# Menu リファレンス

メニューバーやコンテキストメニューを作成する `Menu` ウィジェットについての詳細なリファレンスです。

## 概要

`Menu` ウィジェットは、アプリケーションにメニューバー、プルダウンメニュー、ポップアップメニュー（コンテキストメニュー）を提供するためのコントロールです。階層構造のメニューを作成でき、キーボードショートカット、アクセラレータキー、チェックメニュー、ラジオメニューなどの機能をサポートします。

## 基本的な使用方法

### シンプルなメニューバー

```
 import tkinter as tk
from tkinter import messagebox

def new_file():
    messagebox.showinfo("新規", "新しいファイルを作成します")

def open_file():
    messagebox.showinfo("開く", "ファイルを開きます")

def exit_app():
    app.quit()

app = tk.Tk()
app.title("Menuの例")
app.geometry("400x300")

# メニューバーを作成
menubar = tk.Menu(app)
app.config(menu=menubar)

# ファイルメニューを作成
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=new_file)
file_menu.add_command(label="開く", command=open_file)
file_menu.add_separator()
file_menu.add_command(label="終了", command=exit_app)

app.mainloop()
```

### クラスベースでのメニュー

```
 import tkinter as tk
from tkinter import messagebox

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

        self.title("Menuの例（クラスベース）")
        self.geometry("400x300")

        self.create_menu()

    def create_menu(self):
        # メニューバーを作成
        self.menubar = tk.Menu(self)
        self.config(menu=self.menubar)

        # ファイルメニューを作成
        file_menu = tk.Menu(self.menubar, tearoff=0)
        self.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_separator()
        file_menu.add_command(label="終了", command=self.quit)

        # 編集メニューを作成
        edit_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="編集", menu=edit_menu)
        edit_menu.add_command(label="元に戻す", command=self.undo)
        edit_menu.add_command(label="やり直し", command=self.redo)

    def new_file(self):
        messagebox.showinfo("新規", "新しいファイルを作成します")

    def open_file(self):
        messagebox.showinfo("開く", "ファイルを開きます")
```

```
    def undo(self):
        messagebox.showinfo("元に戻す", "操作を元に戻します")

    def redo(self):
        messagebox.showinfo("やり直し", "操作をやり直します")

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

## 主要なメソッド

| メソッド | 説明 |
|---|---|
| `add_command(label, command, ...)` | コマンドメニュー項目を追加します。 |
| `add_cascade(label, menu, ...)` | サブメニューを追加します。 |
| `add_checkbutton(label, variable, ...)` | チェックボックス型のメニュー項目を追加します。 |
| `add_radiobutton(label, variable, value, ...)` | ラジオボタン型のメニュー項目を追加します。 |
| `add_separator()` | メニューに区切り線を追加します。 |
| `delete(index1, index2=None)` | 指定したメニュー項目を削除します。 |
| `insert_command(index, label, command, ...)` | 指定位置にコマンドメニューを挿入します。 |
| `entryconfig(index, **options)` | メニュー項目の設定を変更します。 |

## 主要なオプション

### コマンドメニューのオプション

| オプション | 説明 |
|---|---|
| `label` | メニュー項目に表示するテキスト。 |
| `command` | メニュー項目が選択されたときに実行される関数。 |
| `accelerator` | ショートカットキーの表示文字列（実際の機能は別途実装が必要）。 |
| `underline` | アクセスキーとして使用する文字のインデックス。 |
| `state` | メニュー項目の状態（`normal`, `active`, `disabled`）。 |
| `font` | フォント。 |
| `foreground` | テキストの色。 |
| `background` | 背景色。 |

### その他のオプション

| オプション | 説明 |
|---|---|
| `tearoff` | メニューをウィンドウから切り離し可能にするかどうか（`0` または `1`）。 |

## 実用的な例

### 完全なメニューシステム

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

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

        self.title("完全なメニューシステム")
        self.geometry("600x400")

        # 状態変数
        self.word_wrap = tk.BooleanVar(value=True)
        self.view_mode = tk.StringVar(value="normal")

        self.create_widgets()
        self.create_menu()
        self.create_context_menu()

        # キーボードショートカットをバインド
        self.bind_shortcuts()

    def create_widgets(self):
        # メインテキストエリア
        self.text_area = tk.Text(self, wrap=tk.WORD, font=("Consolas", 11))
        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",
            relief=tk.SUNKEN,
            anchor=tk.W,
            bg="lightgray"
        )
        self.status_bar.pack(side=tk.BOTTOM, fill=tk.X)

    def create_menu(self):
        # メニューバー
        self.menubar = tk.Menu(self)
        self.config(menu=self.menubar)

        # ファイルメニュー
        file_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="ファイル", menu=file_menu)
        file_menu.add_command(label="新規", command=self.new_file, accelerator="Ctrl+N")
        file_menu.add_command(label="開く", command=self.open_file, accelerator="Ctrl+O")
        file_menu.add_command(label="保存", command=self.save_file, accelerator="Ctrl+S")
        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, accelerator="Ctrl+Q")

        # 編集メニュー
        edit_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="編集", menu=edit_menu)
        edit_menu.add_command(label="元に戻す", command=self.undo, accelerator="Ctrl+Z")
        edit_menu.add_command(label="やり直し", command=self.redo, accelerator="Ctrl+Y")
        edit_menu.add_separator()
        edit_menu.add_command(label="切り取り", command=self.cut, accelerator="Ctrl+X")
        edit_menu.add_command(label="コピー", command=self.copy, accelerator="Ctrl+C")
        edit_menu.add_command(label="貼り付け", command=self.paste, accelerator="Ctrl+V")
        edit_menu.add_separator()
        edit_menu.add_command(label="すべて選択", command=self.select_all, accelerator="Ctrl+A")
        edit_menu.add_command(label="検索", command=self.find, accelerator="Ctrl+F")

        # 表示メニュー
        view_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="表示", menu=view_menu)

        # 折り返しのチェックメニュー
        view_menu.add_checkbutton(
            label="行の折り返し",
            variable=self.word_wrap,
            command=self.toggle_word_wrap
        )
        view_menu.add_separator()

        # 表示モードのラジオメニュー
        view_menu.add_radiobutton(
            label="通常表示",
            variable=self.view_mode,
            value="normal",
            command=self.change_view_mode
        )
        view_menu.add_radiobutton(
            label="フルスクリーン",
            variable=self.view_mode,
            value="fullscreen",
            command=self.change_view_mode
        )

        # ツールメニュー
        tools_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="ツール", menu=tools_menu)
        tools_menu.add_command(label="文字数カウント", command=self.count_characters)
        tools_menu.add_command(label="大文字に変換", command=self.to_uppercase)
        tools_menu.add_command(label="小文字に変換", command=self.to_lowercase)

        # ヘルプメニュー
        help_menu = tk.Menu(self.menubar, tearoff=0)
        self.menubar.add_cascade(label="ヘルプ", menu=help_menu)
        help_menu.add_command(label="使い方", command=self.show_help)
        help_menu.add_command(label="バージョン情報", command=self.show_about)

    def create_context_menu(self):
        # コンテキストメニュー（右クリックメニュー）
        self.context_menu = tk.Menu(self, tearoff=0)
        self.context_menu.add_command(label="切り取り", command=self.cut)
        self.context_menu.add_command(label="コピー", command=self.copy)
        self.context_menu.add_command(label="貼り付け", command=self.paste)
        self.context_menu.add_separator()
        self.context_menu.add_command(label="すべて選択", command=self.select_all)

        # テキストエリアに右クリックイベントをバインド
        self.text_area.bind("<Button-3>", self.show_context_menu)  # Windows/Linux
        self.text_area.bind("<Button-2>", self.show_context_menu)  # macOS

    def bind_shortcuts(self):
        # キーボードショートカット
        self.bind("<Control-n>", lambda e: self.new_file())
        self.bind("<Control-o>", lambda e: self.open_file())
        self.bind("<Control-s>", lambda e: self.save_file())
        self.bind("<Control-q>", lambda e: self.quit())
        self.bind("<Control-z>", lambda e: self.undo())
        self.bind("<Control-y>", lambda e: self.redo())
        self.bind("<Control-x>", lambda e: self.cut())
        self.bind("<Control-c>", lambda e: self.copy())
```

```
        self.bind("<Control-v>", lambda e: self.paste())
        self.bind("<Control-a>", lambda e: self.select_all())
        self.bind("<Control-f>", lambda e: self.find())

    def show_context_menu(self, event):
        # コンテキストメニューを表示
        try:
            self.context_menu.tk_popup(event.x_root, event.y_root)
        finally:
            self.context_menu.grab_release()

    # ファイル操作
    def new_file(self):
        self.text_area.delete(1.0, tk.END)
        self.status_bar.config(text="新しいファイルを作成しました")

    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.status_bar.config(text=f"ファイルを開きました: {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):
        filename = filedialog.asksaveasfilename(
            title="ファイルを保存",
            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.status_bar.config(text=f"ファイルを保存しました: {filename}")
            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 save_as_file(self):
        self.save_file()

    # 編集操作
    def undo(self):
        try:
            self.text_area.edit_undo()
            self.status_bar.config(text="操作を元に戻しました")
        except tk.TclError:
            self.status_bar.config(text="元に戻す操作がありません")

    def redo(self):
        try:
            self.text_area.edit_redo()
            self.status_bar.config(text="操作をやり直しました")
        except tk.TclError:
            self.status_bar.config(text="やり直す操作がありません")

    def cut(self):
        self.text_area.event_generate("<<Cut>>")
        self.status_bar.config(text="切り取りました")

    def copy(self):
        self.text_area.event_generate("<<Copy>>")
        self.status_bar.config(text="コピーしました")

    def paste(self):
        self.text_area.event_generate("<<Paste>>")
        self.status_bar.config(text="貼り付けました")

    def select_all(self):
        self.text_area.tag_add(tk.SEL, "1.0", tk.END)
        self.status_bar.config(text="すべて選択しました")

    def find(self):
        messagebox.showinfo("検索", "検索機能（未実装）")

    # 表示操作
    def toggle_word_wrap(self):
        if self.word_wrap.get():
            self.text_area.config(wrap=tk.WORD)
            self.status_bar.config(text="行の折り返しを有効にしました")
        else:
            self.text_area.config(wrap=tk.NONE)
            self.status_bar.config(text="行の折り返しを無効にしました")

    def change_view_mode(self):
        mode = self.view_mode.get()
        if mode == "fullscreen":
            self.attributes('-fullscreen', True)
            self.status_bar.config(text="フルスクリーンモードに切り替えました")
        else:
            self.attributes('-fullscreen', False)
            self.status_bar.config(text="通常表示に切り替えました")
```

```
    # ツール操作
    def count_characters(self):
        content = self.text_area.get(1.0, tk.END)
        char_count = len(content) - 1  # 末尾の改行を除く
        word_count = len(content.split())
        line_count = content.count('\n')

        messagebox.showinfo(
            "文字数カウント",
            f"文字数: {char_count}\n単語数: {word_count}\n行数: {line_count}"
        )

    def to_uppercase(self):
        try:
            selected_text = self.text_area.get(tk.SEL_FIRST, tk.SEL_LAST)
            self.text_area.delete(tk.SEL_FIRST, tk.SEL_LAST)
            self.text_area.insert(tk.INSERT, selected_text.upper())
            self.status_bar.config(text="大文字に変換しました")
        except tk.TclError:
            messagebox.showwarning("警告", "テキストを選択してください")

    def to_lowercase(self):
        try:
            selected_text = self.text_area.get(tk.SEL_FIRST, tk.SEL_LAST)
            self.text_area.delete(tk.SEL_FIRST, tk.SEL_LAST)
            self.text_area.insert(tk.INSERT, selected_text.lower())
            self.status_bar.config(text="小文字に変換しました")
        except tk.TclError:
            messagebox.showwarning("警告", "テキストを選択してください")

    # ヘルプ操作
    def show_help(self):
        help_text = """
        tkinter テキストエディタ

        使い方:
        - ファイルメニューから新規作成、開く、保存ができます
        - 編集メニューから基本的な編集操作ができます
        - 表示メニューで表示設定を変更できます
        - ツールメニューで便利な機能を使用できます

        キーボードショートカット:
        - Ctrl+N: 新規
        - Ctrl+O: 開く
        - Ctrl+S: 保存
        - Ctrl+Z: 元に戻す
        - Ctrl+Y: やり直し
        - Ctrl+X: 切り取り
        - Ctrl+C: コピー
        - Ctrl+V: 貼り付け
        - Ctrl+A: すべて選択
        """
        messagebox.showinfo("使い方", help_text)

    def show_about(self):
        messagebox.showinfo(
            "バージョン情報",
            "tkinter テキストエディタ\nバージョン 1.0\n\nPython tkinter で作成"
        )

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

## ベストプラクティス

| プラクティス | 説明 |
|---|---|
| **tearoff=0 の使用** | `tearoff=0` を設定してメニューの切り離し機能を無効にします（現代的なUIでは一般的）。 |
| **適切な区切り** | `add_separator()` を使用して関連するメニュー項目をグループ分けします。 |
| **キーボードショートカット** | `accelerator` オプションでショートカットを表示し、実際のキーバインドも実装します。 |
| **コンテキストメニュー** | 右クリックメニューを提供してユーザビリティを向上させます。 |
| **状態管理** | `state` オプションを使用してメニュー項目の有効/無効を適切に管理します。 |

## 参考リンク

- Python Docs - tkinter.Menu
- TkDocs - Menu