Text リファレンス

複数行のテキスト入力・編集を行う `Text` ウィジェットについての詳細なリファレンスです。

概要

`Text` ウィジェットは、複数行のテキストの表示や編集を行うための強力なコントロールです。単純なテキストエディタから複雑な文書処理まで、様々な用途に使用できます。文字の装飾、検索機能、スクロール機能なども提供します。

基本的な使用方法

シンプルなテキストエディタ

```
import tkinter as tk

def get_text():
    content = text_widget.get("1.0", tk.END)
    print(f"テキストの内容:\n{content}")

app = tk.Tk()
app.title("Textの例")
app.geometry("500x400")

text_widget = tk.Text(app, width=60, height=15)
text_widget.pack(pady=20)

button = tk.Button(app, text="テキストを取得", command=get_text)
button.pack()

app.mainloop()
```

クラスベースでのテキストエディタ

```
import tkinter as tk

class TextApp(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title("Textの例（クラスベース）")
        self.geometry("500x400")

        self.create_widgets()

    def create_widgets(self):
        self.text_widget = tk.Text(self, width=60, height=15)
        self.text_widget.pack(pady=20)

        self.button = tk.Button(self, text="テキストを取得", command=self.get_text)
        self.button.pack()

    def get_text(self):
        content = self.text_widget.get("1.0", tk.END)
        print(f"テキストの内容:\n{content}")

if __name__ == "__main__":
    app = TextApp()
    app.mainloop()
```

主要なオプション

オプション	説明
`width`	Text ウィジェットの幅を文字数で指定します。
`height`	Text ウィジェットの高さを行数で指定します。
`font`	フォントを指定。タプル `("フォント名", サイズ, "スタイル")` や文字列で指定。
`fg`（または `foreground`）	テキストの色。
`bg`（または `background`）	Text ウィジェットの背景色。
`wrap`	行の折り返し方法（`none`, `char`, `word`）。
`state`	Text ウィジェットの状態（`normal`, `disabled`）。
`relief`	境界線のスタイル（`flat`, `raised`, `sunken`, `groove`, `ridge`）。
`borderwidth`（または `bd`）	境界線の幅。

オプション	説明
`selectbackground`	選択されたテキストの背景色。
`selectforeground`	選択されたテキストの前景色。
`insertbackground`	カーソルの色。
`undo`	アンドゥ機能を有効にする (`True` / `False`)。

主要なメソッド

メソッド	説明
`get(start, end=None)`	指定範囲のテキストを取得します。
`insert(index, text)`	指定位置にテキストを挿入します。
`delete(start, end=None)`	指定範囲のテキストを削除します。
`replace(start, end, text)`	指定範囲のテキストを置換します。
`see(index)`	指定位置が見えるようにスクロールします。
`search(pattern, start, end=None)`	パターンでテキストを検索します。
`mark_set(name, index)`	指定位置にマークを設定します。
`mark_unset(name)`	マークを削除します。
`tag_add(tag, start, end=None)`	指定範囲にタグを追加します。
`tag_delete(tag)`	タグを削除します。

インデックスの指定方法

Text ウィジェットでは、テキストの位置を `"行.列"` の形式で指定します。

インデックス	説明
`"1.0"`	1行目の0列目（行の先頭）。
`"2.5"`	2行目の5列目。
`tk.END`	テキストの最後。
`tk.INSERT`	現在のカーソル位置。
`tk.SEL_FIRST`	選択範囲の開始位置。
`tk.SEL_LAST`	選択範囲の終了位置。

実用的な例

スクロールバー付きテキストエディタ

```
 import tkinter as tk
from tkinter import scrolledtext, messagebox, filedialog

class TextEditor(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("テキストエディタ")
        self.geometry("700x500")

        self.filename = None
        self.create_widgets()
        self.create_menu()

    def create_widgets(self):
        # スクロールバー付きテキストウィジェット
        self.text_area = scrolledtext.ScrolledText(
            self,
            wrap=tk.WORD,
            width=80,
            height=25,
            font=("Consolas", 11),
            undo=True  # アンドゥ機能を有効化
        )
        self.text_area.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # ステータスバー
        self.status_bar = tk.Label(
            self,
            text="準備完了",
            anchor=tk.W,
            relief=tk.SUNKEN,
            bg="lightgray"
        )
        self.status_bar.pack(side=tk.BOTTOM, fill=tk.X)
```

```
        # カーソル位置の更新をバインド
        self.text_area.bind('<KeyRelease>', self.update_cursor_position)
        self.text_area.bind('<Button-1>', self.update_cursor_position)

    def create_menu(self):
        menubar = tk.Menu(self)
        self.config(menu=menubar)

        # ファイルメニュー
        file_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="ファイル", menu=file_menu)
        file_menu.add_command(label="新規", command=self.new_file)
        file_menu.add_command(label="開く", command=self.open_file)
        file_menu.add_command(label="保存", command=self.save_file)
        file_menu.add_command(label="名前を付けて保存", command=self.save_as_file)
        file_menu.add_separator()
        file_menu.add_command(label="終了", command=self.quit)

        # 編集メニュー
        edit_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="編集", menu=edit_menu)
        edit_menu.add_command(label="元に戻す", command=lambda: self.text_area.edit_undo())
        edit_menu.add_command(label="やり直し", command=lambda: self.text_area.edit_redo())
        edit_menu.add_separator()
        edit_menu.add_command(label="切り取り", command=lambda: self.text_area.event_generate("<<Cut>>"))
        edit_menu.add_command(label="コピー", command=lambda: self.text_area.event_generate("<<Copy>>"))
        edit_menu.add_command(label="貼り付け", command=lambda: self.text_area.event_generate("<<Paste>>"))
        edit_menu.add_command(label="すべて選択", command=lambda: self.text_area.tag_add(tk.SEL, "1.0", tk.END))

    def new_file(self):
        if messagebox.askokcancel("新規ファイル", "現在の内容は失われます。続行しますか？"):
            self.text_area.delete("1.0", tk.END)
            self.filename = None
            self.title("テキストエディタ - 新規ファイル")

    def open_file(self):
        filename = filedialog.askopenfilename(
            title="ファイルを開く",
            filetypes=[("テキストファイル", "*.txt"), ("すべてのファイル", "*.*")]
        )
        if filename:
            try:
                with open(filename, 'r', encoding='utf-8') as file:
                    content = file.read()
                    self.text_area.delete("1.0", tk.END)
                    self.text_area.insert("1.0", content)
                    self.filename = filename
                    self.title(f"テキストエディタ - {filename}")
            except Exception as e:
                messagebox.showerror("エラー", f"ファイルを開けませんでした:\n{e}")

    def save_file(self):
        if self.filename:
            try:
                content = self.text_area.get("1.0", tk.END)
                with open(self.filename, 'w', encoding='utf-8') as file:
                    file.write(content)
                messagebox.showinfo("保存完了", "ファイルが保存されました。")
            except Exception as e:
                messagebox.showerror("エラー", f"ファイルを保存できませんでした:\n{e}")
        else:
            self.save_as_file()

    def save_as_file(self):
        filename = filedialog.asksaveasfilename(
            title="名前を付けて保存",
            defaultextension=".txt",
            filetypes=[("テキストファイル", "*.txt"), ("すべてのファイル", "*.*")]
        )
        if filename:
            try:
                content = self.text_area.get("1.0", tk.END)
                with open(filename, 'w', encoding='utf-8') as file:
                    file.write(content)
                self.filename = filename
                self.title(f"テキストエディタ - {filename}")
                messagebox.showinfo("保存完了", "ファイルが保存されました。")
            except Exception as e:
                messagebox.showerror("エラー", f"ファイルを保存できませんでした:\n{e}")

    def update_cursor_position(self, event=None):
        cursor_position = self.text_area.index(tk.INSERT)
        line, column = cursor_position.split('.')
        self.status_bar.config(text=f"行: {line}, 列: {column}")

if __name__ == "__main__":
    app = TextEditor()
    app.mainloop()
```

ベストプラクティス

プラクティス	説明
スクロールバーの追加	長いテキストを扱う場合は、`tkinter.scrolledtext.ScrolledText` を使用するか、独自にスクロールバーを追加します。

プラクティス	説明
インデックスの理解	Text ウィジェットのインデックス形式 `"行.列"` を正しく理解して使用します。
タグの活用	テキストの装飾や特別な動作には、タグ機能を活用します。
アンドゥ機能	ユーザビリティ向上のため、 `undo=True` オプションを設定してアンドゥ機能を有効にします。
イベントバインディング	キーボードやマウスイベントをバインドして、リアルタイムな操作を実現します。

参考リンク

- Python Docs - tkinter.Text
- TkDocs - Text