        ris += contaNodi(figlio)

    return ris

t = Tree(5, [
        Tree(3, [
            Tree(1, [])
        ]),
        Tree(2, []),
        Tree(7, [
            Tree(1, []),
            Tree(5, [
                Tree(4, [])
            ]),
            Tree(9, [])
        ])
    ])
```

```
t.stampa()

print()
print(contaNodi(t))
```

7.4.3 Scrivi una funzione in grado di contare le foglie di un albero.

Soluzione

```
# 9.4.3 Scrivi una funzione in grado di
# contare le foglie di un albero.

class Nodo:
    def __init__(self, val, figli = None) -> None:
        self.val = val
        if figli == None:
            self.figli = []
        else:
            self.figli = figli

    def __repr__(self) -> str:
        return f'{self.val}'

    def numFoglie(self):
        if self.figli == []:
            return 1
        else:
            ris = 0
            for figlio in self.figli:
                ris += figlio.numFoglie()
            return ris


radice = Nodo(1, [Nodo(2, [Nodo(10)]), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.numFoglie())
radice = Nodo(1, [Nodo(2), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.numFoglie())
```

Soluzione con albero binario

```
class Nodo:
    def __init__(self, val, sx=None, dx=None) -> None:
        self.val = val
        self.sx = sx
        self.dx = dx

    def __repr__(self) -> str:
        return f'{self.val}'

    def numNodi(self):
        if self.sx == None and self.dx == None:
            return 1
        else:
            ris = 0
            if self.sx != None:
                ris += self.sx.numNodi()
            if self.dx != None:
                ris += self.dx.numNodi()
            return ris


radice = Nodo(1, [Nodo(2, [Nodo(10)]), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.numNodi())
```

```
radice = Nodo(1, [Nodo(2), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.numNodi())
```

7.4.4 Scrivi una funzione in grado di verificare la profondità massima di un albero

Soluzione

```
# 9.4.4 - Scrivi una funzione in grado di verificare la
# profondità massima di un albero

class Nodo:
    def __init__(self, val, figli = None) -> None:
        self.val = val
        if figli == None:
            self.figli = []
        else:
            self.figli = figli

    def __repr__(self) -> str:
        return f'{self.val}'

    def profMax(self):
        if self.figli == []:
            return 0
        else:
            risFigli = [figlio.profMax() for figlio in self.figli]
            return max(risFigli) + 1

    def profMax2(self, prof = 0):
        if self.figli == []:
            return prof
        else:
            return max([figlio.profMax2(prof+1) for figlio in self.figli])



radice = Nodo(1, [Nodo(2, [Nodo(10)]), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
print(radice.profMax())
print(radice.profMax2())
radice = Nodo(1, [Nodo(2, [Nodo(10, [Nodo(15)]), Nodo(3)]), Nodo(4, [Nodo(19)]), Nodo(9)])
print(radice.profMax())
print(radice.profMax2())
```

7.4.5 Scrivi una funzione in grado di dire se un certo valore è contenuto nell'albero

Variante: viene restituita la profondità in cui si trova il nodo

Soluzione

```
from alberi import Nodo

def contiene(nodo, val):
    if nodo.dato == val:
        return True

    for figlio in nodo.figli:
        if contiene(figlio, val):
            return True
```

```
    return False

# Variante: viene restituita la profondità in cui si trova il nodo
def contiene2(nodo, val):
    if nodo.dato == val:
        return [True, 0]

    for figlio in nodo.figli:
        ris = contiene2(figlio, val)
        if ris[0] == True:
            return [True, ris[1]+1]

    return [False, None]

a = Nodo(5, [
        Nodo(1, [
            Nodo(10, []),
            Nodo(4, [])
        ]),
        Nodo(2, []),
        Nodo(3, [
            Nodo(7, [])
        ])
    ])

print(contiene(a, 22))
print(contiene(a, 3))
print(contiene2(a, 22))
print(contiene2(a, 3))
```

7.4.6 Definiamo un albero come "artussiano" se è composto solo da

1. nodi foglia
2. nodi con un solo figlio
3. nodi con due figli aventi lo stesso numero di discendenti

Codificare una funzione che riceve in input un albero e determina se l'albero è "artussiano".

7.4.7 Supponiamo che percorrendo un cammino dalla radice alle foglie si totalizzi un punteggio pari alla somma dei valori contenuti nei nodi percorsi.

Scrivere una funzione maxPunti che calcola il punteggio massimo che possiamo totalizzare percorrendo un cammino dalla radice alle foglie.

Soluzione

```
from alberi import Nodo

def maxPunti(nodo):
    if nodo.figli == []:
        return nodo.dato

    risfigli = [maxPunti(figlio) for figlio in nodo.figli]
    return max(risfigli)+nodo.dato
```

```
a = Nodo(5, [
        Nodo(1, [
            Nodo(10, []),
            Nodo(4, [])
        ]),
        Nodo(2, []),
        Nodo(3, [
            Nodo(7, [])
        ])
    ])

print(maxPunti(a))
```

7.4.8 Supponiamo che percorrendo un cammino dalla radice alle foglie si totalizzi un punteggio pari alla somma dei valori contenuti nei nodi percorsi.

Vogliamo percorrere l'albero dalla radice ad una foglia totalizzando esattamente un certo punteggio: né far meno, né sforare. Scrivere una funzione esisteCammino che dati un albero ed un intero k, dice se esiste un cammino dalla radice ad una foglia che permette di totalizzare esattamente k punti. (difficile)

Soluzione

```
from alberi import Nodo

def esisteCammino(nodo, x):
    if nodo.figli == []:
        if nodo.dato == x:
            return True
        else:
            return False

    for figlio in nodo.figli:
        if esisteCammino(figlio, x-nodo.dato):
            return True

    return False


a = Nodo(5, [
        Nodo(1, [
            Nodo(10, []),
            Nodo(4, [])
        ]),
        Nodo(2, []),
        Nodo(3, [
            Nodo(7, [])
        ])
    ])

print(esisteCammino(a, 7))
```

7.4.9 Scrivi e usa una funzione che conta quanti sono i possibili cammini dalla radice alle foglie di un albero.

Soluzione

```
class Nodo:
    def __init__(self, val, figli = None) -> None:
```

```
        self.val = val
        if figli == None:
            self.figli = []
        else:
            self.figli = figli

    def __repr__(self) -> str:
        return f'{self.val}'

    def numCammini(self):
        if self.figli == []:
            return 1
        else:
            risFigli = [figlio.numCammini() for figlio in self.figli]
            return sum(risFigli)
            # ris = 0
            # for figlio in self.figli:
            #     ris += figlio.numCammini()
            return ris

def numCammini(radice):
    ris = 0
    coda = [radice]
    while len(coda) > 0:
        curr = coda[0]
        if curr.figli == []:
            ris += 1
        else:
            for figlio in curr.figli:
                coda.append(figlio)
        coda.pop(0)
    return ris

def numCammini2(radice):
    ris = 0
    pila = [radice]
    while len(pila) > 0:
        curr = pila[-1]
        if curr.figli == []:
            ris += 1
        else:
            for figlio in curr.figli:
                pila.append(figlio)
        pila.pop()
    return ris

# radice1 = Nodo(1, [Nodo(2, [Nodo(10)]), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
radice = Nodo(1, [Nodo(2, [Nodo(10, [Nodo(15)]), Nodo(3)]), Nodo(4, [Nodo(19), Nodo(5)]),
Nodo(9)])

print(radice.numCammini())
print(numCammini(radice))
```

7.4.10 Scrivi e usa una funzione che calcola la somma dei valori contenuti ad ogni diversa profondità di un albero.

Soluzione

```
# 9.4.9 - Scrivi e usa una funzione che calcola la somma dei
# valori contenuti ad ogni diversa profondità di un albero.

class Nodo:
    def __init__(self, val, figli = None) -> None:
        self.val = val
```

```
        if figli == None:
            self.figli = []
        else:
            self.figli = figli

    def __repr__(self) -> str:
        return f'{self.val}'

    def sommeProf(self, ris = None, prof = 0):
        if ris == None:
            ris = []

        if len(ris) <= prof:
            ris.append(self.val)
        else:
            ris[prof] += self.val

        for figlio in self.figli:
            figlio.sommeProf(ris, prof+1)

        return ris


# radice1 = Nodo(1, [Nodo(2, [Nodo(10)]), Nodo(4, [Nodo(19)]), Nodo(9, [Nodo(36), Nodo(44)])])
radice = Nodo(1, [Nodo(2, [Nodo(10, [Nodo(15)]), Nodo(3)]), Nodo(4, [Nodo(19), Nodo(5)]),
Nodo(9)])

print(radice.sommeProf())
```

7.4.11 Scrivi una funzione in grado di stampare nel modo migliore possibile il contenuto di un albero.

Soluzione per alberi generici

```
class Nodo:
    def __init__(self, dato = None, figli = None) -> None:
        self.dato = dato
        self.figli = figli
        if figli == None:
            self.figli = []


    def __repr__(self, prof=0) -> str:
        ris = ''
        for i in range(prof):
                ris += '    '
        ris += f"{self.dato}\n"
        for i, figlio in enumerate(self.figli):
                ris += f"{figlio.__repr__(prof+1)}"
        return ris

a = Nodo(5, [
        Nodo(1, [
            Nodo(10, []),
            Nodo(4, [])
        ]),
        Nodo(2, []),
        Nodo(3, [
            Nodo(7, [])
        ])
    ])

print(a)
```

Soluzione per alberi binari

```
class Nodo:
    def __init__(self, dato = None, figli = None) -> None:
        self.dato = dato
        self.figli = figli
        if figli == None:
            self.figli = []


    def __repr__(self, prof=0) -> str:
        ris = ''
        for i in range(prof):
                ris += '    '
        ris += f"{self.dato}\n"
        for i, figlio in enumerate(self.figli):
                ris += f"{figlio.__repr__(prof+1)}"
        return ris

a = Nodo(5, [
        Nodo(1, [
            Nodo(10, []),
            Nodo(4, [])
        ]),
        Nodo(2, []),
        Nodo(3, [
            Nodo(7, [])
        ])
    ])

print(a)
```

7.4.12 La larghezza di un albero è il numero massimo di nodi che stanno tutti al medesimo livello. Scrivi e usa una funzione che calcola la larghezza di un albero.

7.4.13 "Visitare" un albero significa esaminare sequenzialmente tutti i suoi nodi. Ci sono tre tipi principali di visite:

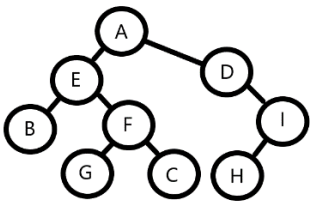
1. Nella visita in ordine simmetrico si visita il sottoalbero sinistro, quindi si esamina la radice e infine si visita il sottoalbero destro.
2. Nella visita in ordine anticipato si esamina prima la radice e quindi si visitano il sottoalbero sinistro e quello destro.
3. Nella visita in ordine posticipato si visitano prima il sottoalbero sinistro poi quello destro e infine si esamina la radice.

Gli ordini di visita di un albero binario T di 9 nodi sono i seguenti:

1. A, E, B, F, G, C, D, I, H (anticipato)
2. B, G, C, F, E, H, I, D, A (posticipato)
3. B, E, G, F, C, A, D, H, I (simmetrico).

Istanzia l'albero binario T.

Soluzione



7.4.14 Scrivi le funzioni per effettuare le visite (e le stampe) anticipata, differita e simmetrica di un albero binario

Variante: invece che fare la stampa, restituisci la lista dei valori.

Soluzione

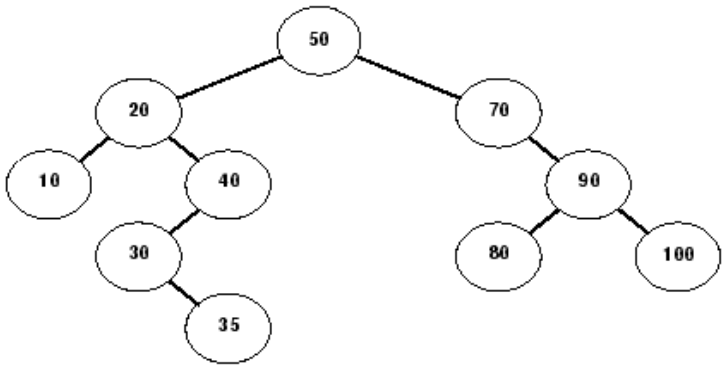
```
class Nodo:
    def __init__(self, dato = None, sx = None, dx = None) -> None:
        self.dato = dato
        self.sx = sx
        self.dx = dx

    def __repr__(self, prof=0, lato = '') -> str:
        ris = ''
        for i in range(prof):
                ris += '    '
        ris += f"{lato}{self.dato}\n"
        if self.sx != None:
            ris += f"{self.sx.__repr__(prof+1, 'sx: ')}"
        if self.dx != None:
            ris += f"{self.dx.__repr__(prof+1, 'dx: ')}"
        return ris

    def anticipata(self):
        ris = [self.dato]
        if self.sx != None:
            ris += self.sx.anticipata()
        if self.dx != None:
            ris += self.dx.anticipata()
        return ris

    def differita(self):
        ris = []
        if self.sx != None:
            ris += self.sx.differita()
        if self.dx != None:
            ris += self.dx.differita()
        ris.append(self.dato)
        return ris

    def simmetrica(self):
        ris = []
        if self.sx != None:
            ris += self.sx.simmetrica()
        ris.append(self.dato)
        if self.dx != None:
            ris += self.dx.simmetrica()
        return ris


a = Nodo(1,
        Nodo(2,
            Nodo(3),
            Nodo(4,
```

```
                Nodo(5),
                Nodo(6)
            )
        ),
        Nodo(7,
            dx = Nodo(8,
                Nodo(9)
            )
        )
    )

print(a)
print("Anticipata:", a.anticipata())
print("Differita:", a.differita())
print("Simmetrica:", a.simmetrica())
```

Un albero binario di ricerca (ABR) è un albero binario tale che:
sui valori delle chiavi dei suoi nodi è definito un ordinamento totale;
soddisfa la seguente proprietà: per ogni nodo n dell'albero:

- tutte le chiavi dei nodi contenuti nel sottoalbero sinistro di n hanno valore strettamente minore della chiave contenuta in n,
- tutte le chiavi dei nodi contenuti nel sottoalbero destro di n hanno valore strettamente maggiore della chiave contenuta in n.

Un esempio, in cui le chiavi sono interi con l'ovvio ordinamento...



Si noti che in un ABR non ci possono essere due nodi con la stessa chiave.

Un'importante proprietà di un ABR è la seguente: la visita in ordine simmetrico di un albero binario di ricerca genera una sequenza crescente di tutte le sue chiavi.

7.4.15 Data una sequenza di numeri interi casuali, costruisci il corrispettivo albero binario di ricerca.

Soluzione

```
from random import randint

class Nodo:
    def __init__(self, dato = None, sx = None, dx = None) -> None:
        self.dato = dato
        self.sx = sx
        self.dx = dx

    def __repr__(self, prof=0, lato = '') -> str:
```

```
        ris = ''
        for i in range(prof):
                ris += '    '
        ris += f"{lato}{self.dato}\n"
        if self.sx != None:
            ris += f"{self.sx.__repr__(prof+1, 'sx: ')}"
        if self.dx != None:
            ris += f"{self.dx.__repr__(prof+1, 'dx: ')}"
        return ris

    def aggiungi(self, dato):
        if dato < self.dato:
            if self.sx == None:
                self.sx = Nodo(dato)
            else:
                self.sx.aggiungi(dato)
        if dato > self.dato:
            if self.dx == None:
                self.dx = Nodo(dato)
            else:
                self.dx.aggiungi(dato)

a = None
numeri = [randint(0, 20) for _ in range(20)]
print(f"Numeri da inserire: {numeri}")

a = Nodo(numeri[0])
print(f"\ninserisco {numeri[0]}")
print("L'albero diventa:")
print(a)

for numero in numeri[1:]:
    print(f"\ninserisco {numero}")
    a.aggiungi(numero)
    print("L'albero diventa:")
    print(a)
    print()
```

7.4.16 Dato un albero binario, i cui nodi contengono elementi interi, si scriva una funzione per ottenere l'albero inverso, ovvero un albero in cui il figlio destro (con relativo sottoalbero) e scambiato con il figlio sinistro (con relativo sottoalbero).

Soluzione

7.4.17 Scrivi una funzione che sia in grado di ricevere un albero binario e aggiungere ad ogni foglia dell'albero un figlio contenente la somma dei valori che appaiono nel cammino dalla radice a tale foglia.

Soluzione

```
class Nodo:
    def __init__(self, val, sx = None, dx = None) -> None:
        self.val = val
        self.sx = sx
        self.dx = dx

    def __repr__(self, prof=0, lato = '') -> str:
        ris = ''
        for i in range(prof):
                ris += '    '
        ris += f"{lato}{self.val}\n"
```

```
        if self.sx != None:
            ris += f"{self.sx.__repr__(prof+1, 'sx: ')}"
        if self.dx != None:
            ris += f"{self.dx.__repr__(prof+1, 'dx: ')}"
        return ris


    def f(self, somma = 0):
        somma += self.val
        if self.sx == None and self.dx == None:
            self.sx = Nodo(somma)
        else:
            if self.sx != None:
                self.sx.f(somma)
            if self.dx != None:
                self.dx.f(somma)



a = Nodo(1,
        Nodo(2,
            Nodo(3),
            Nodo(4,
                Nodo(5),
                Nodo(6)
            )
        ),
        Nodo(7,
            dx = Nodo(8,
                Nodo(9)
            )
        )
    )

print(a)
a.f()
print()
print(a)
```

7.4.18 Un albero binario completo di altezza k è un albero binario in cui tutti i nodi, tranne le foglie, hanno esattamente due figli, e tutte le foglie si trovano al livello k. Scrivi una funzione che verifica se l'albero è completo.

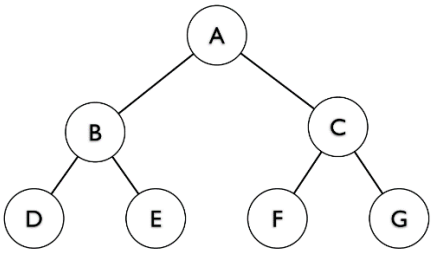
Soluzione

```
class Nodo:
    def __init__(self, val, sx=None, dx=None) -> None:
        self.val = val
        self.sx = sx
        self.dx = dx

    def __repr__(self, prof=0, lato="") -> str:
        ris = ""
        for i in range(prof):
            ris += "   "
        ris += f"{lato}{self.val}\n"
        if self.sx != None:
            ris += f"{self.sx.__repr__(prof+1, 'sx: ')}"
        if self.dx != None:
            ris += f"{self.dx.__repr__(prof+1, 'dx: ')}"
        return ris

    def completo(self, prof = 0, lista = None):
        if lista == None:
```

```
            lista = []
        if prof >= len(lista):
            lista.append(1)
        else:
            lista[prof] += 1
        if self.sx != None:
            self.sx.completo(prof+1, lista)
        if self.dx != None:
            self.dx.completo(prof+1, lista)

        print(self.val, prof, lista)

        for i in range(len(lista)-1):
            if lista[i] * 2 != lista[i+1]:
                return False
        return True



a = Nodo(1,
        Nodo(2,
            Nodo(3),
            Nodo(4,
                Nodo(5),
                Nodo(6)
            )
        ),
        Nodo(7,
            dx = Nodo(8,
                Nodo(9)
            )
        )
    )
print(a)
print(a.completo())

a = Nodo(1,
        Nodo(2,
            Nodo(3),
            Nodo(4)
        ),
        Nodo(7,
            Nodo(8),
            Nodo(9)
        )
    )
print(a)
print(a.completo())
```

7.4.19 Dato un albero binario T, definiamo altezza minimale di un nodo v la minima distanza di v da una delle foglie del suo sottoalbero. Scrivi e usa una funzione che riceve in input un nodo e restituisce la sua altezza minimale.

7.4.20 In un albero binario, definiamo lunghezza di cammino come la somma delle distanze dei nodi dalla radice. Ad esempio, la lunghezza del cammino dell'albero nella seguente figura è pari a 10



Scrivi e usa una funzione che riceve un albero e restituisce la lunghezza di cammino dell'albero

8. Pygame

9. Altro

9.1 Sistema

Questo tipo di programmi non rientra nel normale percorso di studio ma siccome è utile e interessante lo aggiungo per gli interessati. Parte degli esercizi con le relative soluzioni sono presi da https://www.programmareinpython.it/esercizi-python/

9.1.1 Scrivi una funzione che fornisca in output il nome del Sistema Operativo utilizzato con eventuali relative informazioni sulla release corrente.

Soluzione

```
import platform


def sys_info():
    print("Il Sistema attualmente in uso è: " + platform.system())
    print("Info Release: " + platform.release())
```

9.1.2 Scrivi una funzione che calcoli la somma (espressa in MB) delle dimensioni dei file presenti nella cartella di lavoro.

Soluzione

```
import os


def file_size():
    totale = 0
    cartella = os.getcwd()
    for file in os.listdir(cartella):
        totale += os.path.getsize(os.path.join(cartella, file))
    print(f"La somma delle dimensioni dei file presenti nella cartella '{cartella}' è:
{(totale/1048576):.2f}MB")
```

La cosa più utile di questo programma è la funzione os.path.join

9.1.3 Scrivi una funzione "postino" che sia in grado di spedire delle eMail tramite Gmail! (aiuto: puoi usare il modulo smtplib)

Soluzione

```
# soluzione da ricontrollare e forse correggere

import smtplib

def postino():
    print("""
    Questa è la funzione Postino: spedisce eMail utilizzando Gmail!
    Server: smtp.gmail.com
    Porta: 587
    Si richiedono: Username, Password, Destinatario, Oggetto e Messaggio da inviare.
    """)

    username = input("Inserisci il tuo username: ")
    password = input("Inserisci la tua password: ")
    destinatario = input("Inserisci l'email del destinatario: ")
    oggetto = input("Inserisci l'oggetto della mail: ")
    messaggio = input("Ora puoi inserire il messaggio che vuoi inviare: ")
    contenuto = f"Subject: {oggetto}\n\n{messaggio}"
    print("Sto effettuando la connessione col Server...")
    email = smtplib.SMTP("smtp.gmail.com",587)
    email.ehlo()
    email.starttls()
    email.login(username,password)
    print("Sto inviando...")
    email.sendmail(username, destinatario, contenuto)
    email.quit()
    print("Messaggio Inviato!")
```

9.1.4 Scrivi una funzione "cercatrice" che scansioni un dato percorso di sistema alla ricerca di file di tipo pdf.

La funzione dovrà avere le seguenti caratteristiche:

Il percorso fornito dovrà essere anzitutto validato, in quanto deve portare a una cartella esistente

La funzione dovrà fornire un elenco dei file pdf (con/relativo/percorso) man mano che questi vengono trovati

In fine la funzione dovrà fornire in output il totale dei file .pdf che sono stati trovati durante la scansione.

Soluzione

```
import os


def cercatrice(percorso):
    if not os.path.isdir(percorso):
        print(f"Il percorso inserito '{percorso}' risulta non essere un percorso idoneo. Verifica
e riprova, grazie.\n")
        return None
    contatore = 0
    print(f"Sto effettuando la scansione di '{percorso}' alla ricerca di file .pdf\n")
    for cartella, sottocartelle, files in os.walk(percorso):
        for file in files:
            if file.endswith(".pdf"):
                pdf = os.path.join(cartella,file)
                print(f"Trovato file pdf: {pdf}\n")
                contatore += 1
    print(f"\nScansione Ultimata! Ho trovato {contatore} files con estensione pdf.")
```

9.1.5 Scrivi una funzione "file_backup" che sia in grado di effettuare copie di backup di determinati tipi di file, con le seguenti caratteristiche:

Percorso da scansionare, di backup e tipologia di file da copiare dovranno essere passati dall'utente tramite input

Il programma dovrà verificare la presenza o meno di una cartella di backup al percorso fornito, e qualora questa non fosse presente dovrà crearla

La funzione dovrà anche gestire l'eventuale scelta da parte dell'utente, di un percorso da scansionare che non esiste

Soluzione

```
import os
import shutil


def file_backup():
    print("Ciao! Questo script effettua copie di backup per file di un'estensione passata come
input.\n")
    percorso = input("Inserisci il percorso da scansionare: ")
    if not os.path.isdir(percorso):
        print(f"Il percorso inserito '{percorso}' risulta non essere un percorso idoneo. Verifica
e riprova, grazie.\n")
        return file_backup()
    estensione = input("Che tipologia di file desideri salvare? [esempio: .jpg .pdf .epub]: ")
```

```
    backup_folder = input("Inserisci la cartella dove desideri salvare i tuoi file: ")
    if not os.path.isdir(backup_folder):
          os.makedirs(backup_folder)
    contatore = 0
    print(f"Sto effettuando la scansione di '{percorso}' alla ricerca di file '{estensione}'\n")
    for cartella, sottocartelle, files in os.walk(percorso):
        for file in files:
            if file.endswith(estensione):
                match = os.path.join(cartella,file)
                print(f"Trovato file {estensione}: {match}")
                shutil.copy(match,backup_folder)
                contatore += 1
    print("Copia Terminata")
    print(f"Ho trovato '{contatore}' files con estensione {estensione}, ora disponibili anche in
'{backup_folder}'")
```