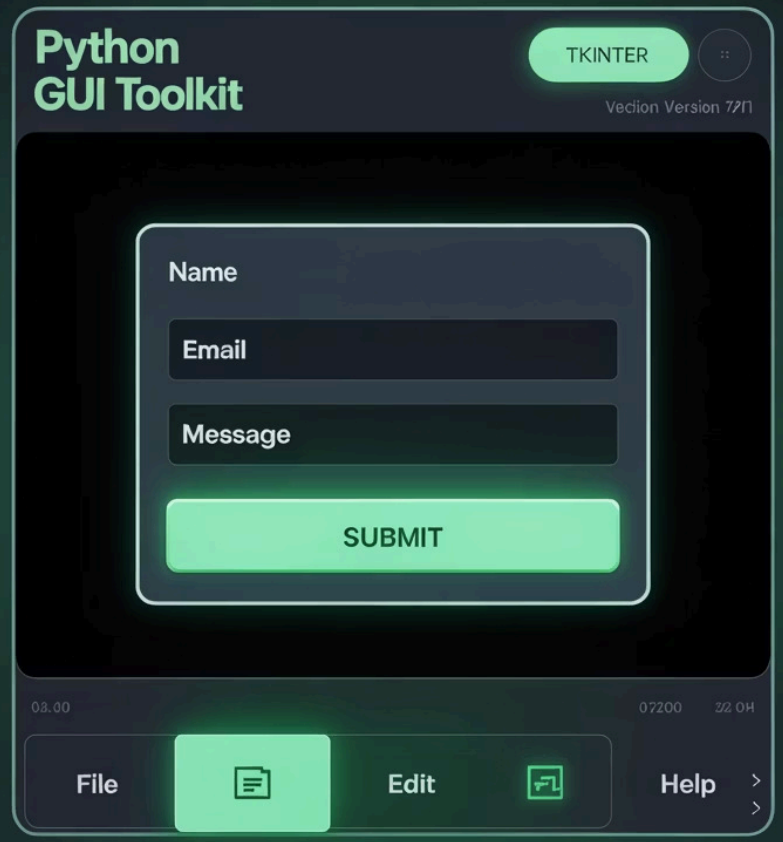


Creare Interfacce Grafiche con Tkinter: Guida per Principianti

Benvenuti a questo corso introduttivo su Tkinter, la libreria standard di Python per la creazione di interfacce grafiche utente (GUI). Questa presentazione vi guiderà attraverso i passaggi fondamentali per costruire la vostra prima applicazione con finestre, pulsanti, etichette e campi di testo.

Imparerete a creare elementi interattivi, personalizzarli e organizzarli in un'interfaccia funzionale. Alla fine di questo percorso, avrete tutte le competenze necessarie per sviluppare applicazioni desktop con interfacce intuitive e professionali.

Creazione della Finestra Principale

Il primo passo per creare un'applicazione Tkinter è generare la finestra principale che conterrà tutti gli altri elementi dell'interfaccia. Questa finestra funziona come un contenitore per tutti i widget che aggiungeremo successivamente.

Per creare la finestra principale, dobbiamo prima importare la libreria Tkinter e poi inizializzare un oggetto Tk(). Questo oggetto rappresenta la finestra dell'applicazione e sarà il punto di partenza per costruire la nostra interfaccia.

```
import tkinter as tk

# Creazione della finestra principale
finestra = tk.Tk()

# Avvio del loop principale
finestra.mainloop()
```

Il metodo mainloop() è fondamentale: mantiene la finestra aperta e gestisce tutti gli eventi dell'interfaccia utente, come i clic del mouse e la pressione dei tasti.

Personalizzazione del Titolo della Finestra

Codice

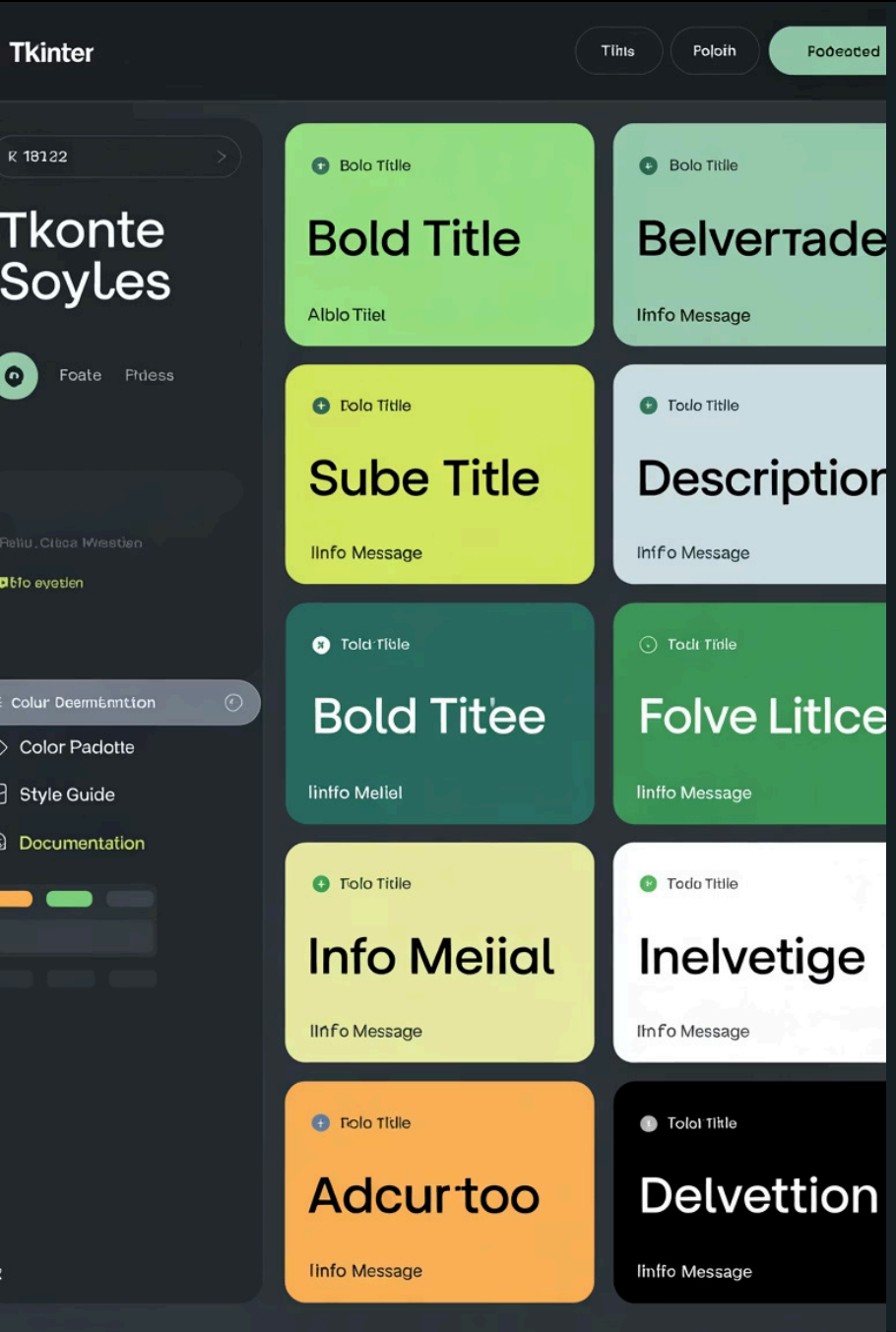
```
import tkinter as tk

finestra = tk.Tk()
finestra.title("La Mia Prima Applicazione")

finestra.mainloop()
```

Il metodo .title() ci permette di impostare il testo che apparirà nella barra del titolo della finestra. È importante scegliere un titolo descrittivo che rifletta lo scopo dell'applicazione.

Personalizzazioni Aggiuntive

Oltre al titolo, possiamo personalizzare altri aspetti della finestra principale:

- Dimensioni con geometry("larghezzaxaltezza")
- Icona con iconbitmap("percorso.ico")
- Colore di sfondo con configure(bg="colore")
- Impedire il ridimensionamento con resizable(False, False)



Aggiunta di Etichette (Label)

Creazione Base

Le etichette (Label) sono widget utilizzati per visualizzare testo o immagini non modificabili dall'utente. Sono componenti statici ideali per titoli, descrizioni o istruzioni.

```
etichetta = tk.Label(finestra,
text="Benvenuto!")
etichetta.pack()
```

Personalizzazione

Possiamo personalizzare l'aspetto delle etichette con vari parametri come font, colore del testo e dello sfondo, allineamento e padding.

```
etichetta = tk.Label(
    finestra,
    text="Titolo Principale",
    font=("Arial", 16, "bold"),
    fg="blue",
    bg="lightgray"
)
```

Posizionamento

Le etichette devono essere posizionate nella finestra utilizzando uno dei gestori di layout come pack(), grid() o place() che vedremo più avanti.

Personalizzazione delle Label

Font Personalizzati

Puoi definire famiglia, dimensione e stile del font usando una tupla: font=("Helvetica", 12, "bold"). Gli stili disponibili includono "bold", "italic" e "underline", che possono essere combinati.

Colori

Il parametro fg imposta il colore del testo, mentre bg definisce il colore dello sfondo. Puoi usare nomi di colore come "red" o codici esadecimali come "#FF5733".

Dimensioni e Padding

Con width e height imposti la dimensione, mentre padx e pady aggiungono spazio attorno al testo. Il parametro wraplength controlla quando il testo deve andare a capo.

Allineamento

Il parametro anchor determina dove posizionare il testo all'interno della Label, usando punti cardinali come "n", "s", "e", "w", "center".

La personalizzazione delle Label è fondamentale per creare interfacce esteticamente gradevoli e funzionali, rendendo l'informazione facilmente leggibile all'utente.

Creazione di Pulsanti (Button)

Creazione Base

I pulsanti sono elementi interattivi che eseguono azioni quando vengono cliccati. Si creano con la classe Button e richiedono almeno il parametro parent (la finestra o il frame che li contiene).

```
pulsante = tk.Button(finestra, text="Clicca qui")
pulsante.pack()
```

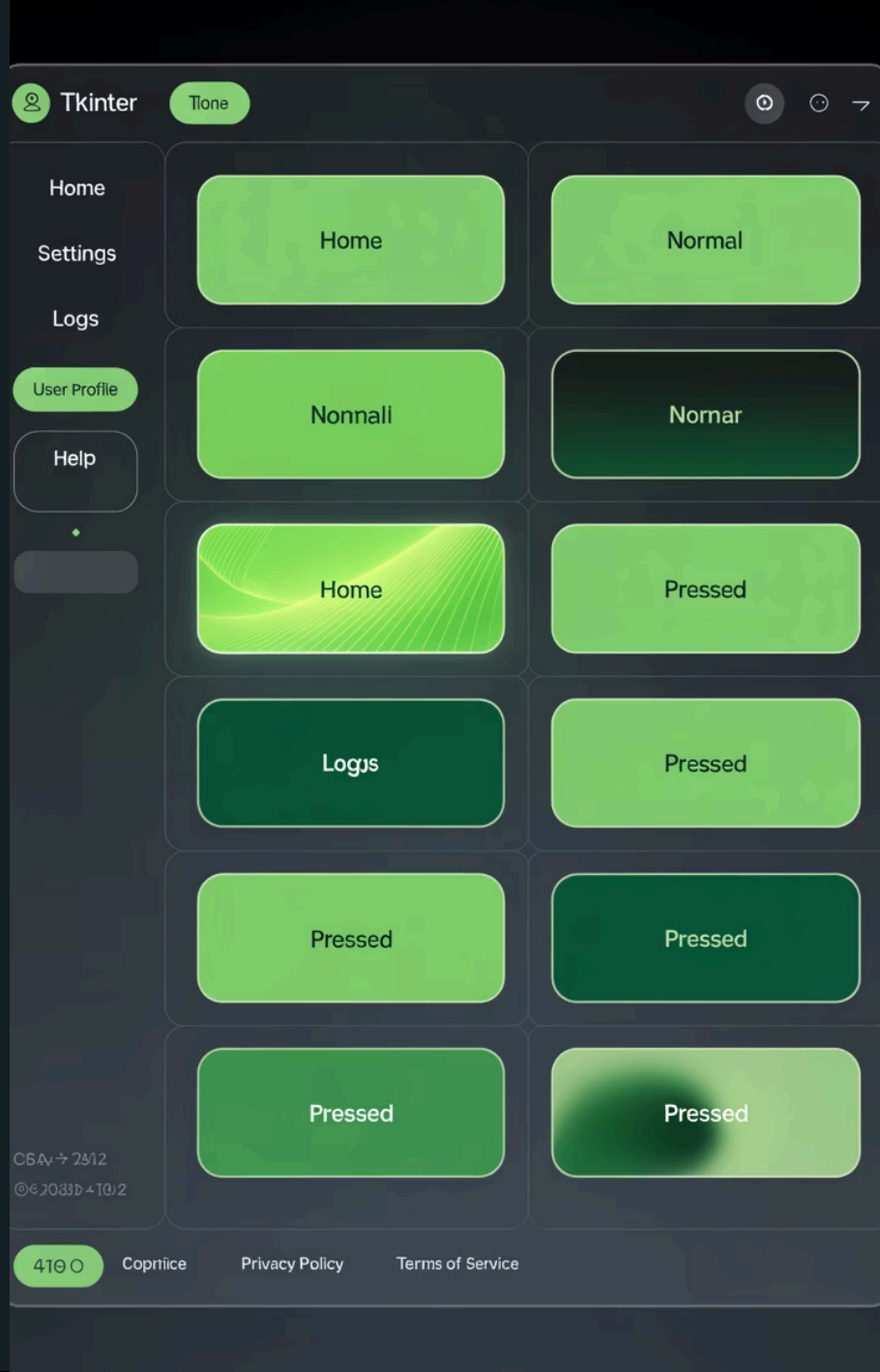
Personalizzazione

Come per le etichette, è possibile personalizzare l'aspetto dei pulsanti con parametri come font, fg, bg, width, height, borderwidth e relief per il tipo di bordo (flat, raised, sunken, ridge).

Collegamento alla Funzione

Per rendere il pulsante funzionale, dobbiamo collegarlo a una funzione da eseguire quando viene premuto, usando il parametro command.

```
def azione():
    print("Pulsante premuto!")

pulsante = tk.Button(
    finestra,
    text="Esegui",
    command=azione
)
```



Campi di Input (Entry)

Creazione

Il widget Entry permette di inserire una singola riga di testo. È ideale per raccogliere input brevi come nomi, email o password.

```
campo_testo =
tk.Entry(finestra)
campo_testo.pack()
```

Configurazione

Possiamo configurare vari aspetti: larghezza con width, colore di sfondo con bg, colore del testo con fg, e bordo con borderwidth.

```
campo_testo = tk.Entry(
    finestra,
    width=30,
    bg="white",
    fg="black",
    borderwidth=2
)
```

Mascheramento

Per i campi password, possiamo nascondere il testo impostando il parametro show con un carattere che sostituirà visivamente ciò che viene digitato.

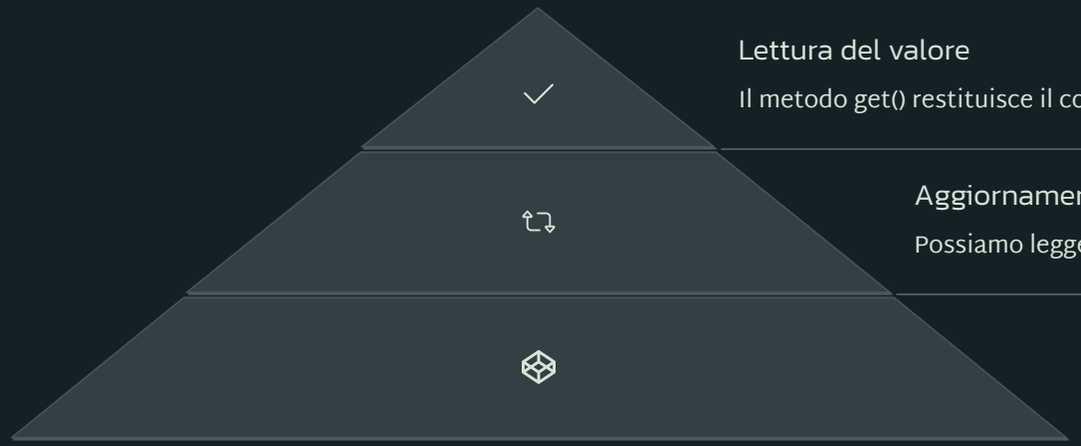
```
password = tk.Entry(finestra,
show="*")
```

Recupero Dati

Per ottenere il testo inserito, utilizziamo il metodo get() che restituisce il contenuto attuale del campo Entry.

```
testo_inserito =
campo_testo.get()
```

Recupero del Testo dai Campi Entry



Lettura del valore

Il metodo get() restituisce il contenuto attuale del campo

Aggiornamento in tempo reale

Possiamo leggere il valore in qualsiasi momento

Implementazione in una funzione

Tipicamente usato all'interno dell'handler di un evento

Quando vogliamo elaborare l'input dell'utente, dobbiamo recuperare il testo dal campo Entry. Questo viene fatto tipicamente in risposta a un evento, come il clic su un pulsante. Ecco un esempio completo di come implementare questa funzionalità:

```
def elabora_input():
    nome_utente = campo_nome.get()
    if nome_utente:
        risultato.config(text=f"Benvenuto, {nome_utente}!")
    else:
        risultato.config(text="Per favore, inserisci il tuo nome.")

campo_nome = tk.Entry(finestra, width=20)
campo_nome.pack(pady=10)

pulsante = tk.Button(finestra, text="Conferma", command=elabora_input)
pulsante.pack(pady=5)

risultato = tk.Label(finestra, text="")
risultato.pack(pady=10)
```

Aree di Testo Multilinea (Text)

Creazione Base

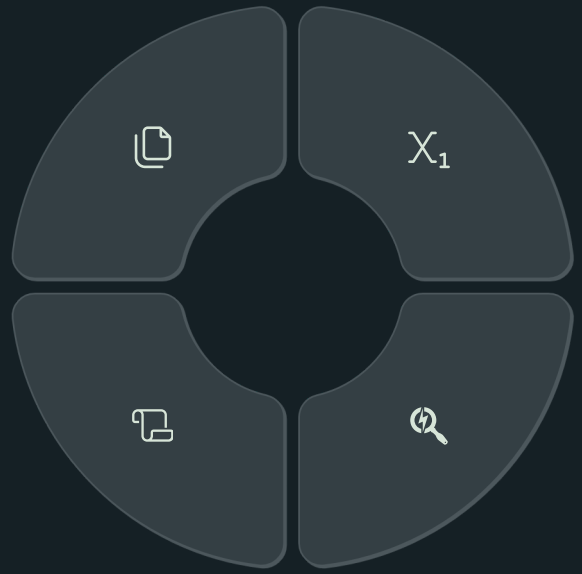
Il widget Text permette l'inserimento di testo su più righe, ideale per note, commenti o documenti.

```
area_testo = tk.Text(finestra,
                     height=5,
                     width=30)
area_testo.pack()
```

Aggiunta Scrollbar

Per testi lunghi, è utile aggiungere una barra di scorrimento verticale.

```
scrollbar = tk.Scrollbar(finestra)
area_testo.config(yscrollcommand=
                  scrollbar.set)
scrollbar.config(command=
                 area_testo.yview)
```



Inserimento Contenuto

Possiamo precompilare il widget con del testo usando il metodo insert().

```
area_testo.insert(tk.END,
    "Scrivi qui il tuo testo...")
```

Recupero Contenuto

Per ottenere tutto il testo inserito, usiamo il metodo get() specificando gli indici di inizio e fine.

```
contenuto = area_testo.get("1.0", tk.END)
```

Recupero del Contenuto dai Campi Text

Indici di posizione

Text usa un sistema di indici in formato "linea.carattere"

Recupero completo

Per ottenere tutto il testo dall'inizio alla fine

Recupero parziale

Per estrarre solo porzioni specifiche del testo

Il widget Text utilizza un sistema di indici speciale per identificare le posizioni nel testo. L'indice "1.0" si riferisce alla prima riga (1), primo carattere (0). L'indice tk.END rappresenta la fine del testo.

```
# Recupero completo del testo
contenuto_completo = area_testo.get("1.0", tk.END)

# Recupero della prima riga
prima_riga = area_testo.get("1.0", "1.end")

# Recupero di un intervallo specifico (dalla riga 2, carattere 5 alla riga 3, carattere 10)
frammento = area_testo.get("2.5", "3.10")

# Nota: il testo recuperato contiene sempre un carattere newline alla fine
# Per rimuoverlo, possiamo usare:
testo_pulito = area_testo.get("1.0", tk.END).strip()
```

Questi metodi di recupero sono essenziali quando dobbiamo elaborare il testo inserito dall'utente o salvarlo in un file.

Gestione del Layout con Pack

3

Parametri principali

side, fill, expand sono i parametri fondamentali per controllare il posizionamento con pack()

4

Direzioni possibili

side può essere impostato su "top" (default), "bottom", "left" o "right" per determinare il lato di ancoraggio

2

Livelli di espansione

La combinazione di fill e expand controlla come i widget si adattano allo spazio disponibile

Il gestore di layout pack() è il modo più semplice per organizzare i widget in una finestra Tkinter. Funziona impilando i widget uno dopo l'altro nella direzione specificata dal parametro side.

```
# Widget in pila verticale (dall'alto verso il basso)
label1 = tk.Label(finestra, text="Prima etichetta")
label1.pack()  # Equivalente a pack(side="top")

label2 = tk.Label(finestra, text="Seconda etichetta")
label2.pack()

# Widget che si espande per riempire lo spazio disponibile
entry = tk.Entry(finestra)
entry.pack(fill="x", padx=10)  # Riempie in orizzontale con padding

# Widget che si espande in entrambe le direzioni
text = tk.Text(finestra, height=5)
text.pack(fill="both", expand=True, padx=10, pady=10)
```

Collegamento di Funzioni ai Pulsanti

Definizione della Funzione di Callback

Il primo passo è definire una funzione che verrà eseguita quando l'utente interagisce con il pulsante. Questa funzione può eseguire qualsiasi operazione, come modificare l'interfaccia, elaborare dati o comunicare con sistemi esterni.

```
def saluta_utente():
    nome = campo_nome.get()
    messaggio.config(text=f"Ciao, {nome}!")
```

Creazione del Pulsante con Command

Quando creiamo il pulsante, utilizziamo il parametro command per associare la nostra funzione. È importante passare il riferimento alla funzione senza parentesi, poiché non vogliamo eseguirla immediatamente ma solo quando il pulsante viene premuto.

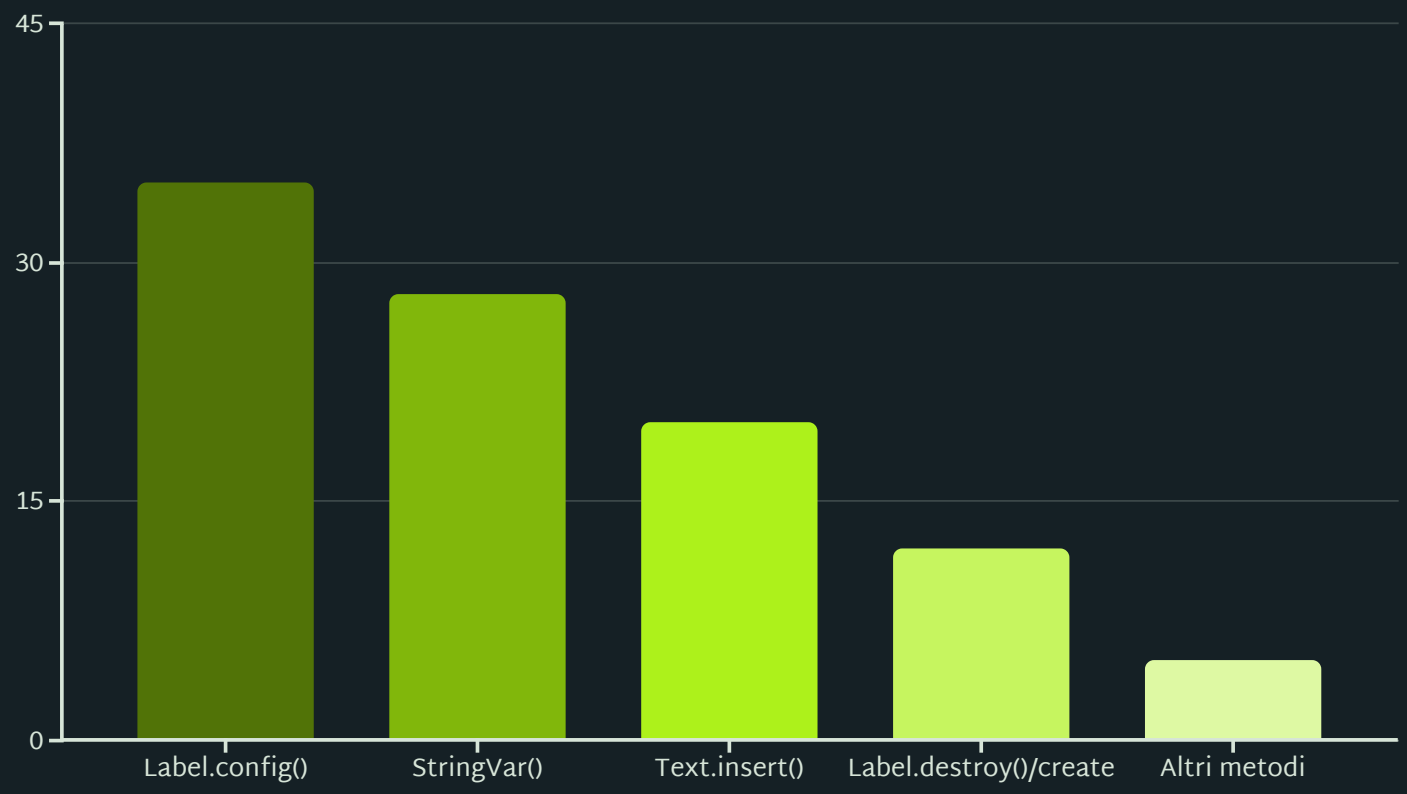
```
pulsante_saluto = tk.Button(
    finestra,
    text="Saluta",
    command=saluta_utente
)
```

Passaggio di Parametri (se necessario)

Se abbiamo bisogno di passare parametri alla funzione, possiamo utilizzare lambda per creare una funzione anonima che incapsula la chiamata con i parametri desiderati.

```
pulsante_colore = tk.Button(
    finestra,
    text="Cambia Colore",
    command=lambda:
cambia_colore("blue")
)
```

Visualizzazione Dinamica del Testo



La visualizzazione dinamica del testo è una funzionalità essenziale nelle applicazioni interattive. Con Tkinter, possiamo aggiornare il contenuto di una Label in risposta agli eventi dell'utente, come il clic su un pulsante o l'inserimento di testo in un campo.

Il metodo più comune è utilizzare config() per modificare le proprietà di un widget esistente. Questo approccio è più efficiente rispetto alla creazione di nuovi widget ogni volta che il contenuto cambia. Ecco un esempio completo:

```
def aggiorna_messaggio():
    nome = campo_nome.get()
    età = campo_età.get()
    if nome and età:
        try:
            età_num = int(età)
            risultato.config(
                text=f"Ciao {nome}, hai {età_num} anni!",
                fg="green"
            )
        except ValueError:
            risultato.config(
                text="L'età deve essere un numero!",
                fg="red"
            )
    else:
        risultato.config(
            text="Compila tutti i campi!",
            fg="orange"
        )
```

Organizzazione con Frame

Frame con Bordi

I Frame possono avere diversi stili di bordo impostando il parametro relief. Le opzioni includono "flat" (default), "raised", "sunken", "ridge", "solid" e "groove". Combinato con borderwidth, questo permette di creare interfacce visivamente strutturate.

Organizzazione Complessa

I Frame sono fondamentali per creare layout complessi, permettendo di raggruppare widget correlati e applicare gestori di layout differenti in diverse sezioni dell'interfaccia. Questo approccio modulare semplifica la gestione di interfacce con molti elementi.

LabelFrame

Una variante speciale è il LabelFrame, che aggiunge automaticamente un titolo al bordo del frame. Questo è particolarmente utile per categorizzare visivamente i controlli dell'interfaccia, migliorando l'usabilità per l'utente finale.

I Frame sono contenitori utilizzati per raggruppare widget correlati e organizzare l'interfaccia in sezioni logiche. Funzionano come "mini finestre" all'interno della finestra principale, con il proprio sistema di coordinate e layout.

```
# Creazione di due frame separati
frame_superiore = tk.Frame(finestra, bg="lightblue", padx=10, pady=10)
frame_superiore.pack(fill="x")

frame_inferiore = tk.Frame(finestra, bg="lightgreen", padx=10, pady=10)
frame_inferiore.pack(fill="both", expand=True)

# Aggiunta di widget ai frame
tk.Label(frame_superiore, text="Dati personali").pack()
tk.Entry(frame_superiore).pack(pady=5)

tk.Button(frame_inferiore, text="Salva").pack(side="right")
tk.Button(frame_inferiore, text="Annulla").pack(side="right", padx=5)
```

Esempio Completo: Interfaccia di Base

Elemento	Funzione
Finestra principale	Contenitore di base per l'applicazione
Frame superiore	Contiene i campi di input
Frame inferiore	Contiene l'area di testo e i pulsanti
Label e Entry	Raccolta dei dati dell'utente
Area Text	Visualizzazione dei risultati
Pulsanti	Controllo dell'applicazione

Ecco un esempio completo che combina tutti gli elementi visti finora in un'applicazione funzionale. Questa applicazione raccoglie informazioni dall'utente e le visualizza in un'area di testo quando si preme il pulsante.

```
import tkinter as tk

def elabora_dati():
    nome = campo_nome.get()
    età = campo_età.get()
    commento = area_commento.get("1.0", tk.END).strip()

    risultato = f"Nome: {nome}\nEtà: {età}\nCommento: {commento}"
    area_risultato.delete("1.0", tk.END)
    area_risultato.insert("1.0", risultato)

# Creazione della finestra principale
root = tk.Tk()
root.title("Esempio Completo Tkinter")
root.geometry("400x500")

# Frame superiore per i campi di input
frame_input = tk.Frame(root, padx=20, pady=10)
frame_input.pack(fill="x")

tk.Label(frame_input, text="Nome:").grid(row=0, column=0, sticky="w", pady=5)
campo_nome = tk.Entry(frame_input, width=30)
campo_nome.grid(row=0, column=1, pady=5)

tk.Label(frame_input, text="Età:").grid(row=1, column=0, sticky="w", pady=5)
campo_età = tk.Entry(frame_input, width=30)
campo_età.grid(row=1, column=1, pady=5)

tk.Label(frame_input, text="Commento:").grid(row=2, column=0, sticky="nw", pady=5)
area_commento = tk.Text(frame_input, height=4, width=30)
area_commento.grid(row=2, column=1, pady=5)

# Frame inferiore per i risultati e pulsanti
frame_output = tk.Frame(root, padx=20, pady=10)
frame_output.pack(fill="both", expand=True)

tk.Label(frame_output, text="Risultato:").pack(anchor="w")
area_risultato = tk.Text(frame_output, height=10, width=40)
area_risultato.pack(fill="both", expand=True, pady=10)

# Frame per i pulsanti
frame_pulsanti = tk.Frame(root, padx=20, pady=10)
frame_pulsanti.pack(fill="x")

pulsante_elabora = tk.Button(frame_pulsanti, text="Elabora Dati", command=elabora_dati)
pulsante_elabora.pack(side="left", padx=5)

pulsante_cancella = tk.Button(frame_pulsanti, text="Cancella",
                              command=lambda: [campo_nome.delete(0, tk.END),
                                               campo_età.delete(0, tk.END),
                                               area_commento.delete("1.0", tk.END),
                                               area_risultato.delete("1.0", tk.END)])
pulsante_cancella.pack(side="left", padx=5)

root.mainloop()
```