TkinterでGUIを作ろう

相川利樹（日本語訳）

更新履歴
2016/12/20 初稿
2019/12/26 更新　各プログラムのやや詳しい説明と実行例の追加

（原題）
Tkinter Tutorial
https://www.python-course.eu/python_tkinter.php

(原著者)
Bernd Klein

目 次

第1章　はじめに

「Python には Tk はないの？」とか、「Tkinter は Tk と同じなの？」とかいう質問を度々受けてきた。勿論 Python にも Tk はあります。Tk が使えない Python は多くのユーザにとって魅力が少ないものになってしまう。Python では Tk は Tkinter と呼ばれていて、より正確には Tkinter は Tk のための Python インターフェースである。Tkinter は “Tk interface” の略称である。

Tk 自体は *John Ousterhout* によってスクリプト言語 Tcl の GUI の機能を持った拡張として開発されたものである。最初の公開は 1991 年であった。Tk は他の同様なツールキットに比較して学ぶ易く従って利用し易いこともあって 1990 年代の大いに流行った。それ故に Tcl を離れて Tk を使いたいというプログラマーが沢山にいた。このような状況から多くのコンピュータ言語が Tk をバインドするようになった。Python では Tkinter がそれである。

Tk は以下のようなウィジェットを提供している：

- button
- canvas
- checkbutton
- combobox
- entry
- frame
- label
- labelframe
- listbox
- menu
- menubutton

- message
- notebook
- tk_optionMenu
- panedwindow
- progressbar
- radiobutton
- scale
- scrollbar
- separator
- sizegrip
- spinbox
- text
- treeview

また以下のような最上位に位置する画面構成を提供している：

- tk_chooseColor - pops up a dialog box for the user to select a color.
- tk_chooseDirectory - pops up a dialog box for the user to select a directory.
- tk_dialog - creates a modal dialog and waits for a response.
- tk_getOpenFile - pops up a dialog box for the user to select a file to open.
- tk_getSaveFile - pops up a dialog box for the user to select a file to save.
- tk_messageBox - pops up a message window and waits for a user response.
- tk_popup - posts a popup menu.
- toplevel - creates and manipulates toplevel widgets.

また Tk は三つのアイテム配置管理法を提供している：

- place - which positions widgets at absolute locations
- grid - which arranges widgets in a grid
- pack - which packs widgets into a cavity

第2章　今日は、Tkinter!

最も簡単なウィジェットの一つである Label からはじめよう。Label は Tkinter のウィジェットクラスの一つであり、テキストや画像を表示するだけの機能を持っている。

“Hello World” と表示するのは一般的であるが、この伝統に従うが表示は「今日は、Tkinter!」とする。

以下の Python スクリプトは一つのウインドウを作りそこに「今日は、Tkinter!」と表示するために Tkinter を使う。

```
#coding utf-8
import tkinter as tk

root = tk.Tk()
w = tk.Label(root, text="今日は、Tkinter!")
w.pack()

root.mainloop()
```

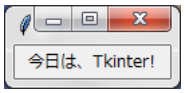

図 2.1: Windows7 上での実行例

詳細説明

Tk ツールキットを含んでいる tkinter モジュールは常にインポートされる必要がある。ここではその名前を慣例に従って tk と名前を変更している。

```
import tkinter as tk
```

次に tkinter を初期化するために Tk のルートウィジェットを一つ具現化しなけれ

ばならない。このウィジェットはタイトル一つのバー付きのウインドウである。このルートウィジェットは他の全てのウィジェットに先立って具現化されなければならず、プログラムのなかではルートウィジェットは一つだけ存在できる。

```
root = tk.Tk()
```

次は Label ウィジェットである。最初の引数は親ウインドウの名前であり、今のばあいは root である。つまりこの Label ウィジェットはルートウィジェットの子ウィジェットになる。キーワード付きの引数 text は表示がテキストであることを示し且つ表示する内容を特定している。

```
w = tk.Label(root, text="今日は、Tkinter!")
```

pack メソッドは Tk にウインドウの大きさをテキストに合うように指示している。

```
w.pack()
```

実際のテキスト表示があるウインドウは Tkinter がイベント待ちのループ（イベントループ）に入るまで現れることはない。

```
root.mainloop()
```

この例ではこのイベントループはユーザがウインドウを閉じる作業をするまで継続する。

2.1 Label に画像を表示

既に述べたように、ラベルはテキストと画像も入れることができる。以下の例では二つのラベル、一つはテキスト、もう一つは画像を入れた例である。

```
#coding: utf-8
import tkinter as tk

root = tk.Tk()
logo = tk.PhotoImage(file="python_logo_small.gif")

w1 = tk.Label(root, image=logo).pack(side="right")

explanation = """現在までのところ GIF と PPM/PGM
形式のみ許されている、しかし他の形式の画像ファイル
の表示を可能とするインターフェースが存在するので
他の形式でも簡単に追加できる。"""

w2 = tk.Label(root,
```

```
                justify=tk.LEFT,
                padx = 10,
                text=explanation).pack(side="left")
root.mainloop()
```

実行例：

図 2.2: テキストと画像を含むラベル

引数名 justify はテキストの左寄せ（LEFT）、右寄せ（RIGHT）、中央（CENTER）を指示するために使われるものである。padx はテキスト周りの水平方向に「詰め物」を入れる値を指示するためのものである。既定値は 1（単位はピクセル）。pady も同様に垂直方向の指示に使う。
二つの引数 justify（既定値は中央）と padx 無しでこの例を実行すると以下に実行例になる：

図 2.3: テキストと画像を含むラベル

画像の上に重ねてテキストを表示したい？これも問題なしでできる。一つのラベルが必要でこれに画像とテキストを同時に入れる。そのままでは一つの画像を入れるラベルの設定で、compound（合成）オプションを選択し表示すべきテキストを指定すると画像の上にテキストが表示される。この compound オプションの値を CENTER にすると、テキストは画像の正面に上書きされる。実例を示す：

```
#coding: utf-8
import tkinter as tk
```

```
root = tk.Tk()
logo = tk.PhotoImage(file="python_logo_small.gif")


explanation = """現在までのところ GIF と PPM/PGM
形式のみ許されている、しかし他の形式の画像ファイル
の表示を可能とするインターフェースが存在するので
他の形式でも簡単に追加できる。"""

w = tk.Label(root,
             compound = tk.CENTER,
             text=explanation,
             image=logo).pack()
root.mainloop()
```

実行例：

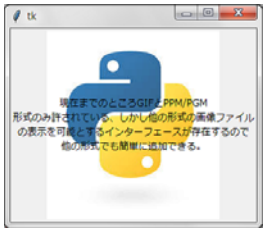

図 2.4: テキストと画像の合成

もう一つの例。画像を右に配置、左には左右に 10 の詰め物をして左詰めのテキストを配置する：

```
w = tk.Label(root,
             compound = tk.RIGHT
             justify = tk.LEFT
             padx = 10
             text=explanation,
             image=logo).pack()
```

compound オプションを tk.BOTTOM、tk.LEFT、 tk.RIGHT そして tk.TOP と選択すると画像がテキストに対して対応する位置に配置される。

2.2 色付きラベルと種々のフォント

label や canvas のような幾つかの Tk ウィジェットではテキスト表示で特定のフォントを使うことができる。それには font 引数を設定すればよい。フォントは動かしているコンピュータに依存することを考慮する必要がある。

その引数の与え方は ("Helvetica","16","bold") ようにフォント名、サイズ、特性を文字列のタプルとして与える。

テキストの色、背景色は引数 fg と bg の文字列の色名で与える。

```
#coding: utf-8
import tkinter as tk

root = tk.Tk()
tk.Label(root,
 text="ＭＳ 明朝を赤で",
 fg = "red",
 font = ("ＭＳ 明朝","16","bold")).pack()
tk.Label(root,
 text="Green Text in Helvetica Font",
 fg = "light green",
 bg = "dark green",
 font = ("Helvetica", "16", "bold italic")).pack()
tk.Label(root,
 text="Blue Text in Verdana bold",
 fg = "blue",
 bg = "yellow",
 font =( "Verdana", "10", "bold")).pack()

root.mainloop()
```

実行例

図 2.5: 色付きラベルとフォント各種

2.3　ラベルを動的に表示

以下の例では「停止」ボタンが押されるまで 1 秒毎に 1 増やす数字の表示が現れる。

```
#coding: utf-8
import tkinter as tk
counter = 0
def counter_label(label):
  def count():
    global counter
    counter += 1
    label.config(text=str(counter))
    label.after(1000, count)
  count()

root = tk.Tk()
root.title("Counting Seconds")
label = tk.Label(root, fg="green")
label.pack()
counter_label(label)
button = tk.Button(root, text='停止', width=25, command=root.destroy)
button.pack()
root.mainloop()
```

関数 counter_label では単に関数 count を呼ぶだけである。この関数 count はこの関数内で定義されている。この定義では大域変数 counter の値をインクリメントしその値でラベルのテキストを更新し、after メソッドで 1000 ミリ秒待って関数 count を呼び出す。つまり関数 count は再帰関数（無限ループ）になっている。関数 counter_label はメインで呼び出される。

実行例

もう一つのウィジェットの button は後に詳しく述べる。

図 2.6: ラベルを動的に表示

第3章　ボタン・ウィジェット

様々なボタンの機能に対応できる標準的なボタン・ウィジェットである。ユーザがマウスでクリックするとそのボタンを「オン」できるような機能をもったウィジェットである。テキストや画像を持たせる機能もあるが一つのフォントのみの表示となる。

ボタンが選択されたときに実行されるPythonの関数やメソッドを関連付けることもできる。. 以下の例ではボタンを二つ表示し、一つはこのアプリケーションの終了、他はテキスト “Tkinter is esy to use”をモニターに表示させる：

```
import tkinter as tk


def write_slogan():
    print("Tkinter is easy to use!")

root = tk.Tk()
frame = tk.Frame(root)
frame.pack()

button = tk.Button(frame,
                   text="QUIT",
                   fg="red",
                   command=quit)
button.pack(side=tk.LEFT)
slogan = tk.Button(frame,
                   text="Hello",
                   command=write_slogan)
slogan.pack(side=tk.LEFT)

root.mainloop()
```

（訳注：ボタンが選択されたときに実行されるPythonの関数やメソッドの関連付

けは command 引数で与える。上の例では command=quit と command=write_slogan がそれである。この引数に代入するものは関数オブジェクト quit と write_slogan である。write_slogan は関数オブジェクトであり、write_slogan() は関数の戻り値である。二つの違いに注意。）
実行例

図 3.1: ボタンを表示

第4章　種々のクラス

幾つかのウィジェット（テキスト入力ウィジェット、ラジオボタンなど）は特別な選択肢（variable、textvariable、onvalue、offvalue によってアプリケーションの変数と直接繋がることができる。何か理由でこの変数が変更になったり、この変数に結合しているウィジェットの状態が更新されたときの双方で動作する。これらの制御変数は Python の普通の変数のように動作する。Python の普通の変数を variable や txtvariable に渡すことはできない。Tkinter モジュールで定義されているクラス Variable のサブクラスである変数だけが可能である。それらは：

- x = StringVar() : 文字列を保持する、既定値は空文""
- x = IntVar() : 整数を保持する、既定値は 0
- x = DoubleVar() : 浮動小数点数を保持する、既定値 0.0
- x = BooleanVar() :ブーリアンを保持する、False に対して 0 を、True に対して 1 を返す

このような変数の現在の値を読みためには get() メソッドを使い、値の設定では set() メソッドを使う。

第5章　メッセージ・ウィジェット

このウィジェットは短いメッセージを表示することに使える。メッセージ・ウィジェットは機能的にはラベルウィジェットに似ているが表示の仕方はもっと融通が利くものになっている。例えば文章の途中でフォントを変更できる。また複数行に渡る文章も表示できる。長いテキストは自動的に折り返され字詰めされる。しかし実際はフォントはウィジェットに付随していてそれが全てである。もしも複数のフォントを使いたければテキストウィジェットを使うとよい。兎も角メッセージ・ウィジェットの実例を示す：

```
#coding: utf-8
import tkinter as tk
master = tk.Tk()
whatever_you_do = "あなたがしようとしていることが些細なことであっても、\
それを成し遂げることは非常に大事なことである。\n \
-マハトマ・ガンジー"

msg = tk.Message(master, text = whatever_you_do)
msg.config(bg='lightgreen', font=('ＭＳ 明朝', 16, 'bold'))
msg.pack()
tk.mainloop()
```

実行例：

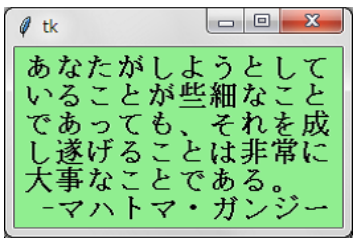

図 5.1: メッセージ・ウィジェット

以下にメッセージ・ウィジェットの引数オプションの一覧を示す：

anchor: The position, where the text should be placed in the message widget: N, NE, E, SE, S, SW, W, NW, or CENTER. The Default is CENTER.

aspect: Aspect ratio, given as the width/height relation in percent. The default is 150, which means that the message will be 50

background: The background color of the message widget. The default value is system specific.

bg: Short for background.

borderwidth; Border width. Default value is 2.

bd: Short for borderwidth.

cursor: Defines the kind of cursor to show when the mouse is moved over the message widget. By default the standard cursor is used.

font: Message font. The default value is system specific.

foreground: Text color. The default value is system specific.

fg: Same as foreground.

highlightbackground; Together with highlightcolor and highlightthickness, this option controls how to draw the highlight region.

highlightcolor: See highlightbackground.

highlightthickness: See highlightbackground.

justify: Defines how to align multiple lines of text. Use LEFT, RIGHT, or CENTER. Note that to position the text inside the widget, use the anchor option. Default is LEFT.

padx: Horizontal padding. Default is -1 (no padding).

pady: Vertical padding. Default is -1 (no padding).

relief: Border decoration. The default is FLAT. Other possible values are SUNKEN, RAISED, GROOVE, and RIDGE.

takefocus: If true, the widget accepts input focus. The default is false.

text: Message text. The widget inserts line breaks if necessary to get the requested aspect ratio. (text/Text)

textvariable: Associates a Tkinter variable with the message, which is usually a StringVar. If the variable is changed, the message text is updated.

width: Widget width given in character units. A suitable width based on the aspect setting is automatically chosen, if this option is not given.

第6章　ラジオボタン・ウィジェット

選択ボタンとも呼ばれるラジオボタンは予め設定された選択肢の内から一つのみを選択させたいときにと使う。ラジオボタンはテキストや画像を含ませることができるが使えるフォントは一つだけである。ラジオボタンが選択されたときに実行される関数やメソッドを設定することもできる。

ラジオボタンの由来は古いラジオで放送局を選択するのに使ったボタンである。一つのボタンを押すと他のボタンは浮き上がり、押したボタンのみが押された状態になっている。

簡単な実例を示す：

```
#coding: utf-8
import tkinter as tk

root = tk.Tk()

v = tk.IntVar()

tk.Label(root,
        text="""プログラミング言語を \n  \
一つ選択していください。:""",
        justify = tk.LEFT,
        padx = 20).pack()
tk.Radiobutton(root,
               text="Python",
               padx = 20,
               variable=v,
               value=1).pack(anchor=tk.W)
tk.Radiobutton(root,
               text="Perl",
               padx = 20,
               variable=v,
```

```
                value=2).pack(anchor=tk.W)

root.mainloop()
```

変数 v は IntVal クラスにインスタンスで各々のラジオボタンに variable オプションに引数の値として設定されている。ユーザがラジオボタンの一つを選択すると、そのボタンの value の値が v に代入される。そしてそのボタンは on の状態になる。それ以外のボタンは off の状態になる。
実行例

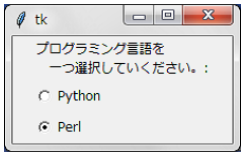

図 6.1: 簡単なラジオボタン

　ラジオボタンが多くなってくると少し工夫がいる。以下の例では text に入れる文字列と value に入れる整数値をタプルのペアとしてこのタプルのリストを作り、複数のボタンを for ループで作る。
　実例を示す：

```
#coding: utf-8
import tkinter as tk

root = tk.Tk()

v = tk.IntVar()
v.set(1)  # value の値が 1 のものを選択。つまり Python

languages = [
    ("Python",1),
    ("Perl",2),
    ("Java",3),
    ("C++",4),
    ("C",5)
]
```

```
def ShowChoice():
    print(v.get())

tk.Label(root,
        text="""好みのプログラミング言語を \n   \
一つ選択していください。:""",
         justify = tk.LEFT,
         padx = 20).pack()

for language, val in languages:
    tk.Radiobutton(root,
                   text=language,
                   padx = 20,
                   variable=v,
                   command=ShowChoice,
                   value=val).pack(anchor=tk.W)


root.mainloop()
```

commnad オプションはユーザがそのボタンを選択したときに実行される関数を設定するためのものである。この例では関数 ShowChoice。IntVar クラスのメソッド set はその引数の整数値を v に設定するためのもの。get は今 v が持っている整数値（つまりはユーザが選択したボタンの value の値）を得るためのものである。

実行例：

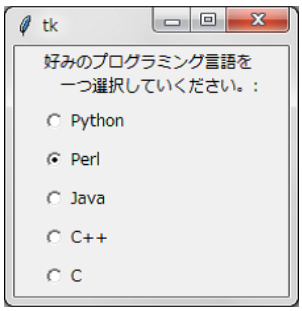

図 6.2: ラジオボタン・ウィジェット

（訳注：この例では関数 ShowChoice は選択されたラジオボタンの value の整数値の print するようになっているが、これを選択され言語名（文字列）を表示す

るように変えてみる。実例を示す (関数 ShowChoice のみ)：

```
def ShowChoice():
    for language, val in languages:
        if val == v.get():
            print(language)
            break
```

この例でも分るように関数 ShowChoice は関数の外の変数（v、languages ）を直接に参照している。このような関数の使いかたに抵抗のあるひとは後述するように無名関数を使うとよい。）

6.1　インジケーター

ラジオボタンの特殊型として穴のないラジオボタン（インジケーター）を作ってみる。これはラジオボタンの indicatoron オプションを 0(既定値は 1）にするとできる。for ループの部分のみを示す：

```
tk.Radiobutton(root,
                  text=language,
                  indicatoron = 0,
                  width = 20,
                  padx = 20,
                  variable=v,
                  command=ShowChoice,
                  value=val).pack()
```

少し雰囲気が違う表示ができる。

実行例：

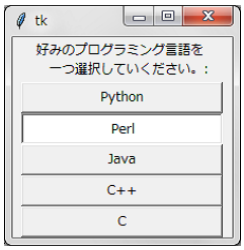

図 6.3: インジケーター

第7章　チェクボックス

チェクボックスまたはチックボックスはユーザが複数の選択肢から複数を選択するようなことを可能とするウィジェットである。複数の選択肢を選択できる点でラジオボタンと異なる。

チェクボックスは通常白い正方形で示される。ユーザがその選択肢をクリックすると選択されたことを示すチックマーク (check mark) や × がそこに現れる。選択しないと白の正方形のままである。チェクボックスの意味を記述した短い語句はそのチェクボックスに添えて表示される。

Tkinter のチェックボックス・ウィジェットはテキストや画像を含むことができる。またボタンはこのボタンが押されたときに起動される関数やメソッドを割り振ることもできる。テキストは複数行に渡って構わない。

以下の例では二つのチェックボックスを持つ。

```
#coding: utf-8
from tkinter import *

master = Tk()
var1 = IntVar()
Checkbutton(master, text="男性", variable=var1).grid(row=0, sticky=W)
var2 = IntVar()
Checkbutton(master, text="女性", variable=var2).grid(row=1, sticky=W)
mainloop()
```

ボックスの状態（チェックされているか、いないか）は変数 var1 と var2 でわかる。

簡単すぎるので少し工夫をする。一つのラベルを追加し、さらに二つのボタン（一つは終了のボタン、もう一つはチェックボックスの状態（選択されているかどうか）を print 文で表示するボタン）を付ける。

```
#coding: utf-8
from tkinter import *
master = Tk()
```

```
def var_states():
   print("男性: %d\n 女性: %d" % (var1.get(), var2.get()))

Label(master, text="男性？女性？").grid(row=0, sticky=W)
var1 = IntVar()
Checkbutton(master, text="男性", variable=var1).grid(row=1, sticky=W)
var2 = IntVar()
Checkbutton(master, text="女性", variable=var2).grid(row=2, sticky=W)

Button(master, text='終了', command=master.quit)    \
                                                    .grid(row=3, sticky=W
Button(master, text='表示', command=var_states)     \
                                                    .grid(row=4, sticky=W
mainloop()
```

ボタンではそれが押されたときに実行される関数およびメソッドが定義できる。終了ボタンではこのアプリケーションを終了させるメソッド master.quit が設定されているし、表示ボタンでは関数 var_states が実行されるよう設定されている。この関数で使われている変数 var1 及び var2 はこの関数の外にあるメインで定義されている大域変数であることにも注意すること。

実行例：さて実用的な状況ではチェックボックスの数は多くなるのでループを使っ

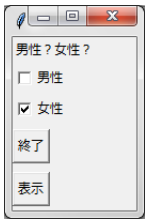

図 7.1: チェックボックス

てチェックボックスを定義する必要がある。例としてコンピュータ言語 ['Python', 'Ruby', 'Perl', 'C++'] と自然言語 [' 英語',' 日本語'] を選択するチェックボックスを作ってみる。今回は複数のチェックボックスを収納するフレームの一種であるチェックバーをクラス定義 Checkbar にして使う。

```
#coding: utf-8
from tkinter import *
class Checkbar(Frame):
```

```
   def __init__(self, parent=None, picks=[], side=LEFT, anchor=W):
      Frame.__init__(self, parent)
      self.vars = []
      for pick in picks:
         var = IntVar()
         chk = Checkbutton(self, text=pick, variable=var)
         chk.pack(side=side, anchor=anchor, expand=YES)
         self.vars.append(var)
   def state(self):
      return map((lambda var: var.get()), self.vars)

if __name__ == '__main__':
   root = Tk()
   lng = Checkbar(root, ['Python', 'Ruby', 'Perl', 'C++'])
   tgl = Checkbar(root, [' 英語',' 日本語'])
   lng.pack(side=TOP,  fill=X)
   tgl.pack(side=LEFT)
   lng.config(relief=GROOVE, bd=2)

   def allstates():
      print(list(lng.state()), list(tgl.state()))
   Button(root, text=' 終了', command=root.quit).pack(side=RIGHT)
   Button(root, text=' 確認', command=allstates).pack(side=RIGHT)
   root.mainloop()
```

（訳注：クラス Checkbar の定義は継承した Frame クラスの属性やメソッドをそのまま引継ぎ、唯一の変更は init 関数で、それも殆どが Frame クラスのそれを引き継ぐ。

```
      Frame.__init__(self, parent)
```

クラス Checkbar はフレームの親クラスの root と言語のリストを引数としてインスタンスを具現化する。そのクラス内ではこのリストは変数 picks に渡される。一つの Checkbar 内の一つ一つの言語 Checkbtton の状態（選択されているか、いないか）は変数 self.vars にリストとして保存される。

二つ付けるボタンは一つは簡単で「終了」ボタンでそれを押すとこのアプリケーションが終了する。一方、「確認」ボタンが押されると関数 allstates が実行される。この関数は lng.state()、tgl.state() の両関数の結果をリストとして可視化し print とする。

各チェックボックスの状態は self.vars にリストとして格納されている変数に対するメソッド get() で得られるので、その要素を例えば var という変数に代入して var.get() とすると状態の値（1:選択されている；0:選択されていない）がえられる。

```
def state(self):
      return map((lambda var: var.get()), self.vars)
```

無名関数 lambda は仮引数が var で戻り値が var.get() の簡単な関数である。この関数の引数にリスト self.vars 要素の一つをこの関数の仮引数にわたすと戻り値としてその状態値がえられる。それを全ての要素について横断的に行うのが map 関数である。ただし map 関数の戻り値は抽象的な map オブジェクトであり。これを可視できるリストに変えるのは list 関数である。）

実行例：

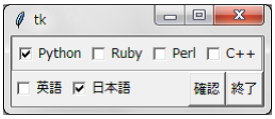

図 7.2: チェックボックス

第8章　エントリー

　ユーザが入力した文字テキストを受け取る役割をするウィジェットである。このウィジェットでは一行に収まるテキストの入力が可能である。表示窓より長い文字列の入力では表示窓に表示される文字列の部分の前後は隠れてしまうが、左右の矢印キーで文字列の表示部分を変えることができる。

```
from tkinter import *

master = Tk()
Label(master, text="First Name").grid(row=0)
Label(master, text="Last Name").grid(row=1)

e1 = Entry(master)
e2 = Entry(master)

e1.grid(row=0, column=1)
e2.grid(row=1, column=1)

mainloop( )
```

実行例
　これで文字列入力窓はできた。

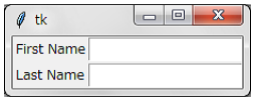

図 8.1: 簡単なエントリー

　ユーザはこの窓に文字を入力できる。しかしプログラムはどのようにしてこの文字列にアクセスできるのか？如何にしたら入力文字列を読むことができるか？
　簡単に言うと get() メソッドが探していたものである。

上のサンプルにボタンを二つ追加する。一つ「終了」のためのものであり、もう一つは現在の入力情報をモニターに表示するものである。後者のボタンはボタンが押される度に関数show_entry_field()が実行される。さらにこのヴァージョンではエントリー・フィールドには最初に既定値が設定される。それは

```
e1.insert(10, "太郎")
e2.insert(10, "浦島")
```

で与える。さらにエントリー・フィールドの内容を表示した後には現在のエントリー・フィールドの内容を消去する。これにはdeleteメソッドを使う。delete(0,END)とするとフィールドの内容を全て消去できる。

```
#coding: utf-8
import tkinter as tk

def show_entry_fields():
   print("名前は: %s\n 姓名は: %s" % (e1.get(), e2.get()))
   e1.delete(0,tk.END)
   e2.delete(0,tk.END)

master = tk.Tk()
tk.Label(master, text="名前").grid(row=0)
tk.Label(master, text="姓名").grid(row=1)

e1 = tk.Entry(master)
e2 = tk.Entry(master)
e1.insert(10,"太郎")
e2.insert(10,"浦島")

e1.grid(row=0, column=1)
e2.grid(row=1, column=1)

tk.Button(master, text='終了', command=master.quit)  \
               .grid(row=3, column=0, sticky=tk.W, pady=4)
tk.Button(master, text='表示', command=show_entry_fields) \
               .grid(row=3, column=1, sticky=tk.W, pady=4)

tk.mainloop( )
```

実行例
これで基本的なことはできた。

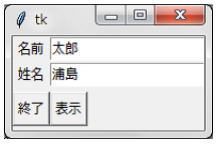

図 8.2: 基本的なエントリー

次のサンプルはエントリー窓が多くなった実用的な例である。ここではエントリー・フィールドは Python のリストとして扱われる。

```
#coding: utf-8
import tkinter as tk
fields = '姓', '名', '職業', '国籍'

def fetch(entries):
   for entry in entries:
      field = entry[0]
      text  = entry[1].get()
      print('%s: "%s"' % (field, text))

def makeform(root, fields):
   entries = []
   for field in fields:
      row = tk.Frame(root)
      lab = tk.Label(row, width=15, text=field, anchor='w')
      ent = tk.Entry(row)
      row.pack(side=tk.TOP, fill=tk.X, padx=5, pady=5)
      lab.pack(side=tk.LEFT)
      ent.pack(side=tk.RIGHT, expand=tk.YES, fill=tk.X)
      entries.append((field, ent))
   return entries

if __name__ == '__main__':
   root = tk.Tk()
   ents = makeform(root, fields)
   root.bind('<Return>', (lambda event, e=ents: fetch(e)))
```

```
b1 = tk.Button(root, text='Show',
       command=(lambda e=ents: fetch(e)))
b1.pack(side=tk.LEFT, padx=5, pady=5)
b2 = tk.Button(root, text='Quit', command=root.quit)
b2.pack(side=tk.LEFT, padx=5, pady=5)
root.mainloop()
```

実行例

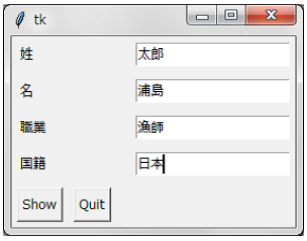

図 8.3: 少し実用的なエントリー

（訳注：エントリーウィジェットを作る項目は fields という文字列のタプルに入れてある。関数 makeform ではこの項目のタプルを横断的に処理して各項目ごとにフレーム（Frame)、つまり枠組みを作りこれを基準にフィールド、エントリーのイメージ表示をする。さらにフィールドとエントリーを対のタプルとして全ての項目についてそれをリストにして戻している。

“Show” とラベルされたボタンと “Quit” とラベルされたボタンが設定される。前者のボタンではボタンが押されると

```
(lambda e=ents: fetch(e))
```

で定義される関数が実行される。この無名関数による関数オブジェクトの定義は以下の例で説明する：

```
import math
square_root = lambda x: math.sqrt(x)

print(square_root(10.0))
```

関数定義を無名関数を使って行っている。ラムダ関数からの戻り値は関数オブジェクトである。それを square_root という変数に代入している。これと同様に fetch 関数を実引数 ents を与えて実行する関数オブジェクトが command に代入される。

後者のボタンはこのアプリケーションの終了である。

またこの例ではエントリーでユーザーがリターン・キー (Return) を入力したときに現在の入力値を表示するイベント処理も設定されている。このようなイベント処理は後に詳しく触れる。）

第9章　カンバス

カンバスウィジェットはグラフィックスの部品を提供している。線、円、画像やその他多くのウィジェットを表示する機能がある。

9.1　線

第一の例は線を描く例である。メソッド create_line(coords, options) が直線の描くものである。座標 coords は四つの整数 x1,x2,y1,y2 で与えられる。引き続くオプションは無しでも構わないが例えば線の色を指定するには fill="#476042" とするとよい。

```
import  tkinter as tk
master = tk.Tk()

canvas_width = 80
canvas_height = 40
w = tk.Canvas(master,
           width=canvas_width,
           height=canvas_height)
w.pack()
y = int(canvas_height / 2)
w.create_line(0, y, canvas_width, y, fill="#476042")
master.mainloop()
```

実行例

図 9.1: 直線

カンバスの横幅と縦幅は canvas_width と canvas_height で与えられている。直線はこのカンバスを上下に二分する水平な線である。

次の例は長方形を含む例である。長方形の位置と大きさは第 1 に左上の座標 (x,y)、次に右下の座標（x,y）で与える。

```
import tkinter as tk

master = tk.Tk()
w = tk.Canvas(master, width=200, height=100)
w.pack()

w.create_rectangle(50, 20, 150, 80, fill="#476042")
w.create_rectangle(65, 35, 135, 65, fill="yellow")
w.create_line(0, 0, 50, 20, fill="#476042", width=3)
w.create_line(0, 100, 50, 80, fill="#476042", width=3)
w.create_line(150,20, 200, 0, fill="#476042", width=3)
w.create_line(150, 80, 200, 100, fill="#476042", width=3)

master.mainloop()
```

実行例

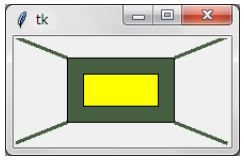

図 9.2: 長方形

9.2　テキスト

カンバスのテキストを表示する例である。メソッド create_text() がそれで最初の二つのパラメターは表示するテキストのカンバス座標である。既定値ではテキストはこの座標を中心になるように表示される。選択肢 “anchor option” のよってこれを変えることができる。例えば anchor = NW とすると座標はテキストの左上となる。

```
import tkinter as tk

canvas_width = 200
canvas_height = 100

colours = ("#476042", "yellow")
box=[]

for ratio in ( 0.2, 0.35 ):
   box.append( (canvas_width * ratio,
                canvas_height * ratio,
                canvas_width * (1 - ratio),
                canvas_height * (1 - ratio) ) )

master = tk.Tk()
w = tk.Canvas(master,
           width=canvas_width,
           height=canvas_height)
w.pack()

for i in range(2):
   w.create_rectangle(box[i][0], box[i][1],box[i][2],box[i][3], \
                              fill=colours[i])

w.create_line(0, 0,
              box[0][0], box[0][1],
              fill=colours[0],
              width=3)
w.create_line(0, canvas_height,
              box[0][0], box[0][3],
              fill=colours[0],
              width=3)
w.create_line(box[0][2],box[0][1],
              canvas_width, 0,
              fill=colours[0],
              width=3)
```

```
w.create_line(box[0][2], box[0][3],
              canvas_width, canvas_height,
              fill=colours[0], width=3)

w.create_text(canvas_width / 2,
              canvas_height / 2,
              text="Python")
master.mainloop()
```

実行例

図 9.3: テキスト

9.3　卵形

長方形の左下座標 x0,y0 と右上座標 x1,y1 に収まる卵形を描く。

```
id = C.create_oval ( x0, y0, x1, y1, option, ... )
```

円は正方形に収まる卵形である。従って座標 x,y を中心に半径 r の円を描く関数は以下のようになる：

```
id = canvas.create_oval(x-r,y-r,x+r,y+r)
```

9.4　手書き描画

マウスがドラッグされていること検出しそのマウスの座標に極小さい（半径＝１）の円を描き、この操作を続けることでマウスドラッグの軌跡を描く。

```
#coding utf-8
import tkinter as tk
```

```
canvas_width = 500
canvas_height = 150

def paint( event ):
   python_green = "#476042"
   x1, y1 = ( event.x - 1 ), ( event.y - 1 )
   x2, y2 = ( event.x + 1 ), ( event.y + 1 )
   w.create_oval( x1, y1, x2, y2, fill = python_green )

master = tk.Tk()
master.title( "描画" )
w = tk.Canvas(master,
           width=canvas_width,
           height=canvas_height)
w.pack(expand = tk.YES, fill = tk.BOTH)
w.bind( "<B1-Motion>", paint )

message = tk.Label( master, text = "マウスをドラッグすると軌跡が残る" )
message.pack( side = tk.BOTTOM )

master.mainloop()
```

マウスドラッグの検出は"<B1-Motion>"のイベントであり、このイベントを検出したときのイベント処理は関数 paint である。実行例

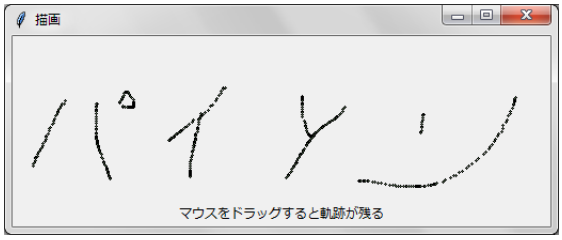

図 9.4: 手書き描画

9.5 多角形

多角形を描きたいときには create_polygon に最低でも三つの座標値を与えるとできる。以下は簡単な例である。

```
import tkinter as tk

canvas_width = 200
canvas_height =200
python_green = "#476042"

master = tk.Tk()

w = tk.Canvas(master,
            width=canvas_width,
            height=canvas_height)
w.pack()

points = [0,0,canvas_width,canvas_height/2, 0, canvas_height]
w.create_polygon(points, outline=python_green,
             fill='yellow', width=3)

master.mainloop()
```

実行例

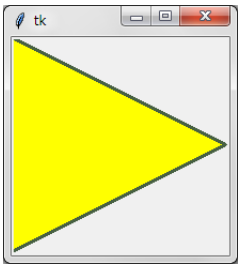

図 9.5: 三角形

クリスマスシーズンでないかもしれないが、次は星を表現してみる。最初はプログラミングのスキルが殆んどない直接的な手法で座標値を与えたものである。

```
import tkinter as tk

canvas_width = 200
canvas_height =200
```

```
python_green = "#476042"

master = tk.Tk()

w = tk.Canvas(master,
            width=canvas_width,
            height=canvas_height)
w.pack()

points = [100, 140, 110, 110, 140, 100, 110, 90, 100, \
          60, 90, 90, 60, 100, 90, 110]

w.create_polygon(points, outline=python_green,
             fill='yellow', width=3)

master.mainloop()
```

実行例

図 9.6: 星

この方法はあまり上手いものではない。もしも「星」の大きさを変えたいと思ったときは座標値を直接に書き直す必要が出てくる。そこで座標値の与えかたと関数とする。

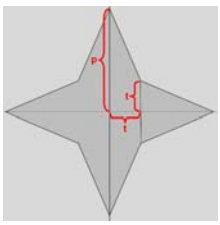

図 9.7: 説明

上記の図で分るように星の中心の座標が与えられるとこの「星」型は二つのパラメター（p,t）を与えると描くことができる。この手法でできる「星」型の一例を以下に示す。

```
import tkinter as tk

canvas_width = 400
canvas_height =400
python_green = "#476042"

def polygon_star(canvas, x,y,p,t, outline=python_green, \
                 fill='yellow', width = 1):
   points = []
   for i in (1,-1):
      points.extend((x,      y + i*p))
      points.extend((x + i*t, y + i*t))
      points.extend((x + i*p, y))
      points.extend((x + i*t, y - i * t))

   print(points)

   canvas.create_polygon(points, outline=outline,
                         fill=fill, width=width)

master = tk.Tk()

w = tk.Canvas(master,
           width=canvas_width,
           height=canvas_height)
w.pack()

p = 50
t = 15

nsteps = 10
step_x = int(canvas_width / nsteps)
step_y = int(canvas_height / nsteps)
```

```
for i in range(1, nsteps):
   polygon_star(w,i*step_x,i*step_y,p,t,outline='red',fill='gold', width=3)
   polygon_star(w,i*step_x,canvas_height - i*step_y,p,t,outline='red', \
                fill='gold', width=3)

master.mainloop()
```

実行例

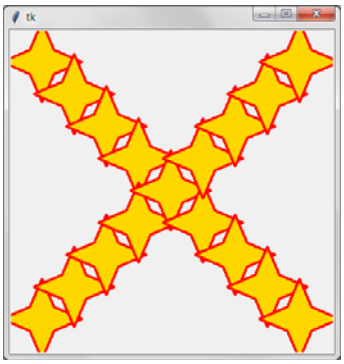

図 9.8: 星、星、星....

9.6 画像の貼り付け

メソッド create_bitmap() でカンバス上にビットマップ画像を表示できる。以下の小さいビットマップは tkinter の組み込み画像で全ての処理系で使うことができる。

```
import tkinter as tk

canvas_width = 300
canvas_height =80

master = tk.Tk()
canvas = tk.Canvas(master,
           width=canvas_width,
           height=canvas_height)
canvas.pack()
```

```
bitmaps = ["error", "gray75", "gray50", "gray25", "gray12", \
           "hourglass", "info", "questhead", "question", "warning"]
nsteps = len(bitmaps)
step_x = int(canvas_width / nsteps)

for i in range(0, nsteps):
   canvas.create_bitmap((i+1)*step_x - step_x/2,50, bitmap=bitmaps[i])

master.mainloop()
```

実行例

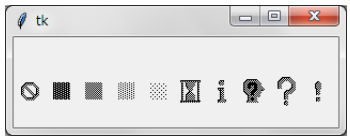

図 9.9: 組み込み画像

今度は普通の画像をカンバスに貼り付ける例である。
create_image(x0,y0, options ...) を使うが直接画像ファイルを受け付けるのでは無く、メソッド PhotoImage() を使ってイメージオブジェクトを作成そのイメージオブジェクトを使って画像を表示する。

```
import tkinter as tk

canvas_width = 200
canvas_height =300

master = tk.Tk()

canvas = tk.Canvas(master,
           width=canvas_width,
           height=canvas_height)
canvas.pack()

img = tk.PhotoImage(file="rocks.png")
```

```
canvas.create_image(20,20, anchor=tk.NW, image=img)

master.mainloop()
```

実行例

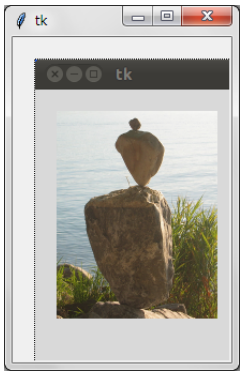

図 9.10: 画像の表示

第10章　スライダー

スライダーは可動表示板を動かすことにようて値を設定できるTkinterオブジェクトである。スライダーは横にも縦にも配置することができる。設定はScaleメソッドを使う。スケールウィジェットはユーザがそれに付いているノブを動かすことで数値を選択できる機能を持っている。スケールの最小値、最大値、分解能はパラメータで設定できる。また縦位置にするか横位置にするかも設定できる。

スケールウィジェットはユーザが期待される数値幅の中で入力すべき値を知っているときにはエントリーウィジェットの代替として使える。

簡単な実例：

```
import tkinter as tk

master = tk.Tk()
w = tk.Scale(master, from_=0, to=42)
w.pack()
w = tk.Scale(master, from_=0, to=200, orient=tk.HORIZONTAL)
w.pack()

master.mainloop()
```

これで簡単なスラダーができたが、これでは実用にならない。プログラムで現在指定されている値を尋ねることができる必要がある。これにはgetメソッドを使う。前サンプルにボタンを付けてボタンが押されたらその時点で指定されているスライダーの値を表示する機能を追加しよう。

```
import tkinter as tk

def show_values():
    print (w1.get(), w2.get())

master = tk.Tk()
w1 = tk.Scale(master, from_=0, to=42)
```

```
w1.pack()
w2 = tk.Scale(master, from_=0, to=200, orient=tk.HORIZONTAL)
w2.pack()
tk.Button(master, text='Show', command=show_values).pack()

master.mainloop()
```

変数 w1 と w2 は関数 show_values の外部で定義され、この関数で参照されていることに注意。

スラダーに目盛りを付けることもできる。それは選択肢 tickinterval に値を設定すればよい。

```
import tkinter as tk

def show_values():
    print (w1.get(), w2.get())

master = tk.Tk()
w1 = tk.Scale(master, from_=0, to=42, tickinterval=20)
w1.set(19)
w1.pack()
w2 = tk.Scale(master, from_=0, to=200,tickinterval=100, orient=tk.HORIZONTAL)
w2.set(23)
w2.pack()
tk.Button(master, text='Show', command=show_values).pack()

master.mainloop()
```

実行例

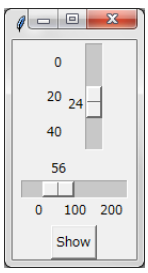

図 10.1: スラダー

第11章　テキストウィジェット

テキストウィジェットは複数行に渡るテキストに関わる領域の目的で使われる。Tkinter のテキストウィジェットは大変に強力かつ柔軟で様々の目的に使える。

最も簡単な例からはじめよう。二行三十文字が入るテキスト領域を Text クラスを使って作る。そのインスタンス t として、そのインスタンスの insert メソッドを使って実際のテキストを設定し、それを表示させる。

```
#coding utf-8
import tkinter as tk

root = tk.Tk()
t = tk.Text(root, height=2, width=30)
t.pack()
t.insert(tk.END, "たった二行の\n テキストウィジェット。\n")
root.mainloop()
```

実行例

図 11.1: テキストウィジェット

もう少し長めのテキストを表示させてみよう。表示領域は前例の同じである。

```
import tkinter as tk

root = tk.Tk()
t = tk.Text(root, height=2, width=30)
t.pack()

quote = """HAMLET: To be, or not to be--that is the question:
```

```
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune
Or to take arms against a sea of troubles
And by opposing end them. To die, to sleep--
No more--and by a sleep to say we end
The heartache, and the thousand natural shocks
That flesh is heir to. 'Tis a consummation
Devoutly to be wished."""

t.insert(tk.END, quote)
root.mainloop()
```

実行して分るようにこのテキストの最初の二行分しか表示されない。これでは意図したものではない。

表示領域を拡大させて解決させることもあるが、ここではy方向にスクロール・バーを付けることで問題の解決を図ることにする。

11.1　スクロール・バー

クラスScrollbarを使う。スクロール・バーウィジェットの親ウィジェットをパラメータとして与えインスタンスsを作り配置する。それと別にテキストを表示するテキストウィジェットtを作成・配置する。そしてこの二つのインスタンス間で相互に相手の存在を認識させる。

```
import tkinter as tk

root = tk.Tk()
s = tk.Scrollbar(root)
t = tk.Text(root, height=4, width=50)
s.pack(side=tk.RIGHT, fill=tk.Y)
t.pack(side=tk.LEFT, fill=tk.Y)
s.config(command=t.yview)
t.config(yscrollcommand=s.set)
quote = """HAMLET: To be, or not to be--that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune
Or to take arms against a sea of troubles
```

```
And by opposing end them. To die, to sleep--
No more--and by a sleep to say we end
The heartache, and the thousand natural shocks
That flesh is heir to. 'Tis a consummation
Devoutly to be wished."""

t.insert(tk.END, quote)
root.mainloop()
```

実行例

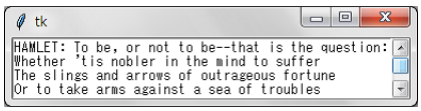

図 11.2: スクロールバー付きテキスト

次はテキストウィジェットに画像を貼り付けることにしよう。

```
import tkinter as tk

root = tk.Tk()

text1 = tk.Text(root, height=20, width=30)
photo=tk.PhotoImage(file='yjimage.gif')
text1.insert(tk.END,'\n')
text1.image_create(tk.END, image=photo)

text1.pack(side=tk.LEFT)

text2 = tk.Text(root, height=20, width=50)
scroll = tk.Scrollbar(root, command=text2.yview)
text2.configure(yscrollcommand=scroll.set)
text2.tag_configure('bold_italics', font=('Arial', 12, 'bold', 'italic'))
text2.tag_configure('big', font=('Verdana', 20, 'bold'))
text2.tag_configure('color', foreground='#476042', \
                    font=('Tempus Sans ITC', 12, 'bold'))
text2.tag_bind('follow', '<1>',  \
               lambda e, t=text2: t.insert(tk.END, "Not now, maybe later!"))
```

```
text2.insert(tk.END,'\nWilliam Shakespeare\n', 'big')
quote = """
To be, or not to be that is the question:
Whether 'tis Nobler in the mind to suffer
The Slings and Arrows of outrageous Fortune,
Or to take Arms against a Sea of troubles,
"""
text2.insert(tk.END, quote, 'color')
text2.insert(tk.END, 'follow-up\n', 'follow')
text2.pack(side=tk.LEFT)
scroll.pack(side=tk.RIGHT, fill=tk.Y)

root.mainloop()
```

（訳注：ここでメソッド tag_config は insert メソッドに付ける名札であり、テキストに付随する属性を定義している。また、メソッド tag_bind は該当するテキストがマウスの左ボタンでクリックされたとき処理すべきコマンドを記述する。ここではテキスト画面の “follow-up” 文字を左クリックする実行される関数

```
lambda e, t=text2: t.insert(tk.END, "Not now, maybe later!")
```

が設定される。）

実行例

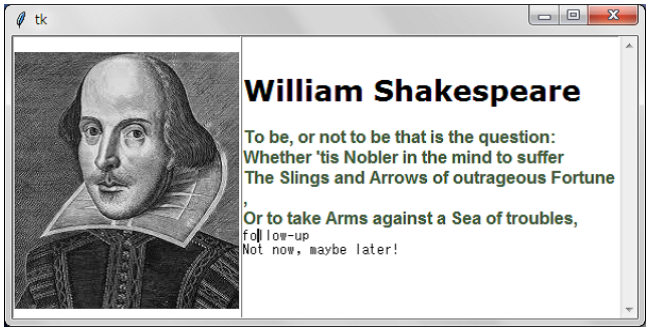

図 11.3: 画像付きテキストウィジェット

第12章　ダイアログウィジェット

Tk(勿論 Tkinter も）ダイアログ（問答）（警告表示、エラー表示及びファイル選択、色選択）のためのウィジェットを提供している。ダイアログはその内容に合わせた小さなアイコン付きのウィジェットとして現れる。またユーザに文字、整数値、浮動小数点数値を入力させるダイアログもある。

12.1　Messagebox

典型的な GUI で遭遇する例を取り上げダイアログウィジェットを紹介する。
以下のような二つのボタンを持つ画面があるとする：

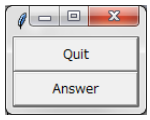

図 12.1: ユーザ作成窓

これに対してこの “Quit” ボタンを選択すると「是非」確認のダイアログが出てくる仕組みを作る。それには askyesno ウィジェットを使う。
実例

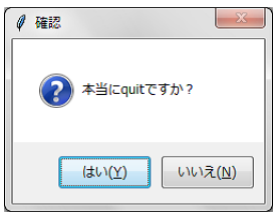

図 12.2: askyesno ウィジェット

このウィジェットは疑問符付きの窓で「はい」と「いいえ」のボタンが付いている。

「はい」というボタンを押すと、「まだ Quit は実装されていません」という警告を出したいとする。それには showwarning ウィジェットを使う。
　実例

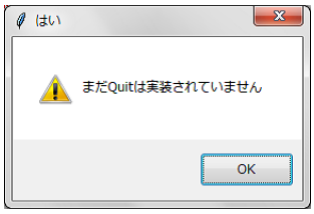

図 12.3: showwarning ウィジェット

このウィジェットは警告マーク付きの窓で窓を閉じるボタンが一つ付いている。
　一方「いいえ」を選択すると、「Quit はキャンセルされました」と表示したい。それには showinfo ウィジェットを使う。
　実例

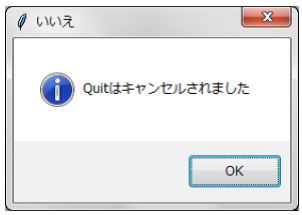

図 12.4: showinfo ウィジェット

このウィジェットは情報マーク付きの窓で窓を閉じるボタンが一つ付いている。
　最初のユーザ作成窓で、Answer ボタンを押したときにはエラー表示で「Answer は利用できません」と表示するとする。それには showerror ウィジェットを使う。
　実例

このウィジェットはエラーマーク付きの窓で窓を閉じるボタンが一つ付いている。
　以上のような機能を持つプログラムの一例を以下に示す：

```
#coding utf-8
import tkinter as tk
from tkinter  import messagebox as tkMB
```

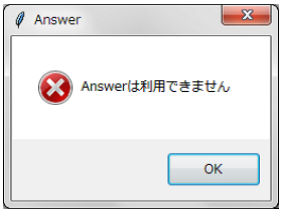

図 12.5: showinfo ウィジェット

```
def answer():
    tkMB.showerror("Answer", "Answer は利用できません")

def callback():
    if tkMB.askyesno('確認', '本当に quit ですか？'):
        tkMB.showwarning('はい', 'まだ Quit は実装されていません')
    else:
        tkMB.showinfo('いいえ', 'Quit はキャンセルされました')

tk.Button(text='Quit', command=callback).pack(fill=tk.X)
tk.Button(text='Answer', command=answer).pack(fill=tk.X)
tk.mainloop()
```

（訳注：これらのダイアログウィジェットはマクロ化されているので、これらを利用するとプログラムを短く書くことができる。
　またこれらのダイアログウィジェットはモジュール tkinter のサブもサブモジュール messagebox に収録されているので

```
from tkinter  import messagebox as tkMB
```

と別個にインポートしておく。）
　その他のダイアログウィジェットの一覧を以下列記する：

- askokcancel(title=None, message=None, **options)
  Ask if operation should proceed; return true if the answer is ok
- askquestion(title=None, message=None, **options)
  Ask a question
- askretrycancel(title=None, message=None, **options)
  Ask if operation should be retried; return true if the answer is yes

- askyesno(title=None, message=None, **options)
  Ask a question; return true if the answer is yes
- askyesnocancel(title=None, message=None, **options)
  Ask a question; return true if the answer is yes, None if cancelled.
- showerror(title=None, message=None, **options)
  Show an error message
- showinfo(title=None, message=None, **options)
  Show an info message
- showwarning(title=None, message=None, **options)
  Show a warning message

12.2 Filedialog

ファイルを開く、閉じるまたはディレクトリーを変更するなどのファイル関連の操作を持たないアプリケーションはないだろう。そのような操作のためのダイアログを纏めたモジュールが filedialog である。

ファイルの一覧を示しファイルを選択する特殊なダイアログの一例を示す。

```
#coding utf-8
import tkinter as tk
from tkinter import filedialog as tkFD



def callback():
    name= tkFD.askopenfilename()
    print(name)

errmsg = 'Error!'
w=tk.Button(text='ファイルを開く', command=callback)
w.pack(fill=tk.X)
tk.mainloop()
```

実行例

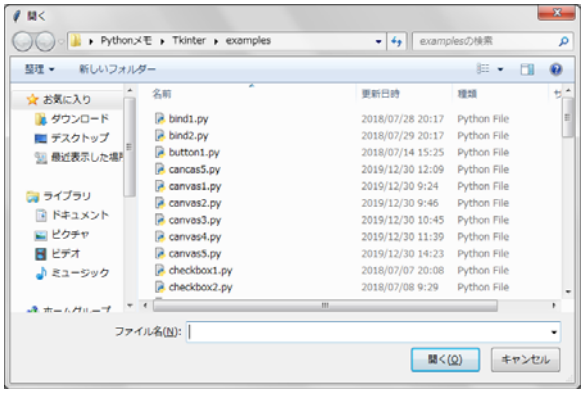

図 12.6: askopenfilename ウィジェット

このプログラムでは最初に「ファイルを開く」という小さなボタンが現れ、それを押すと実行例のような画面がでる。そしてファイルを選択するとそのファイル名がモニターに表示される。

12.3 Colorchooser

同様に特殊なダイアログでカラーの一覧から特定のカラーを選択するダイアログの一例である。

```
import tkinter as tk
from tkinter import colorchooser as tkCC

def callback():
    result = tkCC.askcolor(color="#6A9662",
                           title = "Bernd's Colour Chooser")
    print (result)

root = tk.Tk()
tk.Button(root,
        text='色選択',
        fg="darkgreen",
        command=callback).pack(side=tk.LEFT, padx=10)
tk.Button(text='終了',
        command=root.quit,
        fg="red").pack(side=tk.LEFT, padx=10)
```

```
tk.mainloop()
```

実行例

図 12.7: askcolor ウィジェット

このプログラムでは最初に「色選択」と「終了」という二と小さなボタンが現れ、その「色選択」ボタンを押すと実行例のような画面がでる。そして色を選択するとその色の RGB の値（0-255）がタプルと 8 進数とでモニターに表示される。

第13章　レイアウト管理

この章ではレイアウト管理または配置管理と呼ばれるシステムについて触れる。Tkinterではこの目的のために三つの異なったシステムを提供している。それらは

- pack
- grid
- place

大事なことはこれらを混合して使ってはならないことである。これらのシステムの目的は

- 種々のウィジェットをスクリーン上に配置する。
- 稼動しているウインドウシステムにたいして種々のウィジェットを登録する。
- 種々のウィジェットの表示の方法を管理する。

13.1　pack

packが三つの管理システムの中で最も簡単なものである。ウィジェットの表示の詳細なしにシステムは種々のウィジェットの相互の位置関係を宣言し配置する。gridやplaceに比較すると細かな管理はできないが、簡単なアプリケーションでは使うべきである。最も簡単な例から始めよう。

```
from tkinter import *

root = Tk()

Label(root, text="Red Sun", bg="red", fg="white").pack()
Label(root, text="Green Grass", bg="green", fg="black").pack()
Label(root, text="Blue Sky", bg="blue", fg="white").pack()
```

```
mainloop()
```

実行例

図 13.1: 名札のパック 1

この例では親ウィジェット root に三つのラベルウィジェットを貼り付けた。メソッド pack() を選択肢なしで用いた。こうすると pack は三つのラベルウィジェットを上から下へそして中央に配置する。さらにラベルの幅は文字の長さによっていることも分る。

　ちょっとした改良を試みる。ラベルの幅を親ウィジェットの横幅に合わせてみる。これには fill = X オプションを使う。

```
import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill=tk.X)
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill=tk.X)
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill=tk.X)

root.mainloop()
```

実行例

図 13.2: 名札のパック 2

「詰め物」(pad) の効果。細部は触れないがラベルの上下左右に空白を設定することもできる。一例を示す。

```
import tkinter as tk
root = tk.Tk()
w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill=tk.X,padx=10)
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill=tk.X,padx=10,pady=5)
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill=tk.X,padx=10)
root.mainloop()
```

実行例 垂直方向に「詰め物」を入れることもできる。padx を pady にすればよい。

図 13.3: 名札のパック 3

名札と文字との間に「詰め物」を入れることもできる。これはパラメター ipadx を設定するとよい。実行例

図 13.4: 名札のパック 4

三つのラベルウィジェットを横に並べてみる。それには side オプションを使う。三つの順に左から横のに並べるには

```
import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="red", bg="red", fg="white")
```

```
w.pack(padx=5, pady=10, side=tk.LEFT)
w = tk.Label(root, text="green", bg="green", fg="black")
w.pack(padx=5, pady=20, side=tk.LEFT)
w = tk.Label(root, text="blue", bg="blue", fg="white")
w.pack(padx=5, pady=20, side=tk.LEFT)

root.mainloop()
```

実行例

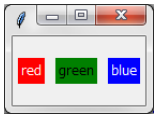

図 13.5: 名札のパック 5

side = RIGHT とすると三つのラベルを右から順に並べることもできる。

13.2 place

配置管理 place では詳細な配置を可能にしている。それらは親ウィジェットに対して絶対的な座標表示で指示することもできるし、相対的に指示することもできる。一例をしめすがこの例ではラベルの色を乱雑に選ぶといった本質的でない部分もある。

```
import tkinter as tk
import random

root = tk.Tk()
# width x height + x_offset + y_offset:
root.geometry("170x200+30+30")

languages = ['Python','Perl','C++','Java','Tcl/Tk']
labels = range(5)
for i in range(5):
   ct = [random.randrange(256) for x in range(3)]
   brightness = int(round(0.299*ct[0] + 0.587*ct[1] + 0.114*ct[2]))
   ct_hex = "%02x%02x%02x" % tuple(ct)
```

```
    bg_colour = '#' + "".join(ct_hex)
    l = tk.Label(root,
                 text=languages[i],
                 fg='White' if brightness < 120 else 'Black',
                 bg=bg_colour)
    l.place(x = 20, y = 30 + i*30, width=120, height=25)

root.mainloop()
```

この例では

```
root.geometry("170x200+30+30")
```

によって親ウィジェットの位置とサイズを指定している。幅が 170 で高さが 200 の窓である。ラベルは

```
 l.place(x = 20, y = 30 + i*30, width=120, height=25)
```

で配置される。始点が x=20、y=30 で幅が 120 で高さが 25 のラベルである。このように配置場所を座標で指定する。

実行例

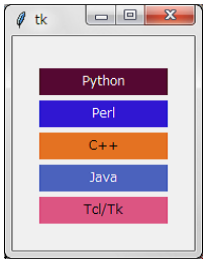

図 13.6: place による配置

13.3 grid

配置管理の 3 番手は grid である。このレイアウト管理は親ウィジェットを二次元のテーブルと見なし、そのテーブルの行 (row) 番号と列 (column) 番号を手がかりに複数の子ウィジェットを配置する。この格子のサイズは未定で配置される子ウィジェットの大きさを勘案して自動的に決められる。簡単な例を示す。

```
 from tkinter import *

colours = ['red','green','orange','white','yellow','blue']

r = 0
for c in colours:
    Label(text=c, relief=RIDGE,width=15).grid(row=r,column=0)
    Entry(bg=c, relief=SUNKEN,width=10).grid(row=r,column=1)
    r = r + 1

mainloop()
```

実行例

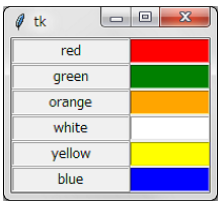

図 13.7: grid による配置

grid システムは pack と place の中間に位置し最も使いやすい。

第14章　メニュー

「メニュー」という言葉を前にすると大抵の人はレストランのメニューを重い浮かべる。レストランのメニューとコンピュータのメニューは一見あまり似ていないと思われるが共通する特徴が多い。レストランのメニューでは主菜、飲み物などの大項目がありその下提供する品物が並んでいる。一方コンピュータのメニューでは「ファイル」「編集」などの大項目がありその下に関連するコマンドが並ぶ。

（訳注：プルダウン・メニューのほかにポップアップ・メニューというものもある。画面の任意の場所でマウスの右クリックで出てくる（ポップアップ）メニューの出し方でマウスを所定の位置まで動かす手間がない点で GUI として興味深いものである。Tkinter によるポップアップ・メニューの例を追加した。）

14.1　プルダウン・メニュー

簡単なファイルオープンのためのメニューの例をしめす。

```
import tkinter as tk
from tkinter.filedialog import askopenfilename
def NewFile():
    print("New File!")
def OpenFile():
    name = askopenfilename()
    print(name)
def About():
    print("This is a simple example of a menu")

root = tk.Tk()
menu = tk.Menu(root)
root.config(menu=menu)
filemenu = tk.Menu(menu)
menu.add_cascade(label="File", menu=filemenu)
```

```
filemenu.add_command(label="New", command=NewFile)
filemenu.add_command(label="Open...", command=OpenFile)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.quit)

helpmenu = tk.Menu(menu)
menu.add_cascade(label="Help", menu=helpmenu)
helpmenu.add_command(label="About...", command=About)

root.mainloop()
```

実行例

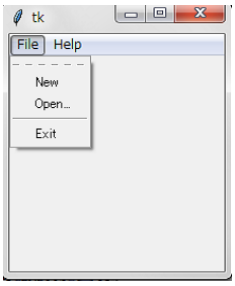

図 14.1: プルダウン・メニュー

14.2　ポップアップ・メニュー

次にポップアップ・メニューの一例を示す。これはウィジェットの内容に従って表現されるメニューでそのウィジェットをマウスで右クリックすると躍り出るメニューになっていることが多い。ここの例はラベルウィジェットに対して設定されたポップアップメニューである。

```
#coding utf-8
import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="右クリックでメニュー表示", \
          width=25, height=10)
```

```
w.pack()

# create a menu
popup = tk.Menu(root, tearoff=0)
popup.add_command(label="次へ") # , command=next) etc...
popup.add_command(label="前へ")
popup.add_separator()
popup.add_command(label="ホーム")

def do_popup(event):
    # display the popup menu
    try:
        popup.tk_popup(event.x_root, event.y_root, 0)
    finally:
        # make sure to release the grab (Tk 8.0a1 only)
        popup.grab_release()

w.bind("<Button-3>", do_popup)

b = tk.Button(root, text="終了", command=root.destroy)
b.pack()

root.mainloop()
```

実行例

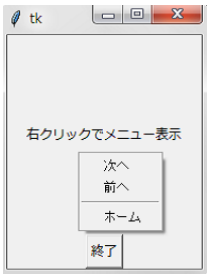

図 14.2: ポップアップ・メニュー

第15章　イベントと結合

Tkinter のアプリケーションは mainloop メソッドによって起動されたイベントループの中に留まっている。そしてユーザによるキーボードやマウスの操作によるイベント発生を待ってる。

Tkinter はプログラマーがこのイベント発生にたいして対処するための機構を提供している。つまり個々のウィジェットで発生が期待されるイベントに対してイベント処理の Python プログラムを結合させることができるようになっている。その一般型は

```
widget.bind(event, handler)
```

である。ウィジェット（widget）に関連つけられたイベント（event）に対してこのイベントが発生した際の処理関数（hander）を結合させることができるわけである。簡単な例を示す。

```
from tkinter import *
def hello(event):
    print("Single Click, Button-l")
def quit(event):
    print("Double Click, so let's stop")
    import sys; sys.exit()

widget = Button(None, text='Mouse Clicks')
widget.pack()
widget.bind('<Button-1>', hello)
widget.bind('<Double-1>', quit)
widget.mainloop
```

この例では二つのイベントが使われている。一つは<Button-1>でマウスの左ボタンによるクリックであり、二つ目は<Double-1>でマウスの左ボタンのダブルクリックである。それぞれのイベント処理関数は hello と quit である。これらの関数は event を引数とする関数でそれ以外の引数は取れない。hander は関数オブジェクトであることに注意。

もう一つの例を示す：

```
from tkinter import *

def motion(event):
  print("Mouse position: (%s %s)" % (event.x, event.y))
  return

master = Tk()
whatever_you_do = "Whatever you do will be insignificant, \
but it is very important that you do it.\n(Mahatma Gandhi)"
msg = Message(master, text = whatever_you_do)
msg.config(bg='lightgreen', font=('times', 24, 'italic'))
msg.bind('<Motion>',motion)
msg.pack()
mainloop()
```

この例ではイベントは<Motion>、つまりマウスの動きを検出したことを示すイベント、このイベント処理関数は motion である。メッセージボックス内でマウスを動かすとこのイベントを検出したことがわかる。どの位の頻度でこのイベントを検出するのかは使っているコンピュータの処理能力による。

その他のイベントについて纏めておく：

<Button>：ウィジェット上でマウスボタンの一つが押された。どのボタンかは細部パートで指定する。左ボタンの場合は<Button-1>、中ボタンでは<Button-2>となり、最も右のボタンでは<Button-3> となる。ホイール付きのマウスでスクロール・アップは<Button-4>、スクロール・ダウンは<Button-5>となる。マウスボタンを押したままでいると Tkinter は自動的にマウスを押さこむ。この状態で Motion や Release のようなマウスイベントが検出されるとそれはマウスが当該のウィジェットの外にあってもイベント検出の信号を送る。<ButtonPress> と表現されることもあるし、省略して単に<1>と書かれることもある。

<Motion>：マウスが動いていることによるイベント。ボタンを押しながらの移動の検出は<B1-Motion>などを使う。ウィジェットに相対的なマウスの現在の座標は event.x、event.y で得られる。

<ButtonRelease>: マウスボタンが離されたときに発生するイベント。ボタンを特定するには<ButtonRelease-1> などとする。

<Double-Button>: マウスをダブルクリックしたときに発生するイベント。これもボタンを特定するには<Double-Button-2> などとする。

<Enter>：マウスの矢印がウィジェット内に入ったとき発生する。

<Leave>：マウスの矢印がウィジェットの外にでたときに発生する。

<FocusIn>：エントリーのようなキー入力が期待されるウィジェットでキー入力の状態が実現したときに発生する。実例を示す：

```
from tkinter import *

def disp_state(event,en):
    print('Focusin',en)

master = Tk()
Label(master, text="First Name").grid(row=0)
Label(master, text="Last Name").grid(row=1)

e1 = Entry(master)
e2 = Entry(master)

e1.bind("<FocusIn>", lambda e, en='entry1': disp_state(e, en))
e2.bind("<FocusIn>", lambda e, en='entry2': disp_state(e, en))

e1.grid(row=0, column=1)
e2.grid(row=1, column=1)

mainloop( )
```

<FocusOut>：キーボードフォーカスの状態が当該のウィジェットから他のウィジェットに移行したときに当該ウィジェットで発生する。

<Return> ：ユーザがエンターキーを叩くと発生する。キーボード上の如何なるキーも結合させることができるがスペシャルキーといわれるものがある。その一つがエンターキーである。

`<Key>`： キーボード上に如何なるキーが叩かれても発生する。キーの判別はイベント変数 `event` の “char member” によって判別できる。スペシャルキーに対しては空白。

`a`： “a” キーが叩かれたことで発生する。印字可能なキーの全てが同様な仕様で使える。例外は「スペースキー」(`<space>`）と「小なりキー」(`<less>`)。`1` はキーボード結合であるが`<1>`はマウスボタン結合であるので注意。

`<Shift-Up>`： シフトキーを押しながら上向き矢印を押したとき発生。“Shift” の替わりの “Alt” そして “Control” キーも使える。

`<Configure>`： ウィジェットのサイズが変更したとき発生する。新しいサイズは `event` インスタンスの属性である `width` と `height` で分る。

おわりに

Tkinter の包括的な説明があり、且つ退屈しないで学べる文献があるかなと探しているときに出会ったのがこの文献である。原文献は講習会の資料のようでプログラムの詳しい説明がないので訳注として多少の説明を加えた。

原文献では “Mastermind” のプログラムを作成する項目があったが tkinter の学習とは少し離れているので省略した。

訳者