Tkinter

1. Tkinter
2. Moduli
3. Storia
4. Esercizio 1 convertitore di misura: label, button, entry, listbox
5. Canvas in tkinter
6. Costrutto `if __name__ == "__main__":`
7. Turtle incorporato dentro tkinter

1. Tkinter in Python

Cos'è Tkinter?
Tkinter è la libreria standard di Python per la creazione di interfacce grafiche utente (GUI). Fornisce un modo semplice e potente per sviluppare applicazioni desktop con un aspetto nativo su diverse piattaforme, tra cui Windows, macOS e Linux.
Perché è così famosa?

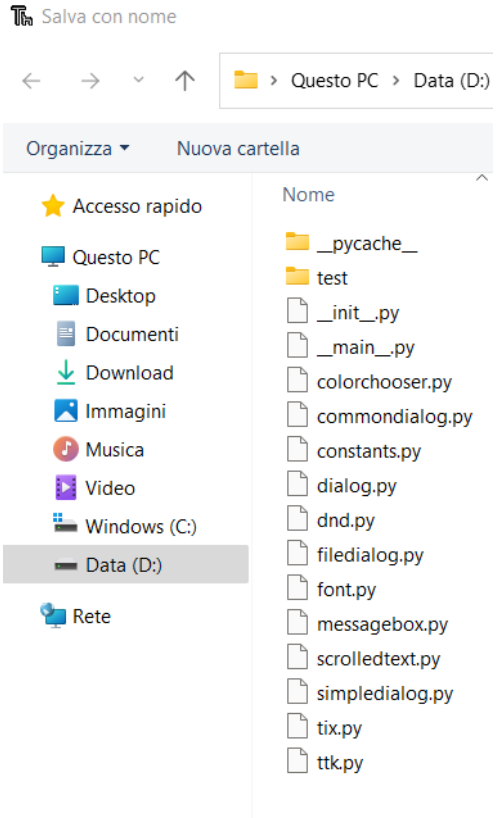
- **Inclusa nella libreria standard di Python**: Tkinter fa parte della libreria standard di Python, il che significa che è già disponibile in qualsiasi installazione di Python. Ciò la rende estremamente accessibile e facile da usare per i principianti.
- **Semplicità**: Tkinter è relativamente semplice da imparare e utilizzare, soprattutto per chi ha già familiarità con Python. La sua sintassi è intuitiva e offre un buon equilibrio tra funzionalità e facilità d'uso.
- **Multipiattaforma**: le applicazioni Tkinter funzionano su diverse piattaforme senza richiedere modifiche significative al codice. Ciò consente agli sviluppatori di creare applicazioni desktop che possono essere eseguite su diversi sistemi operativi.
- **Ampia gamma di widget**: Tkinter offre una vasta gamma di componenti predefiniti, come pulsanti, etichette, caselle di testo, menu e altro ancora. Questi componenti consentono di creare interfacce utente complesse e interattive.
- **Comunità di supporto**: Tkinter ha una vasta comunità di utenti e sviluppatori che offrono supporto, documentazione ed esempi di codice. Ciò rende facile trovare aiuto e risorse quando si lavora con Tkinter.

Aspetti negativi di Tkinter

- **Aspetto obsoleto**: L'aspetto predefinito dei widget Tkinter può sembrare un po' datato rispetto alle interfacce utente moderne. Tuttavia, è possibile personalizzare l'aspetto dei widget utilizzando i temi.
- **Limitazioni**: Tkinter potrebbe non essere la scelta migliore per applicazioni con interfacce utente estremamente complesse o per chi richiede prestazioni grafiche elevate.

In sintesi
Tkinter è una libreria eccellente per chi vuole iniziare a creare applicazioni desktop con Python. La sua semplicità, la sua portabilità e la sua inclusione nella libreria standard di Python la rendono una scelta popolare tra i principianti e gli sviluppatori che cercano un modo rapido e semplice per creare GUI.

2. Moduli




```python
class Button(Widget):
    """Button widget."""

    def __init__(self, master=None, cnf={}, **kw):
        """Construct a button widget with the parent MASTER.

        STANDARD OPTIONS

            activebackground, activeforeground, anchor,
            background, bitmap, borderwidth, cursor,
            disabledforeground, font, foreground
            highlightbackground, highlightcolor,
            highlightthickness, image, justify,
            padx, pady, relief, repeatdelay,
            repeatinterval, takefocus, text,
            textvariable, underline, wraplength

        WIDGET-SPECIFIC OPTIONS

            command, compound, default, height,
            overrelief, state, width
        """
        Widget.__init__(self, master, 'button', cnf, kw)

    def flash(self):
        """Flash the button.

        This is accomplished by redisplaying
        the button several times, alternating between active and
        normal colors. At the end of the flash the button is left
        in the same normal/active state as when the command was
        invoked. This command is ignored if the button's state is
        disabled.
        """
        self.tk.call(self._w, 'flash')

    def invoke(self):
        """Invoke the command associated with the button.

        The return value is the return value from the command,
        or an empty string if there is no command associated with
```

```python
class Widget(BaseWidget, Pack, Place, Grid):
    """Internal class.

    Base class for a widget which can be posi
    Pack, Place or Grid."""
    pass
```

La libreria Tkinter in Python è composta da diversi moduli, ognuno con funzioni specifiche per la creazione di interfacce grafiche. Ecco i moduli principali:

- **tkinter**:
  È il modulo principale che fornisce le classi per creare i widget di base, come finestre, pulsanti, etichette e caselle di testo.
- **tkinter.ttk**:
  Introdotto nelle versioni più recenti di Tk, offre widget con un aspetto più moderno e conforme agli standard del sistema operativo.
- **tkinter.messagebox**:
  Consente di creare finestre di dialogo per visualizzare messaggi di avviso, errore o conferma.
- **tkinter.filedialog**:
  Fornisce funzioni per aprire e salvare file tramite finestre di dialogo standard del sistema operativo.
- **tkinter.colorchooser**:
  Permette di selezionare colori tramite una finestra di dialogo interattiva.
- **tkinter.font**:
  Fornisce le classi per personalizzare i tipi di carattere utilizzati nei widget.
- **tkinter.scrolledtext**:
  Consente di visualizzare e modificare testi lunghi con barre di scorrimento integrate.

- **tkinter.simpledialog**:
  offre finestre di dialogo semplici per ottenere input dall'utente.

Questi moduli lavorano insieme per fornire un set completo di strumenti per la creazione di interfacce grafiche in Python.

3. Storia di tkinter

La storia di Tkinter è strettamente legata a quella di Tcl/Tk, il toolkit GUI su cui si basa. Ecco una panoramica della sua evoluzione:

- **Origini in Tcl/Tk**:
  Tkinter è un "wrapper" Python per Tcl/Tk, un toolkit GUI multipiattaforma creato da John Ousterhout negli anni '80.
  Tcl (Tool Command Language) è un linguaggio di scripting, e Tk è la sua estensione per la creazione di interfacce grafiche.
- **Integrazione in Python**:
  Tkinter è stato incluso nella libreria standard di Python, rendendolo disponibile "out-of-the-box" per tutti gli sviluppatori Python.
  Questa inclusione ha contribuito significativamente alla sua popolarità, specialmente tra i principianti.
- **Evoluzione e aggiornamenti**:
  Nel corso degli anni, Tkinter è stato aggiornato per supportare le nuove versioni di Tcl/Tk e per migliorare la sua funzionalità.
  tkinter.ttk è stato introdotto per fornire widget con un aspetto più moderno e conforme agli standard del sistema operativo.
- **Ruolo nella comunità Python**:
  Nonostante la presenza di librerie GUI più moderne, Tkinter rimane una scelta popolare per progetti semplici e per l'apprendimento della programmazione GUI.
  La sua semplicità e la sua disponibilità lo rendono uno strumento prezioso per gli sviluppatori Python.
- **Tkinter oggi**:
  Tkinter continua ad essere mantenuto e aggiornato come parte della libreria standard di Python.
  Sebbene possa non essere la scelta ideale per applicazioni GUI complesse, rimane uno strumento affidabile e facile da usare per molte esigenze.

In sintesi, Tkinter ha una storia lunga e consolidata, strettamente legata all'evoluzione di Python. La sua semplicità e la sua inclusione nella libreria standard lo hanno reso uno strumento fondamentale per la programmazione GUI in Python.

4. Canvas

Spiegazione di cos'è una Canvas in Tkinter, un esempio di utilizzo e come caricare un'icona e un'immagine:

Cos'è una Canvas in Tkinter?

In Tkinter, la Canvas è un widget versatile che permette di disegnare forme, immagini, testo e altri elementi grafici all'interno di una finestra. È come una **tela virtuale** su cui puoi dipingere e manipolare oggetti grafici.

Esempio di utilizzo

Questo esempio mostra come creare una finestra con una Canvas, disegnare un rettangolo, un cerchio e un testo, e caricare un'immagine e un'icona:

```
import tkinter as tk
from PIL import Image, ImageTk

def esempio_canvas():
    finestra = tk.Tk()
    finestra.title("Esempio Canvas")

    # Carica l'icona
```

```
    try:
        icona = tk.PhotoImage(file="icona.png")
        finestra.iconphoto(True, icona)
    except tk.TclError:
        print("Icona non trovata.")

    # Crea una Canvas
    canvas = tk.Canvas(finestra, width=400, height=300)
    canvas.pack()

    # Disegna un rettangolo
    canvas.create_rectangle(50, 50, 200, 150, fill="lightblue")

    # Disegna un cerchio
    canvas.create_oval(250, 50, 350, 150, fill="lightgreen")

    # Aggiungi del testo
    canvas.create_text(200, 250, text="Ciao dalla Canvas!", font=("Arial",
14))

    # Carica un'immagine
    try:
        immagine_pil = Image.open("immagine.jpg")
        immagine = ImageTk.PhotoImage(immagine_pil)
        canvas.create_image(100, 250, image=immagine)
    except FileNotFoundError:
        print("Immagine non trovata.")

    finestra.mainloop():
          esempio_canvas()
```

```
6. Il costrutto if __name__ == "__main__":
```

```
if __name__ == "__main__":
```

In Python, l'istruzione if __name__ == "__main__": è un costrutto condizionale che verifica se il modulo Python corrente viene eseguito come programma principale o se viene importato come modulo in un altro programma.

Spiegazione dettagliata

- **__name__**: è una variabile speciale in Python che contiene il nome del modulo corrente.
- **__main__**: è un valore speciale che viene assegnato alla variabile __name__ quando il modulo Python viene eseguito come programma principale.

Quindi, l'istruzione if __name__ == "__main__": verifica se il nome del modulo corrente è uguale a __main__. Se la condizione è vera, significa che il modulo viene eseguito come programma principale e il codice all'interno del blocco if viene eseguito. Se la condizione è falsa, significa che il modulo viene importato come modulo in un altro programma e il codice all'interno del blocco if non viene eseguito.

Utilizzo

L'istruzione if __name__ == "__main__": viene comunemente utilizzata per:

- Definire il punto di ingresso principale di un programma Python.
- Eseguire codice di test o di debug solo quando il modulo viene eseguito come programma principale.
- Impedire l'esecuzione di codice indesiderato quando il modulo viene importato come modulo in un altro programma.

Esempio

```
# modulo.py

def mia_funzione():
  print("Funzione eseguita")

if __name__ == "__main__":
  # Questo codice viene eseguito solo se il modulo viene eseguito come
programma principale
  mia_funzione()
  print("Modulo eseguito come programma principale")
```

Se esegui questo modulo direttamente con python modulo.py, l'output sarà:

```
Funzione eseguita
Modulo eseguito come programma principale
```

Tuttavia, se importi questo modulo in un altro programma, il codice all'interno del blocco if non verrà eseguito.

7. turtle incorporato dentro tkinter, differenza tra Screen e turtleScreen:

La differenza tra turtle.Screen e turtle.TurtleScreen in Python:

turtle.Screen

- turtle.Screen() è una funzione che restituisce un oggetto singleton di una sottoclasse di TurtleScreen.
- Questo significa che restituisce sempre la stessa istanza dell'oggetto schermo.
- È pensata per essere usata quando turtle viene usato come strumento autonomo per la grafica.

turtle.TurtleScreen

- turtle.TurtleScreen è la classe base per gli oggetti schermo.
- Quando turtle è incorporato in un programma Tkinter, si usa TurtleScreen per creare più schermi di tartaruga o più canvas di tartaruga all'interno della stessa finestra.
- In sostanza definisce le funzionalità di uno schermo per la grafica della tartaruga.

In parole semplici:

- turtle.Screen() è un modo comodo per ottenere un singolo schermo di tartaruga quando stai solo usando turtle per creare della grafica.
- turtle.TurtleScreen è più utile quando stai integrando turtle in un'applicazione Tkinter più grande e hai bisogno di un controllo più fine sulla creazione e gestione degli schermi.

Ulteriori considerazioni:

- La libreria turtle può essere utilizzata sia come strumento grafico autonomo, sia come strumento grafico integrato in un'applicazione Tkinter.
- Quando si utilizza turtle in modo autonomo, la funzione turtle.Screen() restituisce un oggetto singleton.
- Quando turtle è incorporato in un'applicazione Tkinter, la classe turtle.TurtleScreen viene utilizzata per creare schermi multipli.

Spero che questo ti aiuti a capire la differenza tra turtle.Screen e turtle.TurtleScreen.