



# QaA Ottobre

*Python*

**Domanda uno:**

**Come posso creare un grafico a linee con matplotlib e personalizzare il colore e lo stile della linea?**



Per creare un grafico a linee in matplotlib, si usa la funzione `plt.plot()`, che permette di visualizzare dati in forma di linea continua.

Possiamo personalizzare il grafico specificando il colore, lo stile della linea e altri parametri, passando argomenti come `color` e `linestyle`. Ad esempio, per ottenere una linea tratteggiata blu possiamo specificare `color='blue'` e `linestyle='--'`.

```
1. import matplotlib.pyplot as plt  
2.  
3. x = [0, 1, 2, 3, 4]  
4. y = [0, 1, 4, 9, 16]  
5.  
6. plt.plot(x, y, color='blue', linestyle='--', linewidth=2) # Linea tratteggiata blu  
7. plt.xlabel('X-axis')  
8. plt.ylabel('Y-axis')  
9. plt.title('Grafico a Linee Personalizzato')  
10. plt.show()
```



**Domanda due:**

**Qual è la differenza tra `plt.plot()` e `plt.scatter()` e in quali situazioni è meglio usare uno piuttosto che l'altro?**



La differenza principale tra `plt.plot()` e `plt.scatter()` sta nel modo in cui visualizzano i dati: `plt.plot()` è ideale per rappresentare dati continui come serie temporali o funzioni, mentre `plt.scatter()` è più adatto per rappresentare punti discreti, come misurazioni indipendenti o correlazioni tra variabili. `plt.plot()` crea linee che collegano i punti, mentre `plt.scatter()` mostra solo i punti singoli senza collegarli.



```
1. import matplotlib.pyplot as plt  
2.  
3. x = [1, 2, 3, 4, 5]  
4. y = [2, 4, 6, 8, 10]  
5.  
6. plt.plot(x, y, label='Grafico a Linee') # Linea continua  
7. plt.scatter(x, y, color='red', label='Grafico a Punti') # Punti sparsi  
8. plt.xlabel('X-axis')  
9. plt.ylabel('Y-axis')  
10. plt.legend()  
11. plt.title('Differenza tra Line Plot e Scatter Plot')  
12.  
13. plt.show()
```



**Domanda tre:**

**Come posso aggiungere una legenda e personalizzarne la posizione e lo stile in un grafico?**



In matplotlib, la legenda può essere aggiunta usando `plt.legend()`, e si posiziona automaticamente se i dati hanno un'etichetta (con `label` nei metodi di `plot`). Possiamo specificare la posizione usando il parametro `loc` (ad esempio, `loc='upper left'`).

Inoltre, è possibile personalizzare altri aspetti come la trasparenza e la dimensione della legenda.





```
1.import matplotlib.pyplot as plt  
2.  
3.x = [1, 2, 3, 4, 5]  
4.y1 = [1, 4, 9, 16, 25]  
5.y2 = [2, 4, 6, 8, 10]  
6.  
7.plt.plot(x, y1, label='Quadrati', color='green')  
8.plt.plot(x, y2, label='Doppio', color='purple')  
9.plt.xlabel('X-axis')  
10.plt.ylabel('Y-axis')  
11.plt.title('Grafico con Legenda Personalizzata')  
12.plt.legend(loc='upper left', fontsize='small', frameon=True,  
    shadow=True) # Posizione e stile  
13.plt.show()
```



## **Domanda Quattro:**

**Come posso salvare un grafico generato con matplotlib in diversi formati (ad esempio PNG, PDF) e con una risoluzione specifica?**



Matplotlib permette di salvare i grafici usando `plt.savefig()`, specificando il nome del file e il formato desiderato (come `.png`, `.pdf`, `.jpg`).

Si può anche impostare una risoluzione personalizzata con il parametro `dpi` (dots per inch), ad esempio `dpi=300` per ottenere un'immagine ad alta risoluzione.

Questo metodo consente anche di salvare i grafici senza mostrare la finestra del plot.



```
1. import matplotlib.pyplot as plt  
2.  
3. x = [1, 2, 3, 4, 5]  
4. y = [1, 4, 9, 16, 25]  
5.  
6. plt.plot(x, y, color='green')  
7. plt.xlabel('X-axis')  
8. plt.ylabel('Y-axis')  
9. plt.title('Grafico Salvato')  
10. plt.savefig('grafico_alta_risoluzione.png', dpi=300) #  
    Salva il grafico in PNG con risoluzione 300 dpi  
11. plt.show()
```



**Domanda cinque:**

**Come posso creare un grafico con più subplot e personalizzare ciascun subplot individualmente (titoli, colori, assi)?**



Per creare più subplot, si usa `plt.subplot()`, che permette di dividere la finestra grafica in una griglia e definire il contenuto di ciascuna "cella".

Ad esempio, `plt.subplot(2, 1, 1)` crea il primo di due subplot sovrapposti verticalmente.

Ogni subplot può essere personalizzato singolarmente con etichette, colori, e titoli propri.



```
1. import matplotlib.pyplot as plt
2.
3. x = [1, 2, 3, 4, 5]
4. y1 = [1, 4, 9, 16, 25]
5. y2 = [2, 3, 4, 5, 6]
6.
7. plt.figure(figsize=(10, 5)) # Imposta le dimensioni della figura
8.
9. # Primo subplot
10. plt.subplot(2, 1, 1) # Due righe, una colonna, primo subplot
11. plt.plot(x, y1, color='blue')
12. plt.title('Quadrati')
13. plt.xlabel('X-axis')
14. plt.ylabel('Y-axis')
15.
16. # Secondo subplot
17. plt.subplot(2, 1, 2) # Due righe, una colonna, secondo subplot
18. plt.plot(x, y2, color='red')
19. plt.title('Incremento Lineare')
20. plt.xlabel('X-axis')
21. plt.ylabel('Y-axis')
22.
23. plt.tight_layout() # Adatta i subplot per evitare sovrapposizioni
24.
25. plt.show()
```



**Domanda Sei:**

**Come posso aggiungere annotazioni a un grafico in matplotlib per evidenziare punti specifici o eventi particolari?**





In matplotlib, si possono aggiungere annotazioni ai grafici utilizzando il metodo `plt.annotate()`, che permette di evidenziare punti o eventi specifici con testo e frecce.

Questo è particolarmente utile per spiegare punti dati o segnare valori chiave.

Con `plt.annotate()`, si specifica il testo, le coordinate del punto da annotare (`xy`) e, opzionalmente, la posizione del testo rispetto al punto (`xytext`), insieme allo stile della freccia (`arrowprops`) se desiderato.



```
1. import matplotlib.pyplot as plt  
2.  
3. x = [1, 2, 3, 4, 5]  
4. y = [1, 4, 9, 16, 25]  
5.  
6. plt.plot(x, y, marker='o', color='blue')  
7. plt.xlabel('X-axis')  
8. plt.ylabel('Y-axis')  
9. plt.title('Grafico con Annotazione')  
10.  
11. # Aggiungere un'annotazione per il punto più alto  
12. plt.annotate('Punto massimo', xy=(5, 25), xytext=(3, 20),  
13.           arrowprops=dict(facecolor='black', shrink=0.05),  
14.           fontsize=10, color='red')  
15.  
16. plt.show()
```



**Buon MasterD a tutti**

