Introduction to Python programming

Dictionaries and Functions

- Armando Camerlingo

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani

Introduzione alla Programmazione in Python

Armando Camerlingo

06/06/2025 - Lezione 2

Dizionari

- Usati per contenere delle informazioni che hanno qualche relazione fra di loro;
- I dizionari salvano le informazioni in coppie di chiavi e valori, univocamente definiti;
- Non salvano in un **ordine specifico**, possono restituire le variabili salvate casualmente;

```
dizionario = {chiave_1: valore_1,
              chiave_2: valore_2,
              chiave_3: valore_3}
```

Esempio

```
dict = {'stud_A': 'Aldo',
        'stud_B': 'Bruno',
        'stud_C': 'Carlo',
        'stud_D': 'Dario'
        }
print(dict)
```

Possiamo ottenere le entrate singole del dizionario usando il nome del dizionario e la chiave in parentesi quadre:

ITINERIS

```
print('\nCodice studente: %s' % 'stud_1')
print('Nome: %s' % dict['stud_A'])

print('\nCodice studente: %s' % 'stud_2')
print('Nome: %s' % dict['stud_B'])

print('\nCodice studente: %s' % 'stud_3')
print('Nome: %s' % dict['stud_C'])
```

I dizionari hanno una particolare sintassi per i cicli for in modo da accedere a tutti i suoi elementi

```
for codice, nome in dict.items():
    print('\nCodice studente: %s' %
codice)
    print('Nome: %s' % nome)
```

Modificare un valore

I valori di un dizionario si possono modificare in modo simile alle liste, però invece di usare l'indice si usano le chiavi.

```
print('Studente B: ' + dict['stud_B'])
dict['stud_B'] = 'Benedetta'
print('Studente B: ' + dict['stud_B'])
```

Rimuovere un oggetto

In modo simile alle liste, si può usare del per cancellare il valore associato alla chiave fornita. Rimuove sia la chiave sia il valore.

```
print('elenco studenti: \n', dict)
del dict['stud_C']
print('nuovo elenco studenti: \n', dict)
print(dict)
```

Modificare una chiave

Modificare una chiave è più complesso, perchè la chiave serve per accedere ai valori. Si può fare in 2 step:

- Creare nuova chiave e copiare il valore associato alla chiave da modificare
- Cancellare la coppia associata alla vecchia chiave con `del`

```
print('elenco studenti: \n', dict)
dict['stud_E'] = dict['stud_B'] #creazione nuova
chiave
print('elenco studenti: \n', dict)
del dict['stud_B'] #cancellazione vecchia chiave
print('nuovo elenco studenti: \n', dict)
```

Esercizio 1

Abbiamo i seguenti voti associati ai codici degli studenti salvati in ordine alfabetico in una lista:

```
voti = [25, 18, 15, 22]
```

Salvare questa lista in un dizionario avente come chiave il codice e valori i voti.

Stampare in un elenco il nome dello studente seguito dal voto, utilizzando il dizionario appena creato.

Facoltativo:

Creare un nuovo dizionario che contenga entrambe le informazioni (nome, voto) associato al codice studente a partire dal dizionario già presente e la lista.

Soluzione 1

```
print('elenco studenti :', dict)
voti = [25, 18, 15, 22]
dict_voti = {'stud_A' : voti[0],
'stud_D' : voti[1], 'stud_E' : voti[2]}
print(dict_voti)

i = 0
for codice, nome in dict.items():
    print(nome, ':', dict_voti[codice])
    i += 1
```

Soluzione 2

```
i = 0
print(dict)
for codice, nome in dict.items():
  dict_finale[codice] = [nome, voti[i]]
  i += 1
print(dict_finale)

for codice, info in dict_finale.items():
  print('Codice : ', codice, '\nNome : ',
info[0], '\nVoto : ', info[1] )
```

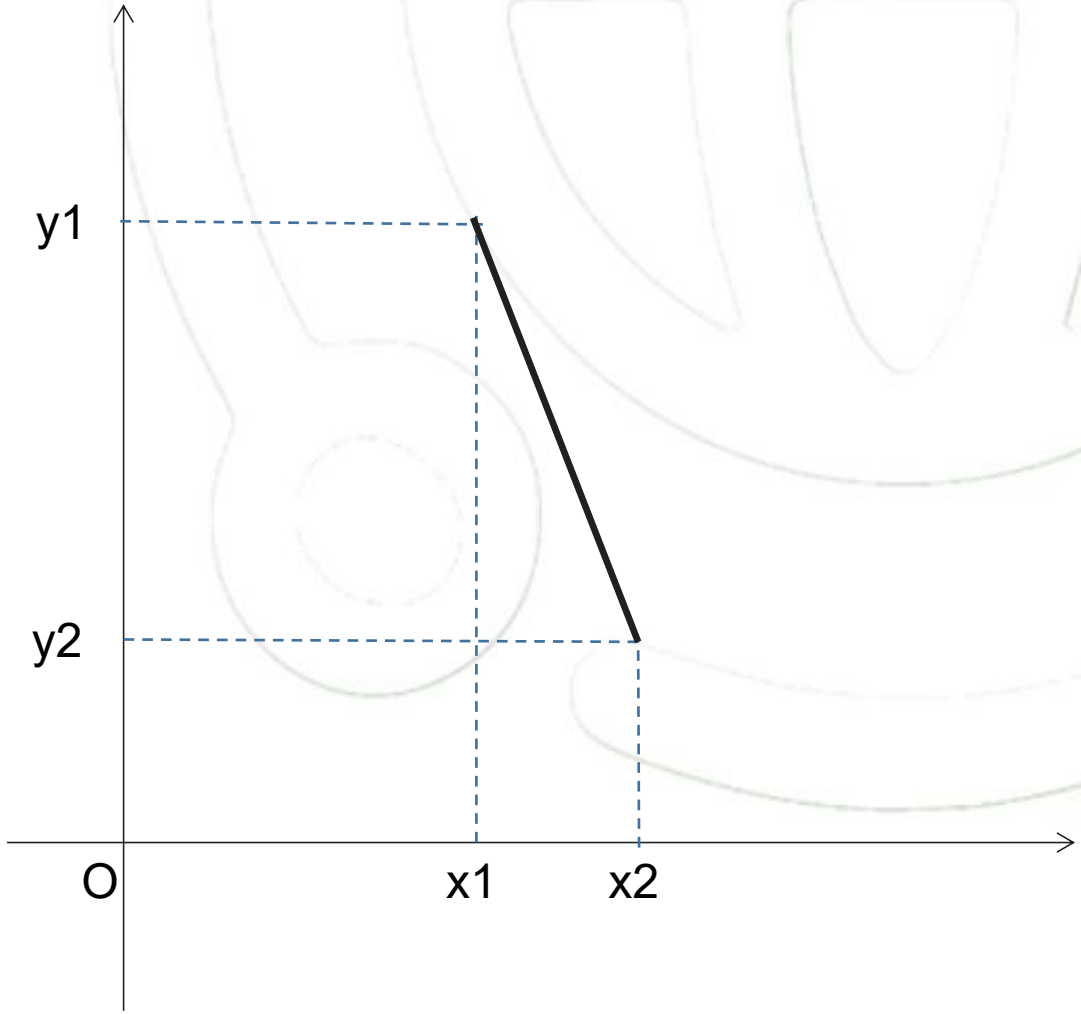
Funzioni

- Sono gruppi di istruzioni collegate fra loro che compiono un’azione specifica;
- Le funzioni possono essere richiamate più volte, rendendo il codice riusabile e evitando ripetizioni nel caso si debba compiere una stessa azione più volte;
- Aiutano a dividere il programma in pezzi più piccoli e modulari, facilitando la gestione e l’ordine di programmi molto grandi;
- Questa modularità aiuta molto anche nel debugging del programma;

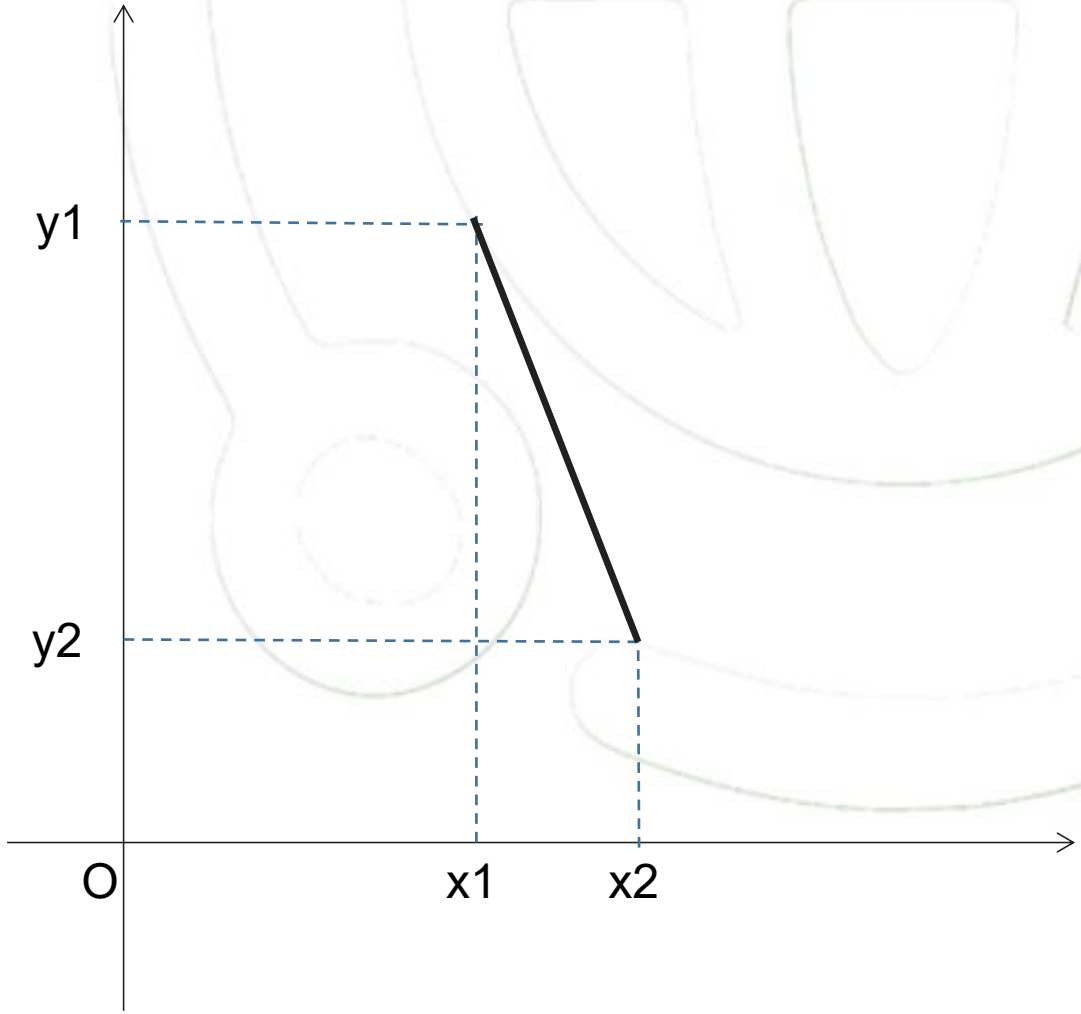
```
def nome_funzione(parametri):
    istruzioni
    return(output) #facoltativo!
```

- La keyword **`def`** indica l'inizio dell'header della funzione ;
- Tramite i **`parametri`** possiamo passare valori ad una funzione;
- Una o più **`istruzioni`** python che compongono il body della funzione. Devono avere tutte la stessa indentazione;
- Se necessario, si può usare l'istruzione **`return()`** per far sì che la funzione restituisca un valore;

```
x1 = 5
y1 = 9
x2 = 7
y2 = 3

dif_x = x1 - x2
dif_y = y1 - y2

dif_x = dif_x**2
dif_y = dif_y**2

D = dx + dy
D = D**(1/2)

print(D)
```



```
def dist(x1,y1,x2,y2):
    dif_x = x1 - x2
    dif_y = y1 - y2

    dif_x = dif_x**2
    dif_y = dif_y**2

    D = dif_x + dif_y
    D = D**(1/2)

    return(D)
```



```
x1 = 5
y1 = 9
x2 = 7
y2 = 3

dist(x1,y1,x2,y2)

#oppure si possono dare direttamente dei
#valori

dist(21,45,13,9)
```

Gestione dei file di testo:

- Il modo più semplice di condividere e conservare dati è tramite lettura e scrittura di file di testo;
- Utilizzeremo dei file già presenti in Colab sotto la directory sample_data;
- Possiamo leggere il file README per avere una descrizione dei datasets disponibili;

```
! cat sample_data/README.md
```

```
! ls -l sample_data
```

```
total 55504
-rwxr-xr-x 1 root root     1697 Jan  1  2000 anscombe.json
-rw-r--r-- 1 root root   301141 Mar 13 13:31 california_housing_test.csv
-rw-r--r-- 1 root root  1706430 Mar 13 13:31 california_housing_train.csv
-rw-r--r-- 1 root root 18289443 Mar 13 13:31 mnist_test.csv
-rw-r--r-- 1 root root 36523880 Mar 13 13:31 mnist_train_small.csv
-rwxr-xr-x 1 root root      962 Jan  1  2000 README.md
```

```
! head sample_data/california_housing_test.csv
```

```
"longitude","latitude","housing_median_age","total_rooms","total_bedrooms",
"population","households","median_income","median_house_value"
-
122.050000,37.370000,27.000000,3885.000000,661.000000,1537.000000,606.00000
0,6.608500,344700.000000
-
118.300000,34.260000,43.000000,1510.000000,310.000000,809.000000,277.000000,
3.599000,176500.000000
-
117.810000,33.780000,27.000000,3589.000000,507.000000,1484.000000,495.00000
0,5.793400,270500.000000
-
118.360000,33.820000,28.000000,67.000000,15.000000,49.000000,11.000000,6.13
5900,330000.000000
-
119.670000,36.330000,19.000000,1241.000000,244.000000,850.000000,237.000000,
2.937500,81700.000000
```

La funzione open()

La funzione `open()` può essere usata per leggere e scrivere file:

```
oggetto_file = open("nome_file", "mode")
```

Al parametro `mode` si possono assegnare i seguenti valori:

- **`r`**: sta per read, lettura, per quando si vuole solo accedere ai dati all'interno del file;
- **`w`**: sta per write, scrittura, per scrivere nuove informazioni nel file. ATTENZIONE: se esiste un file con lo stesso nome, verrà SOVRASCRITTO;
- **`a`**: sta per append, aggiungere, per aggiungere nuovi dati alla fine del file, questi dati verranno automaticamente messi dopo le ultime righe;
- **`s`**: modalità di lettura e scrittura;

Scrittura: la funzione write()

```
file = open("sample_data/file_test.txt", "w")

file.write("Hello World")
file.write("seconda riga")
file.write("...terza riga...")
file.write("...ultima riga")



file.close()
```

```
! ls -l sample_data/file_test.txt
! more sample_data/file_test.txt
```

```
file = open("sample_data/file_test.txt", "w")

file.write("Hello World\n")
file.write("seconda riga\n")
file.write("...terza riga\n...")
file.write("...ultima riga")



file.close()
```

```
! cat sample_data/file_test.txt
```

La funzione read()

```
file = open("sample_data/california_housing_test.csv",
"r")
F = file.read()
file.close()

print(type(F))
print(F)
```

```
F = file.read(20)
file.close()

print(F)
```

L’istruzione `with` può essere usata per evitare di dimenticare di chiudere un file (o un network o un database) alla fine delle modifiche. Non c’è bisogno di chiudere il file con `close()`, viene chiuso automaticatemente dopo che sono state eseguite tutte le istruzioni nel blocco `with`.

```
with open("sample_data/california_housing_test.csv",
"r") as f:
  conto= 0
  for riga in f:
    print(riga)
    conto += 1

    if conto > 5:
      break
```

La funzione readlines()

ITINERIS

```
file = open("sample_data/california_housing_test.csv",
"r")
F = file.readlines()
file.close()

print(type(F))
print(F)
```

La funzione readlines() restituisce le righe del file come lista.

Le Librerie

- Python è basato sui **moduli** per aggiungere funzioni utilizzabili nel codice;
- Ogni modulo può contenere multiple funzioni, di solito raccolte a secondo del loro scopo;
- Utile per non scrivere lo stesso codice per più programmi e soprattutto non riscriverlo se è già disponibile;
- Possono anche contenere dataset e altro;
- Una **libreria** è una collezione di moduli;
- Alcune librerie fondamentali, chiamate Python standard libraries, sono già presenti nell'installazione base di Python.

NumPy e Pandas

- **NumPy** (“Numerical Python”) è una libreria open source usata in quasi tutti gli script di ambito scientifico. NumPy lavora con gli **array** ed è utile quando si lavora in ambito di algebra lineare e matrici;
- **Pandas** (“Python Data Analysis”) è una libreria costruita a partire da NumPy. Utilizzata per la gestione di dataset. La sua struttura di base è il **DataFrame** e permette di manipolare dati in tabelle (p.e. righe osservazioni, colonne variabili);

Toy Dataset

- Utilizzeremo il dataset iris disponibile nella libreria seaborn;

```
import seaborne as sns
iris = sns.load_dataset("iris")
```

- Dataset su misurazioni di diverse tipologie di fiori iris;

Toy Dataset

- Pandas può anche leggere csv da url o file locali e caricarli direttamente in Dataframe;

```
iris_2 = pd.read_csv(
   "https://raw.githubusercontent.com/mwaskom/seaborn-data/refs/heads/master/iris.csv"
)

print(type(iris_2))
```

Controllare il formato dei dati

- con la funzione **`type()`** possiamo vedere che cosa è l'oggetto che stiamo indagando;
- l'attributo **`shape`** restituisce le dimensioni del dataframe, ovvero nel caso di tabelle il numero di righe e il numero di colonne;
- l'attributo **`dtypes`** restituisce che tipo di variabile è il dato contenuto nelle colonne;

```
type(iris)

iris.shape

iris.dtypes
```

Possiamo ricavare i nomi delle colonne e delle righe usando gli attributi del DataFrame

```
# nomi delle colonne, restituiti come indice
iris.columns
```

```
#nomi delle colonne, restituiti come array
iris.columns.values
```

```
# nomi delle righe, restituiti come indice
iris.index
```

```
#nomi delle colonne, restituiti come array
iris.index.values
```

Si può accedere ai dati all'interno del dataframe utilizzando i suoi metodi


```
# stampa prime 5 righe
iris.head()
```

```
# stampa prime 10 righe
iris.head(10)
```

Si può accedere ai dati anche utilizzando i nomi delle colonne

```
# stampa tutta la colonna
iris['petal_length']
```

```
# stampa solo le prime linee della colonna selezionata
iris['petal_length'].head()
```


```
# stampa solo le prime linee della colonna selezionata
iris['petal_length'].head()
```

Si può accedere a parte del dataset utilizzando l'operatore [] per selezionare le righe e le colonne.

```
# Selezionare righe specifiche
iris[0:3]
```

```
# Selezionare le prime 5 righe
iris[:5]
```

```
# Selezionare le ultime 6 righe
iris[-6:]
```

```
# Selezionare righe specifiche
iris[10:13]
```

Si può accedere ai dati utilizzando i metodi:

- **`iloc[]`** utilizza indici basato sugli interi;
- **`loc[]`** utilizza indici basati sulle etichette;

```
# Selezione di una colonna utilizzando iloc
iris.iloc[:, 4]
```

```
# Slicing righe e colonne utilizzando iloc
iris.iloc[1:3, 0:2]
```

```
# Slicing righe e colonne utilizzando loc
# per le righe si utilizzano gli interi, per le colonne i
# loro nomi
iris.loc[1:3, ['sepal_length', 'sepal_width']]
```

Si possono filtrare i dati utilizzando condizioni sui valori contenuti nel DataFrame

```
# Selezioniamo solo le righe associate all'iris versicolor
iris[iris['species'] == 'versicolor']
```

```
# Selezioniamo tutti i record che non si riferiscono a
iris di tipo versicolor
iris[iris['species'] != 'versicolor']
```

```
# Selezionare tutti gli esemplari di specie setosa con
lunghezza petalo
# (petal_length) superiore ai 1.5 cm
iris[(iris['species'] == 'setosa') & (iris['petal_length']
> 1.5)]
```

Esercizio 1

Selezionare tutti gli esemplari di specie virginica con larghezza del sepale (sepal_width) inferiore o uguale a 3.0 e stampare solo le colonne relative al sepale stesso e specie (sepal_length, sepal_width, species)

Soluzione

```
iris[(iris.species == 'virginica') &
(iris.sepal_width <= 3.0)][['sepal_length',
'sepal_width', 'species']]
```

Statistiche sui dati

I DataFrame di Pandas offrono diversi attributi, metodi e funzioni per ottenere statistiche sui dati utili al fine di prepararli per l'analisi

- Valori unici di una colonna

```
# Entrate uniche in una colonna specifica
pd.unique(iris['species'])
```

```
# si possono salvare i valori in un array
specie = pd.unique(iris['species'])
print(specie)
```

- Numero di valori unici di una colonna

```
# Si può utilizzare l'array salvato precedentemente
len(specie)
```

```
# Si può utilizzare il metodo nunique per contarli
iris['species'].nunique()
```

- Statistica descrittiva

Diverse grandezze come media e mediana sui dati contenuti nelle colonne di tipo numerico

```
iris['sepal_length'].describe()
```

```
# Si possono stampare specifice grandezze
iris['sepal_length'].median()
```

- Raggruppare il dato

Si possono raggruppare i dati a partire di valori contenuti nelle colonne.

```
iris.groupby('species')
```

Il dataframe con i dati separati per i valori nelle colonne può essere salvato in un nuovo dataframe

```
species_groupby = iris.groupby('species')
species_groupby.head()
```

- Raggruppare il dato - Statistiche

Una volta raggruppati, si possono contare gli elementi totali per ogni elemento della colonna.

```
species_groupby.count()
```

Allo stesso modo possiamo ottenere le statistiche associate:

```
species_groupby.describe()
```

Si possono raggruppare gli elementi anche per più di una colonna

```
iris.groupby(['species',
'sepal_length'])['petal_length'].count()

type(iris.groupby(['species',
'sepal_length'])['petal_length'].count())
```

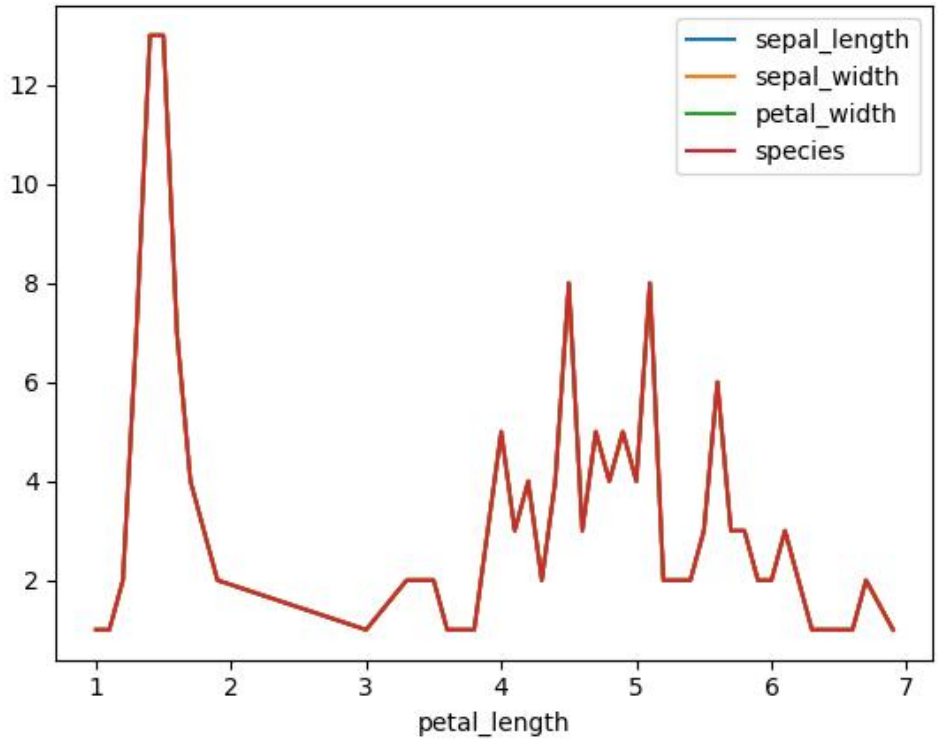
Si possono anche ottenere quanti elementi del dataframe corrispondono ai valori di una determinata colonna

```
species_seplen_count = iris.groupby(['species',
'sepal_length'])['petal_length'].count().unstack('sepal_len
gth')
print(species_seplen_count)
print(type(species_seplen_count))
```

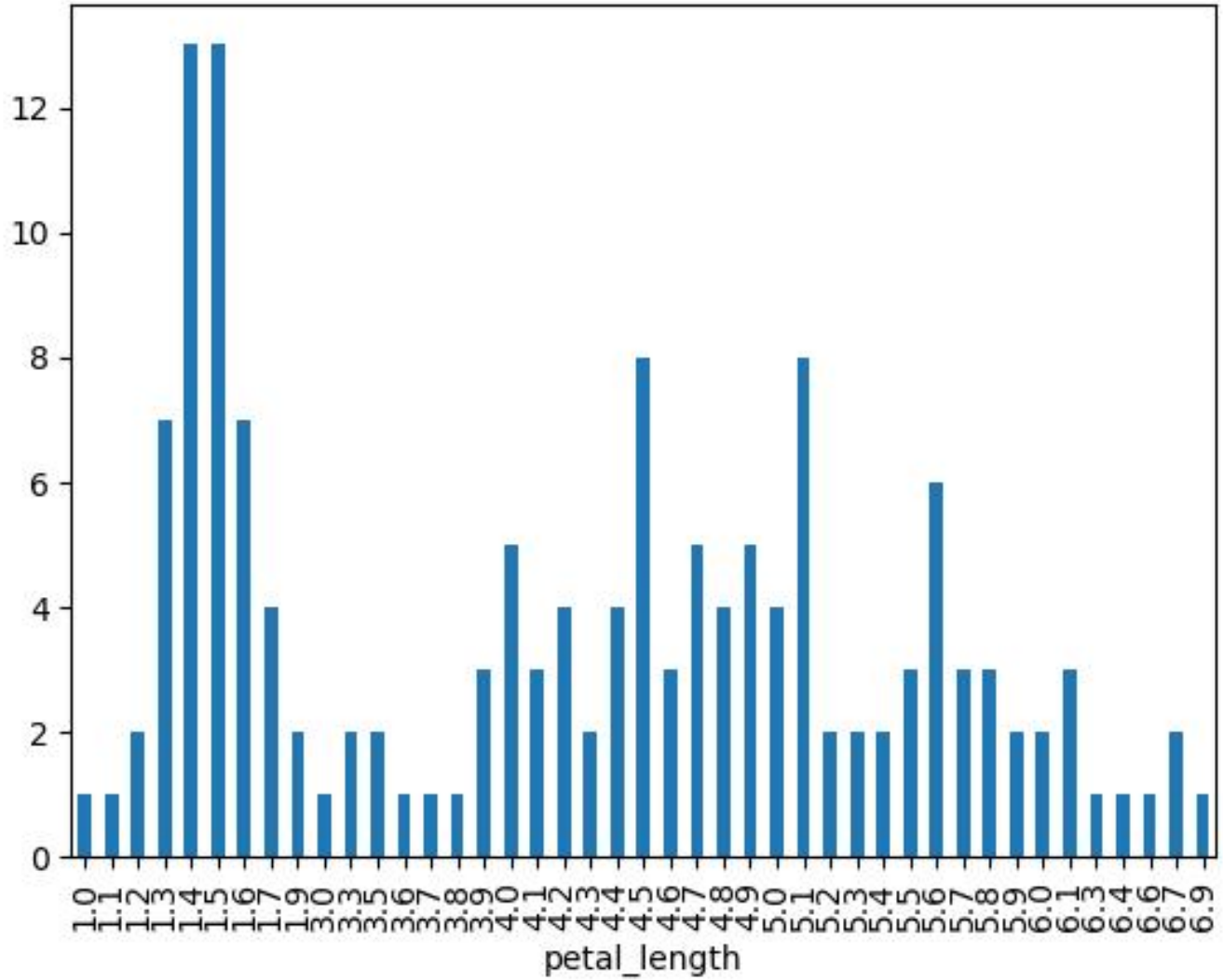
Graficare i dati

Con Pandas si possono velocemente visualizzare i dati in un grafico

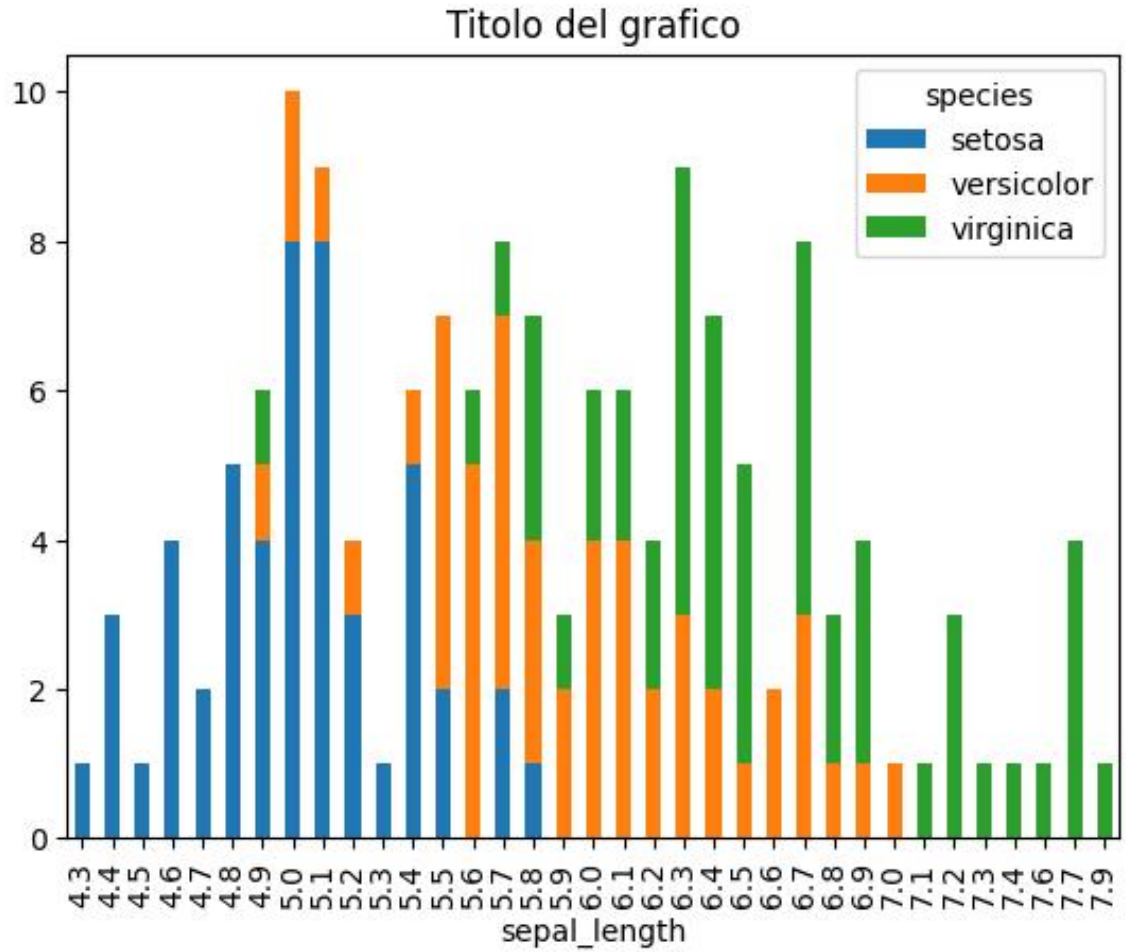
```
petlen_count = iris.groupby('petal_length')['species'].count()
petlen_count.plot()
```



```
petlen_count.plot(kind = 'bar')
```



```
species_seplen_count = iris.groupby(['species',
'sepal_length'])['petal_length'].count().unstack('species')

species_seplen_count.plot(kind = 'bar', stacked=True,
title = 'Titolo del grafico')
```



La libreria “math”

ITINERIS

```
import math
print(math)
```

Libreria contenente funzioni e costanti utili in ambito matematico, come sin(), exp() e pi

```
print(pi)
```

Non funziona, bisogna specificare il modulo da cui recuperare pi

```
print(math.pi)
```

Per evitare di specificare il modulo ogni volta, possiamo usare tre istruzioni

```
from math import pi
print(pi)
```

```
from math import pi as PI_COST
print(PI_COST)
```

```
from math import *
print(pi)
```

La libreria “random”

ITINERIS

```
import random
```

Libreria contenente funzioni per generare sequenze di numeri “randomici”.

Con `randint()` si possono generare numeri interi randomici.

```
randint(start,end)
```

Per esempio, possiamo simulare un lancio di un dado

```
import random
dice_result = random.randint(1,6)
print(dice_result)
```

Esempio di funzione con random:

```
import random

def quanti_6_escono(lanci=10):
  numero_6 = 0
  print("Tirando il dado", lanci, " volte")
  for i in range(lanci):
    dice = random.randint(1,6)
    if dice == 6:
      numero_6 += 1
      print("è uscito un 6")
  frequenza_6 = numero_6/lanci
  return(frequenza_6)

quanti_6_escono(100)
```

Le librerie “numpy” e “matplotlib”

ITINERIS

- Numpy è una delle librerie Python più usate in ambito scientifico. Introduce un oggetto chiamato array, multidimensionale e ad alte performance in termini di memoria e gestione. Insieme ad esso, introduce i tools necessari per maneggiarli ed effettuare diverse operazioni con essi;
- Matplotlib è una libreria basata su Numpy per la visualizzazione di dati con molteplici possibilità di grafici e personalizzazione.

Array Numpy

ITINERIS

L’array è un insieme di valori, tutti dello **stesso tipo**, ed usa un indice di numeri interi non-negativi. Il numero di dimensioni dell’array viene chiamato rango (rank), mentre la forma (shape) di un array è una tupla di interi con la lunghezza dell’array lungo ogni dimensione.

```
import numpy as np

a = np.array([1,2,3])
print(type(a))
print(a.shape)
print(a[0], a[1], a[2])
a[0] = 5
print(a)
```

Array Numpy

ITINERIS

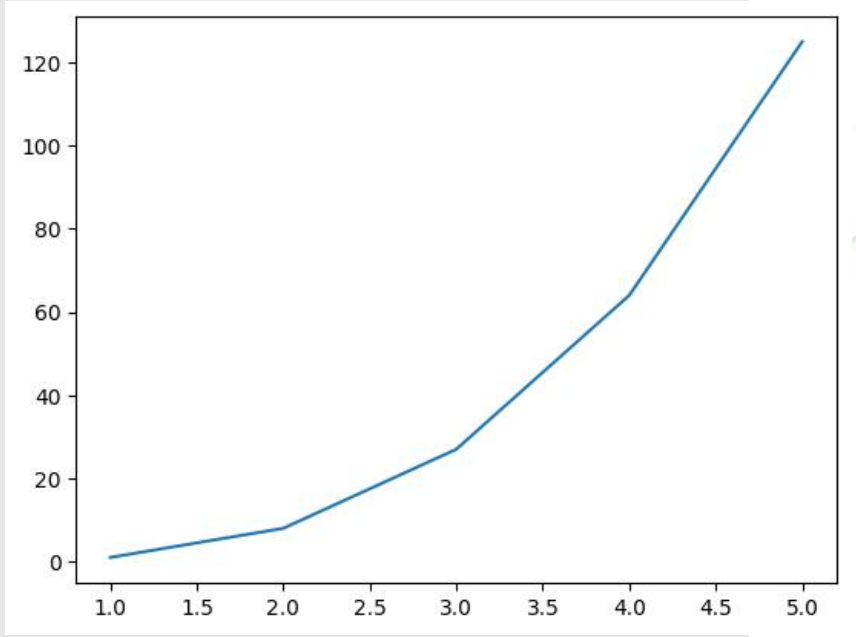
L’array è un insieme di valori, tutti dello **stesso tipo**, ed usa un indice di numeri interi non-negativi. Il numero di dimensioni dell’array viene chiamato rango (rank), mentre la forma (shape) di un array è una tupla di interi con la lunghezza dell’array lungo ogni dimensione.

```
import numpy as np

b = np.array([[1,2,3],[4,5,6]])

print(b.shape)
print(b[0,0], b[0,1] , b[1,0])
```

Matplotlib: il modulo pyplot e il metodo plot()

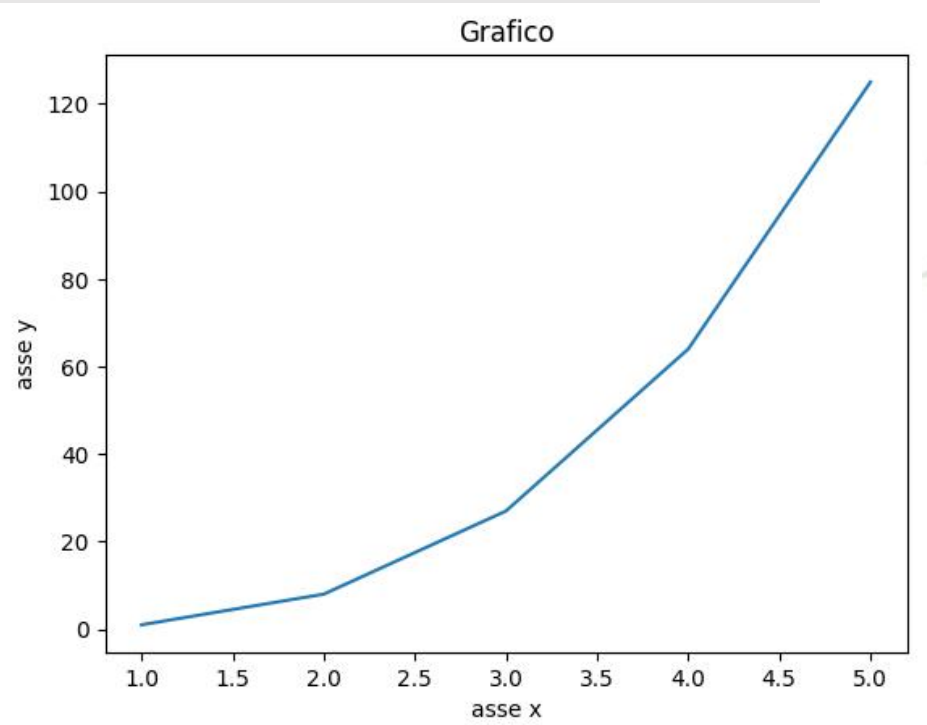
Il metodo **`plot()`** del modulo **`pyplot`** di matplotlib permette di visualizzare i dati con molti tipi di grafico altamente personalizzabile.

```
import matplotlib.pyplot as plt
import numpy as np
#creiamo due array da plottare
x = np.array([1,2,3,4,5])
y = np.array([1,8,27,64,125])

#usiamo il modulo plt per
#creare un grafico
plt.plot(x,y)

#mostra il plot
plt.show()
```



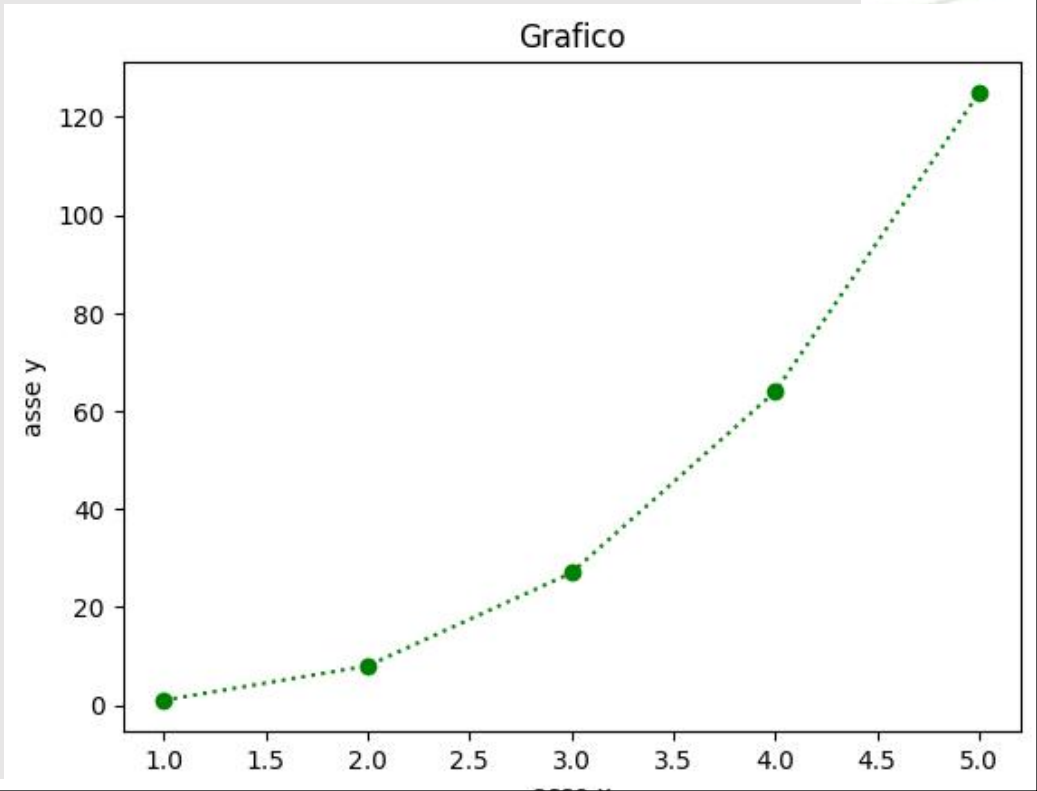
Titolo e Nomi Assi

```
import matplotlib.pyplot as plt
import numpy as np

#creiamo due array da plottare
x = np.array([1,2,3,4,5])
y = np.array([1,8,27,64,125])

#usiamo il modulo plt per creare un grafico
plt.plot(x,y)

#aggiungiamo titolo e nomi assi
plt.title('Grafico')
plt.xlabel('asse x')
plt.ylabel('asse y')

#mostra il plot
plt.show()
```



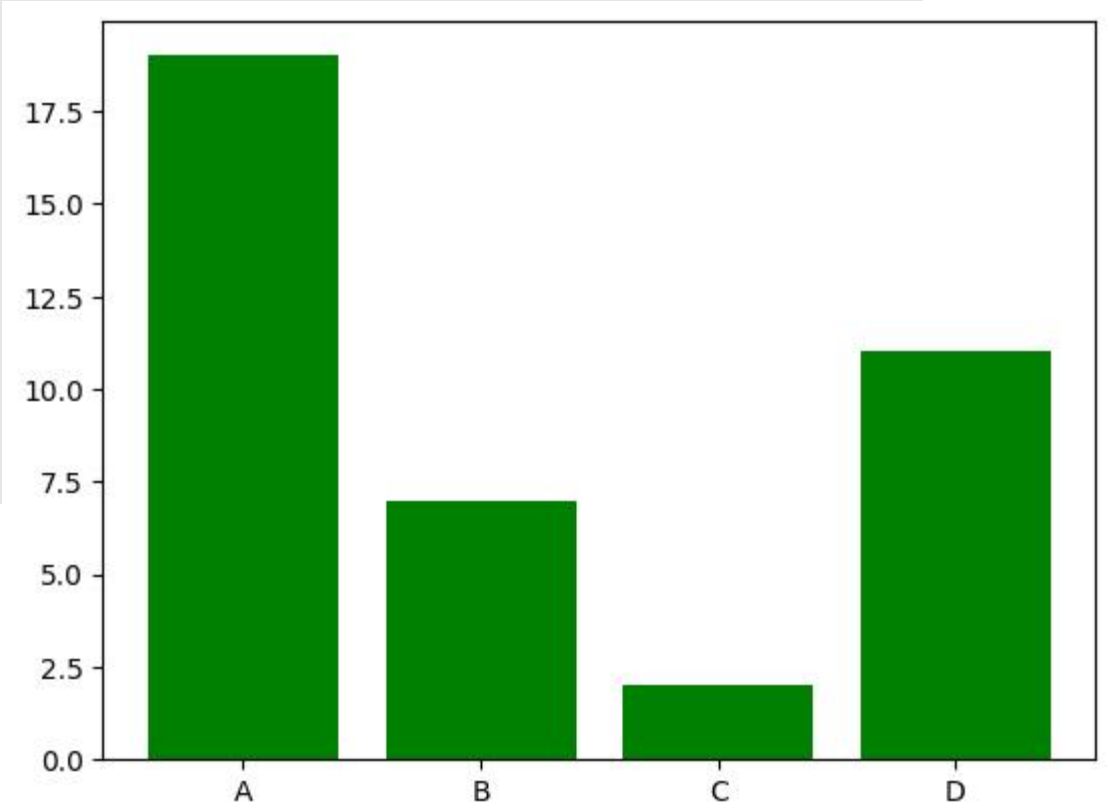
Tratteggio e Colore Linea

```
import matplotlib.pyplot as plt
import numpy as np

#creiamo due array da plottare
x = np.array([1,2,3,4,5])
y = np.array([1,8,27,64,125])

#usiamo il modulo plt per creare un grafico
plt.plot(x,y,":og")

#aggiungiamo titolo e nomi assi
plt.title('Grafico')
plt.xlabel('asse x')
plt.ylabel('asse y')


#mostra il plot
plt.show()
```



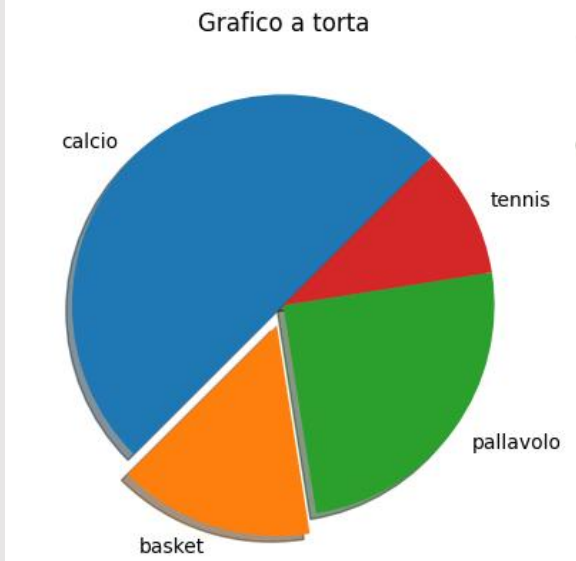
Pyplot: metodo bar()

```
import matplotlib.pyplot as plt

x = ["A", "B", "C", "D"]
y = [19, 7, 2, 11]

#creiamo un grafico a barre
plt.bar(x,y, color="green")

#mostra il plot
plt.show()
```



Pyplot: metodo pie()

```
import matplotlib.pyplot as plt

sport = np.array(['calcio', 'basket', 'pallavolo',
'tennis'])
perc = np.array([50, 15, 25, 10])
explode = np.array([0, 0.1, 0, 0])

#plottiamo un grafico a torta
plt.pie(perc, explode=explode, labels=sport, shadow=True,
startangle=45)

#titolo
plt.title("Grafico a torta")

#mostra il plot
plt.show()
```



Esercizio

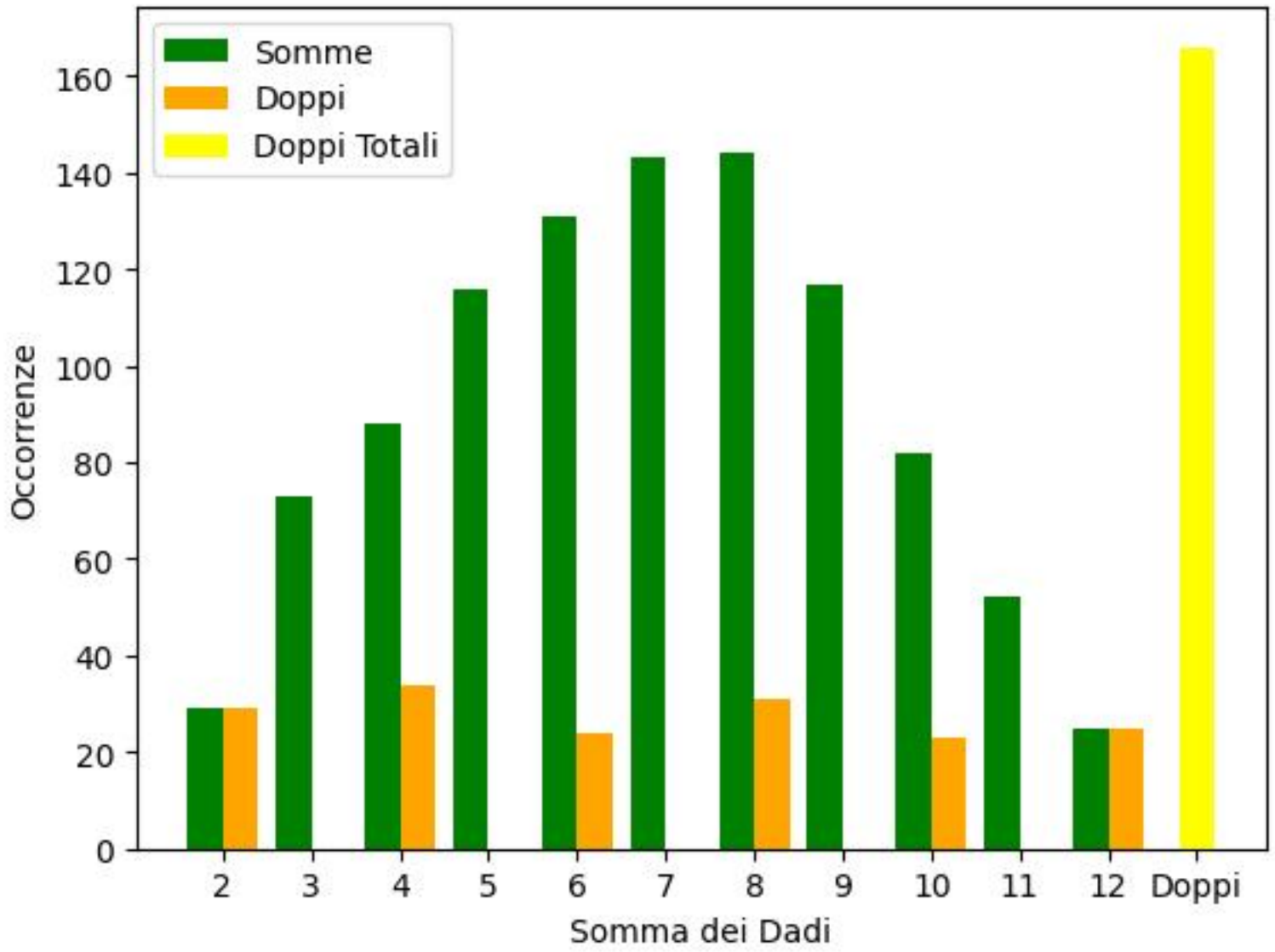
ITINERIS

Scrivere una funzione che simula il lancio di un paio di dadi e ottenere un istogramma dei risultati.

- Servirà una variabile per memorizzare il numero di lanci da simulare. Poi si lanceranno due dadi per lancio;
- Sommare i due numeri ottenuti dai dadi e tenere conto di quante volte escono le somme da 2 a 12 e quante volte abbiamo lo stesso valore sugli stessi dadi.

```
import random
import matplotlib.pyplot as plt
import numpy as np

def lancio_dado():
  return random.randint(1,6)

def lancio_2_dadi(numero_lanci=10):
  risultati = {2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0, 11:0, 12:0}
  doppio_tot = 0
  doppio = {2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0, 11:0, 12:0}
  for i in range(numero_lanci):
    dado_1 = lancio_dado()
    dado_2 = lancio_dado()
    somma = dado_1 + dado_2
    if dado_1 == dado_2:
      doppio[somma] += 1
      doppio_tot += 1
    risultati[somma] += 1

  return risultati, doppio, doppio_tot
```

```
def isto(risultati=dict(), doppio=dict(), doppio_tot=0):
  x = np.array(list(risultati.keys()))
  plt.bar(x-0.2 , risultati.values(), 0.4, color='green', label = 'Somme')
  plt.bar(x+0.2, doppio.values(), 0.4, color='orange', label = 'Doppi')
  plt.bar(13, doppio_tot, 0.4, color='yellow', label = 'Doppi Totali')
  plt.xlabel('Somma dei Dadi')
  plt.ylabel ('Occorrenze')
  xticks = list(risultati.keys())
  xticks.append('Doppi')
  plt.xticks(np.array(range(2,14)), xticks)
  plt.legend()
  plt.show()

risultati, doppio, doppio_tot = lancio_2_dadi(1000)
print(risultati)
print(doppio)
print(doppio_tot)
isto(risultati, doppio, doppio_tot)
```


Somme
Doppi
Doppi Totali
Occorrenze
0
20
40
60
80
100
120
140
160
2
3
4
5
6
7
8
9
10
11
12
Doppi
Somma dei Dadi

THANKS!

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani