Bildschirmsteuerungen

auf voriges Fenster zurückspringen

Will man z.B. nach einer Messagebox wieder auf das vorherige Fenster zurückspringen, hilft folgendes Snippet

```
fenster.lift()
fenster.focus_force()
```

Parameter: fenster = der Fenstername.

Rückgabe: keine
**Beachte: ** bei Funktionen, die an sich kein Fenster haben, das übergeordnete Fenster im Aufruf der Funktion zu übergeben.

Fenster zentrieren

Will man ein Fenster in Breite und Höhe in Abhängigkeit von der Bildschirmauflösung zentrierenk hilft folgende Lösung.

```
# Bildschirmauflösung holen

    screen_width = window.winfo_screenwidth()
    screen_height = window.winfo_screenheight()
    # Falls Breite/Höhe nicht angegeben, skaliert berechnen
    width = width or int(screen_width * scale)
    height = height or int(screen_height * scale)
    # Begrenzung auf Bildschirmgröße
    width = min(width, screen_width)
    height = min(height, screen_height)
    # Zentrierte Position berechnen
    x = (screen_width - width) // 2
    y = (screen_height - height) // 2
    # DPI-Skalierung berücksichtigen (für 4K/HiDPI)
    try:
        window.tk.call('tk', 'scaling', window.winfo_fpixels('1i') / 72.0)
    except Exception:
        pass
    window.geometry(f"{width}x{height}+{x}+{y}")
```

Parameter:
window (tk.Tk | tk.Toplevel): Fensterobjekt
width (int, optional): Exakte Breite
height (int, optional): Exakte Höhe
scale (float, optional): Verhältnis zur Bildschirmauflösung (Standard: 0.8)
Rückgabe:

None

Theemed Widgets

`ttk` bringt einen einheitlichen Ansatz, um über vordefinierte oder eigene „Themes“ das gesamte Look & Feel deiner Anwendung zu steuern.

- Du legst mit `ttk.Style()` einen Style-Manager an.
- Dann definierst du unter einem Style-Namen („`<Name>.<WidgetType>`“) sämtliche Eigenschaften.
- Alle Widgets dieses Typs, die du mit genau diesem Style-Namen erzeugst, übernehmen die Einstellungen automatisch.

```
style = ttk.Style()
# Basis-Theme auswählen (z. B. 'clam', 'alt', 'default', 'classic')
style.theme_use('clam')

# Dann benutzt du ttk.Button mit diesem Style
close_button = ttk.Button(
    root,
    text="Schließen",
    command=root.destroy,
    style='My.TButton'
)
close_button.pack(side=tk.LEFT, padx=5, pady=10)
```

Vorteile von ttk.style

Zentrales Styling für viele Widgets auf einmal

Einheitliche Themes (z. B. helle und dunkle Varianten)

Plattform-nahes Aussehen (Buttons wie auf Windows, macOS, …)

From:
http://wiki.waldhofer.at/ - **Wiki von Franz**

Permanent link:
http://wiki.waldhofer.at/doku.php?id=python:snippets:bildschirm&rev=1746718120

Last update: **2025/05/08 17:28**