

Python tkinter

画面作成

●メイン画面作成

書式

```
from tkinter import *  
root = Tk()  
root.mainloop()
```

GUIを作成するためのtkinterをインポートします。

Tk()オブジェクトを作成するだけで画面の大枠が作れます。

「mainloop()」と記入すると、画面を常に表示することができます。「main lop()」を記載しない場合は、画面は一瞬で消えてしまいます。必ず「mainloop()」を記述してください。

1	from tkinter import *
2	
3	root = Tk()
4	root.mainloop()



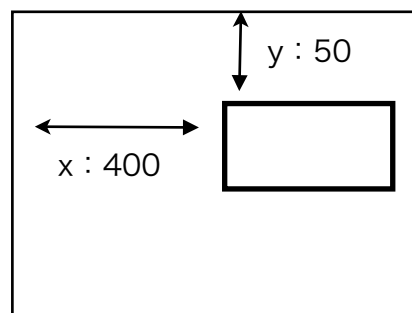
●タイトルと画面サイズ

書式

```
root.title("タイトル")  
root.geometry("横x縦+x+y")
```

画面のタイトルと画面サイズを指定することができます。また、画面を表示する座標も指定することができます。座標は、画面の左上が起点となり、x:画面左端からの距離、y:画面上部からの距離を記述します。

1	from tkinter import *
2	
3	root = Tk()
4	root.title("テスト画面")
5	root.geometry("500x200+400+50")
6	
7	root.mainloop()



e

ラベルの作成

●ラベル

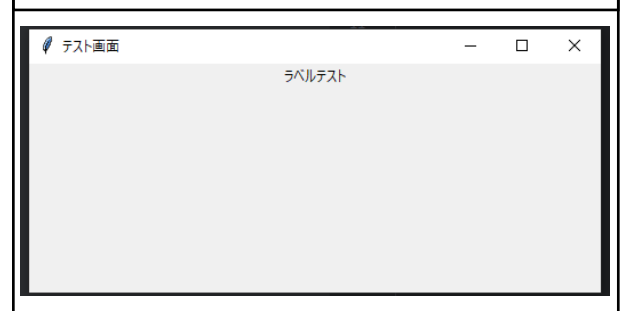
書式

```
lab1=Label(配置場所,text="文字列",オプション(省略可) )  
lab1.pack()          #オブジェクトを設置するメソッド
```

「Label()」でラベルを作成することができます。引数には、配置先の画面と、ラベルの内容を記述します。「pack()」はラベルやボタンなどのオブジェクトを画面に設置するためのメソッドです。詳しい使い方は後述します。

1	from tkinter import *
2	root = Tk()
3	root.title("ラベル表示")
4	root.geometry("500x200")
5	
6	lab1=Label(root,text="ラベルテスト")
7	lab1.pack()
8	
9	root.mainloop()

実行結果



●ラベルのオプション

ラベルやボタンなどのオブジェクトには背景色の変更や文字色の変更などのオプションがあります。オプションは引数に追加することで使用できます。複数追加することもできます。使用しない場合は省略しても問題ありません。

・ anchor=値

表示文字位置の指定をできる「anchor」というオプションがあります。値は東西南北の9種類あります。下記の表を参考にしてください。

anchor=NW	anchor=N	anchor=NE
anchor=W	anchor=CENTER	anchor=E
anchor=SW	anchor=S	anchor=SE

・relief=値

ラベルの枠線の指定をできる「relief」というオプションがあります。下記6種類のデザインがあります。

relief	
raised	ラベルが隆起
ridge	枠線が凸く
sunken	ラベルが凹む
flat	枠なし
groove	枠が凹む
solid	枠が実線



・その他のオプション

オプション	内容
bg='色'	背景色を指定
fg='色'	文字色を指定
width=値	幅のサイズを指定
height=値	高さを指定
font=('フォント', サイズ)	フォントとサイズの指定。指定しない場合はデフォルト設定
padx=値, pady=値	ラベルの枠とテキストの間にスペースを追加

オプションはオブジェクトの作成時に指定することができますが、作成後にも「.configure」を使用することで、オプションを変更することもできます。

書式

lab1.configure(オプション)

6	lab1=Label(root,text="ラベルテスト",bg="red")
7	lab1.pack()
8	lab.configure(bg="red",fg="#ffffff")

テキストボックス作成

●テキストボックス(1行)

書式

en1=Entry(配置場所,オプション(省略可))

「Entry」は1行のみのテキストボックスです。

1	from tkinter import *
2	root = Tk()
3	root.title("テキストボックス表示")
4	root.geometry("500x200")
5	
6	en1=Entry(root)
7	en1.pack()
8	
9	root.mainloop()

●Entryのオプション

ラベルのオプションと同じものも多々あります。

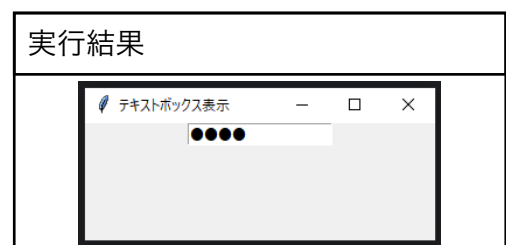
オプション	内容	
bg='色'	背景色を指定	"red","blue"などの指定や"#ff00ff"など 16進数での指定もできる (カラーコードなどを参考)
fg='色'	文字色を指定	
width=値	幅のサイズを指定	
font=('フォント', サイズ)	フォントとサイズの指定。指定しない場合はデフォルト設定	
justify=値	文字揃え (LEFT,CENTER,RIGHT)	
relief=値	テキストボックスの枠線のデザイン(4ページ参照)	
state=値	状態を指定 (DISABLED:使用不可,NORMAL:使用可能)	

・ show="値"

テキストボックスに入力した文字を、指定した文字に置き換えて表示します。パスワードの入力ボックスなどに使用します。

6	en1=Entry(root,show="●")
7	en1.pack()























実行結果



・ cursor="値 "

テキストボックスにマウスホバーした際のカーソルの種類を指定できます。

カーソルの種類

arrow		circle		clock		cross	
dotbox		exchange		fleur		heart	
dot		man		mouse		pirate	
plus		shuttle		sizing		spider	
spraycan		star		target		tcross	
trek		watch					

・ textvariable

書式

変数名=Entry(textvariable=変数)

textvariableはテキストボックスに入力された文字を受け取る変数を指定するオプションです。

下記の例はテキストボックスに文字が入力されて「Enter」キーが押されるテキストボックス内の文字を出力するメソッドです。

1	from tkinter import *
2	def return_push(event):
3	print(strvar.get())
4	strvar.set("更新しました。")
5	
6	root = Tk()
7	root.title("textvariable")
8	root.geometry("300x200")
9	strvar = StringVar()
10	en = Entry(textvariable=strvar)
11	en.bind("<Return>", return_push)
12	en.pack()
13	root.mainloop()

●bind

書式

ウィジェット名.bind(トリガー,イベント名)

bindはボタンではないウィジェット(ラベルやテキストボックス)に対し、イベントを紐づけるメソッドです。上記の例では"<Return>"(「Enter」キー)が押されたら、「return_push」が実行されます。上記以外にも、tikinterで取り扱われるすべてのウィジェットに対し処理ができます。

・トリガー

<Button-1> または <1>	マウスの左クリック
<Button-3> または <3>	マウスの右クリック
<ButtonRelease-1>	マウスの左ボタンを離した時
<ButtonRelease-3>	マウスの右ボタンを離した時
<Return>	キーボードのリターンキーが押された
<KeyPress-H> または h	キーボードのhキーが押された
<Control-c>	Ctrlキーを押しながらキーボードのcキーが押された
<Shift-c>	Shiftキーを押しながらキーボードのcキーが押された
<Alt-c>	Altキーを押しながらキーボードのcキーが押された
<Double-1>	マウスの左ボタンをダブルクリック

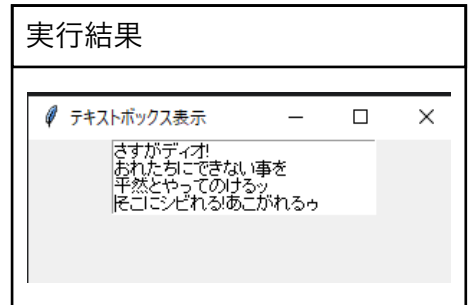
●テキストボックス（複数行）

書式

txt1=Text(配置場所,height=値,オプション(省略可))

複数行入力を行いたい場合は「Text」を使用します。「Entry」では使用できなかった「height」オプションが使用できます。省略もできますが高さはなるべく初期設定してください。

1	from tkinter import *
2	root = Tk()
3	root.title("テキストボックス表示")
4	root.geometry("300x200")
5	
6	txt1=Text(root,font=("",9),width=30,height=4)
7	txt1.pack()
8	
9	root.mainloop()



●Textのオプション

オプション	内容	
bg='色'	背景色を指定	"red","blue"などの指定や"#ff00ff"など16進数での指定もできる (カラーコードなどを参考)
fg='色'	文字色を指定	
width=値	幅のサイズを指定	
font=('フォント', サイズ)	フォントとサイズの指定。指定しない場合はデフォルト設定	
justify=値	文字揃え (LEFT,CENTER,RIGHT)	
relief=値	テキストボックスの枠線のデザイン(4ページ参照)	
state=値	状態を指定 (DISABLED:使用不可,NORMAL:使用可能)	
padx=値, pady=値	テキストボックスの枠とテキストの間にスペースを追加	
height=値	テキストボックスの行数を指定	
tabs=値	タブ位置の指定	
wrap=値	行の折り返し方法の指定 CHAR: 文字単位で折り返す WORD: 単語単位で折り返す NONE: 折り返ししない	

ボタン

●ボタン作成

変数名=Button(配置場所,オプション)

1	from tkinter import *
2	
3	root = Tk()
4	root.title("テスト画面")
5	root.geometry("500x200")
6	bt1=Button(root,text="ボタン1",state="disabled")
7	bt1.pack()
8	root.mainloop()

●Buttonオプション

オプション	内容
text="メッセージ"	ボタンに表示するメッセージ
state	ボタンの状態 state="normal" : 通常状態 (初期設定) state="disabled" : 無効状態
command=関数名	ボタンを押すと関数が実行される

●Entryのオプション

オプション	内容
delete("0",END)	テキストボックスの中身を削除
insert("0", "メッセージ")	テキストボックスにメッセージを追記
get()	テキストボックスの中身を取得

●commandオプション

commandオプションは、ボタンを押したときの処理を指定できます。

試しに、ボタンを押すとテキストボックスにメッセージが表示されるプログラムを作ってみましょう。下記サンプルコードは、「from tkinter import *」を省略してあります。」

1	root = Tk()
2	root.title("テスト画面")
3	root.geometry("500x200")
4	en1=Entry()
5	en1.pack()
6	bt1=Button(root,text="ボタン1",command=en1.insert("0","ボタンを押しました"))
7	bt1.pack()
8	root.mainloop()

上記プログラムを実行すると、ボタンを押してないのに、テキストボックスにボタンを押しましたと表示されます。「command=関数名」でなければなりません。lambdaで式を書き換えてみましょう。

6	bt1=Button(root,text="ボタン1",command=lambda : en1.insert("0", "ボタンを押しました"))
---	--

このようにlambdaを使用することでうまくプログラムが動きます。次は関数を使用して処理を行ってみましょう。

1	def bt_action(self):
2	en1.delete("0",END)
3	en1.insert("0", f"ボタンを押しました{self}")
4	root = Tk()
5	root.title("テスト画面")
6	root.geometry("500x200")
7	en1 = Entry(width=50)
8	en1.pack()
9	bt1 = Button(root, text="ボタン1", command=lambda: bt_action(1))
10	bt1.pack()
11	bt2 = Button(root, text="ボタン2", command=lambda: bt_action(2))
12	bt2.pack()
13	root.mainloop()

引数にナンバーを持たせることで、どのボタンが押されたのか判断ができます。

●ボタンの表示を変える

前述のプログラムを少し改変し、ボタンを押したときに、ボタンの「text」を変更するプログラムを作ってみましょう。ボタンの「text」を変更するにはボタンの「command」の関数の引数にボタン自身のオブジェクトを渡し、「ボタン["text"]="変更後の文字」で変更できます。

1	def bt_action(self):	オブジェクトを引数として渡しているので、オブジェクトの設定変更が可能になる
2	en1.delete("0",END)	
3	en1.insert("0", f"ボタンを押しました{self}")	
4	self["text"]="押されました"	
5	root = Tk()	
6	root.title("テスト画面")	
7	root.geometry("500x200")	
8	en1 = Entry(width=50)	
9	en1.pack()	
10	bt1 = Button(root, text="ボタン1", command=lambda: bt_action(bt1))	ボタンオブジェクトを引数として渡す
11	bt1.pack()	
12	root.mainloop()	

押すたびに、「ON」、「OFF」が切り替わるボタンを作成してみましょう。

1	def bt_action(self):	オブジェクトを引数として渡しているので、self["text"]が使用できる
2	if self["text"]=="ON":	
3	self["text"]="OFF"	
4	else:	
5	self["text"]="ON"	
6	root = Tk()	
7	root.title("ボタン表示切り替え")	
8	root.geometry("500x200")	
9	en1 = Entry(width=50)	
10	en1.pack()	
11	bt1 = Button(root, text="ON", command=lambda: bt_action(bt1))	ボタンオブジェクトを引数として渡す
12	bt1.pack()	
13	root.mainloop()	

画面遷移

●サブ画面作成

今まで、メイン画面を作成してきましたが、メイン画面とサブ画面を作成し、メイン画面とサブ画面を遷移するプログラムを作成しましょう。画面を削除するには、下記構文を使用します。

画面.destroy()

1	def main():	
2	root = Tk()	
3	root.title("メイン画面")	
4	root.geometry("500x200")	
5	lab1 = Label(root,text="メイン画面")	
6	lab1.pack()	
7	bt1 = Button(root, text="サブ画面表示", command=lambda: sub_window())	サブ画面呼び出し
8	bt1.pack()	
9	root.mainloop()	
10		
11	def sub_window():	
12	sub_window = Tk()	
13	sub_window.title("サブ画面")	
14	sub_window.geometry("500x200")	
15	sub_lab1 = Label(sub_window,text="サブ画面")	
16	sub_lab1.pack()	
17	sub_bt1 = Button(sub_window, text="サブ画面削 除", command=lambda: sub_window.destroy())	サブ画面削除
18	sub_bt1.pack()	
19	main()	

サブ画面は使用しない時は、非表示ではなく、削除します。

●画面遷移

前述のプログラムを修正し、サブ画面を呼び出したときに、メイン画面を非表示にするプログラムを作成しなさい。画面を非表示にするには下記構文でできます。画面.withdraw()

1	def visible_change(self,num,tar_screen):	
2	if num==1:	第2引数が1なら
3	sub_window(tar_screen)	サブ画面表示
4	self.withdraw()	メイン非表示
5	else:	
6	self.destroy()	サブ画面削除
7	tar_screen.deiconify()	メイン再表示
8	def main():	
9	root = Tk()	
10	root.title("メイン画面")	
11	root.geometry("500x200+400+200")	
12	lab1 = Label(root,text="メイン画面")	
13	lab1.pack()	
14	bt1 = Button(root, text="サブ画面表示", command=lambda: visible_change(root,1,root))	画面のオブジェクトとメイン画面であることを示す1を引数にする
15	bt1.pack()	
16	root.mainloop()	
17	def sub_window(tar_screen):	
18	sub_window = Tk()	
19	sub_window.title("サブ画面")	
20	sub_window.geometry("500x200+430+250")	
21	sub_lab1 = Label(sub_window,text="サブ画面")	
22	sub_lab1.pack()	
23	sub_bt1 = Button(sub_window, text="メイン画面表示", command=lambda:visible_change(sub_window,2,tar_screen))	画面のオブジェクトとサブ画面であることを示す2を引数にする
24	sub_bt1.pack()	
	main()	

メイン画面は常に動作していなくてはならないので、削除(destroy())ではなく、非表示(withdraw())を使用します。

●関数内関数

サブ画面内で、サブ画面のラベルの値を変更するボタンを作成しなさい。

ラベルの内容を変更するメソッドは、サブ画面内でしか使用しないため、サブ画面の関数内関数で書いたほうがわかりやすく良いです。

1	def sub_window():	
2	sub_window = Tk()	
3	sub_window.title("サブ画面")	
4	sub_window.geometry("500x200")	
5	sub_lab1 = Label(sub_window,text="サブ画面")	
6	sub_lab1.pack()	
7	sub_bt1 = Button(sub_window, text="サブ画面削除", command=lambda:visible_change(sub_window,2))	
8	sub_bt1.pack()	
9	sub_bt2 = Button(sub_window, text="ラベル表記変更", command=lambda: label_change())	
10	sub_bt2.pack()	
11		
12	def label_change():	サブ画面の中だけしか使用 しないのでサブ画面のメソ ッドの中に記述する
13	 sub_lab1["text"] = "ラベル内容変更"	

配置

●grid

ウィジェットを配置するには、pack,grid,placeの3つのメソッドがあります。ここでは、gridについて説明します。gridは縦横に罫線が入っているエクセルのマス目のようなイメージです。

行番号、列番号は「0」から始まります。(row：行番号 column：列番号)

ウィジェット.grid(row = 行番号, column = 列番号, オプション)

1	root = Tk()	
2	root.title("メイン画面")	
3	root.geometry("300x200+800+300")	
4	lab0 = Label(root,text="ログイン")	
5	lab1 = Label(root,text="ID:")	
6	lab2 = Label(root, text="PW:")	
7	en1 = Entry(root,width=30)	
8	en2 = Entry(root,show="●",width=30)	
9	bt1 = Button(root, text="ログイン")	
10	lab0.grid(row = 0, column = 1)	1行目の2列目に配置
11	lab1.grid(row=1, column=0,)	2行目の1列目に配置
12	en1.grid(row=1, column=1,)	2行目の2列目に配置
13	lab2.grid(row=2, column=0)	3行目の1列目に配置
14	en2.grid(row=2, column=1)	3行目の2列目に配置
15	bt1.grid(row=3, column=1)	4行目の2列目に配置
16	root.mainloop()	

●gridオプション

さらに、オプションを設定することで、ウィジェットの配置に自由度が増します。

オプション	内容
columnspan = 値	値で指定された数だけ横と結合する
rowspan = 値	値で指定された数だけ縦と結合する
padx=値	値で指定された数だけ横に余白を設ける
pady=値	値で指定された数だけ縦に余白を設ける
sticky=値	anchorと同じp2を参照

オプションを設定する。

1	root = Tk()	
2	root.title("メイン画面")	
3	root.geometry("300x200+800+300")	
4	lab0 = Label(root,text="ログイン")	
5	lab1 = Label(root,text="ID:")	
6	lab2 = Label(root, text="PW:")	
7	en1 = Entry(root,width=30)	
8	en2 = Entry(root,show="●",width=30)	
9	bt1 = Button(root, text="ログイン")	
10		
11	lab0.grid(row = 0, column = 2)	
12		
13	lab1.grid(row=1, column=0,padx=20, pady=10)	横に20の余白、縦に10の余白
14	en1.grid(row=1, column=1,columnspan = 3,)	3列結合
15		
16	lab2.grid(row=2, column=0,padx=20, pady=10)	横に20の余白、縦に10の余白
17	en2.grid(row=2, column=1,columnspan = 3)	3列結合
18		
19	bt1.grid(row=3, column=2, pady=20)	縦に20の余白
20	root.mainloop()	

配置イメージ↓

	0	1	2	3
0			lab0(row = 0, column = 2)	
1	lab1(row=1, column=0)	en1(row=1, column=1,columnspan = 3)		
2	lab2(row=2, column=0)	en2(row=2, column=1,columnspan = 3)		
3			bt1(row=3, column=2)	

●place

placeは、座標でウィジェットを配置することが出来ます。アプリの左端を原点として指定することが出来ます。

ウィジェット.place(x = 座標, y = 座標, オプション)

1	root = Tk()	
2	root.title("メイン画面")	
3	root.geometry("300x200+800+300")	
4	lab0 = Label(root,text="ログイン")	
5	lab1 = Label(root,text="ID:")	
6	lab2 = Label(root, text="PW:")	
7	en1 = Entry(root,width=30)	
8	en2 = Entry(root,show="●",width=30)	
9	bt1 = Button(root, text="ログイン")	
10		
11	lab0.place(x=130,y=10)	
12		
13	lab1.place(x=20,y=40)	
14	en1.place(x=60,y=40)	
15		
16	lab2.place(x=20,y=70)	
17	en2.place(x=60,y=70)	
18		
19	bt1.place(x=130,y=120)	
20		

●placeオプション

さらに、オプションを設定することで、ウィジェットの配置に自由度が増します。

オプション	内容
relx=値, rely=値	座標指定後、相対的な位置を0.0～1.0で指定
anchor = 値	座標指定後、東西南北を指定p2参照
width=値,height=値	ウィジェットの幅、高さを指定
relwidth=値, relheight=値	ウィジェットの幅、高さを相対的に指定(0.0～1.0)

●pack

packは、ウィジェットを上下左右に積み重ねて配置できます。

ウィジェット.pack(オプション)

●packオプション

さらに、オプションを設定することで、ウィジェットの配置に自由度が増します。

オプション	内容
side	ウィジェットの配置方向を指定(TOP ,BOTTOM,LEFT,RIGHT)
before	既に配置されているウィジェットの前に配置
after	既に配置されているウィジェットの後ろに配置
padx=値,pady=値	ウィジェットの外側に隙間を設定
ipadx=値,ipady=値	ウィジェットの内側に隙間を設定
expand=値	ウィジェットの大きさを領域全体まで広げる(True, False)
fill=値	ウィジェットの幅を指定の方向まで広げる(X,Y,BOTH, NONE)

サブ画面

●Toplevel()

画面遷移でサブ画面とメイン画面の切り替えを行っていますが、これは正確にはサブ画面ではありません。独立した2つの画面を切り替えているだけです。

メイン画面と連動するサブ画面を作成することができます。Toplevel()で作成するとメイン画面と連動するので、メイン画面を閉じるとサブ画面も同時に閉じます。

サブ画面名=Toplevel(オプション)

オプション	内容
bg	ウィンドウの背景色設定
bd	ウィンドウのボーダー幅設定
cursor	カーソルがウィンドウの上に乗った時のカーソルの指定
relief	ウィンドウの外観設定
height,width	ウィンドウの縦横サイズ設定

1	from tkinter import *	
2	def main():	
3	root = Tk()	メイン画面
4	root.title("メイン画面")	
5	root.geometry("500x300+300+100")	
6	def sub_window():	
7	sub=Toplevel()	サブ画面
8	sub.title("サブ画面")	
9	sub.geometry("500x300+900+100")	
10	main()	
11	sub_window()	
12	mainloop()	

メイン画面の「x」ボタンを押し、メイン画面を閉じると、同時にサブ画面も閉じることを確認してください。

ドロップダウンリスト

●combobox

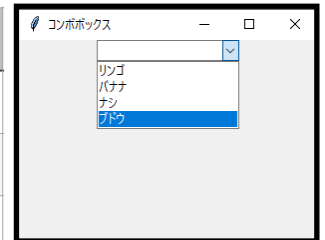
コンボボックスを使用することで、ドロップダウンリストを使用することができます。ドロップダウンリストを使用することで、入力が楽になったり、入力値に誤差が出ないなどのメリットがあります。

from tkinter.ttk import Combobox

リスト=["選択肢1","選択肢2","選択肢3","選択肢4"]

ドロップダウン名=Combobox(配置場所,values=lis,オプション)

オプション	内容
justify	文字列の配置位置(LEFT, CENTER, RIGHT)
values	リストに表示するデータを指定
cursor	カーソルがウィンドウの上に乗った時のカーソルの指定
relief	ウィンドウの外観設定
height,width	ウィンドウの縦横サイズ設定
postcommand	Comboboxの▼をクリックしたときのコールバック関数を指定
.get()	コンボボックスで選択された値を取得
.config(オプション)	配置後、オプションを再設定する



1	from tkinter import *	
2	from tkinter.ttk import Combobox	tkinter.ttkをインポート
3	def main():	
4	root = Tk()	
5	root.title("コンボボックス")	
6	root.geometry("300x200+300+100")	
7	lis=["リンゴ","バナナ","ナシ","ブドウ"]	コンボボックスの内容
8	com=Combobox(root,values=lis,width=30)	
9	bt=Button(root,text="取得", width=10,command=lambda :print(com.get()))	コンボボックスの値を取得
10	com.pack()	
11	bt.pack()	
12	main()	
13	mainloop()	

ウィジェットを配列に格納

●ボタンを配列に格納

GUIアプリを作成していると、ボタンやラベルやテキストボックスを多く使用することがあります。その場合に、たくさんのボタンやラベルなどに同じ処理を行う場合があります。コードが長くなりますので、ボタンなどを配列に格納し、ループで処理を行うことでコードが短くなります。

リスト=[]

for i in range(0,10)

**リスト.append(Button(配置場所,text=f"ボタン{i+1}"),
 command=bt_action(i))**

1	from tkinter import *	
2		
3	def main():	
4	def bt_action(tar_lab,num):	ボタンメソッド
5	x=num+1	
6	def inner():	クロージャ python基本 p27を参照
7	tar_lab["text"]=f"ボタン{x}を押しました。"	
8	return inner	
9		
10	root = Tk()	
11	root.title("ボタンを配列に格納")	
12	root.geometry("200x400+300+100")	
13	lab1=Label(root,text="ボタンを押してください")	
14	lab1.pack()	ボタンを格納するリスト
15	buttons=[]	
16	for i in range(0,10):	
17	buttons.append(Button(root,text=f"ボタン{i+1}", command=bt_action(lab1,i),width=20))	「lambda」は使用しない
18	buttons[i].pack()	
19		
20	main()	
	mainloop()	

チェックボックス

●チェックボックス作成

チェックボックスをチェックしたかどうかはTrue,Falseで戻り値が帰ります。

var=BooleanVar()

チェックボックス名=Checkbutton(配置,text="テキスト",values=var,オプション)

オプション	内容
foreground	文字色の設定
background	背景色の設定
font	フォントの設定
height,width	チェックボックスの縦横サイズ設定
set()	チェックボックスに値を格納
get()	チェックボックスの値を取得(True:チェック有,False:チェック無)

1	from tkinter import *	
2	def main():	
3	root = Tk()	
4	root.title("チェックボックス")	
5	root.geometry("300x200+300+100")	
6	var=BooleanVar()	
7	che=Checkbutton(text="チェックボックス",variable=var)	
8	lab1=Label(root,text="チェックボックスを選択し、 確認ボタンを押してください")	
9	bt1=Button(root,text="確認", command=lambda :bt_action(lab1,var.get()))	varの値をget
10	che.pack()	
11	lab1.pack()	
12	bt1.pack()	
13	def bt_action(tar_lab,val):	
14	tar_lab["text"]="チェック有り" if val==True else "チェック無し"	
15	main()	
16	mainloop()	

ラジオボタン

●ラジオボタン作成

ラジオボタンは複数の選択肢の中から1つだけ選択することができます。

**ラジオボタン名=Checkbutton(配置,text="テキスト",value=値,
variable=グループ名,オプション)**

オプション	内容
justify	文字列の配置位置(LEFT, CENTER, RIGHT)
values	リストに表示するデータを指定
cursor	カーソルがウィンドウの上に乗った時のカーソルの指定
relief	ウィンドウの外観設定
width	ウィンドウの縦横サイズ設定
postcommand	Comboboxの▼をクリックしたときのコールバック関数を指定
get()	コンボボックスで選択された値を取得

キャンバス

●canvas

canvasは、画像やレイアウトを描画するための領域です。まずはcanvasを用意して、そのcanvasに対して操作を行っていく手順になります。

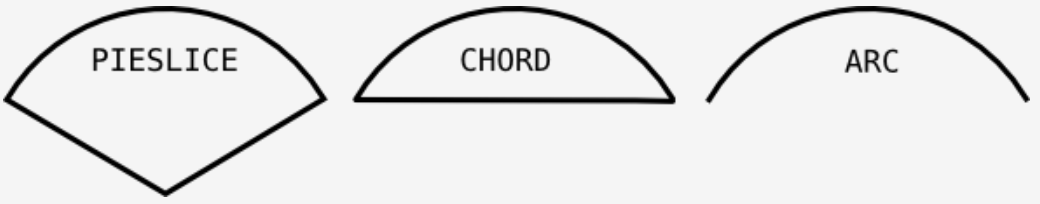
まずはcanvasを作成します。

キャンバス名=Canvas(配置,width=幅,height=高さ)

●弧を描画

まずは弧を描画してみましょう。

オブジェクトID=キャンバス名.create_arc(x0,y0,x1,y1,オプション)

オプション	内容
fill	円の内側の色を指定。デフォルトは透明
outline	線の色。デフォルトは黒
start	スライスの開始角度
extent	スライスの終了角度（startから反時計回り）
state	円の状態。デフォルトはNORMAL。HIDDEN：非表示。DISABLED：無効
style	スライスした時の表示状態。PIESLICE、CHORD、ARC 
width	線の幅。デフォルト値は1

1	from tkinter import *	
2	root = Tk()	
3	root.title("円描画")	
4	root.geometry("300x200+300+100")	
5	can=Canvas(root,width=300,height=200)	big='green'
6	arc1=can.create_arc(10,10,100,100,outline='red')	
7	can.pack()	
8	root.mainloop()	

●弧以外の描画

弧以外にも下記の項目を描画することができます。

関数	内容
<code>create_line(x0,y0,x1,y1,...xn,yn,オプション)</code>	線の描画
<code>create_oval(x0,y0,x1,y1,オプション)</code>	楕円の描画
<code>create_arc(x0,y0,x1,y1,オプション)</code>	弧の描画
<code>create_rectangle(x0,y0,x1,y1,オプション)</code>	矩形の描画
<code>create_polygon(x0,y0,x1,y1,...xn,yn,オプション)</code>	多角形の描画
<code>create_image(x軸,y軸,オプション)</code>	画像の描画
<code>create_bitmap(x軸,y軸,オプション)</code>	ビットマップの描画
<code>create_text(x軸,y軸,オプション)</code>	文字列の描画

オプションは若干の違いあり、リファレンスサイトで確認

<https://anzelg.github.io/rin2/book2/2405/docs/tkinter/index.html>

●画像の描画

画像を描画するには、まず画像ファイルを読み込みます。その後キャンバスに配置します。オプションはいくつかありますが、上記サイトで確認してください。

画像オブジェクト=PhotoImage(file='画像ファイルパス')

オブジェクトID=キャンバス名.create_image(x,y,image=画像オブジェクト,オプション)

1	<code>from tkinter import *</code>	
3	<code>root = Tk()</code>	
4	<code>root.title("画像描画")</code>	
5	<code>root.geometry("300x200+300+100")</code>	
6	<code>can=Canvas(root,width=300,height=200)</code>	
7	<code>img=PhotoImage(file='sample.png')</code>	画像読込
9	<code>can.create_image(0,0,image=img,anchor=NW)</code>	画像オブジェクトをキャンバスに配置
10	<code>can.pack()</code>	
11	<code>can.image=img</code>	ガーベージコレクション対策
12	<code>root.mainloop()</code>	

フォトイメージ

●Photolmage

Photolmageは画像を扱うことができるウィジェットです。取り扱える画像はpam,ppm,gif,pngです。jpegは使用できません。下記にメソッドの一覧を表示します。

関数	内容
cget(オプション)	
画像オブジェクト.configure(オプション)	画像オブジェクトの設定を変える
新しいオブジェクト=画像オブジェクト.copy()	画像オブジェクトをコピーする
新しいオブジェクト=画像オブジェクト.subsample(x,y)	縮小(x,yには比率を入れる)2:半分
新しいオブジェクト=画像オブジェクト.zoom(x,y)	拡大(x,yには比率を入れる)2:倍
戻り値=画像オブジェクト.type()	Photolmageクラスのオブジェクトでなら「photo」が返る
戻り値=画像オブジェクト.height()	画像の高さを取得
戻り値=画像オブジェクト.width()	画像の幅を取得
画像オブジェクト.write(ファイル名,フォーマット,(x,y))	画像を保存(第二、第三引数は省略可能)
戻り値=画像オブジェクト.get(x,y)	指定された座標のRGBデータを取得する
戻り値=画像オブジェクト.put(色,to=(x,y))	指定された座標に色を指定する
画像オブジェクト.blank()	画像オブジェクトを透明にする
boolean=transparency_get(x,y)	透明度の取得(True:完全に透明、False:それ以外)
transparency_set(x,y,boolean)	指定された座標に透明度を設定

なお、Photolmageで取り込んだ画像ファイルをボタンに貼り付けることもできます

1	from tkinter import *	
2	root = Tk()	
3	root.title("ボタンに画像を貼り付け")	
4	root.geometry("300x200+300+100")	
5	img=Photolmage(file='sample_bt.png')	
6	img=img.subsample(1,2)	幅そのまま、高さ半分にリサイズ
7	bt1=Button(root,text="なし",image=img)	
8	bt1.pack()	
	bt1.image=img	ガーベージコレクション対策
9	root.mainloop()	

●PhotoImage(PIL)

Photoimageでは「jpeg」が使用できませんでした。もし「jpeg」を使用する際は、PIL(Pillow)のPhotoImageを使用します。

「jpeg」ファイルをPILで読み込む、読み込んだファイルをPhotoImageファイルに変換するという流れです。

jpegオブジェクト=Image.open(‘画像ファイルパス’)



画像オブジェクト=ImageTk.PhotoImage(jpegオブジェクト)

オブジェクトID=キャンバス名.create_image(x,y,image=画像オブジェクト,オプション)

1	from tkinter import *	
2	from PIL import Image,ImageTk	
3	root = Tk()	
4	root.title("jpeg取り込み")	
5	root.geometry("300x200+300+100")	
6	tmp=Image.open('sample.jpeg')	jpegファイル読込
7	img=ImageTk.PhotoImage(tmp)	PhotoImage形式に変換
8	can=Canvas(width=300,height=200)	
9	can.create_image(0,0,image=img,anchor=NW)	
10	can.pack()	
11	can.image=img	ガーベージコレクション対策
12	root.mainloop()	

タブ (Notobook)

●tkinter.ttkのインポート

タブはノートブックの機能です。ノートを作成してそれにタブ（タグ）を付けるイメージです。タブを使用するには、ttkモジュールをインポートする必要があります。下記いずれかの方法でインポートを行なってください。

```
from tkinter import ttk
from tkinter.ttk import *
import tkinter.ttk as ttk
```

●ttk.Notebook（ノートブックの作成）

タブを作成する前に、ノートブックを作成する必要があります。ノートブックウィジェットは複数のウィンドウを管理し1つのウィンドウを表示します。それぞれの子ウィンドウはタブに関連付けられます。

Notebookオブジェクト名=ttk.Notebook(配置,オプション)

オプション	内容
height	高さ指定
width	幅指定
padding	内部間隔を指定
cursor	マウスホバー時のカーソル指定

●タブを作成する

ノートブックに付けるタブを作成します。今回Frameを使用していますが、CanvasやLabelでも構いません。タブというかルーズリーフをイメージしていただいた方が近いと思います。

タブオブジェクト名=Frame(Notebookオブジェクト名)

●Notebookに追加する

作成したタブをノートブックに追加します。ルーズリーフを追加するイメージ

Notebookオブジェクト名.add(タブオブジェクト名,オプション)

オプション	内容
state	NORMAL:通常,DISABLED:無効,HIDDEN:非表示
sticky	子ウィンドウの配置。N,S,E,W
padding	内部間隔を指定
text	タブに表示する文字
underline	タブに表示した文字に下線を引く
image	タブに画像を表示する
compound	textとimageが指定されている場合の表示様式指定

●タブにラベルなどを配置

タブにラベルやボタンなどを配置できます。配置方法は通常の方法と変わりません。配置場所をタブにするだけです。

Notebookオブジェクト名.add(タブオブジェクト名,オプション)

1	from tkinter import *	
2	import tkinter.ttk as ttk	
3	root = Tk()	
4	root.title("タブ")	
5	root.geometry("300x200+300+100")	
6	note1=ttk.Notebook(root,width=300,height=200)	ノートブック作成
7	tab1=Frame(note1)	タブ1作成
8	tab2=Frame(note1)	タブ2作成
9	note1.add(tab1,text="タブ1")	タブ1追加
10	note1.add(tab2,text="タブ2")	タブ2追加
11	lab1=Label(tab1,text="タブ1 コンテンツ")	ラベルをタブ1に配置
12	lab2=Label(tab2,text="タブ2 コンテンツ")	ラベルをタブ2に配置
13	note1.pack()	
14	lab1.pack()	
15	lab2.pack()	
16	root.mainloop()	

ツリービュー (Treeview)

●tkinter.ttkのインポート

Treeviewもttkに含まれていますので、tkinter.ttkのインポートが必要になります。

●Treeview

Treeviewはテーブル(表)の作成ができます。データベースに保存されているデータを一覧表示する際などに使用すると良いでしょう。

Treeviewオブジェクト名=ttk.Treeview(配置,オプション)

オプション	内容
cloumns	列の識別名
displaycolumns	表示する列の指定及び順番
show	表示する対象。tree:内容、headings:見出し。デフォルトは両方
select mode	EXTENDED:複数選択可(デフォ),BROWSE:一つ選択可,NONE:選択不可
height	表示する行数
padding	内部間隔
cursor	カーソルの指定
style	スタイルの指定
xscrollcomand	横スクロール
yscrollcomand	縦スクロール

●列に識別子を設定

列幅やテキストの配置などの設定を行うため、列に識別子を設定します。

ツリーオブジェクトを作成する時に一緒に設定する方法と後で設定する方法があります。

●列に識別子を設定する

Treeviewオブジェクト名["cloumns"]=("id","name")

●列の設定

列に識別子を設定すると、列の幅やテキストの配置などの設定が行えます。

Treeviewオブジェクト名.cloumns=(列の識別子名,オプション)

オプション	内容
anchor	配置位置
id	列名
width	列の横幅
minwidth	列の最小幅(デフォルト値は20ピクセル)
stretch	列幅変更の可否。True:可能、False:不可能

●列の見出しの設定

Treeviewオブジェクト作成時に、「show=headings」もしくはデフォルトの場合は列の見出しが表示されます。

Treeviewオブジェクト名.heading(列の識別子名,オプション)

オプション	内容
anchor	配置位置
command	関数紐付け
image	見出しの右側に画像を表示
text	見出しのテキスト

1	from tkinter import *	
2	import tkinter.ttk as ttk	
3	root = Tk()	
4	root.title("タブ")	
5	root.geometry("300x200+300+100")	
6	c_name=("id","name","score")	
7	tree1=ttk.Treeview(root)	
8	tree1["columns"]=c_name	
9	tree1.column("#0",stretch=False)	Falseで0番目の列が表示されない
10	tree1.column("id",width=80)	columnの設定
11	tree1.column("name",width=200)	
12	tree1.column("score",width=100)	
13	tree1.heading("#0",text="")	見出しの設定
14	tree1.heading("id",text="ID")	
15	tree1.heading("name",text="名前",anchor=W)	
16	tree1.heading("score",text="点数",anchor=E)	
23	tree1.pack()	
24	root.mainloop()	

●データの挿入

表にデータを挿入します。

Treeviewオブジェクト名.insert(parent="",index="",iid="",values=(カラム1,カラム2,...))

オプション	内容
parent	親アイテムのID
index	新しいアイテムをどこに挿入するか。"end"なら最後、0なら先頭
iid	挿入するアイテムのIDを指定。省略すると自動で割り振られる。

●30ページの例文17~22行に追加

17	tree1.insert(parent="",index="end",iid=0,values=(1,"ラマヌジャン",100))	
18	tree1.insert(parent="",index="end",iid=1,values=(2,"湯川",99))	
19	tree1.insert(parent="",index="end",iid=2,values=(3,"野口",97))	
20	tree1.insert(parent="",index="end",iid=3,values=(4,"保井",98))	
21	tree1.insert(parent="",index="end",iid=4,values=(5,"斎藤",60))	
22	tree1.insert(parent="",index="end",iid=5,values=(6,"竹田",70))	
23	tree1.pack()	
24	root.mainloop()	

●上記をenumerateとdic.items()を使用することで少ない行数で記述できるようになります。

17	name_dic={"ラマヌジャン":100,"湯川":99,"野口":97,"保井":98,"斎藤":60,"竹田":70}	
18	for num,(name,score) in enumerate(data_dic.items(),0)	
19	tree1.insert(parent="",index="end",iid=num,values=(num+1,name,score))	
20	tree1.pack()	
21	root.mainloop()	

●データベースから取ってきた値を表示する場合

データベース：test.db、テーブル名：t_table

t_no	t_name	t_score
1	ラマヌジャン	100
2	湯川	99

17	data_lis=select_data()	タプルをそのままvaluesに入れる
18	for num,data in enumerate(data_lis,0)	
19	tree1.insert(parent="",index="end",iid=num,values=data)	
20	tree1.pack()	
21	root.mainloop()	
22	#sqlite3をインポートすること	
23	def select_data():	データベースから抜き出したデータはタプルのリストになっている [(1,"ラマヌジャン",100), (2,"湯川",99), (3,"野口",97)・・・]
24	con=sqlite3.connect("test.db")	
25	c=con.cursor()	
26	data=execute("select * from t_table")	
27	return data	