

# whole

February 9, 2024

## 1 Introduzione a Python

cos'è python?

Python è un linguaggio di programmazione che permette di scrivere istruzioni per far fare cose al computer in modo semplice e leggibile. Puoi usarlo per fare molte cose diverse, come creare siti web, analizzare dati, o anche fare giochi. È amato dagli sviluppatori perché è facile da imparare e da usare.

## 2 Il Primo Programma

```
[5]: print("Ciao", input("Inserisci il tuo nome: "), "!")
```

Ciao Gabriele !

Comandi usati:

1. print()

stampa l'argomento (il codice fra parentesi)

2. input()

chiede un input all'utente, mostrando a schermo l'argomento. L'input viene **sempre** preso come stringa

```
[6]: nome = "TemplateText"
print(nome)
```

TemplateText

Tipi di dati:

Numeri interi (int): Rappresentano numeri interi senza decimali. Ad esempio: 5, -10, 100.

Numeri decimali (float): Rappresentano numeri con decimali. Ad esempio: 3.14, -0.5, 2.0.

Stringhe (str): Rappresentano sequenze di caratteri, come lettere, numeri o simboli, racchiuse tra virgolette. Ad esempio: "ciao", "123", "Python è divertente!".

Booleani (bool): Possono avere solo due valori: True o False. Sono utilizzati per valutare condizioni e controllare il flusso di un programma.

**Assegnamento di variabili:** Assegnare un valore a una variabile: È come dare un nome a un valore; ad esempio: `x = 5` assegna il valore 5 alla variabile `x`.

Riassegnare una variabile: Puoi cambiare il valore di una variabile assegnandole un nuovo valore; ad esempio: `x = 10` cambia il valore di `x` da 5 a 10.

Utilizzare le variabili: Dopo aver assegnato un valore a una variabile, puoi utilizzare quel valore facendo riferimento al nome della variabile; ad esempio: se `x = 5`, allora `print(x)` stamperà 5.

```
[7]: print("hai inserito " + input("scegli nome della via"))
```

hai inserito Via accademia, 26

Stessi concetti del precedente se si vuole esercitarsi riempire li spazi:

1. `print()`

.....

2. `input()`

.....

```
[8]: nome = input("Inserisci il tuo nome: ")
for contatore in range(3):
    print("Ciao", nome, "!")
```

Ciao Gabriele !

Ciao Gabriele !

Ciao Gabriele !

nuovi comandi usati:

1. `for`

usato per iterare, **deve** essere seguito da una variabile ed a un `in`

2. `range`

Il comando “range” in Python è usato per generare una sequenza di numeri, partendo da un numero di inizio fino a un numero precedente a quello finale specificato, con incrementi fissi, se necessario.

Non ricordi qualche comando? [Clicca qui](#) per la lista completa

### 3 dove le cose si fanno interessanti

#### Le operazioni

```
[9]: numero1 = int(input("Inserisci il primo numero: "))
numero2 = int(input("Inserisci il secondo numero: "))
print("La somma è:", int(numero1 + numero2))
```

La somma è: 44

nuovi comandi usati:

1. `int()`

trasforma la variabile in numero intero(se possibile, altrimenti restituisce errore)

Non ricordi qualche comando? [Clicca qui](#) per la lista completa

### gli operatori aritmetici:

I operatori aritmetici in Python sono simboli utilizzati per eseguire operazioni matematiche :

1. Addizione (+): Somma due numeri.
2. Sottrazione (-): Sottrae il secondo numero dal primo.
3. Moltiplicazione (\*): Moltiplica due numeri.
4. Divisione (/): Divide il primo numero per il secondo (risultato come float).
5. Divisione intera (//): Divide il primo numero per il secondo, restituendo solo la parte intera del risultato.
6. Resto della divisione (%): Restituisce il resto della divisione tra due numeri.
7. Esponenziale (\*\*): Eleva un numero alla potenza di un altro.

### qualche esempio:

```
[10]: nEsempio1, nEsempio2 = 10,3
      #notare: questo metodo per dichiarare più variabili
      # è molto utile per pulire il codice e si può fare
      #con diversi tipi di dato

      print("Addizione (+):", 10 + 3)
      print("Sottrazione (-):", 10 - 3)
      print("Moltiplicazione (*):", 10 * 3)
      print("Divisione (/):", 10 / 3)
      print("Divisione intera (//):", 10 // 3)
      print("Resto della divisione (%):", 10 % 3)
      print("Esponenziale (**):", 10 ** 3)
```

```
Addizione (+): 13
Sottrazione (-): 7
Moltiplicazione (*): 30
Divisione (/): 3.3333333333333335
Divisione intera (//): 3
Resto della divisione (%): 1
Esponenziale (**): 1000
```

## 4 Calcolatrice Python

```
[11]: # Calcolatrice Python con decisioni

operazione = input("Inserisci l'operazione (+, -, *, /): ")

numero1 = float(input("Inserisci il primo numero: "))
```

```

numero2 = float(input("Inserisci il secondo numero: "))

if operazione == "+":
    risultato = numero1 + numero2
elif operazione == "-":
    risultato = numero1 - numero2
elif operazione == "*":
    risultato = numero1 * numero2
elif operazione == "/":
    risultato = numero1 / numero2
else:
    risultato = "Operazione non valida"

print("Il risultato è:", risultato)

```

Il risultato è: 43.0

nuovi comandi utilizzati:

1. if

verifica se la condizione che viene posta dopo è corretta, se lo è esegue il codice

2. else

“altrimenti” il codice da eseguire qual’ora la condizione non fosse vera

Non ricordi qualche comando? [Clicca qui](#) per la lista completa

gli operatori di confronto

- > : Maggiore di
  - Esempio: 5 > 3 restituisce True.
- < : Minore di
  - Esempio: 3 < 5 restituisce True.
- == : Uguale a
  - Esempio: 5 == 5 restituisce True.
- >= : Maggiore o uguale a
  - Esempio: 5 >= 5 restituisce True.
- <= : Minore o uguale a
  - Esempio: 3 <= 5 restituisce True.
- != : Diverso da
  - Esempio: 5 != 3 restituisce True.

**Contare fino a N**

```

[12]: n = int(input("Inserisci un numero intero positivo:"))

for x in range(1,n+1):
    print(x)

```

2  
3

concetti precedentemente usati se si vuole esercitarsi:

1. for  
.....
2. int()  
.....
3. print()  
.....
4. range()  
.....

Non ricordi qualche comando? [Clicca qui per la lista completa](#)

## 5 Somma dei primi N numeri

```
[13]: n = int(input("Inserisci un numero intero positivo: "))  
      somma = 0  
  
      for x in range(1, n+1):  
          somma += n  
      print("La somma dei primi", n, "numeri interi è:", somma)
```

La somma dei primi 6 numeri interi è: 36

concetti precedentemente usati se si vuole esercitarsi:

1. for  
.....
2. int()  
.....
3. print()  
.....
4. range()  
.....

Non ricordi qualche comando? [Clicca qui per la lista completa](#)

## 6 Calcolare il Quadrato dei N Primi Numeri

```
[14]: n = int(input("Inserisci un numero intero positivo: "))

print("Quadrati dei primi", n, "numeri:")

for x in range (1, n+1):
    quadrato = x ** 2
    print("Il quadrato di", x, "è", quadrato)
```

Quadrati dei primi 7 numeri:

Il quadrato di 1 è 1

Il quadrato di 2 è 4

Il quadrato di 3 è 9

Il quadrato di 4 è 16

Il quadrato di 5 è 25

Il quadrato di 6 è 36

Il quadrato di 7 è 49

concetti precedentemente usati se si vuole esercitarsi:

1. for

.....

2. int()

.....

3. print()

.....

4. range()

.....

Non ricordi qualche comando? [Clicca qui per la lista completa](#)

## 7 Parità...

```
[15]: numero = int(input("Inserisci un numero: "))

if numero % 2 == 0:
    print(numero, "é un numero pari.")
else:
    print(numero, "è un numero dispari.")
```

3 è un numero dispari.

concetti precedentemente usati se si vuole esercitarsi:

1. `int()`  
.....
2. `print()`  
.....

## 8 ...Fattoriali e...

```
[16]: numeri = [] #Lista vuota di nome numeri

n = int(input("Quanti numeri vuoi inserire= ")) #Chiede quanti numeri voglio
    ↪ inserire dentro la lista vuota

#chiede un numero per quanto hai detto alla prima domanda
for i in range(n):
    numero = float(input("Inserisci un numero: "))
    numeri.append(numero) #mette il numero inserito dopo quello precedente

media = sum(numeri) / len(numeri) #Somma numeri dentro la lista e divide per
    ↪ lunghezza di quanti elementi ci sono

print("La media dei numeri inseriti (", numeri, ") è:", media, numeri)
```

La media dei numeri inseriti ( [5.0, 4.0, 4.0, 4.0] ) è: 4.25 [5.0, 4.0, 4.0, 4.0]

**Noti niente di nuovo?** Esatto! Una []!

quella è una lista, le liste in Python sono come elenchi di cose che puoi mettere insieme, come una lista della spesa. Puoi mettere dentro le liste numeri, parole o qualsiasi altra cosa, e poi accedervi per leggerle o modificarle come vuoi.

fun fact: le stringhe contano come liste di lettere

Comandi nuovi:

1. `sum()`  
somma tutti i numeri di una lista
2. `len()`  
ti dà la lunghezza di una lista (o stringa) come valore
3. `append()`  
metodo solo delle liste, per aggiungere un elemento in fondo
4. `float()`  
Non ricordi qualche comando? [Clicca qui](#) per la lista completa

## 9 come calcolare il fattoriale

```
[17]: n = int(input("Inserisci un numero intero: "))

fattoriale = 1

if n<0:
    print("Numero Negativo")
elif n ==0:
    print("Il numero di zero è un 1 per definizione")
else:
    for x in range(1, n+1):
        fattoriale*=n
print(f"Il fattoriale di {n} è {fattoriale}")
```

Il fattoriale di 3 è 27

## 10 Gioco dell'Indovinello

```
[18]: import random #importa la libreria del random

numero_da_indovinare =random.randint(1, 100) #spara un numero da 1 a 100
tentativi = 0

#chiede un numero
while True:
    tentativo = int(input("Indovina il numero (1-100): "))
    tentativi += 1

    if tentativo == numero_da_indovinare:
        print("Bravo! Hai indovinato il numero", numero_da_indovinare, "in", □
↪tentativi, "tentativi")
        break
    elif tentativo < numero_da_indovinare:
        print("Il numero è più grande")
    else:
        print("Il numero è più piccolo")
```

Il numero è più grande  
Il numero è più grande  
Il numero è più grande  
Il numero è più grande  
Il numero è più grande  
Il numero è più grande  
Il numero è più grande  
Il numero è più piccolo  
Il numero è più grande



Il numero è più grande  
Il numero è più piccolo  
Bravo! Hai indovinato il numero 98 in 12 tentativi

Import?

Le librerie in Python sono come scatole di strumenti speciali che aggiungono funzionalità extra al linguaggio base. Queste librerie contengono codice già scritto per svolgere compiti comuni, come lavorare con dati, creare grafici, gestire il tempo, e molto altro ancora.

Random è una libreria che ci permette di generare numeri casuali

## 11 MORRA CINESE

```
[19]: import random

mosse = ["carta", "forbice", "sasso"]

computer_mossa = random.choice(mosse)

print("Benvenuti al gioco del Morra Cinese!")
scelta_giocatore = input("Scegli la tua mossa (Carta, forbici, sasso): ")

if scelta_giocatore not in mosse:
    print("Mossa non permessa")
else:
    print("Il computer ha scelto:", computer_mossa)
    if scelta_giocatore == computer_mossa:
        print("Pareggio!")
    elif (scelta_giocatore == "carta" and computer_mossa == "sasso") or \
         (scelta_giocatore == "forbici" and computer_mossa == "carta") or \
         (scelta_giocatore == "sasso" and computer_mossa == "forbici"):
        print("Hai Vinto!")
    else:
        print("Hai Perso!")
```

Benvenuti al gioco del Morra Cinese!

Mossa non permessa

Niente di nuovo qui, per questo esercizio, prova a scrivere cosa pensi ogni funzione faccia, poi confrontale con i risultati qui sotto:

Non ricordi qualche comando? [Clicca qui per la lista completa](#)

## 12 Calcoli più interessanti

### 12.0.1 INDOVINA IL NUMERO DI DADO CASUALE (1 a 6)

```
[20]: import random

# Genera un numero casuale da 1 a 6(simulando un lancio di un dado)
numero_dado = random.randint(1, 6)

# Chiedi all'utente d'indovinare il numero
indovina = int(input("Indovina il numero del dado (da 1 a 6): "))

# Verifica se l'utente ha indovinato correttamente
if indovina < 1 or indovina < 0:
    print("numero non ammesso")
elif indovina == numero_dado:
    print(f"Complimenti il numero del dado era {numero_dado}. Hai indovinato!")
else:
    print(f"Mi dispiace, il numero del dado era {numero_dado}. Meglio fortuna_
    ↪la prossima volta!")
```

Complimenti il numero del dado era 4. Hai indovinato!

Nuovi comandi utilizzati:

1. random.\*

quando si importa una libreria, le funzione in essa possono essere utilizzate semplicemente scrivendo il nome della libreria, mettere un punto e il nome della funzione

2. random.randint()

sceglie un integer casuale nel range immesso

Non ricordi qualche comando? Clicca qui per la lista completa

### 12.0.2 SIMULAZIONE POPOLAZIONE

```
[21]: # Inizializza la popolazione e gli anni
popolazione = int(input("inserisci popolazione iniziale: "))
anni = int(input("Inserisci il numero di anni da simulare: "))
# Tasso di natalità e tasso di mortalità (% annuale)
tasso_natalità = float(input("Inserisci tasso natalità: "))
tasso_mortalità = float(input("Inserisci tasso mortalità: "))

# Simulazione della crescita della popolazione
for anno in range(anni):
    nascite = (popolazione * tasso_natalità) / 100
    morti = (popolazione * tasso_mortalità) / 100
    popolazione += (nascite - morti)
```

```
print(f"Anno {anno}, Popolazione {int(popolazione)}")  
  
print("Simulazione completata")
```

```
Anno 0, Popolazione 110  
Anno 1, Popolazione 110  
Anno 2, Popolazione 110  
Anno 3, Popolazione 110  
Anno 4, Popolazione 110  
Anno 5, Popolazione 110  
Anno 6, Popolazione 110  
Anno 7, Popolazione 110  
Anno 8, Popolazione 110  
Anno 9, Popolazione 110  
Anno 10, Popolazione 110  
Anno 11, Popolazione 110  
Anno 12, Popolazione 110  
Anno 13, Popolazione 110  
Anno 14, Popolazione 110  
Anno 15, Popolazione 110  
Anno 16, Popolazione 110  
Anno 17, Popolazione 110  
Anno 18, Popolazione 110  
Anno 19, Popolazione 110  
Anno 20, Popolazione 110  
Anno 21, Popolazione 110  
Anno 22, Popolazione 110  
Anno 23, Popolazione 110  
Anno 24, Popolazione 110  
Anno 25, Popolazione 110  
Anno 26, Popolazione 110  
Anno 27, Popolazione 110  
Anno 28, Popolazione 110  
Anno 29, Popolazione 110  
Anno 30, Popolazione 110  
Anno 31, Popolazione 110  
Anno 32, Popolazione 110  
Anno 33, Popolazione 110  
Anno 34, Popolazione 110  
Anno 35, Popolazione 110  
Anno 36, Popolazione 110  
Anno 37, Popolazione 110  
Anno 38, Popolazione 110  
Anno 39, Popolazione 110  
Anno 40, Popolazione 110  
Anno 41, Popolazione 110
```

Anno 42, Popolazione 110  
Anno 43, Popolazione 110  
Anno 44, Popolazione 110  
Anno 45, Popolazione 110  
Anno 46, Popolazione 110  
Anno 47, Popolazione 110  
Anno 48, Popolazione 110  
Anno 49, Popolazione 110  
Anno 50, Popolazione 110  
Anno 51, Popolazione 110  
Anno 52, Popolazione 110  
Anno 53, Popolazione 110  
Anno 54, Popolazione 110  
Anno 55, Popolazione 110  
Anno 56, Popolazione 110  
Anno 57, Popolazione 110  
Anno 58, Popolazione 110  
Anno 59, Popolazione 110  
Anno 60, Popolazione 110  
Anno 61, Popolazione 110  
Anno 62, Popolazione 110  
Anno 63, Popolazione 110  
Anno 64, Popolazione 110  
Anno 65, Popolazione 110  
Anno 66, Popolazione 110  
Anno 67, Popolazione 110  
Anno 68, Popolazione 110  
Anno 69, Popolazione 110  
Anno 70, Popolazione 110  
Anno 71, Popolazione 110  
Anno 72, Popolazione 110  
Anno 73, Popolazione 110  
Anno 74, Popolazione 110  
Anno 75, Popolazione 110  
Anno 76, Popolazione 110  
Anno 77, Popolazione 110  
Anno 78, Popolazione 110  
Anno 79, Popolazione 110  
Anno 80, Popolazione 110  
Anno 81, Popolazione 110  
Anno 82, Popolazione 110  
Anno 83, Popolazione 110  
Anno 84, Popolazione 110  
Anno 85, Popolazione 110  
Anno 86, Popolazione 110  
Anno 87, Popolazione 110  
Anno 88, Popolazione 110  
Anno 89, Popolazione 110

```
Anno 90, Popolazione 110
Anno 91, Popolazione 110
Anno 92, Popolazione 110
Anno 93, Popolazione 110
Anno 94, Popolazione 110
Anno 95, Popolazione 110
Anno 96, Popolazione 110
Anno 97, Popolazione 110
Anno 98, Popolazione 110
Anno 99, Popolazione 110
Simulazione completata
```

Niente di nuovo qui, questo semplicemente calcola la crescita della popolazione data la percentuale di nascite e mortalità (supponendo che rimanga sempre uguale)

## 13 Ma nella vita vera?

### 13.1 dire giorno ed ora con datetime

```
[22]: import datetime

today = datetime.datetime.today()
print(f"oggi è il giorno:{today: %d %m %y}, ore:{today: %H %M %S}")
```

oggi è il giorno: 09 02 24, ore: 12 13 00

Nuove librerie:

1. datetime():

La libreria datetime in Python è usata per lavorare con date e orari. Permette di creare, manipolare e formattare date e orari in modo semplice e efficace.

Nuovi comandi utilizzati:

1. datetime.datetime():

serve per rappresentare date e orari in modo combinato.

Non ricordi qualche comando? [Clicca qui](#) per la lista completa

cos'è %d %m %y? Non è altro che la forma in cui gli sviluppatori hanno deciso di far scrivere la data e l'ora nella libreria

#### 13.1.1 Bisogno di aiuto in fisica?

```
[23]: print("Benvenuto nel convertitore di misura!")
scelta = input("Cosa desideri convertire? (meters/foot/kilos/lbs): ").lower()

if scelta == "meters":
    valore = float(input("Inserisci il valore in meters: "))
    risultato = valore * 3.28034
```

```

    print(f"{valore} meters corrispondono = {risultato} foot.")

elif scelta == "foot":
    valore = float(input("Inserisci il valore in foot: "))
    risultato = valore * 0.3048
    print(f"{valore} foot corrispondono = {risultato} meters.")

elif scelta == "chilogrammi":
    valore = float(input("Inserisci il valore in chilogrammi: "))
    risultato = valore * 2.2046
    print(f"{valore} chilogrammi corrispondono = {risultato} lbs.")

elif scelta == "lbs":
    valore = float(input("Inserisci il valore in lbs: "))
    risultato = valore * 3.28034
    print(f"{valore} lbs corrispondono = {risultato} chilogrammi.")

```

Benvenuto nel convertitore di misura!

10.0 meters corrispondono = 32.80339999999996 foot.

Nuovi comandi usati:

1. lower()

trasforma tutto in minuscolo

f prima delle stringhe?

Le f-strings, o stringhe formattate, in Python consentono di inserire facilmente variabili all'interno di una stringa.

## 13.2 FIBONACCI

```

[24]: # Chiedere all'utente d'inserire un numero n
n = int(input("Inserisci un numero n per calcolare l'n-esimo numero di Fibonacci: "))

#inizializzare le variabili per i primi due numeri di fibonacci
a=0
b=1
c=1

# Calcolare l'n-esimo numero di fibonacci
if n <= 0:
    print("il numero deve essere maggiore di 0")
elif n == 1:
    risultato = a
else:
    for iterazione in range(n-3):
        a = b
        b = c

```

```

        c = a + b
    risultato = c
    # Stampare l'n-esimo numero di fibonacci
    print("l'n-esimo numero di fibonacci è:", risultato)

```

l'n-esimo numero di fibonacci è: 144

concetti precedentemente usati se si vuole esercitarsi:

1. `int()`  
.....
2. `print()`  
.....
3. `range()`  
.....
4. `input()`  
.....

Non ricordi qualche comando? [Clicca qui](#) per la lista completa

### 13.2.1 Non ti sembra che il codice sia un po' troppo poco comprensibile?

qui entrano in gioco le funzioni custom, per pulire il codice, si dichiarano con `def < nome >(argomenti)`

## 14 Le Funzioni Custom

```

[25]: def fibonacci(n):
        fib_series = [0, 1]

        while len(fib_series) < n:
            fib_series.append(fib_series[-1] + fib_series[-2])

        return fib_series

```

qui definiamo la funzione `fibonacci()`

```

[26]: fibonacci(9)

```

```

[26]: [0, 1, 1, 2, 3, 5, 8, 13, 21]

```

Così da poterla usare in una sola linea

## 15 Ancora matematica?

```
[27]: import math

def calcola_area_cerchio(raggio):
    return math.pi * (raggio ** 2)

def calcola_area Rettangolo(base, altezza):
    return base * altezza

def calcola_area triangolo(base, altezza):
    return (base * altezza) / 2

print("Benvenuto nella calcolatrice di aree")

scelta = input("Vuoi calcolare l'area di un cerchio (c), rettangolo (r),  
↳ triangolo (t): ").lower()

if scelta == 'c':
    raggio = float(input("Inserisci il raggio del cerchio: "))
    area = calcola_area_cerchio(raggio)
    print(f"L'area del cerchio è {area: .2f}")
elif scelta == 'r':
    base = float(input("Inserisci la base del rettangolo: "))
    altezza = float(input("Inserisci l'altezza del rettangolo: "))
    area = calcola_area Rettangolo(base, altezza)
    print(f"L'area del rettangolo è {area: .2f}")
elif scelta == 't':
    base = float(input("inserisci la base del triangolo: "))
    altezza = float(input("Inserisci l'altezza del triangolo: "))
    area = calcola_area triangolo(base, altezza)
    print(f"l'area del triangolo è {area: -2f}") # {area: -2f} significa di  
↳ stampare il valore AREA ma massimo 2 numeri dopo virgola
else:
    print("Scelta non valida, riprova")
```

Benvenuto nella calcolatrice di aree

L'area del cerchio è 50.27

Nuove librerie:

1. Math

La libreria math in Python fornisce funzioni matematiche predefinite per eseguire operazioni complesse su numeri.

Nuovi comandi:

1. Math.pi()



ritorna come valore il pi greco

## 16 calcolo interessi

```
[28]: def calcola_interessi(importo_iniziale, tasso_interesse, periodi_investimento):  
        importo_finale = importo_iniziale * (1 + tasso_interesse / 100) **  
        ↪periodi_investimento  
        return importo_finale
```

Definiamo la funzione, in modo da pulire il codice

```
[29]: print("Benvenuto nel Calcolatore d'interessi!")  
  
importo = float(input("Inserisci l'importo iniziale: "))  
tasso = float(input("Inserisci il tasso d'interesse annuale (%): "))  
periodo = int(input("Inserisci il periodo d'investimento (anni): "))  
  
importo_finale = calcola_interessi(importo, tasso, periodo)  
  
print(f"l'importo finale dopo {periodo} anni è di {importo_finale:.2f} euro.")
```

Benvenuto nel Calcolatore d'interessi!

l'importo finale dopo 10 anni è di 33.95 euro.

concetti precedentemente usati se si vuole esercitarsi:

1. float()

.....

2. print()

.....

Non ricordi qualche comando? [Clicca qui per la lista completa](#)

## 17 Ancora fisica ?!

### 17.1 forza gravitazionale dei pianeti

```
[30]: def forza_gravitazionale(m1, m2, r):  
        #costante gravitazionale  
        G = 6.67430e-11 #N(m/gk) ~2  
  
        #Calcola dela forza gravitazionale  
        F = (G * m1 * m2) / (r ** 2)  
  
        return F
```

Abbiamo definito la funzione

```
[31]: #esempio di utilizzo
massa_terra = 5.972e24 #kg
massa_luna = 7.342e22 #kg
distanza_terra_luna = 384400000 #metri

forza = forza_gravitazionale(massa_terra, massa_luna, distanza_terra_luna)
print(f"forza gravitazionale tra la terra e la luna: {forza} Newton")
```

forza gravitazionale tra la terra e la luna: 1.9804922390990566e+20 Newton

Niente di nuovo qui

Il programma calcola la forza gravitazionale

## 18 Non solo matematica

### 18.0.1 anagrammi

```
[32]: from itertools import permutations
k=0

def trova_anagrammi(parola):
    anagrammi = ["".join(p) for p in permutations(parola)]
    #unisce elementi in permutations di parola
    #for p in permutations(parola):
    #    anagrammi = ["".join(p)]
    return anagrammi

print("Benvenuto nel risolutore di anagrammi!")

parola_input = input("Inserisci una parola: ").strip().lower()

if len(parola_input) < 2:
    print("Inserisci una parola con almeno 2 caratteri: ")
else:
    anagrammi = trova_anagrammi(parola_input)

    for anagramma in anagrammi:
        if anagramma != parola_input:
            k+=1
            print(anagramma)
    print(f"gli anagrammi di '{parola_input}' sono: '{k}'")
```

Benvenuto nel risolutore di anagrammi!

gli anagrammi di 'casa' sono: '0'

caas

gli anagrammi di 'casa' sono: '1'

csaa

gli anagrammi di 'casa' sono: '2'  
csaa  
gli anagrammi di 'casa' sono: '3'  
caas  
gli anagrammi di 'casa' sono: '4'  
gli anagrammi di 'casa' sono: '4'  
acsa  
gli anagrammi di 'casa' sono: '5'  
acas  
gli anagrammi di 'casa' sono: '6'  
asca  
gli anagrammi di 'casa' sono: '7'  
asac  
gli anagrammi di 'casa' sono: '8'  
aacs  
gli anagrammi di 'casa' sono: '9'  
aasc  
gli anagrammi di 'casa' sono: '10'  
scaa  
gli anagrammi di 'casa' sono: '11'  
scaa  
gli anagrammi di 'casa' sono: '12'  
saca  
gli anagrammi di 'casa' sono: '13'  
saac  
gli anagrammi di 'casa' sono: '14'  
saca  
gli anagrammi di 'casa' sono: '15'  
saac  
gli anagrammi di 'casa' sono: '16'  
acas  
gli anagrammi di 'casa' sono: '17'  
acsa  
gli anagrammi di 'casa' sono: '18'  
aacs  
gli anagrammi di 'casa' sono: '19'  
aasc  
gli anagrammi di 'casa' sono: '20'  
asca  
gli anagrammi di 'casa' sono: '21'  
asac  
gli anagrammi di 'casa' sono: '22'

Nuove librerie:

1. itertools()

La libreria itertools in Python fornisce strumenti per creare, combinare e manipolare iterabili in modo efficiente, permettendo di generare combinazioni, permutazioni, e altri tipi di iterabili.

Nuovi comandi:

1. `.join()`

utilizzato nelle liste per unirne due o più

2. `permutations()`

Python genera tutte le possibili permutazioni degli elementi di un iterabile. Ad esempio, se hai una lista `[1, 2, 3]`, `permutations` genererà tutte le possibili combinazioni di lunghezza 1, 2 e 3 degli elementi `[1, 2, 3]`.

## 18.0.2 Ehy amico, hai quale che Yen per la benzina?

```
[33]: # Definizione dei tassi di cambio

tassi_di_cambio= {
    "dollari": 1.0,
    "euro": 0.85,
    "yen": 110.41,
    # aggiungi altre valute e tassi di cambio se necessario
}

# Chiedi all'utente d'inserire l'importo, la valuta di partenza e la valuta di
↳ destinazione
importo = float(input("Inserisci l'importo da convertire: "))
valuta_di_partenza = input("Inserisci la valuta di partenza: ").lower()
valuta_destinazione = input("Inserisci la valuta di destinazione: ").lower()

#Verifica se le valute sono nel dizionari dei tassi di cambio
if valuta_di_partenza in tassi_di_cambio and valuta_destinazione in
↳ tassi_di_cambio:
    #Calcola il tasso di cambio e l'importo converitto
    tasso_di_cambio = tassi_di_cambio[valuta_destinazione] /
↳ tassi_di_cambio[valuta_di_partenza]
    importo_convertito = importo * tasso_di_cambio

#stampa il risultato
    print(f"{importo} {valuta_di_partenza} sono equivalenti a
↳ {importo_convertito:.2f} {valuta_destinazione}")
else:
    print("Valute non supportate. Assicurati di inserire valute valide.")
```

12.0 yen sono equivalenti a 0.11 dollari

liste con le graffe?

I dizionari in Python sono strutture dati che memorizzano coppie di chiave-valore. Ogni valore è associato a una chiave univoca.

```
[34]: tassi_di_cambio["euro"]
```

```
[34]: 0.85
```

## 19 Quanto è lunga una frase?

```
[35]: # Chiedi all'utente di inserire una frase
frase = input("Inserisci una frase: ")

# Converti la frase in minuscolo per evitare problemi di maiuscole/minuscole
frase = frase.lower()

# Inizializza una lista di lettere dell'alfabeto
alfabeto = 'abcdefghijklmnopqrstuvwxyz'

# Inizializza un dizionario per tenere traccia del conteggio delle lettere
conteggio_lettere = {}

# Itera attraverso ciascuna lettera dell'alfabeto
for lettera in alfabeto:
    # Conta quante volte appare la lettera nell'alfabeto
    conteggio = frase.count(lettera)

    # Aggiungi la lettera e il conteggio al dizionario se la lettera appare
    # almeno una volta
    if conteggio > 0:
        conteggio_lettere[lettera] = conteggio

# stampa il conteggio delle lettere in un formato leggibile
for lettera, conteggio in conteggio_lettere.items():
    print(f"{lettera}: {conteggio}")
```

```
a: 1
c: 3
e: 1
i: 2
l: 3
o: 1
s: 1
u: 1
```

Anche in questo usiamo i dizionari

Nuovi comandi usati:

1. items()

usato per prendere sia chiave che valore di un dizionario singolarmente

Non ricordi qualche comando? [Clicca qui](#) per la lista completa

```
[36]: conteggio_lettere.items()
```

```
[36]: dict_items([('a', 1), ('c', 3), ('e', 1), ('i', 2), ('l', 3), ('o', 1), ('s', 1), ('u', 1)])
```

Mostriamo l'intero conteggio

```
[37]: prodotti = {}  
prodotti["pan bauletto"] = 2  
prodotti["coca cola"] = 3
```

Definiamo dei prodotti facendo una lista e inizializzandola, mostrando come aggiungere valori ad un dizionario

```
[38]: prodotti
```

```
[38]: {'pan bauletto': 2, 'coca cola': 3}
```

## 20 Che ore sono in...

```
[39]: from datetime import datetime  
import pytz  
  
print("Benvenuto nell'orologio mondiale")  
  
# Definisci le città e i relativi fusi orari  
citta_fusi_orari = {  
    "New York": "America/New_York",  
    "Londra" : "Europe/London",  
    "Tokyo" : "Asia/Tokyo",  
    "Sydney" : "Australia/Sidney",  
    "Rio de Janeiro" : "America/Sao_Paulo",  
}  
  
while True:  
    print("\nCittà disponibili:")  
    for citta in citta_fusi_orari.keys():  
        print(citta)  
  
    scelta_citta = input("Inserisci il nome della città per visualizzare l'ora_↵  
↵(o 'esci' per uscire): ").strip()  
    if scelta_citta.lower() == 'esci':  
        break  
    if scelta_citta in citta_fusi_orari.keys():  
        fuso_orario = pytz.timezone(citta_fusi_orari[scelta_citta])
```

```

    ora_corrente = datetime.now(fuso_orario)
    print(f"L'ora corrente a {scelta_citta} è: {ora_corrente.strftime('%H:
↪%M:%S')}")
    else:
        print("citta non valid. Riprova")

```

Benvenuto nell'orologio mondiale

Città disponibili:

New York

Londra

Tokyo

Sydney

Rio de Janeiro

citta non valid. Riprova

Città disponibili:

New York

Londra

Tokyo

Sydney

Rio de Janeiro

L'ora corrente a New York è: 06:14:20

Città disponibili:

New York

Londra

Tokyo

Sydney

Rio de Janeiro

Questo è il vero prima programma che troviamo. Andiamo a spezzarlo in più parti:

1. `from datetime import datetime`

importa la libreria datetime

2. `import pytz`

importa pytz, una libreria con tutti i fusi orari

3. `print("Benvenuto nell'orologio mondiale")`

linea puramente per l'utente, saluta e spiega cosa è il programma

4. `citta_fusi_orari = { "New York": "America/New_York", "Londra" : "Europe/London", "Tokyo" : "Asia/Tokyo", "Sydney" : "Australia/Sidney", "Rio de Janeiro" : "America/Sao_Paulo", }`

Un dizionario con le città e fusi orari che si vogliono, seguendo la lista di fusi che si trova nella documentazione della libreria pytz

```

5. while True:
    mentre Vero è Vero (Quidni lancia il codice per sempre)

6. print("\nCittà disponibili:")
    stampa le città da cui scegliere

7. for citta in citta_fusi_orari.keys(): print(citta)
    stampa la lista di citta in modo da far sapere quali sono disponibili

8. scelta_citta = input("Inserisci il nome della città per      visualizzare
    l'ora (o 'esci' per uscire): ").strip()
    chiede all'utente di scegliere un città o uscire,.strip() elimina gli spazi nell'input dato

9. if scelta_citta.lower() == 'esci': break
    permette di uscire dal programma

10. if scelta_citta in citta_fusi_orari.keys():
    se la scelta dell'utente è presente nelle chiavi

11. fuso_orario      =      pytz.timezone(citta_fusi_orari[scelta_citta])      ora_corrente
    =      datetime.now(fuso_orario)      print(f"L'ora      corrente      a      {scelta_citta}      è:
    {ora_corrente.strftime('%H:%M:%S')}")
    stampa l'ora corrente della città scelta

12. else: print("citta non valid. Riprova")
    se la scelta non è valida dice di riprovare

```

## 21 Ancora dizionari?!

```

[40]: # Funzione principale può avere qualsiasi nome
def paolo():
    print("Mi chiamo Paolo")

if __name__ == "__main__" : #è una condizione logica che "si verifica sempre" e
    ↳pertanto tutto ciò
                                #che risulta indentato a questa ocondizione viene
    ↳eseguita
    paolo()

```

Mi chiamo Paolo

La funzione `if __name__ == "__main__"`: è sempre vera perché `__name__` è una variabile di sistema che è NON modificabile se non volontariamente

```

[41]: #Funzione principale può avere qualsiasi nome
def main():

```



```
print("la funzione principale del codice è stata eseguita, in questa  
↳ funzione possono essere presenti funzioni s")
```

```
if __name__ == "__main__":  
    main()
```

la funzione principale del codice è stata eseguita, in questa funzione possono essere presenti funzioni s

Mostra che main funziona sempre

## 22 Quanto pesi?...

```
[42]: #main  
# funzione per il calcolo del BMI  
def calcola_bmi(peso, altezza):  
    return peso / (altezza ** 2)  
  
#funzione per la valutazione del BMI  
def valuta_bmi(bmi):  
    if bmi < 10.5:  
        return "Sottopeso"  
    elif 10.5 <= bmi < 24.9:  
        return "Normopeso"  
    elif 25 <= bmi < 29.9:  
        return "Sovrappeso"  
    else:  
        return "Obeso"  
  
# Funzione principale  
def main():  
    print("Benvenuto nella calcolatri bmi!")  
    peso = float(input("Inserisci il tuo peso in chilogrammi: "))  
    altezza = float(input("Inserisci la tua altezza in metri: "))  
  
    bmi = calcola_bmi(peso, altezza)  
    valutazione = valuta_bmi(bmi)  
  
    print(f"il tuo BMI è {bmi:.2f}, sei classificato come '{valutazione}'.")  
  
if __name__ == "__main__":  
    main()
```

Benvenuto nella calcolatri bmi!

il tuo BMI è 13.89, sei classificato come 'Normopeso.'

concetti precedentemente usati se si vuole esercitarsi:

1. float()  
.....
2. print()  
.....
3. input()  
.....

Non ricordi qualche comando? [Clicca qui](#) per la lista completa

## 23 ...e quante calorie mangi?

```
[43]: cibo_calorie = {
    "pizza": 285,
    "hamburger": 250,
    "insalata": 100,
    "panino del mc": 335,
    "yogurt": 150
}

#Funzione per calcolare le calorie consumate
def calorie_consumate(cibo, quantita):
    if cibo not in cibo_calorie.keys():
        print("Cibo non presente")
    elif cibo in cibo_calorie:
        calorie_per_100g = cibo_calorie[cibo]
        calorie_totali = (calorie_per_100g / 100) * quantita
        return calorie_totali
    else:
        return 0

#Funzione principale
def main():
    cibo_consumato = []

    while True:
        print("menù:")
        print("\n 1. aggiungi cibo consumato")
        print("\n 2. calcola calorie totali")
        print("\n 3 esci")

        scelta = input("scegli un opzione: ")

        if scelta == "1":
            print("\n")
            for key, value in cibo_calorie.items():
```

```

        print(key,value)

        cibo = input("inserisci il cibo consumato: ").lower()
        quantita = float(input("Inserisci la quantita in grammi: "))
        cibo_consumato.append((cibo,quantita))
        elif scelta == "2":
            calorie_totali = sum(calorie_consumate(c,q) for c, q in
↳cibo_consumato)
            print(f"\ncalore totali consumate: {calorie_totali} calorie")
        elif scelta == "3":
            print("ok")
            break
        else:
            print("\nscelta non valida. riprova")
if __name__ == "__main__":
    main()

```

menù:

1. aggiungi cibo consmato
2. calcola calorie totali
- 3 esci

pizza 285  
hamburger 250  
insalata 100  
panino del mc 335  
yogurt 150  
menù:

1. aggiungi cibo consmato
2. calcola calorie totali
- 3 esci

Cibo non presente

```

-----
TypeError                                Traceback (most recent call last)
Cell In[43], line 49
      47         print("\nscelta non valida. riprova")
      48 if __name__ == "__main__":
----> 49     main()

```

```

Cell In[43], line 41, in main()
    39     cibo_consumato.append((cibo,quantita))
    40 elif scelta == "2":
--> 41     calorie_totali =
    ↪sum(calorie_consumate(c,q) for c, q in cibo_consumato)
    42     print(f"\ncalore totali consumate: {calorie_totali} calorie")
    43 elif scelta == "3":

TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'

```

questo sembra complesso ma è molto semplice, inizializza un dizionario con diversi cibi, definisce una funzione che calcola le calorie consumate e successivamente chiede all'utente di inserire quest'ultimi.

Dopodiché mostra le chilocalorie totali

## 24 Sei uno scrittore? Allora questa è per te

### 24.0.1 Personaggi satorie e romanzi

```

[44]: import random

# Liste di speci, classi, armi e abilità
speci = ["Elfo", "Umano", "Nano", "Orco", "Gnomo"]
classi = ["Guerriero", "Mago", "Ranger", "Ladro", "Chierico"]
armi = ["Spada", "Arco", "Bacchetta magica", "Ascia", "Daga"]
abilità = ["Furtività", "Magia dell'acqua", "Camuffamento", "Estrazione",
    ↪"mineraria", "Incantesimi di guarigione"]

#Gebera un personaggio casuale
specie = random.choice(speci)
classe = random.choice(classi)
arma = random.choice(armi)
abilità_scelte = random.sample(abilità, random.randint(1,3))

#stampa il personaggio generato
print(f"Personaggio Fantasy Generato: ")
print(f"Specie: {specie}")
print(f"Classe: {classe}")
print(f"Arma: {arma}")
print(f"Abilità: {' '.join(abilità_scelte)}")

```

```

Personaggio Fantasy Generato:
Specie: Gnomo
Classe: Ladro
Arma: Ascia
Abilità: Furtività

```

```
[45]: import random
# Liste di speci, classi, armi e abilità
speci = ["Elfo", "Umano", "Nano", "Orco", "Gnomo"]
classi = ["Guerriero", "Mago", "Ranger", "Ladro", "Chierico"]
armi = ["Spada", "Arco", "Bacchetta magica", "Ascia", "Daga"]
abilità = ["Furtività", "Magia dell'acqua", "Camuffamento", "Estrazione_
↳ mineraria", "Incantesimi di guarigione"]

# funzione per creare un personaggio casuale
def crea_personaggio():
    return {
        "Specie": random.choice(speci),
        "Classe": random.choice(classi),
        "Arma": random.choice(armi),
        "Abilità": random.sample(abilità, random.randint(1,3))
    }

# funzione principale
def main():
    personaggio_generato = crea_personaggio()

    print("Personaggi Fantasy Generato:")
    for chiave, valore in personaggio_generato.items():
        if chiave == "Abilità":
            valore = ', '.join(valore)
        print(f"{chiave}: {valore}")

# eseguire la funzione main quando il programma viene eseguito
if __name__ == "__main__":
    main()
```

Personaggi Fantasy Generato:

Specie: UmanoNano

Classe: Mago

Arma: Spada

Abilità: Camuffamento, Furtività, Estrazione mineraria

```
[46]: import random
# Liste di speci, classi, armi e abilità
speci = ["Elfo", "Umano", "Nano", "Orco", "Gnomo"]
classi = ["Guerriero", "Mago", "Ranger", "Ladro", "Chierico"]
armi = ["Spada", "Arco", "Bacchetta magica", "Ascia", "Daga"]
abilità = ["Furtività", "Magia dell'acqua", "Camuffamento", "Estrazione_
↳ mineraria", "Incantesimi di guarigione"]

# funzione per creare un personaggio casuale
def crea_personaggio():
```

```

personaggio={
    "Specie": random.choice(speci),
    "Classe": random.choice(classi),
    "Arma": random.choice(armi),
    "Abilità": random.sample(abilità, random.randint(1,3))
}
return personaggio

#funzione principale
def main():
    personaggio_generato = crea_personaggio()

    print("Personaggi Fantasy Generato:")
    for chiave, valore in personaggio_generato.items():
        if chiave == "Abilità":
            valore = ', '.join(valore)
        print(f"{chiave}: {valore}")

#eseguire la funzione main quando il programma viene eseguito
if __name__ == "__main__":
    main()

```

Personaggi Fantasy Generato:

Specie: Gnomo

Classe: Chierico

Arma: Daga

Abilità: Furtività

Questi due codici generano personaggi; per esercizio, prova a descrivere il loro funzionamento, runnandoli e leggendo il codice

.....

## 25 La letteratura combinatoria

```

[47]: import random

physical_traits = ["capelli nero", "capelli biondi", "occhi cazzurri", "occhi_
↳verdi"]
personality_traits = ["gentile", "introverso", "estroverso", "ottimista"]
backgrounds = ["contadino", "nobile", "artigiano", "commerciale"]
motivations = ["vendetta", "ricchezza", "amore", "vita esterna"]

def genera_personaggio():
    nome = input("Inserisci il nome del personaggio: ")
    aspetto_fisico = random.choice(physical_traits)
    aspetto_personale = random.choice(personality_traits)
    sfondo = random.choice(backgrounds)

```

```

motivazione = random.choice(motivations)

descrizione =f"Nome: {nome}\nAspetto Fisico: {aspetto_fisico}\nAspetto_
↳Personale: {aspetto_personale}\nSfondo: {sfondo}\nMotivazione: {motivazione}"

return descrizione
print("Generatore di personaggi per romanzi")
print(genera_personaggio())

```

Generatore di personaggi per romanzi

Nome: Jacopo

Aspetto Fisico: capelli nero

Aspetto Personale: ottimista

Sfondo: artigiano

Motivazione: ricchezza

Nuovi Elementi:

1. random.choice

seleziona fra le opzioni date quelle più possibili

Non ricordi qualche comando? [Clicca qui per la lista completa](#)

## 26 citazione del giorno

```

[48]: #database di citazioni
citazioni = [
    "lao",
    "A",
    "B",
    "C",
]

#Funzione per generare una citazione casuale
def genera_citazione():
    return random.choice(citazioni)

#funzione principale
def main():
    print("Benvenuto nel generatore di citazioni")
    input("Premi invio per ottenere una citazione causale...")

    citazione = genera_citazione()
    print(f"citazione del giorno: {citazione}")

if __name__ == "__main__":
    main()

```

Benvenuto nel generatore di citazioni  
citazione del giorno: B

Nuovi Elementi:

1. random.choice

seleziona fra le opzioni date quelle più possibili

Non ricordi qualche comando? [Clicca qui per la lista completa](#)

## 27 GENERATORE FRASI DA INFLUENCER

```
[49]: import random
#lista di frammenti di citazioni famose (più brevi)
frammenti = [
    "ciao",
    "come va?",
    "dio",
    "grazioso",
    "madonna",
    "blabla",
    "lalalal",
    "pipipipip",
    "blululullulu"
]
#funzione per creare nuove citazioni rimenscolando i frammenti
def crea_citazione():
    num_frammenti = random.randint(5,7) #scegli un numero casuale di frammenti
    ↪da utilizzare
    citazione_rimescolata = random.sample(frammenti, num_frammenti)
    nuova_citazione = " ".join(citazione_rimescolata)
    return nuova_citazione

#genera una nuova nuova citazione
nuova_citazione = crea_citazione()
print("Nuova citazione generata:")
print(nuova_citazione)
```

Nuova citazione generata:

madonna blabla come va? lalalal blululullulu pipipipip grazioso

Randint ti dà un intero casuale fra quelli che metti nell'input

Non ricordi qualche comando? [Clicca qui per la lista completa](#)



## 27.1 Frasi da influencer

```
[50]: import random

frammenti = [
    "ciao",
    "come va?",
    "dio",
    "grazioso",
    "madonna",
    "blabla",
    "lalalal",
    "pipipipip",
    "blululullulu"
]
#funzione per crrare nuove citazioni mescolando i frammenti
def crea_citazione():
    num_frammenti = random.randint(4,7) #scegli un umero casuale di frammenti
    ↪ da utilizzare
    citazione_rimescolata = random.sample(frammenti, num_frammenti)
    nuova_citazione = " ".join(citazione_rimescolata)
    return nuova_citazione

#genera una nuova citazione
def main():
    nuova_citazione = crea_citazione()
    print("Nuova citazione generata:")
    print(nuova_citazione)

if __name__ == "__main__":
    main()
```

Nuova citazione generata:

come va? madonna blululullulu pipipipip blabla lalalal

## 27.2 Generatore di Poesie

```
[51]: import random

#liste di parole predefinite per la generazione dlela poesia
aggettivi = ["dolce", "serena", "profondo", "luminoso", "gentile"]
sostantivi = ["amore", "mare", "cielo", "vento", "sogno"]
verbi = ["danza", "splende", "abbraccia", "canta", "sorride"]

#genera una poesia casuale
def genera_poesia():
    verso1 = f"Il {random.choice(aggettivi)} {random.choice(sostantivi)}
    ↪ {random.choice(verbi)}."
```

```
verso2 = f"Il {random.choice(aggettivi)} {random.choice(sostantivi)}  
↳{random.choice(verbi)}."  
verso3 = f"Nel {random.choice(aggettivi)} {random.choice(sostantivi)}"
```

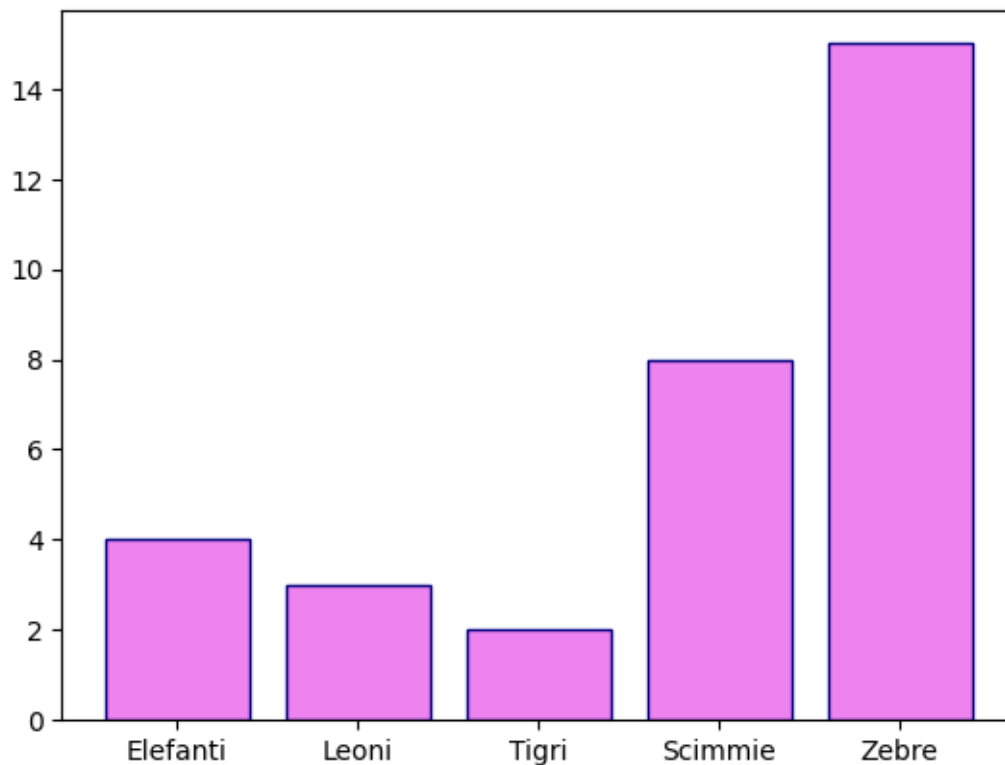
## 28 Ma... Le AI?

Ci arriveremo, ma prima abbiamo bisogno di saper manipolare i dati.

Come?

### 28.0.1 con i grafici

```
[52]: import matplotlib.pyplot as plt  
  
animali = ["Elefanti", "Leoni", "Tigri", "Scimmie", "Zebre"]  
numero_animali = [4, 3, 2, 8, 15]  
plt.bar(animali, numero_animali, color="violet", edgecolor="navy")  
plt.show()
```



nuove librerie:

1. matplotlib.pyplot

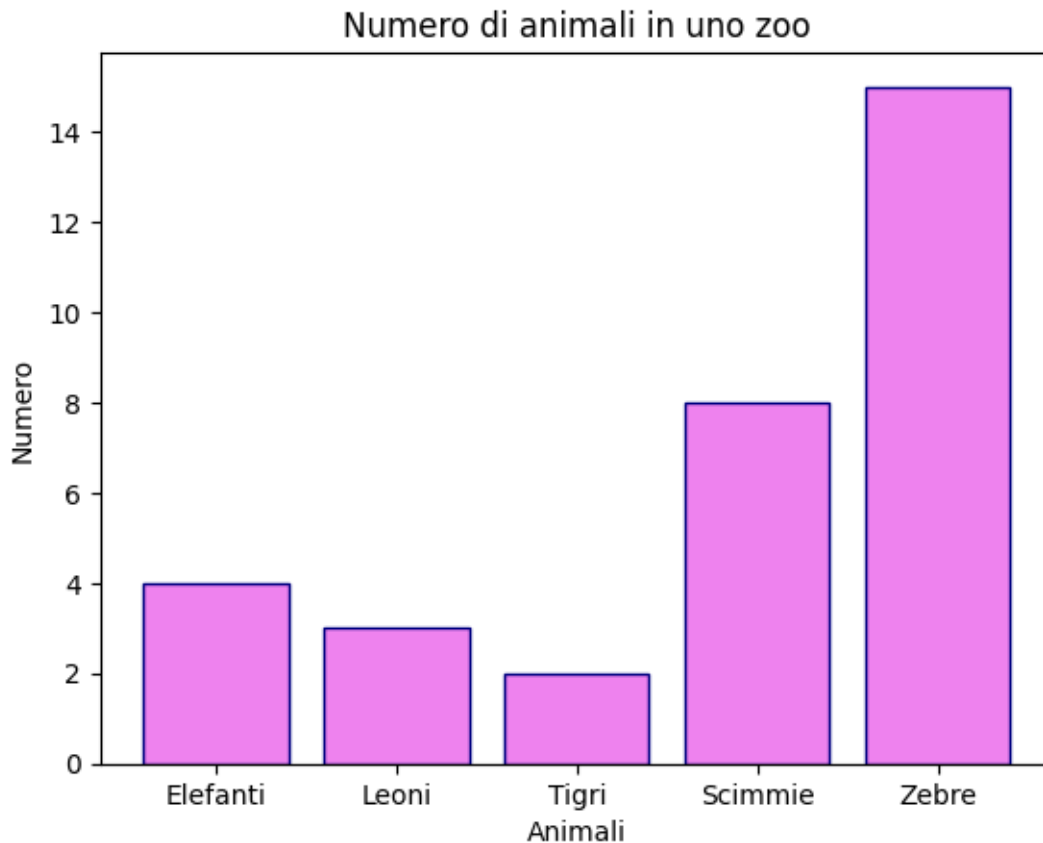
serve a fare grafici (plot)

1. `import matplotlib.pyplot as plt`: Importa la libreria `matplotlib.pyplot` con l'alias `plt`, che consente di accedere alle funzioni per la creazione di grafici.
2. `animali = ["Elefanti", "Leoni", "Tigri", "Scimmie", "Zebre"]`: Definisce una lista di stringhe contenenti i nomi degli animali.
3. `numero_animali = [4, 3, 2, 8, 15]`: Definisce una lista di numeri che rappresentano il numero di animali di ciascuna specie corrispondente alla lista degli animali.
4. `plt.bar(animali, numero_animali, color="violet", edgecolor="navy")`: Crea un grafico a barre utilizzando i dati forniti. Gli argomenti sono:
  - `animali`: La lista degli animali sull'asse x.
  - `numero_animali`: La lista dei numeri di animali sull'asse y.
  - `color="violet"`: Specifica il colore delle barre, in questo caso, violetto.
  - `edgecolor="navy"`: Specifica il colore del bordo delle barre, in questo caso, blu scuro.
5. `plt.show()`: Mostra il grafico a barre.

```
[53]: import matplotlib.pyplot as plt

animali = ["Elefanti", "Leoni", "Tigri", "Scimmie", "Zebre"]
numero_animali = [4, 3, 2, 8, 15]
plt.bar(animali, numero_animali, color="violet", edgecolor="navy")
plt.title('Numero di animali in uno zoo')
plt.xlabel('Animali')
plt.ylabel('Numero')

plt.show()
```



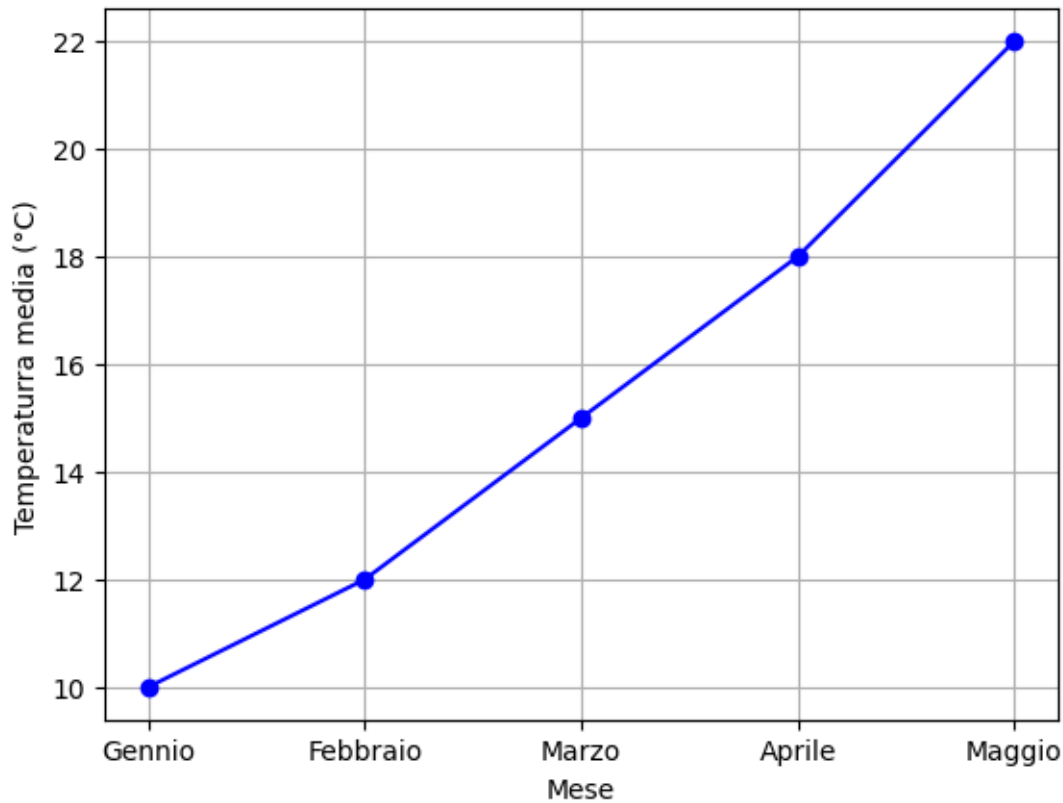
5. `plt.title('Numero di animali in uno zoo')`: Aggiunge un titolo al grafico.

6. `plt.xlabel('Animali')`: Aggiunge un'etichetta sull'asse x.

7. `plt.ylabel('Numero')`: Aggiunge un'etichetta sull'asse y.

```
[54]: import matplotlib.pyplot as plt

mese = ["Gennaio", "Febbraio", "Marzo", "Aprile", "Maggio"]
temperatura_media = [10, 12, 15, 18, 22]
plt.plot(mese, temperatura_media, marker='o', linestyle='-', color="blue")
plt.xlabel('Mese')
plt.ylabel('Temperatura media (°C)')
plt.grid(True)
plt.show()
```



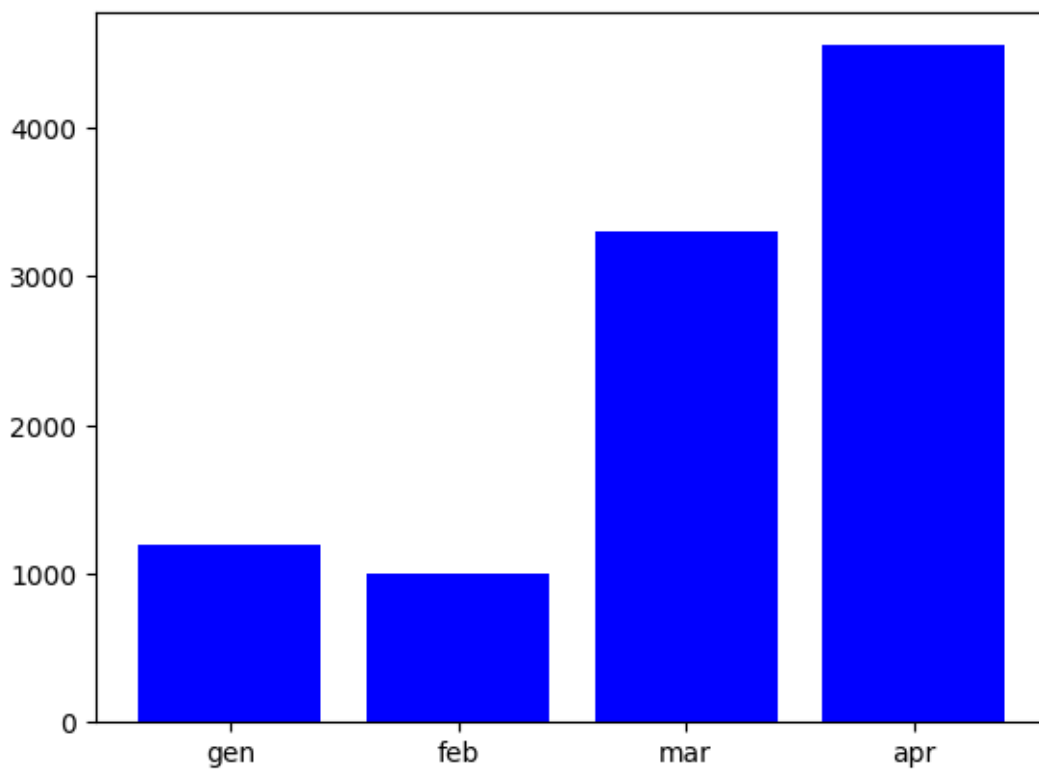
1. `import matplotlib.pyplot as plt`: Importa la libreria `matplotlib.pyplot` con l'alias `plt`, che consente di accedere alle funzioni per la creazione di grafici.
2. `mese = ["Gennaio", "Febbraio", "Marzo", "Aprile", "Maggio"]`: Definisce una lista di stringhe contenenti i nomi dei mesi.
3. `temperatura_media = [10, 12, 15, 18, 22]`: Definisce una lista di numeri che rappresentano le temperature medie mensili.
4. `plt.plot(mese, temperatura_media, marker='o', linestyle='-', color="blue")`: Crea un grafico a linee utilizzando i dati forniti. Gli argomenti sono:
  - `mese`: La lista dei mesi sull'asse x.
  - `temperatura_media`: La lista delle temperature medie sull'asse y.
  - `marker='o'`: Specifica il marker da utilizzare per i punti sul grafico, in questo caso, cerchi.
  - `linestyle='-'`: Specifica lo stile della linea, in questo caso, una linea continua.
  - `color="blue"`: Specifica il colore delle linee, in questo caso, blu.
5. `plt.xlabel('Mese')`: Aggiunge un'etichetta sull'asse x.
6. `plt.ylabel('Temperatura media (°C)')`: Aggiunge un'etichetta sull'asse y.
7. `plt.grid(True)`: Attiva la griglia di sfondo nel grafico.

8. `plt.show()`: Mostra il grafico a linee con tutte le personalizzazioni.

```
[55]: import matplotlib.pyplot as plt
```

```
vendite_mensili={  
    "gen":1200,  
    "feb":1000,  
    "mar":3300,  
    "apr":4555  
}
```

```
plt.bar(vendite_mensili.keys(), vendite_mensili.values(), color="blue")  
plt.show()
```

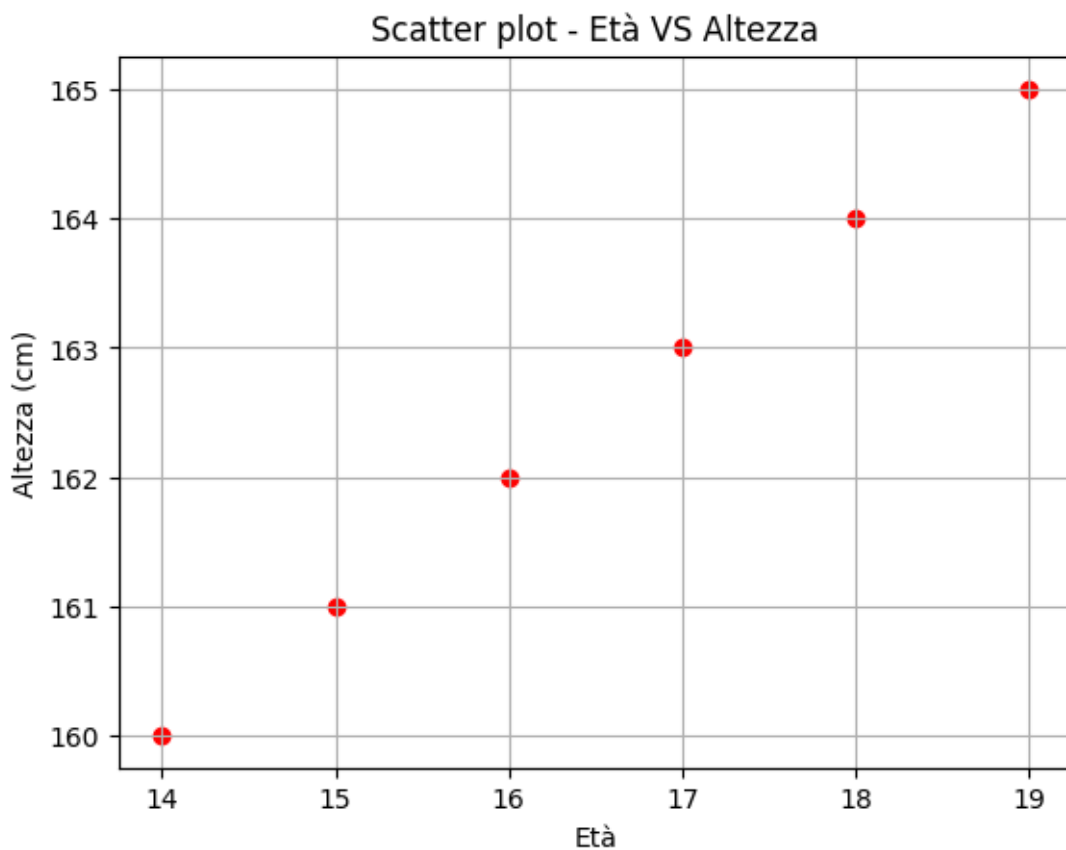


1. `import matplotlib.pyplot as plt`: Importa la libreria `matplotlib.pyplot` con l'alias `plt`, che consente di accedere alle funzioni per la creazione di grafici.
2. `vendite_mensili`: È un dizionario che contiene le vendite mensili, con le chiavi rappresentanti l'abbreviazione del mese e i valori rappresentanti il numero di vendite.
3. `plt.bar(vendite_mensili.keys(), vendite_mensili.values(), color="blue")`: Crea un grafico a barre utilizzando le chiavi del dizionario come etichette sull'asse x e i valori come altezze delle barre sull'asse y. L'argomento `color="blue"` specifica il colore delle barre come blu.

4. `plt.show()`: Mostra il grafico a barre.

```
[56]: import matplotlib.pyplot as plt
eta = [14, 15, 16, 17, 18, 19]
altezza = [160, 161, 162, 163, 164, 165]
plt.scatter(eta, altezza, color='red', marker='o')
plt.title('Scatter plot - Età VS Altezza')
plt.xlabel('Età')
plt.ylabel('Altezza (cm)')
plt.grid(True)
plt.show
```

```
[56]: <function matplotlib.pyplot.show(close=None, block=None)>
```



1. `import matplotlib.pyplot as plt`: Importa la libreria `matplotlib.pyplot` con l'alias `plt`, che consente di accedere alle funzioni per la creazione di grafici.
2. `eta = [14, 15, 16, 17, 18, 19]`: Definisce una lista di età.
3. `altezza = [160, 161, 162, 163, 164, 165]`: Definisce una lista di altezze.
4. `plt.scatter(eta, altezza, color='red', marker='o')`: Crea un grafico di dispersione

(scatter plot) utilizzando i dati forniti. Gli argomenti sono:

- `eta`: Lista delle età sull'asse x.
- `altezza`: Lista delle altezze sull'asse y.
- `color='red'`: Specifica il colore dei punti, in questo caso, rosso.
- `marker='o'`: Specifica il tipo di marker, in questo caso, cerchi.

5. `plt.title('Scatter plot - Età VS Altezza')`: Aggiunge un titolo al grafico.
6. `plt.xlabel('Età')`: Aggiunge un'etichetta sull'asse x.
7. `plt.ylabel('Altezza (cm)')`: Aggiunge un'etichetta sull'asse y.
8. `plt.grid(True)`: Attiva la griglia di sfondo nel grafico.
9. `plt.show()`: Mostra il grafico di dispersione con tutte le personalizzazioni.

## 29 E adesso IA

I dataset

Un dataset è un insieme strutturato di dati che è organizzato in righe e colonne, simile a una tabella.

```
[57]: import pandas as pd

# Dataset con dati mancanti rappresentati da None o NaN
dataset = [
    {"età": 25, "punteggio": 90, "ammesso": 1},
    {"età": None, "punteggio": 85, "ammesso": 0},
    {"età": 28, "punteggio": None, "ammesso": 1},
    {"età": None, "punteggio": 75, "ammesso": 1},
    {"età": 23, "punteggio": None, "ammesso": None},
    {"età": 23, "punteggio": 77, "ammesso": None},
]
df = pd.DataFrame(dataset) #dataframe è una tabella
df
```

```
[57]:
```

	età	punteggio	ammesso
0	25.0	90.0	1.0
1	NaN	85.0	0.0
2	28.0	NaN	1.0
3	NaN	75.0	1.0
4	23.0	NaN	NaN
5	23.0	77.0	NaN

Questo codice utilizza la libreria **pandas** per creare un **DataFrame**, che è una struttura dati tabellare fornita da Pandas. Ecco una spiegazione dettagliata:

1. `import pandas as pd`: Importa la libreria **pandas** con l'alias **pd**, che consente di accedere alle funzioni e agli oggetti di Pandas.



2. **dataset**: È una lista di dizionari. Ogni dizionario rappresenta una riga del dataset con le chiavi che corrispondono alle colonne della tabella e i valori rappresentanti i dati delle colonne. Alcuni valori sono mancanti, rappresentati da **None**.
3. **pd.DataFrame(dataset)**: Crea un DataFrame utilizzando i dati forniti. Un DataFrame è una struttura dati tabellare di Pandas, dove ogni riga rappresenta un'osservazione e ogni colonna rappresenta una variabile. Questa funzione trasforma il dataset in una tabella organizzata in righe e colonne.
4. **df**: Assegna il DataFrame creato a una variabile chiamata **df**.
5. **df**: Mostra il DataFrame, visualizzando i dati in formato tabellare.

In questo DataFrame, le colonne sono “età”, “punteggio” e “ammesso”, e ogni riga contiene i dati corrispondenti. I valori mancanti sono rappresentati da **None** nella tabella.

Nuove librerie:

1. Pandas  
libreria che permette di creare un dataframe

`df['punteggio']`

Mostra il dataframe, nella colonna punteggio

```
[58]: #identificazione delle righe con dati mancanti
righe_con_dati_mancanti = df[df.isnull().any(axis=1)]
righe_con_dati_mancanti
```

```
[58]:      età  punteggio  ammesso
1   NaN      85.0      0.0
2  28.0      NaN      1.0
3   NaN      75.0      1.0
4  23.0      NaN      NaN
5  23.0      77.0      NaN
```

1. **righe\_con\_dati\_mancanti = df[df.isnull().any(axis=1)]**: Questa riga di codice utilizza il metodo `.isnull()` per creare un DataFrame booleano in cui ogni cella è **True** se il valore è mancante (NaN o None), altrimenti è **False**. Successivamente, il metodo `.any(axis=1)` viene utilizzato per controllare se almeno un valore mancante è presente in ciascuna riga. Questo restituisce una serie booleana indicando se ci sono dati mancanti in ogni riga. Infine, questa serie booleana viene usata per filtrare il DataFrame originale **df**, restituendo solo le righe che contengono dati mancanti.
2. **righe\_con\_dati\_mancanti**: Questo codice restituisce il DataFrame contenente solo le righe che contengono dati mancanti.

In sostanza, questa riga di codice serve a individuare le righe del DataFrame che hanno almeno un valore mancante.

```
[59]: #Conta quante righe con dati mancanti ci sono in totale
totale_dati_mancanti = righe_con_dati_mancanti.shape[0] #0=righe, 1=colonne
```

```
totale_dati_mancanti
```

[59]: 5

Questo codice conta il numero totale di righe con dati mancanti nel DataFrame filtrato `righe_con_dati_mancanti`. Ecco una spiegazione:

1. `totale_dati_mancanti = righe_con_dati_mancanti.shape[0]`: La proprietà `.shape` restituisce una tupla che rappresenta le dimensioni del DataFrame. L'elemento `[0]` di questa tupla corrisponde al numero di righe nel DataFrame. Pertanto, `righe_con_dati_mancanti.shape[0]` restituisce il numero totale di righe nel DataFrame `righe_con_dati_mancanti`, che rappresentano i dati mancanti.
2. `totale_dati_mancanti`: Questa variabile contiene il numero totale di righe con dati mancanti nel DataFrame filtrato.

```
[60]: print('righe con dati mancanti:')
      print(righe_con_dati_mancanti)
      print('Totale dati mancanti: ',totale_dati_mancanti)
```

```
righe con dati mancanti:
   età  punteggio  ammesso
1  NaN      85.0      0.0
2 28.0      NaN      1.0
3  NaN      75.0      1.0
4 23.0      NaN      NaN
5 23.0      77.0      NaN
Totale dati mancanti: 5
```

```
[61]: import pandas as pd

# Dataset con dati mancanti rappresentati da None o NaN
dataset = [
    {"nome": "Alice", "età": 25, "punteggio": 90, "email": "alice@email.com"},
    {"nome": "Bob", "età": 22, "punteggio": None, "email": None},
    {"nome": "Charlie", "età": 28, "punteggio": 75, "email": "charlie@email.
↪com"},
]

# Converti il dataset in un DataFrame
df = pd.DataFrame(dataset)
df
```

```
[61]:
```

	nome	età	punteggio	email
0	Alice	25	90.0	alice@email.com
1	Bob	22	NaN	None
2	Charlie	28	75.0	charlie@email.com

```
[62]: #Rimuovi le righe con dati mancanti
df1=df.dropna(inplace=False) #crea un nuovo dataframe in questo caso senza riga
                                     #se è true allora sostituisce il nuovo dataframe
↳ in generale con quello nuovo
df1
```

```
[62]:      nome  età  punteggio      email
0   Alice   25      90.0  alice@email.com
2  Charlie   28      75.0  charlie@email.com
```

```
[63]: import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# Genera dati di esempio
data = {
    'Variable1': [1, 2, 3, 4, 5],
    'Variable2': [1, 2, np.nan, 4, np.nan],
    'Missing_Column': ['A', 'B', 'A', 'C', np.nan]
}
# Crea un DataFrame
df = pd.DataFrame(data)
df1=pd.DataFrame()
df
```

```
[63]:      Variable1  Variable2  Missing_Column
0           1          1.0             A
1           2          2.0             B
2           3          NaN             A
3           4          4.0             C
4           5          NaN           NaN
```

```
[64]: #trattamento dei missing values nelle variabili numeriche
numeric_cols = df.select_dtypes(include=['number'])
numeric_cols
```

```
[64]:      Variable1  Variable2
0           1          1.0
1           2          2.0
2           3          NaN
3           4          4.0
4           5          NaN
```

```
[65]: #trattamento dei missing values nelle variabili numeriche
numeric_cols = df.select_dtypes(include=['number']) #include colonne numeriche
↳ e fa vedere solo categoriche
```

```
numeric_cols.columns
```

```
[65]: Index(['Variable1', 'Variable2'], dtype='object')
```

```
[66]: df1[numeric_cols.columns] = df[numeric_cols.columns].fillna(df[numeric_cols.
      ↪columns].mean())
df1

#crea colonne con stessi nomi (var1 e var2) = assegna valori delle colonne df_
      ↪originale(mean=media)
```

```
[66]:
```

	Variable1	Variable2
0	1	1.000000
1	2	2.000000
2	3	2.333333
3	4	4.000000
4	5	2.333333

```
[67]: #trattamento dei missing values nelle variabili categoriche
categorical_cols = df.select_dtypes(exclude=['number']) #esclude colonne_
      ↪numeriche e fa vedere solo categoriche
categorical_cols.columns
```

```
[67]: Index(['Missing_Column'], dtype='object')
```

```
[68]: df1[numeric_cols.columns] = df[numeric_cols.columns].fillna(df[numeric_cols.
      ↪columns].mean().iloc[1])
df1
#iloc[] = gestisce indici
```

```
[68]:
```

	Variable1	Variable2
0	1	1.000000
1	2	2.000000
2	3	2.333333
3	4	4.000000
4	5	2.333333

## 30 RICORDA QUESTO

```
[69]: print(f"il primo coi valori mancanti \n{df} \ne")
```

```
il primo coi valori mancanti
  Variable1  Variable2  Missing_Column
0         1         1.0             A
1         2         2.0             B
2         3         NaN             A
```

3	4	4.0	C
4	5	NaN	NaN
e			

## 31 RICORDA QUESTO

```
[70]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Genera dati di esempio
data = {
    'Feature1': [1, 2, np.nan, 4, 5],
    'Feature2': [np.nan, 2, 3, 4, np.nan],
    'Feature3': [1, np.nan, 3, 4, 5]
}
# Crea un DataFrame
df = pd.DataFrame(data)
df
```

```
[70]:
```

	Feature1	Feature2	Feature3
0	1.0	NaN	1.0
1	2.0	2.0	NaN
2	NaN	3.0	3.0
3	4.0	4.0	4.0
4	5.0	NaN	5.0

```
[71]: df.isnull()
```

```
[71]:
```

	Feature1	Feature2	Feature3
0	False	True	False
1	False	False	True
2	True	False	False
3	False	False	False
4	False	True	False

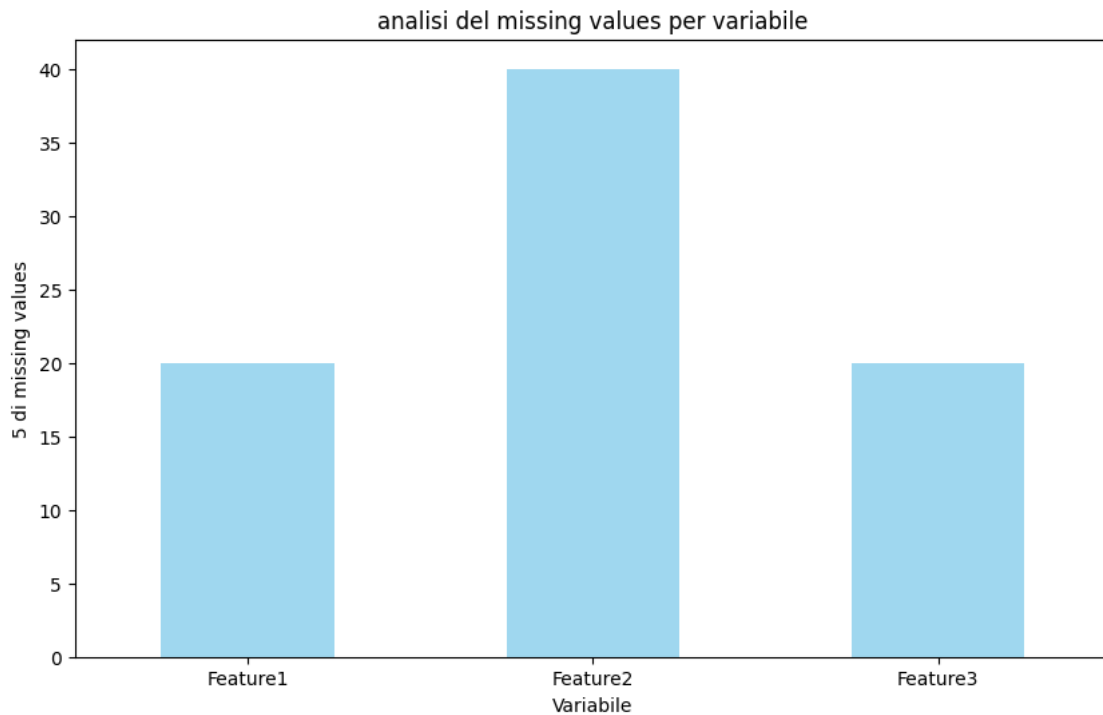
```
[72]: df.isnull().sum()
```

```
[72]: Feature1    1
Feature2    2
Feature3    1
dtype: int64
```

```
[73]: missing_percent = df.isnull().sum() / len(df) * 100
missing_percent
```

```
[73]: Feature1    20.0  
      Feature2    40.0  
      Feature3    20.0  
      dtype: float64
```

```
[74]: #Calcola la percentuale di righe con missing values per ciascuna variabile  
missing_percent= (df.isnull().sum()) / len(df) * 100  
  
#crea il grafico a barre  
plt.figure(figsize=(10,6))  
missing_percent.plot(kind='bar', color='skyblue', alpha=0.8)  
plt.xlabel('Variabile')  
plt.ylabel('5 di missing values')  
plt.title('analisi del missing values per variabile')  
plt.xticks(rotation=0)  
plt.show()
```



```
[75]: import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Genera dati di esempio  
data = {
```

```

    'Feature1': [1, 2, np.nan, 4, 5],
    'Feature2': [np.nan, 2, 3, 4, np.nan],
    'Feature3': [1, np.nan, 3, 4, 5]
}

# Crea un DataFrame
df = pd.DataFrame(data)

# Calcola la matrice di missing values
missing_matrix = df.isnull()
missing_matrix

```

```

[75]:
   Feature1  Feature2  Feature3
0     False      True     False
1     False     False      True
2      True     False     False
3     False     False     False
4     False      True     False

```