Scale リファレンス

数値をスライダーで選択する `Scale` ウィジェットについての詳細なリファレンスです。

概要

`Scale` ウィジェットは、指定された範囲内の数値をスライダーで選択できるコントロールです。水平または垂直に配置でき、連続的または離散的な値の選択に使用されます。音量調整、明度調整、数値パラメータの設定などに適しています。

基本的な使用方法

シンプルなスケール

```
import tkinter as tk

def on_scale_change(value):
    print(f"スライダーの値: {value}")

app = tk.Tk()
app.title("Scaleの例")
app.geometry("300x200")

# 水平スケールの作成
scale = tk.Scale(
    app,
    from_=0,
    to=100,
    orient=tk.HORIZONTAL,
    command=on_scale_change
)
scale.set(50)  # 初期値を設定
scale.pack(pady=20)

app.mainloop()
```

クラスベースでのスケール

```
import tkinter as tk

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

        self.title("Scaleの例（クラスベース）")
        self.geometry("300x200")

        self.create_widgets()

    def create_widgets(self):
        # 水平スケールの作成
        self.scale = tk.Scale(
            self,
            from_=0,
            to=100,
            orient=tk.HORIZONTAL,
            command=self.on_scale_change
        )
        self.scale.set(50)  # 初期値を設定
        self.scale.pack(pady=20)

    def on_scale_change(self, value):
        print(f"スライダーの値: {value}")

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

主要なオプション

オプション	説明
`from_`	スケールの最小値。
`to`	スケールの最大値。
`orient`	スケールの向き（`tk.HORIZONTAL` または `tk.VERTICAL`）。
`resolution`	スケールの分解能（最小変化量）。デフォルトは1。
`variable`	`tk.IntVar` や `tk.DoubleVar` で値を管理。

オプション	説明
`command`	値が変更されたときに実行される関数。
`length`	スケールの長さをピクセルで指定。
`width`	スケールの幅をピクセルで指定。
`label`	スケールの上（または左）に表示するラベルテキスト。
`showvalue`	現在の値を表示するかどうか（`True` / `False`）。
`tickinterval`	目盛りの間隔。0の場合は目盛りを表示しない。
`font`	ラベルのフォント。
`fg`（または `foreground`）	テキストの色。
`bg`（または `background`）	背景色。
`relief`	境界線のスタイル。
`borderwidth`（または `bd`）	境界線の幅。

主要なメソッド

メソッド	説明
`get()`	現在の値を取得します。
`set(value)`	値を設定します。

実用的な例

色調整アプリケーション

```
 import tkinter as tk

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

        self.title("カラーミキサー")
        self.geometry("400x450")

        self.create_widgets()

    def create_widgets(self):
        # タイトル
        tk.Label(self, text="RGB カラーミキサー", font=("Arial", 16, "bold")).pack(pady=10)

        # 色表示用のキャンバス
        self.color_canvas = tk.Canvas(self, width=200, height=100, bg="black")
        self.color_canvas.pack(pady=10)

        # RGB値表示用のラベル
        self.rgb_label = tk.Label(self, text="RGB: (0, 0, 0)", font=("Arial", 12))
        self.rgb_label.pack(pady=5)

        # 16進値表示用のラベル
        self.hex_label = tk.Label(self, text="HEX: #000000", font=("Arial", 12))
        self.hex_label.pack(pady=5)

        # RGBスライダーのフレーム
        sliders_frame = tk.Frame(self)
        sliders_frame.pack(pady=20)

        # 赤 (Red) スライダー
        red_frame = tk.Frame(sliders_frame)
        red_frame.pack(pady=5)
        tk.Label(red_frame, text="Red:", width=6, fg="red", font=("Arial", 10, "bold")).pack(side=tk.LEFT)
        self.red_scale = tk.Scale(
            red_frame,
            from_=0,
            to=255,
            orient=tk.HORIZONTAL,
            length=200,
            command=self.update_color
        )
        self.red_scale.pack(side=tk.LEFT, padx=10)

        # 緑 (Green) スライダー
        green_frame = tk.Frame(sliders_frame)
        green_frame.pack(pady=5)
        tk.Label(green_frame, text="Green:", width=6, fg="green", font=("Arial", 10, "bold")).pack(side=tk.LEFT)
        self.green_scale = tk.Scale(
            green_frame,
            from_=0,
            to=255,
            orient=tk.HORIZONTAL,
            length=200,
            command=self.update_color
        )
        self.green_scale.pack(side=tk.LEFT, padx=10)
```

```
        # 青 (Blue) スライダー
        blue_frame = tk.Frame(sliders_frame)
        blue_frame.pack(pady=5)
        tk.Label(blue_frame, text="Blue:", width=6, fg="blue", font=("Arial", 10, "bold")).pack(side=tk.LEFT)
        self.blue_scale = tk.Scale(
            blue_frame,
            from_=0,
            to=255,
            orient=tk.HORIZONTAL,
            length=200,
            command=self.update_color
        )
        self.blue_scale.pack(side=tk.LEFT, padx=10)

        # プリセットボタン
        preset_frame = tk.Frame(self)
        preset_frame.pack(pady=10)

        presets = [
            ("赤", 255, 0, 0),
            ("緑", 0, 255, 0),
            ("青", 0, 0, 255),
            ("黄", 255, 255, 0),
            ("紫", 255, 0, 255),
            ("水色", 0, 255, 255),
            ("白", 255, 255, 255),
            ("黒", 0, 0, 0)
        ]

        for i, (name, r, g, b) in enumerate(presets):
            if i % 4 == 0:
                row_frame = tk.Frame(preset_frame)
                row_frame.pack()

            btn = tk.Button(
                row_frame,
                text=name,
                width=8,
                command=lambda r=r, g=g, b=b: self.set_color(r, g, b)
            )
            btn.pack(side=tk.LEFT, padx=2, pady=2)

        # 初期色を設定
        self.update_color()

    def update_color(self, event=None):
        red = self.red_scale.get()
        green = self.green_scale.get()
        blue = self.blue_scale.get()

        # RGB値を16進数に変換
        hex_color = f"#{red:02x}{green:02x}{blue:02x}"

        # キャンバスの色を更新
        self.color_canvas.configure(bg=hex_color)

        # ラベルを更新
        self.rgb_label.config(text=f"RGB: ({red}, {green}, {blue})")
        self.hex_label.config(text=f"HEX: {hex_color.upper()}")

    def set_color(self, red, green, blue):
        self.red_scale.set(red)
        self.green_scale.set(green)
        self.blue_scale.set(blue)
        self.update_color()

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

計算機と設定パネル

```
 import tkinter as tk
import math

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

        self.title("関数計算機")
        self.geometry("500x400")

        self.create_widgets()

    def create_widgets(self):
        # タイトル
        tk.Label(self, text="三角関数計算機", font=("Arial", 16, "bold")).pack(pady=10)

        # メインフレーム
        main_frame = tk.Frame(self)
        main_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)

        # 左側：コントロールパネル
        control_frame = tk.Frame(main_frame)
        control_frame.pack(side=tk.LEFT, fill=tk.Y, padx=(0, 20))

        # 角度スライダー（0〜360度）
```

```
        tk.Label(control_frame, text="角度 (度)", font=("Arial", 12, "bold")).pack(pady=(0, 5))
        self.angle_scale = tk.Scale(
            control_frame,
            from_=0,
            to=360,
            orient=tk.VERTICAL,
            length=200,
            tickinterval=90,
            command=self.update_calculation
        )
        self.angle_scale.pack()

        # 振幅スライダー
        tk.Label(control_frame, text="振幅", font=("Arial", 12, "bold")).pack(pady=(20, 5))
        self.amplitude_scale = tk.Scale(
            control_frame,
            from_=0.1,
            to=3.0,
            resolution=0.1,
            orient=tk.VERTICAL,
            length=150,
            command=self.update_calculation
        )
        self.amplitude_scale.set(1.0)
        self.amplitude_scale.pack()

        # 右側：結果表示
        result_frame = tk.Frame(main_frame)
        result_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        # 結果表示エリア
        tk.Label(result_frame, text="計算結果", font=("Arial", 14, "bold")).pack(pady=(0, 10))

        self.result_text = tk.Text(result_frame, width=30, height=20, font=("Consolas", 10))
        self.result_text.pack(fill=tk.BOTH, expand=True)

        # リセットボタン
        reset_frame = tk.Frame(self)
        reset_frame.pack(pady=10)

        tk.Button(reset_frame, text="リセット", command=self.reset_values).pack(side=tk.LEFT, padx=5)
        tk.Button(reset_frame, text="ランダム設定", command=self.random_values).pack(side=tk.LEFT, padx=5)

        # 初期計算
        self.update_calculation()

    def update_calculation(self, event=None):
        angle_deg = self.angle_scale.get()
        amplitude = self.amplitude_scale.get()

        # 度をラジアンに変換
        angle_rad = math.radians(angle_deg)

        # 三角関数の計算
        sin_val = amplitude * math.sin(angle_rad)
        cos_val = amplitude * math.cos(angle_rad)
        tan_val = amplitude * math.tan(angle_rad) if abs(math.cos(angle_rad)) > 1e-10 else float('inf')

        # 結果テキストを作成
        result = f"角度: {angle_deg}°\n"
        result += f"ラジアン: {angle_rad:.4f}\n"
        result += f"振幅: {amplitude}\n"
        result += f"{'-' * 25}\n\n"
        result += f"sin({angle_deg}°) × {amplitude} = {sin_val:.4f}\n"
        result += f"cos({angle_deg}°) × {amplitude} = {cos_val:.4f}\n"

        if tan_val == float('inf'):
            result += f"tan({angle_deg}°) × {amplitude} = undefined\n\n"
        else:
            result += f"tan({angle_deg}°) × {amplitude} = {tan_val:.4f}\n\n"

        # 特殊角度の判定
        special_angles = {0: "0°", 30: "30°", 45: "45°", 60: "60°", 90: "90°",
                          120: "120°", 135: "135°", 150: "150°", 180: "180°",
                          210: "210°", 225: "225°", 240: "240°", 270: "270°",
                          300: "300°", 315: "315°", 330: "330°", 360: "360°"}

        if angle_deg in special_angles:
            result += f"* {special_angles[angle_deg]} は特殊角です\n\n"

        # 座標系での位置
        x = cos_val
        y = sin_val
        result += f"座標: ({x:.4f}, {y:.4f})\n"

        # 象限の判定
        if x > 0 and y >= 0:
            quadrant = "第1象限"
        elif x <= 0 and y > 0:
            quadrant = "第2象限"
        elif x < 0 and y <= 0:
            quadrant = "第3象限"
        elif x >= 0 and y < 0:
            quadrant = "第4象限"
        else:
            quadrant = "軸上"

        result += f"位置: {quadrant}"

        # テキストエリアを更新
```

```
            self.result_text.delete(1.0, tk.END)
            self.result_text.insert(1.0, result)

        def reset_values(self):
            self.angle_scale.set(0)
            self.amplitude_scale.set(1.0)
            self.update_calculation()

        def random_values(self):
            import random
            random_angle = random.randint(0, 360)
            random_amplitude = round(random.uniform(0.1, 3.0), 1)

            self.angle_scale.set(random_angle)
            self.amplitude_scale.set(random_amplitude)
            self.update_calculation()

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

ベストプラクティス

プラクティス	説明
適切な範囲設定	`from_` と `to` を用途に応じて適切に設定します。
分解能の調整	`resolution` オプションで適切な精度を設定します。小数が必要な場合は0.1などを指定します。
リアルタイム更新	`command` オプションを使用してスライダーの変更をリアルタイムで反映します。
変数との連携	`variable` オプションを使用してアプリケーションの状態と同期できます。
視覚的フィードバック	スライダーの変更に応じて視覚的な変化を提供してユーザビリティを向上させます。

参考リンク

- Python Docs - tkinter.Scale
- TkDocs - Scale