目 次

1 ウィンドウの生成 **3**
1.1 サンプルプログラム 3
1.2 解説 3
1.3 Toplevel 4
1.4 扱われる問題の範囲 4
1.5 ウィジェットかそれともインスタンスか? 6

2 Frame 6
2.1 Frame の機能 6
2.2 Frame のオプション 6
2.3 Frame のメソッド 6

3 Label 7
3.1 Label の機能 7
3.2 Label のオプション 7

4 Message 8
4.1 Message の機能 8
4.2 Message のオプション 8

5 Dialog 9
5.1 Dialog の機能 9
5.2 Dialog のオプション 10

6 Button 10
6.1 Button の機能 10
6.2 Button のオプション 11
6.3 Button のメソッド 12

7 Checkbutton 13
7.1 Checkbutton の機能 13
7.2 Checkbutton のオプション 14
7.3 Checkbutton のメソッド 16

8 Radiobutton 16
8.1 Radiobutton の機能 16
8.2 Radiobutton のオプション 18
8.3 Radiobutton のメソッド 19

9 Menu **20**
9.1 Menu の機能 20
9.2 Menu のオプション 25

9.3 Menu の add_separator メソッド 26
9.4 Menu の add_command メソッド 26
9.5 Menu の add_cascade メソッド 27
9.6 Menu の add_checkbutton メソッド 27
9.7 Menu の add_radiobutton メソッド 28

10 Menubutton 29
10.1 Menubutton の機能 29
10.2 Menubutton 33

11 Listbox 34
11.1 Listbox の機能 34
11.2 ビュー 36
11.3 Listbox のオプション 37
11.4 Listbox のメソッド 38

12 Entry 42
12.1 Entry の機能 42
12.2 Entry のオプション 46
12.3 Entry のメソッド 47

13 Text 50
13.1 Text の機能 50
13.2 Text のオプション 51

14 Scale **52**
14.1 Scale の機能 52
14.2 Scale のオプション 53
14.3 Scale のメソッド 54

15 Scrollbar 54
15.1 Scrollbar の機能 54
15.2 Scrollbar のオプション 57
15.3 Scrollbar のメソッド 58

16 Image 58
16.1 Image の機能 58

17 補足 **61**

A ウィジェット共通オプション **63**

B ウィジェット共通メソッド **67**

C カーソル名 **68**

Python における GUI の構築法 II
— ウィジェット各論 —

有澤 健治

平成 17 年 2 月 28 日

この記事は 2000 年に発表された筆者の記事の再録である[1]。 Python によるグラフィクスの解説の日本語版がまだ出ていないので役に立つと考え Web 上に公開する。なおこの記事の実行環境は古いが、解説の内容は現在でもほとんど修正を必要としていない。また末尾の参考文献には現在では当然載せるべき書物が載っていない。当時のまま載せている。必要に応じて脚注に最近の状況をコメントしてある。

はじめに

Python でマウスとグラフィクスを使用した利用者向けのインターフェース (GUI) を構成するには `Widget` クラスを使用する。Python の `Widget` クラスはボタンやメニューなど GUI 構築に必要とする殆どのウィジェット (widget: Window gadget から派生した造語でウィンドウを構成する機能部品を意味する) を提供する。

Python の `Widget` クラスに関しては、参考文献 [2] および [3] からある程度窺い知る事ができる。(参考文献 [2] は [3] の日本語訳である。) しかしながらこの著書は Python の `Widget` クラスを解説していると言うよりも、GUI の構築を例としたクラス構成法の解説に力点が置かれている。そのために `Widget` クラスの解説は省略され、また個々のウィジェットの利用法を示すサンプルプログラムすら添えられていない。我々が必要とするのはウィジェット一つひとつについての利用法の解説である。

Python の `Widget` クラスのシンタックスは `python/Lib/lib-tk/Tkinter.py` の Python のソースライブラリを参照すれば判明する。

さらに Python の `Widget` クラスの使用法は UNIX 版 Python のソースプログラムに付属する幾つかのデモプログラム `Python/Demo/tkinter/Guido/` と `Python/Demo/tkinter/Matt/` からもある程度窺い知ることができる。(何れもクラス構成の中での使用方法である。) 他方 Python の `Widget` クラスは Tcl/Tk を利用しているので Tcl/Tk の解説書 (参考文献 [4],[5]) から Python での使用法を推測する事ができる。そして現在の所、Tcl/Tk の解説書だけが Python の `Widget` クラスの使い方を知る手掛かりとなっている。

Tcl/Tk の解説書としては文献 [4] が基本的ではあるが個々のウィジェットに関しては文献 [5] の方がよく整理され、かつ詳しい。この論文は文献 [5] に記述されている各ウィジェット

[1]「Python による GUI の構築法 II — ウィジェット各論 —」(「Com」Vol.11, No.2、愛知大学情報処理センター、2000 年 9 月) 但し、読みやすくするための改訂と校正は行なっている

のオプションとメソッドが Python についても同様に成立するか否かを吟味するところから出発している。Tcl と Python は文法体系が異なるので文献 [5] の記述は必ずしも当てはまらない。またこの文献は版組の際に発生したと推測される誤りをかなり含んでいる。これらをひとつひとつ吟味して行く必要があった。

吟味して行くにあたって、Python における正確なオプション名とメソッド名を知る必要があった。そのためにこの論文で採用した方法は「第 17 節 補足」で説明されている。この方法によると Python において実際に存在するオプションとメソッドの名前は判明するが、残念ながら、それらが実際にいかなる効果を持っているかまでは分らない。効果の確認できないオプションとメソッドも含まれるのである。従ってこの論文では以下の原則でオプションとメソッドの説明を行う事にした。

1. 文献 [5] を基本にし、Python には存在しないオプションとメソッドは除外する。
2. 効果の確認できないオプションとメソッドは除外する。
3. 文献 [5] に含まれていないオプションとメソッドでも効果の確認のとれているオプションとメソッドは追加する。

この論文がほぼ完成した頃、筆者は文献 [5] よりも良くできた Tcl/Tk の文献が存在する事に気が付いた。Tcl/Tk の配布ファイルに添付されているマニュアルである (文献 [12])。これはマニュアルなのでサンプルプログラムは付いていないが Tcl/Tk に関しては完全な情報が含まれている。従って筆者の解説からさらに先に進もうとする読者はこのマニュアルで補って頂きたい。ウィジェットの解説ではウィジェットを表示するだけでは不十分である。大切な事は、プログラムの中でユーザのアクションを知る事である。この解説では Python における GUI の基本的な構成法をウィジェット毎に整理した。またウィジェットの働きのエッセンスを示す為にあえて初等的なプログラミング方法を採用した。

言葉使いについて注意を促しておこう。筆者はウィジェットと `Widget` を区別している。ウィジェットは普通名詞であるのに対して `Widget` は Python が扱うクラスの名称の一つである。従って `Widget` はそのまま書く必要があり、先頭の W を小文字に変更する事すら許されない。同様な区別はフレームと `Frame` など多くの用語に及んでいる。

`Text` クラスの解説は最小限に留めた。この解説は単独に扱うだけの価値を持っている。また完全な解説を行うにはあまりにも多くの記述が要求される。`Canvas` クラスの解説は省略した。既に筆者はこの解説を他で行っている (参考文献 [7] と [8]) ので、それを参照されたい。`Image` クラスは `Tkinter` のモジュールではあるが `Widget` に属するクラスではない。しかしウィジェットと共に用いられるのでついでに解説した。この解説に現われるサンプルプログラムは全て WindowsNT と Windows98 で動作の確認が行われている。UNIX でも (文字フォントの微妙な違いを別にすれば) 同じ様に動作するはずである。Python の実行環境の構築法とサンプルプログラムの実行方法に関しては他書 (例えば参考文献 [7] や [8]) を参照されたい。ここでは Python の実行環境を持っており、Python のプログラミング経験のある読者を想定している。

1　ウィンドウの生成

1.1　サンプルプログラム

まず最初にウィンドウがどのように生成されるかを次のプログラムを例にして説明する。

```
from Tkinter import *

a=Tk()
a.title("alice")
Checkbutton(a,text='Alice',variable='v').pack()

b=Toplevel(a)
b.title("bob")
Checkbutton(b,text='Bob',variable='v').pack()

c=Tk()
c.title("carol")
Checkbutton(c,text='Carol',variable='v').pack()

mainloop()
```

図 1: プログラム `top1.py`

このプログラムは次の図に示す様に 3 つのウィンドウ a, b, c を生成し、各々のウィンドウでチェックボタンを表示している。

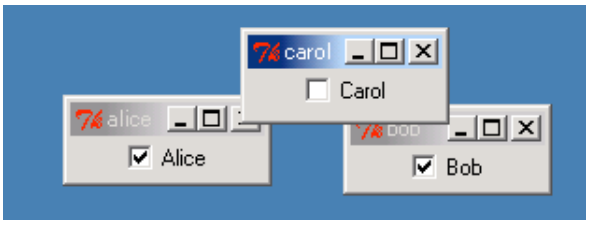

図 2: `Tk()` と `Toplevel()` によって生成されたウィンドウ

1.2　解説

ウィンドウのタイトルバーには各々 alice, bob, carol と表示され、またチェックボタンの説明ラベルには 各々 Alice, Bob, Carol と表示されている。これら 3 つのウィンドウはプログラムの中では各々変数 a, b, c で表されている。そこで以下の説明を容易にする為にこれらのウィンドウを a, b, c を呼ぶことにする。3 つのウィンドウは見かけは良く似てはいるが機能的には基本的な違いがある。その違いは次の様にすれば見つかるであろう。

1. チェックボタンをマウスでクリックしてみよう。すると a と b のチェックマークは連動するが、c のチェックマークは独立している。

2. タイトルバーの右端の ×印をマウスでクリックしてウィンドウを消去してみよう。すると次の事が分る。

   - a を消去すると b も消去される
   - b を消去しても a は消去されない
   - c は a, b と独立している

1.3 Toplevel

こうした違いはプログラムの中でのウィンドウの生成のされ方の違いから発生したのである。ウィンドウを生成したのは次の 3 つの文である。

```
a=Tk()
b=Toplevel(a)
c=Tk()
```

これら Tk や Toplevel は何れも Tkinter モジュールの中で定義されているクラス名である。Python ではクラス名を使ってそのクラスに属するオブジェクトを生成する。引数の与え方は関数とよく似ているがオブジェクトの生成子であるから関数と呼ばずにコンストラクタと呼ばれる。2 つのコンストラクタ Tk と Toplevel は見かけ上同じようなウィンドウを生成するのだが役割が異なる。Tk は独立したウィンドウを生成する。他方 Toplevel は親ウィンドウを指定し、親子関係にあるウィンドウの間で変数を共有する。親が死ねば子も死ぬ。子が死んでも親は死なない。

チェックボタンを題材にしたのは変数の共有関係を簡単に示す事ができるからである。Checkbutton の引数の中の variable='v' によって v と名付けられた変数が生成される。このプログラムでは 3 つのウィンドウで同じ名前が使用されている。そして Checkbutton の仕様では同一の変数によって指定されたチェック欄は連動する。従ってこのプログラムの実験から分る事は、 Tk によって生成されたウィンドウ相互の間で同じ名前の変数が使用されていても、それらは関係の無い変数として扱われると言う事である。

1.4 扱われる問題の範囲

この解説では Tkinter の Widget クラスの解説をできるだけ簡単な例題で説明する予定である。そうした例題ではたった 1 個のウィンドウだけが扱われている。Python は簡単な問題は簡単に処理できる様に設計されている。次節以降で採り上げる例題では Tk も Toplevel もプログラムの中に現れない。書かなくても暗黙の内にこっそり実行してくれるのである。従ってそのような簡単な例題だけを見ていると実際の応用で発生する複雑な事態、即ち多数のウィンドウの相互の関係が見えなくなる。これらの例題はどれも 1 個のウィンドウだけしか扱っていないのである。

ウィジェットを生成する場合には常にどのウィンドウに張り付けるかが問題になる。例題の

```
Checkbutton(a,text='Alice',variable='v').pack()
```

のように張り付けるウィンドウを明示的に指定すれば問題は無いが Python では張り付けるウィンドウを省略できる。即ち

```
Checkbutton(None,text='Alice',variable='v').pack()
```

あるいは同じ事だが

```
Checkbutton(text='Alice',variable='v').pack()
```

と書くことができる。この様に張り付けるウィンドウが指定されていない場合には省略時の親ウィンドウに張り付けられる。省略時の親ウィンドウは次の様にして決定される。

1. ウィジェットを生成する前に Tk が実行されると、それによって生成されたウィンドウが省略時の親ウィンドウになる。

2. Tk が実行される前にウィジェットが生成されると、こっそりと Tk が実行され、それによって生成されたウィンドウが省略時の親ウィンドウになる。

読者はこの事を例題の Checkbutton の第 1 引数を省略するなり None で置き換えるなりして確認することができる。コンストラクタ Toplevel の親ウィンドウの指定を省略した場合もここに述べた省略時の親ウィンドウが使用される。この事も同様に確認してみるがよい。

さて第 2 節から第 16 節においてウィジェットを生成する以下のコンストラクタを説明する。

```
Frame, Label, Message, Dialog, Button, Checkbutton, Radiobutton
Menu, Menubutton, Listbox, Entry, Text, Scale, Scrollbar, Image
```

この内最後の Image を除けば、残りは何れも共通の使い方をする。Checkbutton を例に説明すると

```
Checkbutton(w,text='Alice',variable='v').pack()
```

のように第 1 引数に親ウィジェットを書く。第 1 引数が省略されると None が指定されたと見なされ、結局は省略時の親ウィンドウを指定したと見なされる。残りの引数はオプションを与えている。オプションの名前と許される値はウィジェット全体に共通するものもあるし、ウィジェットに固有のものもある。またウィジェットによっては省略できないオプションもある。それらは個別に解説される。ここでの pack() は詰め込みの方法を指定している。pack() の使い方は文献 [10] に解説済みなので省略する。Checkbutton と pack を一括して実行しないで、

```
b=Checkbutton(w,text='Alice',variable='v')
b.pack()
```

のように分けて実行する事もできる。これはウィジェットを表す変数 b (ここでは b としているが、この名称は何でもよい) を後に使用する必要がある場合の使い方である。どのような場合にこのような事が発生するかは次節以降の例題を見て行けば分るであろう。

1.5 ウィジェットかそれともインスタンスか?

ところで Python における GUI プログラミングを解説すると同じ事を 2 つの面から見る事になる。1 つの側面は利用者が目で見る GUI の仕様であり、他方の側面はそれを実現するプログラムコードである。そして、どちらの側から見るかによって同じものを異なる言葉で語る事になる。GUI の面から見れば、例えば、「`b` はチェックボタンのウィジェットである」と言う。プログラムコードの面から見れば「`b` はチェックボタンのインスタンスである」と言う。厳密に言えば、「`b` は `Widget` クラスの派生クラスである `Checkbutton` クラスのインスタンス」とでも言うべきなのであろうが、そのような厳密さは煩わしいだけである。従ってこの解説ではオブジェクト指向プログラミング言語の難しい用語はできるだけ使わずに、「`b` はチェックボタンのウィジェットである」という言い方に統一する事にする。

2 Frame

2.1 Frame の機能

`Frame` はフレームを生成する。フレームは枠だけのウィジェットであり、複数のウィジェットを纏める役割を果たす。フレームは通常 `pack` メソッドと共に用いられる。(筆者は既に `Frame` の使い方に関する解説を文献 [10] で行っている。詳しくはこの文献を見よ。)

2.2 Frame のオプション

□ `background`
　値: 色名 またはヌル文字列
　意味: 背景色。但しヌル文字列 (長さがゼロの文字列) を指定すると透明となる。

□ `height`
　値: 非負整数 (省略時の値は 0)
　意味: フレームの高さ。単位はピクセル。

□ `width`
　値: 非負整数 (省略時の値は 0)
　意味: フレームの横幅。単位はピクセル。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

`bd bg borderwidth cursor highlightbackground highlightcolor highlightthickness relief takefocus`

2.3 Frame のメソッド

`Frame` に固有なメソッドは存在しない。

3 Label

3.1 Label の機能

Label はラベルを生成する。ラベルは単に文字列を表示するためのウィジェットである。ラベルが表示する文字列は 1 行だけで構成される、ごく簡単な文字列であると想定されている。以下にラベルを使用した簡単なプログラム例とその実行結果を挙げる。

```
from Tkinter import *
Label(None, text='Hello GUI World!').pack()
mainloop()
```

図 3: プログラム label1.py

図 4: Tk() と Toplevel() によって生成されたウィンドウ

3.2 Label のオプション

□ height
　値: 非負整数
　意味: ラベルの高さ。単位は文字数。

□ width
　値: 非負整数
　意味: ラベルの横幅。単位は文字数。

□ wraplength
　値: 非負整数
　意味: 文字列を折り畳む時の幅。単位はピクセル。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

anchor background bitmap bd bitmap borderwidth cursor
fg font foreground highlightbackground highlightcolor
highlightthickness image justify padx pady relief takefocus
text textvariable underline

4 Message

4.1 Message の機能

Message はメッセージウィジェットを生成する。メッセージウィジェットはメッセージを表示するためのウィジェットである。メッセージとラベルはよく似ているが、メッセージはラベルに比べて長い文字列で扱う事が想定されている。以下にメッセージを使用した簡単なプログラム例とその実行結果を挙げる。

```
from Tkinter import *
s='''This is a message window.
It is useful when you want to show meassage constructed with more than one
line of text in a window.
You cannot edit this window.'''
Message(None,text=s,font='times 14').pack()
mainloop()
```

図 5: プログラム message1.py

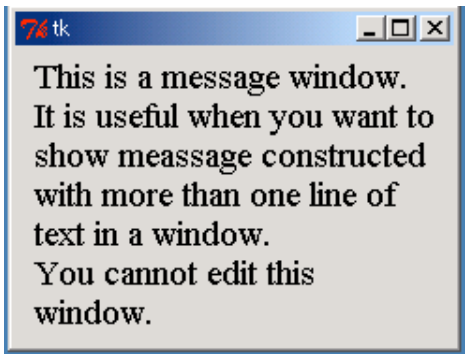

図 6: メッセージ

4.2 Message のオプション

□ aspect

値: 実数 (省略時の値は 150)

意味: 100*(メッセージの横幅)/(メッセージの高さ)。(この式での横幅と高さはの単位はピクセル)

□ width

値: 非負整数意味: メッセージウィンドウの横幅。単位は文字数。

□ justify

bf 値: 'left', 'right', 'center' または LEFT, RIGHT, CENTER (省略時の値は LEFT)

意味: テキストの行を左揃え、右揃え、あるいはセンター配置のいずれで行うかを指定する。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

anchor background bd bg borderwidth cursor font foreground highlightbackground highlightcolor highlightthickness padx pady relief takefocus text textvariable

5 Dialog

5.1 Dialog の機能

Dialog はダイアログを生成する。ダイアログは Tkinter モジュールのウィジェットではない。しかしながら GUI プログラミングにおいてよく使用されるので一緒に解説する。ダイアログウィジェットは警告を表示し、利用者に選択を迫る場合に使用される事が多い。多くの場合選択肢は “Yes”, “No”, “Cancel” である。以下にダイアログを使用した簡単なプログラム例とその実行結果を挙げる。

```
from Tkinter import *
from Dialog import Dialog
f=Frame()
f.pack()
Label(f,text='TEST of Diallog').pack()
a=Dialog(None,
    title='Sample of Dialog',
    text='Push a Button',
    bitmap='',
    font='Times',
    default=0,
    strings=('Yes','No','Cancel'))
# Then a.num is one of 0,1,2 coresponding to 'Yes','No','Cancel'
print a.num
mainloop()
```

図 7: プログラム dialog1.py

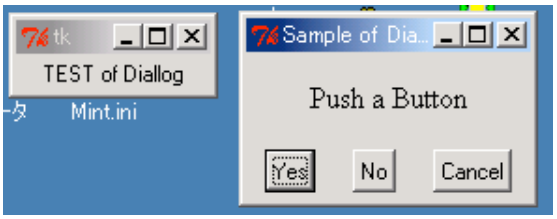

図 8: ダイアログ

このプログラムの 2 行目

```
from Dialog import Dialog
```

が存在する事によってダイアログが利用可能になる。

このプログラムはまず Label を実行し、その後に Dialog を実行している。同一のプログラムからこの 2 つを実行しているにも関わらず、ラベル (“TEST of Dialog”) を表示しているウィンドウと、ダイアログ (“Push a Button”) を表示しているウィンドウが分離している事に注目すべきである。

Dialog が実行されると利用者がダイアログによって提示された選択肢の 1 つを選択しない限りプログラムの実行は先に進まない。プログラミングにおいては、利用者が何を選択したかを知る必要がある。選択結果は Dialog のインスタンス変数 (ダイアログを表す変数で、今の場合は a) の中のメンバー変数 num で与えられる。この例ではこの値は 0 と 1 と 2 をとる。この数字は Dialog の呼び出しの際に指定された string オプションの値の配列要素の番号である。このプログラムでは Dialog の実行後に単に a.num の値を出力しているだけであるが、実際にはこの値に応じた処理を行う事になる。

実際のプログラムではダイアログは利用者が危険な選択を行った場合に利用者の真意を確認する為に使用されるであろう。その場合には Dialog は何かの関数の中から呼び出される事になるがメンバー変数 num の値を調べて実際の処理を決めていけばよい。

5.2 Dialog のオプション

□ title
　値: 文字列
　意味: ダイアログウィンドウのタスクバーに表示する標題。

□ text
　値: 文字列
　意味: ダイアログボックスに表示するメッセージ。

□ font
　値: フォント名
　意味: text で指定した文字列を表示する時のフォント名。

□ bitmap
　値: ビットマップ名 (ここに与える文字列については付録 A を見よ)
　意味: ダイアログボックスに表示するビットマップ。このビットマップはダイアログボックスの左に表示される。

□ default
　値: 非負整数
　意味: マウスクリックの代わりにリターンキーを打つ場合に選択される選択肢。配列要素の番号で指定する。

6 Button

6.1 Button の機能

Button はボタンウィジェットを生成する。ボタンはマウスの左ボタンでクリックする事に

よって、プログラムの中で指定された関数を実行するウィジェットである。典型的なボタンのデザインを図 9 に示す。この図に示したボタンは例えば button1.py によって生成される。

図 9: ボタン

```
from Tkinter import *
def cmd():
    print "Pushed"
b = Button(None,text="Push",command=cmd)
b.pack()
mainloop()
```

図 10: プログラム button1.py

ボタンの上の文字列の表示は text オプションによって指定される。command オプションがボタンが押された時に実行される関数を指示する。このプログラムではボタンが押される事によって単に “Pushed” が表示される。実際の応用ではこの部分は問題毎に異なるであろう。

6.2 Button のオプション

☐ activebackground
　値: 色名
　意味: ボタンが押された時の背景色。

☐ activeforeground
　値: 色名
　意味: ボタンが押された時の前景色 (文字の色)。

☐ command
　値: 関数名
　意味: ボタンが押された時に実行される関数の名称。

☐ default
　値: 'normal', 'active', 'disabled'
　意味: 強調枠の見え方を指定する。この指定は hightlightthickness で 0 でない値が指定された場合にのみ有効である。'normal' 及び 'disabled' では強調枠は見えない。'active' では強調枠が見え、その時の強調枠の色は highlightcolor で指定された色である。

☐ height
　値: 非負整数
　意味: ボタンの高さ。単位は文字数。

□ state

値: 'normal', 'active', 'disabled' (省略時の値は 'normal')

意味: ボタンが生成された時の状態を指定する。'normal' は普通の状態である。'disabled' が指定された時にはボタンは反応しない。'active' が指定された時には activebackground で指定された色となる。

□ width

値: 非負整数

意味: ボタンの横幅。単位は文字数。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

anchor background db bg bitmap borderwidth cursor
disableforeground fg font foreground highlightbackground highlightcolor
highlightthickness image justify padx pady relief takefocus text
textvariable underline wraplength

図 11 に default オプションの 3 つの指定に伴うボタンの見え方の違いを示す。文献 [12] によると 'normal' と 'disabled' は異なっていてもよいと書いてあるが、実際には違いは見つからない。ボタンでは入力フォーカスがあっても強調枠の色は変化せず、その代わり入力フォーカスの存在が破線で示される。

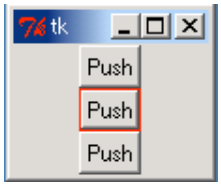

図 11: ボタンの強調枠

これを生成したプログラムコードを次に示す。

```
from Tkinter import *
def cmd():
print "Pushed"
Button(None,text="Push",command=cmd,default='normal').pack()
Button(None,text="Push",command=cmd,default='active',highlightcolor='red').pack()
Button(None,text="Push",command=cmd,default='disabled').pack()
mainloop()
```

6.3 Button のメソッド

□ flash()

機能: ボタンを点滅させる。点滅させるボタンは activebackground で点滅色を指定しておく事。

□ invoke()

機能: ボタンを押した時の動作をさせる。

例

```
from Tkinter import *
def cmd1():
    print "Left Button is Pushed"
def cmd2():
    print "Right Button is Pushed"
    b.flash()
    b.invoke()
f=Frame()
f.pack()
b=Button(f,text="Left",command=cmd1,activebackground='red')
b.pack(side=LEFT)
Button(f,text="Right",command=cmd2).pack(side=RIGHT)
mainloop()
```

7 Checkbutton

7.1 Checkbutton の機能

Checkbutton はチェックボタンを生成する。チェックボタンは ON、OFF が切り替わるボタンである。ON 状態ではチェックマークがチェック欄に表示される。OFF 状態ではチェック欄は空白である。チェック欄は本質的にボタンと同じ役割を果たしているのでチェックボタンと呼ばれる。

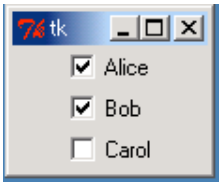

図 12: チェックボタン

この図に示した 3 つのチェックボタンは例えば図 13 のプログラム checkbutton1.py によって生成される。

このプログラムではボタンを揃えるために pack メソッドで anchor='w' を指定している。ボタンが押された時に実行される関数は Checkbutton の command オプションで指定する。ボタンの状態は getvar を使って知る事ができる。getvar は ON 状態の時には 1、OFF 状態の時には 0 を返す。getvar の引数は Checkbutton の variable オプションで与えた文字列である。この文字列はボタンのグループ化のために使用される。即ち幾つかのチェック欄が関連しており、同じ値をとる必要がある場合には同じ文字列を与える。

このプログラムでは r1, r2, r3 に getvar を作用させたが必ずしもその必要はない。フレーム f に作用させても同じ結果が得られるし、その方がエレガントである。また、このプログラムではチェックボタン毎に異なる関数を command オプションで指定したが必ずしもその必要はない。またそもそも Checkbutton に command オプションが必要だと言う訳でもない。

多数のチェックボタンがある場合には、むしろ図 14 のプログラム checkbutton2.py の様に、アクションを定める関数は 1 個で済ませ、“OK” ボタンを別に追加する方が実際的であろう。

```
from Tkinter import *
def cmd1():
    print "alice"
    print r1.getvar('v1')
def cmd2():
    print "bob"
    print r2.getvar('v2')
def cmd3():
    print "carol"
    print r3.getvar('v3')
f=Frame()
f.pack()
r1 = Checkbutton(f,text="Alice",variable='v1',command=cmd1)
r1.pack(anchor="w")
r2 = Checkbutton(f,text="Bob",variable='v2',command=cmd2)
r2.pack(anchor="w")
r3 = Checkbutton(f,text="Carol",variable='v3',command=cmd3)
r3.pack(anchor="w")
mainloop()
```

図 13: プログラム checkbutton1.py

```
from Tkinter import *
def cmd():
    print f.getvar('v1')
    print f.getvar('v2')
    print f.getvar('v3')
f=Frame()
f.pack()
Checkbutton(f,text="Alice",variable='v1').pack(anchor="w")
Checkbutton(f,text="Bob",variable='v2').pack(anchor="w")
Checkbutton(f,text="Carol",variable='v3').pack(anchor="w")
Button(None,text="OK",command=cmd).pack()
mainloop()
```

図 14: プログラム checkbutton2.py

getvar メソッドは個々のボタンではなく、それらの親ウィジェットであるフレームに作用させている事に注意しよう。variable オプションで指定された変数は全てのウィジェットで共有される。それ故、多数のウィジェットを扱うプログラムでは名前の衝突に対する注意が必要である。

7.2 Checkbutton のオプション

□ command

値: 関数名

意味: チェックボタンが押された時に実行される関数の名称。

□ height

値: 非負整数

意味: チェックボタンの高さ。単位は文字数。

□ indicatoron または indicator

値: YES または NO (省略時の値は YES)

意味: チェックボタンのデザインとしてインジケータを使用するか、それとも、通常のボタンを使用するかを指定する。

□ onvalue

値: 任意

意味: ボタンの ON 状態を 1 でない値に設定したい時にこのオプションを設定する。

□ offvalue

値: 任意

意味: ボタンの OFF 状態を 0 でない値に設定したい時にこのオプションを設定する。

□ state

値: 'normal', 'active', 'disabled' (省略時の値は 'normal')

意味: 'disabled' が指定された時にはチェックボタンは反応しない。'active' が指定された時にはチェックボタンの初期状態の色は activebackground で指定された色となる。

□ selectcolor

値: 色名

意味: ボタンが ON 状態の時のボタンの色を指定する。

□ selectimage

値: イメージ

意味: selectimage で指定されたイメージと image で指定されたイメージが対になってチェックボタンの ON/OFF を表示する。

□ variable

値: 文字列

意味: チェックボタンを区別する為に使用される。異なるチェックボタンに同じ文字列を与えた場合には ON/OFF の状態が連動する。

□ width

値: 非負整数

意味: チェックボタンの横幅。単位は文字数。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

activebackground activeforeground anchor background bd bg bitmap
borderwidth cursor disableforeground fg font foreground highlightbackground
highlightcolor highlightthickness image justify
padx pady relief takefocus text textvariable underline wraplength

7.3 Checkbutton のメソッド

□ deselect()
機能: チェックボタンを OFF 状態にする。

□ flash()
機能: チェックボタンを点滅させる。点滅させるボタンは activebackground で点滅色を指定しておく事。

□ getvar(*s*)
引数 *s*: variable オプションで指定した文字列
機能: チェックボタンの状態 (0 または 1) を返す。

□ invoke()
機能: チェックボタンを押した時の動作をさせる。

□ select()
機能: チェックボタンを ON 状態にする。

□ setvar(*s*,*v*)
引数 *s*: variable オプションで指定した文字列
引数 *v*: 0 または 1
機能: チェックボタンの状態 (0 または 1) を定める。

□ toggle()
機能: チェックボタンの ON/ OFF を反転する。

8 Radiobutton

8.1 Radiobutton の機能

Radiobutton はラジオボタンを生成する。普通にラジオボタンと言われるものは、複数のボタンが組になっていて、その中の 1 つだけを ON 状態に設定できる 1 組のボタンを言う。ボタンの一つを ON にすると、それまでに ON 状態であったボタンは自動的に OFF 状態となる。相互に排他的に作用するのである。

ラジオボタンの通常の応用ではあまり発生しない事だが、Python では 1 組のラジオボタンの中で連動するボタンのグループも定義できる。

相互に干渉しない複数のラジオボタンの組が必要になる場合もある。例えば答を 1 つだけ選ぶ試験問題の場合には独立した幾つかの問題が与えられているのが普通であり、このような試験をコンピュータを利用して行う場合である。

次に Radiobutton を使用したサンプルプログラム (図 15) とその実行結果 (図 16) を載せる。

このサンプルは 2 組のラジオボタンを定義している。左に表示されているボタンは

```
r1=Radiobutton(f,text="Alice",variable='a',value='alice')
r1.pack(anchor='w')
```

```
from Tkinter import *
def cmd():
print [f.getvar('a'),g.getvar('b')]

f=Frame(relief='ridge',borderwidth=3)
f.pack(side=LEFT)
Label(f,text='person').pack()
r1 = Radiobutton(f,text="Alice",variable='a',value='alice')
r1.pack(anchor='w')
r2 = Radiobutton(f,text="Bob",variable='a',value='bob')
r2.pack(anchor='w')
r3 = Radiobutton(f,text="Carol",variable='a',value='carol')
r3.pack(anchor='w')

g = Frame(relief='ridge',borderwidth=3)
g.pack(side=LEFT)
Label(g,text='pet').pack()
s1 = Radiobutton(g,text="Dog",variable='b',value='dog')
s1.select()
s1.pack(anchor='w')
s2 = Radiobutton(g,text="Cat",variable='b',value='cat')
s2.pack(anchor='w')
s3 = Radiobutton(g,text="Python",variable='b',value='python',
state='disabled')
s3.pack(anchor='w')

Button(None,text='OK',command=cmd).pack(side=BOTTOM)
mainloop()
```

図 15: プログラム radiobutton1.py

```
r2=Radiobutton(f,text="Bob",variable='a',value='bob')
r2.pack(anchor='w')
r3=Radiobutton(f,text="Carol",variable='a',value='carol')
r3.pack(anchor='w')
```

によって生成されている。これらの 3 つが 1 つの組を形成する事は variable の値が同一である事によって表現されている。そして value の値が異なる事によってラジオボタンが相互に排他的に働く。もしも "Alice" と "Bob" の value の値が同じであれば、この 2 つは同一の ON/OFF 状態となる。anchor で 'w' が指定されているのはボタンの位置を揃える為である。1 組のラジオボタンの状態は variable で指定された文字列を使用して知る事ができる。そのためには以下の様に getvar メソッドをラジオボタンのインスタンス変数 (ラジオボタンを表す変数) に作用させる:

```
r1.getvar("a")
r2.getvar("a")
r3.getvar("a")
```

この 3 つは同一の値を返す。即ち、現在選択されている "a" の値 (value)、例えば Bob が ON であれば "bob" を返す。従って 3 つの内の 1 つだけを調べれるだけでよいが、ボタンの親であるフレームのインスタンス変数を使っても値を知る事ができる:

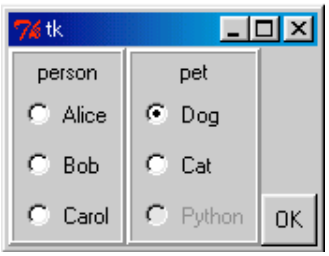

図 16: 2 組のラジオボタン

```
f.getvar("a")
```

そしてこの方がエレガントである。variable オプションで指定された変数は全てのウィジェットで共有される。それ故、多数のウィジェットを扱うプログラムでは名前の衝突に対する注意が必要である。実行結果の図で “Dog”, “Cat”, “Python” と表示されている一組のラジオボタンは

```
s1 = Radiobutton(g,text="Dog",variable='b',value='dog')
s1.select()
s1.pack(anchor='w')
s2 = Radiobutton(g,text="Cat",variable='b',value='cat')
s2.pack(anchor='w')
s3 = Radiobutton(g,text="Python",variable='b',value='python',
state='disabled')
s3.pack(anchor='w')
```

によって生成される。プログラムが起動された段階で “Dog” に選択マークが付いているのは

```
s1.select()
```

による。また “Python” が選択不可能になっているのは

```
state='disabled'
```

の指定による。このサンプルプログラムでは OK ボタンが押されたタイミングでボタンの状態を調べている。もしもラジオボタンが選択された時に直ちに何かの関数を実行する必要があれば Radiobutton に command オプションを設定すればよい。variable オプションは省略する事もできる。その場合には "selectButton" が variable の値となる。

8.2 Radiobutton のオプション

□ command

値: 関数名

意味: ラジオボタンが押された時に実行される関数の名称。

□ height
値: 非負整数
意味: ラジオボタンの高さ。単位は文字数。

□ indicatoron または indicator
値: YES または NO (省略時の値は YES)
意味: ラジオボタンのデザインとしてインジケータを使用するか、それとも、通常のボタンを使用するかを指定する。

□ selectimage
値: イメージ
意味: selectimage で指定されたイメージと image で指定されたイメージが対になってラジオボタンの ON/OFF を表示する。

□ state
値: 'normal', 'active', 'disabled' (省略時の値は 'normal')
意味: 'disabled' が指定された時にはラジオボタンは反応しない。'active' が指定された時にはラジオボタンの初期状態の色は activebackground で指定された色となる。

□ selectcolor
値: 色名
意味: ボタンが ON 状態の時のボタンの色を指定する。

□ value
値: 任意データ
意味: 連動するラジオボタンのグループを定義するのに使用される。

□ variable
値: 文字列 (省略時の値は "selectButton")
意味: 独立した一組のラジオボタンを定義するのに使用される。

□ width
値: 非負整数
意味: ラジオボタンの横幅。単位は文字数。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

activebackground activeforeground anchor background bd bg bitmap
borderwidth cursor disableforeground fg font foreground
hightlightbackground highlightcolor highlightthickness
image justify padx pady relief takefocus text textvariable
underline wraplength

8.3 Radiobutton のメソッド

□ deselect()
機能: ラジオボタンを OFF 状態にする。

□ `flash()`
機能: ラジオボタンを点滅させる。点滅させるボタンは `activebackground` で点滅色を指定しておく事。

□ `getvar(s)`
引数 *s*: `variable` オプションで指定した文字列
機能: ラジオボタンの状態を `value` オプションで指定された値によって返す。

□ `invoke()`
機能: ラジオボタンを押した時の動作をさせる。

□ `select()`
機能: ラジオボタンを ON 状態にする。

□ `setvar(s,v)`
引数 *s*: `variable` オプションで指定した文字列
引数 *v*: `value` オプションで指定した値
機能: ラジオボタンの状態を定める。

9 Menu

9.1 Menu の機能

`Menu` はメニューを生成する。メニューとは項目が与えられ、それを選択することによって何かが実行される仕組みであり、通常は選択をマウス操作によって行う。`Menu` が生成するメニューの視覚的なデザインは OS ごとに異なる。例えば Windows の `Menu` は Windows の標準的なメニューのデザインに従って項目を横に並べて配置する (図 17)。そして Windows の場合にはメニューはウィンドウの一番上側に配置される。この図では 3 つの選択項目 “Alice”,

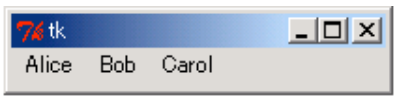

図 17: Windows のメニュー

“Bob”, “Carol” が表示されている。これらの項目をマウスでクリックすると何かが実行される様になっているのである。項目が選択された場合にはサブメニューを表示させることも可能である。その場合には選択された項目の下側にサブメニューが表示される。`Menu` の仕様はかなり複雑である。配布パッケージの中の次の 2 つのプログラムが参考となろう。

```
Demo/tkinter/guido/AttrDialog.py
Demo/tkinter/matt/menu-all-types-of-entries.py
```

さて、ここに挙げたメニューは次のプログラムによって作成されている。プログラムを実行して項目 Alice をマウスでクリックしてみよう。するとコンソールに `alice` と表示されるであろう。プログラム中の

```
from Tkinter import *
def cmd1():
    print "alice"
def cmd2():
    print "bob"
def cmd3():
    print "carol"
b=Tk()
Frame(b,width=200).pack()
m=Menu(b)
m.add_command(label="Alice",command=cmd1)
m.add_command(label="Bob",command=cmd2)
m.add_command(label="Carol",command=cmd3)
b.configure(menu=m) # same as: b["menu"]=m
mainloop()
```

図 18: プログラム menu1.py

```
m.add_command(label="Alice",command=cmd1)
```

によってメニューの中に項目 Alice が追加され、Alice と実行される関数 cmd1 が結びつけられる。そして cmd1 の中の文

```
print "alice"
```

によってコンソールに alice が表示されるのである。Bob や Carol についても同様である。今の場合には単に文字列 "alice" や "bob" や "carol" をコンソールに表示しているだけであるが実際の応用では問題に即した処理に置き換えられるべきであろう。

このプログラムには Tk が現れている。Tk は第 1 節に述べられている様に明示的にウィンドウを生成する。生成されたウィンドウは

```
b=Tk()
```

によって変数 b で参照できる様になっている。また

```
Frame(b,width=200).pack()
```

はウィンドウの幅を確保する為に挿入されている。もしもこれがないとメニューは折り返されて表示されるであろう。実際の応用でウィンドウの横幅はアプリケーションで使用する他のウィジェット、例えば Canvas ウィジェットや Text ウィジェットなどで定まるので、このように Frame でわざわざ横幅を確保する必要はないはずである。

メニューはウィンドウ b の中に生成される必要がある。従って Menu の親ウィジェットには次の様にウィンドウ b を指定する。

```
m=Menu(b)
```

メニューは pack できない。この点でメニューは他のウィジェットと異なっている。これはメニューの配置位置が定められている事と関係している。メニューの場合には pack の代わりにウィンドウを指定して

```
b.configure(menu=m)
```

を実行する。(あるいはプログラム中のコメントにあるように b["menu"]=m を実行してもよい。) このプログラムでどうしても Tk を使用しなければならない理由はウィンドウに対して configure を実行する必要があるからである。

さて、このプログラム例では最も単純なメニューを扱った。即ち、このメニューに現われる項目はどれも単純なボタンと同じ様に振る舞う。即ち、外観こそボタンの形をしていないが、項目を選択した時に何か或る関数を実行するのみである。しかしながらメニューに対する応用上の要求は多くの場合サブメニューを表示してくれるメニューにある。即ちメニューのある項目を選択したら再びメニューが現れるようなメニューである。そのようなメニューをカスケードメニューと言う。次の図にカスケードメニューの例を載せる。

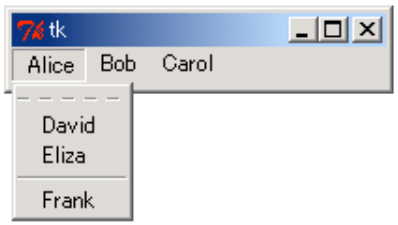

図 19: カスケードメニュー

カスケードメニューを生成するには add_command ではなく add_cascade を使用する。図 19 は次のプログラムによってを生成される。

```
from Tkinter import *
def cmd2():
    print "bob"
def cmd3():
    print "carol"
def cmd4():
    print "david"
def cmd5():
    print "eliza"
def cmd6():
    print "frank"

b=Tk()
Frame(b,width=200).pack()
n=Menu(b)
n.add_command(label="David",command=cmd4)
n.add_command(label="Eliza",command=cmd5)
n.add_separator()
n.add_command(label="Frank",command=cmd6)
m=Menu(b,type="tearoff")
m.add_cascade(label="Alice",menu=n)
m.add_command(label="Bob",command=cmd2)
m.add_command(label="Carol",command=cmd3)
b.configure(menu=m) # same as: b["menu"]=m

mainloop()
```

図 20: プログラム menu2.py

このプログラムの

```
m.add_cascade(label="Alice",menu=n)
```

に注目しよう。これは Alice の項目がカスケードメニューであることを指定している。そしてこの項目が選択された時に表示されるサブメニューが n で指定されている。変数 n には前もってメニューが与えられる必要がある。それはプログラム中の以下の文によって行われている。

```
n=Menu(b)
n.add_command(label="David",command=cmd4)
n.add_command(label="Eliza",command=cmd5)
n.add_separator()
n.add_command(label="Frank",command=cmd6)
```

図 19 には Eliza と Frank の間に横線が引かれているが、この横線は add_separator によって生成している。さらにこの図には David の上に破線が引かれている。この破線をマウスでクリックするとサブメニューが切り離される (図 21)。これは

```
m=Menu(b,type="tearoff")
```

の中の type="tearoff" の指定による。

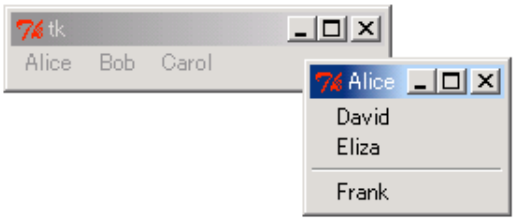

図 21: 切り離されたサブメニュー

Menu のサブメニューではこの他にラジオボタンやをチェックボタン扱う事が可能である。次に示すのはサブメニューでラジオボタンとチェックボタンを扱うプログラム例 (図 22) とその実行例 (図 23) でである。

図 23 を見て分る様にメニューの中のボタンのデザインは極めてシンプルである。単にチェックマークが付くに過ぎない。残念ながら Python の現在の版にはこのチェックマークの表示にバグがあるらしく、マウスカーソルをラジオボタンやチェックボタンの項目に持って行くまでは、このマークが見えない[2]。

プログラムを実行し Alice のサブメニューを表示すると今度は David と Eliza がラジオボタン、Frank がチェックボタンになっている。David をクリックするとコンソールに "radio: D" のメッセージが表示されるであろう。これはプログラムの中の次の文の作用による。

```
n.add_radiobutton(label="David",command=cmd4,variable='v',value='D')
```

[2] このバグは Python 2.3 では取れている

```
from Tkinter import *
def cmd2():
    print "bob"
def cmd3():
    print "carol"
def cmd4():
    print "radio:", n.getvar('v')
def cmd5():
    print "check:", n.getvar('u')
b=Tk()
Frame(b,width=200).pack()

n=Menu(b)
n.add_radiobutton(label="David",command=cmd4,variable='v',value='D')
n.add_radiobutton(label="Eliza",command=cmd4,variable='v',value='E')
n.add_separator()
n.add_checkbutton(label="Frank",command=cmd5,variable='u')
m=Menu(b,type="tearoff")
m.add_cascade(label="Alice",menu=n)
m.add_command(label="Bob",command=cmd2)
m.add_command(label="Carol",command=cmd3)
b.configure(menu=m) # same as: b["menu"]=m

mainloop()
```

図 22: プログラム menu3.py

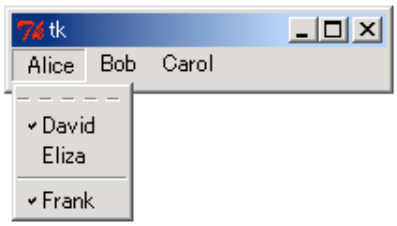

図 23: メニューの中のラジオボタンとチェックボタン

David をクリックすると関数 cmd4 が実行され、その関数の中で

```
print "radio:", n.getvar('v')
```

が実行される。ここに現れる n.getvar('v') が add_radiobutton の variable オプションでで与えた変数 v の値 (value オプションで与えられている) を返し、それがコンソールに表示されているのである。

項目 Frank をクリックするとコンソールに "check: 1" あるいは "check: 0" が表示されるであろう。この表示はトグルになっている。これはチェックボタンの性質である。プログラム中の

```
n.add_checkbutton(label="Frank",command=cmd5,variable='u')
```

が Frank がチェックボタンである事を定めている。これをクリックすると cmd5 が実行され、関数 cmd5 の中で

```
print "check:", n.getvar('u')
```

が実行される。ここに現れる n.getvar('u') が add_checkbutton の variable オプションでで与えた変数 u の値 を返す。チェックボタンの場合にはこの値はボタンの ON/OFF を表し、1 または 0 である。

プログラム中の add_radiobutton と add_checkbutton の引数に現れている variable オプションと value オプションの与え方と意味は第 7 節と第 8 節で扱った Checkbutton や Radiobutton と同じなのでこれ以上の解説は省略する。

9.2 Menu のオプション

□ height
　値: 非負整数
　意味: メニューの高さ。単位は文字数。

□ width
　値: 非負整数
　意味: メニューの横幅。単位は文字数。

□ postcommand
　値: 関数名
　意味: メニューが表示されるたびに実行される関数を指定する。

□ selectcolor
　値: 色名
　意味: このオプションはメニューの中にチェックボタンまたはラジオボタンが使用された場合にのみ意味を持っている。インジケータの色を指定する。

□ tearoff
　値: YES, NO (省略時の値は YES)
　意味: メニューの中に、メニューをメニューボタンから分離する選択肢である破線 "-----" を含めるか否かを指定する。

□ tearoffcommand
　値: 関数
　意味: ティアオフエントリを選択した時に実行される関数を指定する。
　注意: この関数は 2 つの引数を持つ様に定義しなくてはならない。(しかし引数から得られる情報は役にはたたないであろう。)

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

activebackground activeborderwidth activeforeground
background bd bg borderwidth cursor disableforeground fg font
foreground relief

9.3 Menu の add_separator メソッド

用法: add_separartor()

戻り値: None

機能: メニューの項目 (メニューエントリー) を区切るための線を挿入する。オプションは存在しない。

9.4 Menu の add_command メソッド

用法: add_command()

戻り値: None

機能: メニューに項目 (メニューエントリー) を追加する。

add_command は以下のオプションを持つ

- □ accelarator]
  値: 文字列
  意味: メニュー項目の右側に表示する文字列を指定する。

- □ command
  値: 関数名
  意味: メニュー項目が選択された時に実行される関数を指定する。

- □ label
  値: 文字列
  意味: メニュー項目として表示する文字列。

- □ selectimage
  値: ビットマップ
  意味: 選択状態になった時に表示するビットマップ。

- □ state
  値: "normal", "active", "disabled" (省略時の値は "normal")
  意味: メニューの表示色を指定する。"normal" では文字を foreground で指定された色で、背景を background で指定された色で表示する。"active" では文字を activeforeground で指定された色で、背景を activebackground で指定された色で表示する。"disabled" を指定するとこの項目は反応しない。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

activebackground activeforeground background bitmap font foreground image underline

9.5 Menu の add_cascade メソッド

用法: add_cascade()

戻り値: None

機能: メニューの項目としてカスケードメニューを追加する。カスケードメニューとは項目を選択した時にサブメニューが表示される階層的な構造を持つメニューである。

add_cascade は以下のオプションを持つ

□ command
値: 関数名
意味: サブメニューが表示される前に実行される関数を指定する。

□ label
値: 文字列
意味: メニュー項目として表示する文字列。

□ menu
値: サブメニュー
意味: サブメニューとして結合するメニュー。

以上の他に add_command と共通の以下のオプションが存在する。詳細は add_command を参照せよ。

accelerator selectimage state

また以下のオプションも存在する。これらについては付録 A を参照せよ。

activebackground activeforeground background bitmap font foreground image underline

9.6 Menu の add_checkbutton メソッド

用法: add_checkbutton()

戻り値: None

機能: メニューの項目としてチェックボタンを追加する。

add_checkbutton は以下のオプションを持つ

□ command
値: 関数名
意味: サブメニューが表示される前に実行される関数を指定する。

□ indicatoron または indicator
値: YES , NO (省略時の値は YES)
意味: インジケータ (チェックマーク) を表示するか否かを指定する。

□ label
　値: 文字列
　意味: メニュー項目として表示する文字列。

□ offvalue
　値: 任意の値
　意味: エントリが選択されなかった場合にエントリ変数に代入する値を指定する。

□ onvalue
　値: 任意の値
　意味: エントリが選択された場合にエントリ変数に代入する値を指定する。

□ selectcolor
　値: 色名
　意味: インジケータ (チェックマーク) の色を指定する。

□ variable
　値: 文字列
　意味: チェックボタンの状態を表す変数名を指定する。この変数は広域変数である。

以上の他に add_command と共通の以下のオプションが存在する。詳細は add_command を参照せよ。

accelerator selectimage state

また以下のオプションも存在する。これらについては付録 A を参照せよ。

activebackground activeforeground background bitmap font foreground image underline

9.7 Menu の add_radiobutton メソッド

用法: add_radiobutton()

戻り値: None

機能: メニューの項目としてラジオボタンを追加する。

add_radiobutton は以下のオプションを持つ。

□ command
　値: 関数名
　意味: サブメニューが表示される前に実行される関数を指定する。

□ indicatoron または indicator
　値: YES , NO (省略時の値は YES)
　意味: インジケータ (チェックマーク) を表示するか否かを指定する。

☐ `label`
　値: 文字列
　意味: メニュー項目として表示する文字列。

☐ `selectcolor`
　値: 色名
　意味: インジケータ (チェックマーク) の色を指定する。

☐ `value`
　値: 任意の値
　意味: エントリが選択された時に `variable` オプションで指定された変数に代入する値を指定する。

☐ `variable`
　値: 文字列
　意味: チェックボタンの状態を表す変数名を指定する。この変数は広域変数である。

以上の他に `add_command` と共通の以下のオプションが存在する。詳細は `add_command` を参照せよ。

`accelerator selectimage state`

また以下のオプションも存在する。これらについては付録 A を参照せよ。

`activebackground activeforeground background bitmap font foreground image underline`

10 Menubutton

10.1 Menubutton の機能

`Menubutton` はメニューを表示するためのボタン (メニューボタン) を生成する。`Menubutton` は `Menu` と共に用いられ、`Menu` がメニューを生成する。メニューの中にはメニューの項目 (メニューエントリ) が表示される。以下に `Menubutton` と `Menu` が生成する典型的なメニューの外観を示す。

図 24: メニューボタンとメニュー

この図では “Person” と表示されているのが `Menubutton` が生成したボタンであり、これをマウスで選択するとその下に `Menu` によって生成されたメニューが表示されるのである。マウ

スカーソルがメニューの項目の上に来ると、その項目は色が変化する。通常は背景が青色に変化するがこの色は Menu の activebackground オプションによって変更可能である。マウスカーソルが指している項目をアクティブであると言う。この図では項目 “Bob” がアクティブになっている。メニュー項目の先頭の破線 “-----” を選択すると次の図 25 の様にメニューが切り離される。

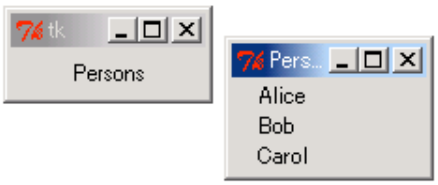

図 25: メニューボタンから切り離されたメニュー

切り離されたメニューは 1 つのウィンドウの様に振る舞い、スクリーンの任意の位置に移動する事ができる。Menu と Menubutton の仕様はかなり複雑である。配布パッケージの中の次の 2 つのプログラムが参考となろう。

Demo/tkinter/guido/AttrDialog.py
Demo/tkinter/matt/menu-all-types-of-entries.py

さて、ここに挙げたメニューは次のプログラム menubutton1.py によって作成されている。

```
from Tkinter import *
def cmd1():
    print "alice"
def cmd2():
    print "bob"
def cmd3():
    print "carol"

b = Menubutton(None,text="Persons")
b.pack()
m=Menu(b)
m.add_command(label="Alice",command=cmd1)
m.add_command(label="Bob",command=cmd2)
m.add_command(label="Carol",command=cmd3)
b.configure(menu=m)  # same as: b["menu"]=m
mainloop()
```

図 26: プログラム menubutton1.py

プログラムの中でのメニューボタンは Menubutton で記述される。即ち

```
b = Menubutton(None,text="Persons")
```

は、このメニューボタンの親ウィジェットが None で、このボタンには “Person” のラベルが張られる事を指示している。このメニューボタンを選択した時に現われるメニューは

```
m=Menu(b)
```

によって指定される。メニューに示される項目は add_command メソッドによって構成される。このプログラムに現われる 3 つの文

```
m.add_command(label="Alice",command=cmd1)
m.add_command(label="Bob",command=cmd2)
m.add_command(label="Carol",command=cmd3)
```

が 3 つのメニュー項目 “Alice”, “Bob”, “Carol” をメニューに追加しているのである。これらの中の command オプションはメニュー項目が選択された時に実行される関数を指定する。このプログラムではこれらの関数は単に文字列 "alice" や "bob" や "carol" をコンソールに表示しているだけであるが実際の応用では問題に即した処理に置き換えられるべきであろう。

さて、このプログラム例では最も単純なメニューを扱った。即ち、このメニューに現われる項目はどれも単純なボタンと同じ様に振る舞う。即ち、外観こそボタンの形をしていないが、項目を選択した時に何か或る関数を実行するのみである。Python の Tkinter モジュールではこうした単純な動作を行う項目の他に以下の 5 つの種類の項目 (エントリ) を扱う事が可能である。

1. セパレータ (separator) エントリ
2. カスケード (cascade) エントリ
3. チェックボタン (check button) エントリ
4. ラジオボタン (radio button) エントリ
5. ティアオフ (tear off) エントリ

これらの内、ティアオフエントリは破線 “-----” の表示で示されるエントリであり、プログラム例で暗黙の内に設定されている。次に示すのはこの 5 つを全て扱ったプログラム例とその実行例でである。

この例ではメニューボタンを選択した時に表示されるメニュー項目 “male” と “female” はカスケードエントリである。即ち、これらの項目はサブメニューの入口なのである。Python のカスケードメニューはこの上にマウスカーソルが来ると自動的にサブメニューが表示される。項目 “male” と “female” の間にはセパレータが使用されている。セパレータは単に視覚的なデザインの為に使用されるのに過ぎない。項目 “male” と “female” と セパレータがこの順に配置されている事はプログラムの中の

```
menu=Menu(mbutton)
menu.add_cascade(label="male",menu=male_menu)
menu.add_separator()
menu.add_cascade(label="female",menu=female_menu)
mbutton["menu"]=menu
```

によって判断される。項目 “male” のサブメニューの中には 2 つの項目 “Bob” と “David” が存在する。これらの項目が “male” のサブメニューであることはプログラムの中の

```
male_menu=Menu(mbutton)
male_menu.add_checkbutton(label="Bob",command=cmd1,variable='B')
male_menu.add_checkbutton(label="David",command=cmd1,variable='D')
```

```
from Tkinter import *
def cmd1():
    print male_menu.getvar('B')
    print male_menu.getvar('D')
def cmd2():
    print female_menu.getvar('V')
mbutton = Menubutton(None,text="persons")
mbutton.pack()
male_menu=Menu(mbutton)
male_menu.add_checkbutton(label="Bob",command=cmd1,variable='B')
male_menu.add_checkbutton(label="David",command=cmd1,variable='D')
female_menu=Menu(mbutton)
female_menu.add_radiobutton(label="Alice",command=cmd2,variable='V',value='A')
female_menu.add_radiobutton(label="Carol",command=cmd2,variable='V',value='C')
menu=Menu(mbutton)
menu.add_cascade(label="male",menu=male_menu)
menu.add_separator()
menu.add_cascade(label="female",menu=female_menu)
mbutton["menu"]=menu   # same as: mbutton.configure(menu=menu)
mainloop()
```

図 27: プログラム menubutton2.py

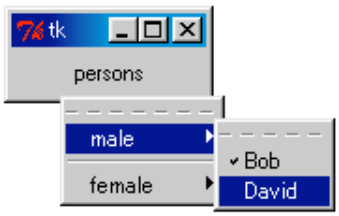

図 28: プログラムの実行例

及び、 male_menu をカスケードメニューに結びつける記述

```
menu.add_cascade(label="male",menu=male_menu)
```

によって実現されている。項目 “Bob” と “David” は一見すると普通の項目の様に見えるが、実はチェックボタンである。実際、マウスでこれらの項目を選択するとチェックマークが付いたり消えたりするのが分る。項目 “female” のサブメニューの中にも 2 つの項目 “Alice” と “Carol” を見つける事ができる。これらも一見すると普通の項目の様に見えるが、実はラジオボタンである。実際、マウスでこれらの項目を選択すると “Alice” と “Carol” のうちのいずれか一つだけにチェックマークが付く。プログラムの中ではこれらのメニューは

```
female_menu=Menu(mbutton)
female_menu.add_radiobutton(label="Alice",command=cmd2,variable='V',value='A')
female_menu.add_radiobutton(label="Carol",command=cmd2,variable='V',value='C')
```

及び、 female_menu をカスケードメニューに結びつける記述

```
menu.add_cascade(label="female",menu=female_menu)
```

によって実現されている。メニューの項目としてのチェックボタンとラジオボタンのアクションは command オプションによって指定される。さらに、これらのボタンの利用に当たって

は variable オプションや value オプションが必要である。これらのオプションの使い方はチェックボタンやラジオボタンを単独に使う場合と同じであるので第 7 節と第 8 節を参照すればよい。

Python のメニューは動的なメニュー構成を行う様には設計されていない。即ち、Python のメニューはプログラムを記述する時点で全てのメニュー項目が決定されている事を想定している。従って例えばファイル一覧からファイルを選択するなど、プログラムを実行するまでは確定しない項目を扱う問題には適さない。そのような問題は次節で扱うリストボックスに頼る必要がある。

カスケードメニューを構築するプログラムコードはボトムアップ式、即ちメニューの末端要素から定義していく必要がある。もしもトップダウン式、即ちメニューがマウス操作によって現われる順にプログラムコードを書くと未定義変数を使用する事になり直ちにエラーになるであろう。

10.2 Menubutton

のオプション

- ☐ height
  値: 非負整数
  意味: メニューボタンの高さ。単位は文字数。

- ☐ width
  値: 非負整数
  意味: メニューボタンの横幅。単位は文字数。

- ☐ text
  値: 文字列
  意味: メニューボタンに表示する文字列。

- ☐ menu
  値: メニュー
  意味: メニューボタンによって生成されるメニューを指定する。用法はプログラム例を参考にせよ。

- ☐ indicatoron または indicator
  値: YES または NO (省略時の値は NO)
  意味: メニューボタンの右に小さな四角を表示する。(下図参照)

図 29: インジケータ付きメニューボタン

□ direction

値: 'above', 'below', 'left', 'right', 'flush' (省略時の値は 'below')

意味: メニューをメニューボタンのどの位置に表示するかを指定する。(注意: 'flush' は重なる様に表示するはずだが実際には 'below' と同じ動作をしている。また 'left' と 'right' は働いていない。)

□ state

値: 'normal' , 'active' , 'disabled' (省略時の値は 'normal')

意味: 'active' はマウスをメニューボタンに重ねた時と同じ状態にする。この場合 activeforeground や activebackground が指定されていれば、指定された色で表示されるであろう。'disabled' はメニューボタンの反応を禁止する。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

activebackground activeforeground anchor
background bd bg bitmap borderwidth cursor disableforeground
fg font foreground highlightbackground highlightcolor highlightthickness
image justify padx pady relief takefocus text textvariable
underline wraplength

11 Listbox

11.1 Listbox の機能

Listbox はリストボックスを生成する。リストボックスは複数個の項目の中から 1 つあるいは複数の項目を選択するウィジェットである。1 つの項目の選択だけを許すのか、それとも複数の項目の選択を許すかはオプションによって指定できる。1 つの項目の選択だけを許す場合には機能的にはラジオボタンとよく似ている。複数の項目の選択を許す場合には機能的にはチェックボタンとよく似ている。ラジオボタンやチェックボタンでできることはリストボックスでも代用できる。

リストボックスはラジオボタンやチェックボタンと異なり項目を動的に構成あるいは変更できる。従ってプログラムを実行するまでは定まらない項目を処理するのに適している。(もしもラジオボタンやチェックボタンで同様の処理を行おうとすればウィンドウを新たに生成する必要に迫られるだろう。)

以下にリストボックスを使用した簡単なプログラム例とその実行結果を載せる。

プログラムを見て行く事にしよう。リストボックスは Listbox によって生成される。

```
b = Listbox(f)
```

ボックス中の項目は Listbox のメソッドである insert によって生成される。

```
b.insert(0,"Alice")
b.insert(1,"Bob")
b.insert(2,"Carol")
```

```
from Tkinter import *
def cmd():
    print b.curselection()
f=Frame()
f.pack()
b = Listbox(f)
b.pack()
b.insert(0,"Alice")
b.insert(1,"Bob")
b.insert(2,"Carol")
c=Button(f,text="OK",command=cmd)
c.pack()
mainloop()
```

図 30: プログラム listbox1.py

図 31: リストボックスとボタン

ここに insert の最初の引数は挿入する項目の位置を表している。もしも既にこの位置の項目が存在するならば、insert で指定された項目はその位置に割り込み、それまでに存在していたこの位置以降の項目は 1 つづつ後ろにずれていく。

特に指定しなければリストボックスは browse モードになる。このプログラム例では browse モードになっているので 1 個の項目しか選択できない。複数の項目の選択を許すには Listbox で selectmode オプションを設定する必要がある。複数の選択項目を許す場合には選択操作は複雑になる。操作の方法は selectmode オプションの欄で解説する。

リストボックス中の選択された項目はメソッド curselection によって調べる事が可能である。プログラム例の中の記述

```
print b.curselection()
```

が curselection の使い方を示している。この例では単に curselection の戻り値を出力しているだけであるが、実際には問題に即した処理が必要である。

メソッド curselection は選択された項目の位置をタプル形式で返す。例えば図 9a において (プログラムで複数選択が許されていると仮定して) 項目 “Alice” と “Carol” が選択された状態で OK ボタンをマウスでクリックすれば curselection はタプル形式で ('0','2') を返すであろう。タプルの要素は数字ではなく文字列であることに注意する。これは奇妙

なプログラミングインターフェースであり、プログラマは 関数 `int` または `atoi` を使って `curselection` が返したタプル要素の型変換を行う必要に迫られるであろう。

リストボックスに表示し切れない程多数の項目が存在する場合、利用者は隠れている項目を表示させる必要に迫られる。リストボックスの横幅を越えるような長い項目を含んでいる場合にも同様な問題が発生する。UNIX では 3 ボタンマウスが標準であり、中央のボタンがこの問題の解決のために使用される。MS-Windows の Python も実は 3 ボタンマウスを想定した仕様になっている。しかしながら MS-Windows では 3 ボタンマウスは一般的ではないので[3]実用的な応用に当たってはリストボックスにスクロールバーを付加した方が良いであろう。このための具体的な方法は第 15 節の `Scrollbar` で述べる。

ここでは項目を選択した後に “OK” ボタンで関数を実行させたが、項目を選択しただけで何かある関数を実行できるようなリストボックスを作るには `bind` メソッドを使用すればよい。`bind` に関しては別の機会に解説する予定である。

11.2 ビュー

リストボックスの項目の中に、リストボックスの横幅を越える程の長い文字列が含まれていたり、リストボックスの項目数がリストボックス高さから発生する制約を越えた場合には、ボックスには全体の一部しか表示されない。ボックスに見えている部分をビュー (view) と言う。次の図 32 は項目一覧の全体を囲む矩形 (外側の大きな矩形) と、ボックス (内部の小さな矩形) の関係を示している。影で示す領域が隠れて見えていない部分である。`xview` メソッドおよび `yview` メソッドによってビューを調整できる。調整の必要がなければビューの解説は読み飛ばしても構わない。

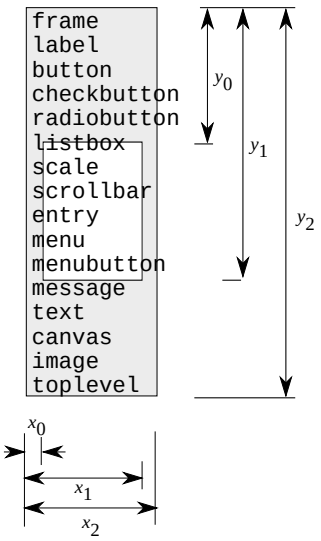

図 32: ウィンドウ

[3]現在では 3 ボタンマウスの機能を持っているインテリマウスが広く普及している。

図に示す様に、全体を囲む矩形の横幅を x_2 縦幅を y_2 とし、この矩形の左上端を起点とした座標系を考え、その下での view の左上端の座標を (x_0, y_0) 、右下端の座標を (x_1, y_1) とする。座標 (x_0, y_0) に関しては $0 \le x_0 \le x_2$ および $0 \le y_0 \le y_2$ の制限が付いている。この図の様に、ボックスの横幅が x_2 より小さいか等しい場合にはボックスの横位置はさらに $x_1 \le x_2$ の制限が付く。同様に、ボックスの縦幅が y_2 より小さいか等しい場合にはボックスの縦位置は $y_1 \le y_2$ の制限が付く。ボックスの横幅が x_2 より大きい場合には x_0 は 0 に固定される。同様に、ボックスの縦幅が y_2 より大きい場合には y_0 は 0 に固定される。いずれにせよ、x_1 と x_2 の大きい方を x_m 、y_1 と y_2 の大きい方を y_m とすると、$0 \le x_0 \le x_1 \le x_m$ $0 \le y_0 \le y_1 \le y_m$ である。そこで、ボックス中での見え方は横方向に関しては 2 つの実数 x_0/x_m と x_1/x_m によって、縦方向に関してはやはり 2 つの実数 y_0/y_m と y_1/y_m によって特徴付けられる。

11.3 `Listbox` のオプション

□ `height`

値: 非負整数

意味: リストボックスの高さ。単位は文字数。

□ `width`

値: 非負整数

意味: リストボックスの横幅。単位は文字数。

□ `selectmode`

値: `"single"`, `"browse"`, `"multiple"`, `"extended"` (省略時の値は `"browse"`)

意味: `"single"` と `"browse"` は 1 つの項目のみを選択するのに使用される。`"multiple"` と `"extended"` は複数個の項目の選択を許す。`"browse"` と `"extended"` を指定するとマウスの左ボタンを押しながら選択項目を変更できる。

`"extended"` モードの選択操作は複雑である。このモードでは SHIFT キーや CTRL キーを押しながらマウスの操作ができる。SHIFT キーを併用した場合には 1 つの範囲の選択のみを許すが、CTRL キーとの併用では複数の範囲の選択が可能である。`"single"` と `"browse"` ではリストボックスの中でマウスの左ボタンを押しながらマウスカーソルを上下に移動する事によって隠れている項目を表示させる事が可能である。他方 `"multiple"` では隠れた項目を表示させるには SHIFT キーまたは CNTRL キーを押しながらマウスの左ボタンを押してマウスカーソルを操作する必要がある。`"extended"` モードでは隠れた項目を表示させる良い方法は存在しない。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

```
background bd bg borderwidth cursor exportselection font
foreground highlightbackground highlightcolor
highlightthicknes relief selectbackground selectborderwidth
setgrid takefocus xscrollcommand yscrollcommand
```

11.4 Listbox のメソッド

以下の説明ではインデックスとは 0,1,2,⋯ などの整数を指す[4] 。この数字は項目に表示されている文字列の配列の添字である。また項目位置 (あるいは単に位置) とはリストボックス中の項目の位置を次の形式で表現したものである。

- 0,1,2,⋯ などの整数
- '0','1','2',⋯ などの 10 進数で表された文字列
- "active", "anchor", "end" の文字列
- "@座標" 形式の文字列

ここに、非負整数はインデックスを表している。また 10 進数で表された文字列も同様である。

マウスをリストボックス内でドラッグすると、マウスの左ボタンを押した時の項目の位置と離した時の項目の位置は異なる。押した時の項目の位置は "anchor" で表されている。そして離した時の項目の位置が "active" である。マウスによるクリック動作では同一項目の上でボタンを押す動作と離す動作を行うので "anchor" と "active" は同じ位置を表している。選択されている項目をマウスの左ボタンでクリックすると選択が取り消されるので "active" は必ずしも選択された項目を意味しない事に注意する。

文字列 "end" は最後の項の次の位置を表す。例えば図 31 では "end" は "Carol" の位置ではなく、実際には存在しないその次の位置である。(項目番号では 4 であり、これは項目数と一致している。) 最後の形式の"@座標" とは、例えば "@100,50" のように指定された文字列でありピクセルを単位として指定された座標位置 (500, 50) の項目の位置を意味する。

□ activate(*n*)

引数 *n*: 項目位置
戻り値: None
機能: リストボックスの位置 *n* の項目を active にする。

□ bbox(*n*)

引数 *n*: 項目位置
戻り値: 4 つの整数を要素とするタプル
機能: 位置 *n* で指定された項目内の文字列を囲む矩形の座標位置とサイズをピクセル単位で返す。図 31 を例にとると、位置 0 の項目は "Alice" と表示されている。実験によると bbox(0) は (4,4,23,13) を返す。この 4 つの数字は文字列 "Alice" を囲む矩形のピクセル座標上の位置と大きさを表している。(4,4) が矩形の左上隅の座標であり、(23,13) が横幅と高さを表している。(Canvas クラスの Rectangle の指定方法を参考にせよ。) なお実験は次の様に行えばよい。図 30 のプログラム listbox1.py の

```
print b.curselection()
```

の後に、

[4]Tcl/Tk ではここに言う「項目位置」の事を「インデックス」と呼んでいる (文献 [4],[5])。ところが Python においてはインデックスとは配列の添字であり、非負整数の値をとる。他方 Tcl/Tk においては配列は連想配列であるためにその添字はもともと文字列を許すのである。

```
print b.bbox("active")
```

を挿入すれば、マウスをクリックした項目位置の bbox の値が出力される。

□ curselection()

戻り値: 文字列を要素とするタプル

機能: リストボックスの項目の内、選択されているものの一覧をタプル形式で返す。例えばインデックスが 0 の項目とインデックスが 2 の項目が選択されていれば ('0','2') を返す。タプルの各要素は数値ではなく文字列である事に注意する。

□ delete(*n*)

引数 *n*: 項目位置

戻り値: None

機能: リストボックスの位置 *n* の項目を削除する。

□ delete(*n*, *m*)

引数 *n*,*m*: 項目位置

戻り値: None

機能: リストボックスの位置 *n* の項目から *m* の項目までを削除する。

□ get(*n*)

引数 *n*: 項目位置

戻り値: 文字列

機能: リストボックスの項目位置 *n* の文字列を返す。

□ get(*n*,*m*)

引数 *n*,*m*: 項目位置

戻り値: 文字列

機能: リストボックスの項目位置 *n* から項目位置 *m* までの文字列をタプルとして返す。

□ index(*n*)

引数 *n*: 項目位置

戻り値: 整数

機能: 項目位置を与えて対応したインデックスを返す。

□ insert(*n*, *s*, ...)

引数 *n*: 挿入位置

引数 *s*: 項目の表示文字列

戻り値: None

機能: リストボックスの項目 *n* で指定された位置に文字列 *s* の項目を挿入する。複数個の文字列を指定できる。挿入前に *n* の後にあった項目は、挿入された項目の個数だけ後ろへずれる。

□ nearest(*y*)

引数 *y*: *y* 座標のピクセル値

戻り値: 整数

機能: *y* で指定された座標に最も近い項目番号を返す。

□ `scan_mark(x, y)`

引数 x: ピクセル値による x 座標
引数 y: ピクセル値による y 座標
戻り値: `None`
機能: `scan_dragto` を見よ。

□ `scan_dragto(x, y)`

引数 x: ピクセル値による x 座標
引数 y: ピクセル値による y 座標
戻り値: `None`
機能: `scan_markdragto` で指定された座標 (x, y) と `scan_dragto` で指定した座標 (x, y) の差の 10 倍だけリストボックスをスクロールさせる。

□ `see(n)`

引数 n: 項目位置
戻り値: `None`
機能: 位置 n の項目がリストボックスの中に見える様にする。

□ `selection_anchor(n)`

引数 n: 項目位置
戻り値: `None`
機能: 位置 n の項目を `anchor` として指定する。`anchor` とはドラッグの開始位置である。

□ `selection_clear(n, m)`

引数 n,m: 項目位置
戻り値: `None`
機能: 位置 n の項目から位置 m の項目までを非選択状態にする。

□ `selection_set(n)`

引数 n: 項目位置
戻り値: `None`
機能: 位置 n の項目を選択状態にする。

□ `selection_set(n, m)`

引数 n,m: 項目位置
戻り値: `None`
機能: 位置 n の項目から位置 m の項目までを選択状態にする。

□ `selection_includes(n)`

引数 n: 項目位置
戻り値: 0 または 1
機能: 位置 n の項目が選択状態であれば 1 を、非選択状態なら 0 を返す。

□ `size()`

戻り値: 整数
機能: リストボックスの項目の個数を返す。

□ `xview()`

戻り値: 0 と 1 の間の 2 つの実数を要素とするリスト。

機能: 図 32 の $[x_0/x_m, x_1/x_m]$ を返す。これはボックス内に見えている部分の割合を表している。

□ `xview(`*n*`)`

引数 *n*: 文字位置

戻り値: `None`

機能: リストボックスに見えている文字の先頭が文字位置 *n* になるようにする。

□ `xview('moveto', `*f*`)`

引数 *f*: 0 から 1 の間の実数

戻り値: `None`

機能: リストボックスの上端側に隠れている部分を *f* で指定された割合にする。

□ `xview('scroll', `*n*`, `*s*`)`

引数 *n*: 整数

引数 *s*: `'units'` または `'pages'`

戻り値: `None`

機能: リストボックスを *n* で指定された分だけスクロールさせる。第 3 引数で `'units'` を指定した場合には行数を単位とし、`'pages'` を指定した場合にはリストボックスの縦の長さを単位とする。

□ `yview()`

戻り値: 0 と 1 の間の 2 つの実数を要素とするリスト。

機能: 図 32 の $[y_0/y_m, y_1/y_m]$ を返す。これはリストボックスに見えている項目の割合を表している。リストの先頭の要素は上端の、リストの第 2 の要素は下端の全体に対する割合である。例えば項目が全体で 100 個あるとし、リストボックス内には項目番号 10 から 20 まで見えているとせよ。その場合には [0.1, 0.2] が返される。

□ `yview(`*n*`)`

引数 *n*: 項目位置

戻り値: `None`

機能: リストボックスに見えている項目の先頭が項目位置 *n* になるようにする。

□ `yview('moveto', `*f*`)`

引数 *f*: 0 から 1 の間の実数

戻り値: `None`

機能: リストボックスの上端側に隠れている部分を *f* で指定された割合にする。

□ `yview('scroll', `*n*`, `*s*`)`

引数 *n*: 整数

引数 *s*: `'units'` または `'pages'`

戻り値: `None`

機能: リストボックスを *n* で指定された分だけスクロールさせる。第 3 引数で `'units'`

を指定した場合には行数を単位とし、'pages' を指定した場合にはリストボックスの縦の長さを単位とする。

12 Entry

12.1 Entry の機能

Entry はエントリを生成する。エントリは文字列をデータとしてプログラムに渡すために使用される。エントリウィジェットは図 34 に示す様にキーボードからの入力欄を持っている。入力終了の合図として OK ボタンと組み合わせて使われる事もあるが、単にリターンキーを打てばよい様にもできる。どのように終了の合図を与えるかはプログラムが決定する。

入力終了の合図を送る前であれば入力欄は訂正が可能である。訂正はマウスやキーを使用して行える。これはエントリウィジェット自体の仕様であるからプログラムでは何も指定する必要はない。

次に Entry を用いた簡単なプログラムとその実行結果を紹介する。このプログラムでは入力終了の合図として OK ボタンとリターンキーを併用している。

```
from Tkinter import *
def doit():
    print e.get()
def do(event):
    print e.get()
f=Frame()
f.pack()
w = Label(f, text='Your name: ',font='Times')
w.pack(side=LEFT)
e=Entry(f)
e.pack(side=RIGHT,expand=YES,fill=X)
e.bind("<Key-Return>",do)
b=Button(None,text="OK",command=doit)
b.pack()
mainloop()
```

図 33: プログラム entry1.py

図 34: エントリとボタン

プログラム entry1.py に現われる

```
e=Entry(f)
```

がエントリを生成する。このプログラムではリターンキーを入力終了の合図にするために bind メソッドを用いて

```
e.bind("<Key-Return>",do)
```

で入力欄にリターンキーが打たれた時に実行される関数 do を指定している。第一引数の "<key-Return>" はリターンキーに反応する事を指定している。もしも ESC キーに反応する様に設計するなら "<Key-Escape>" 、TAB キーなら "<Key-Tab>" である。この書き方は Tcl/Tk に由来するが、ここではこれ以上の解説をしない。キーにバインドされた関数 do は単に

```
e.get()
```

の値を出力している。ここに現われる get メソッドが現在入力欄に表示されている文字列を返すのである。

関数 do は 1 つの引数 event を持っている。bind メソッドが関数 do を呼び出す時には、この引数にイベントに関する情報を渡す。この中には例えばイベントが発生した時刻に関する情報などが含まれる。しかしこの問題に関してもここではこれ以上立ち入らない事にする。

複数個の入力欄が存在する時には入力の順序を利用者に案内した方が親切である。このような例として例えばユーザ名とパスワードの入力を促す場合が挙げられる。言葉で案内しなくても、カーソルが自動的に入力すべき入力欄に位置付けられ、単にキーを打ち込むだけでその入力欄に打った文字が表示される。そして一つの欄の入力が完了するとカーソルはさらに次に入力すべき入力欄に位置付けられる。このような入力欄を実現する為に focus メソッドが準備されている。

次のプログラムとそれに続く図は focus メソッドの使い方とその実行結果を示している。このプログラムを起動すると上の入力欄にカーソルが自動的に位置付けられる。そしてキーボードから “bob” を打ち込き、リターンキーを打つとカーソルはさらに自動的に下の入力欄に移動する。そして “alice” と打ち込んだ後の状態が図 36 である。

```
from Tkinter import *
def do1(event):
    print e1.get()
    e2.focus()
def do2(event):
    print e2.get()
    e1.focus()
f=Frame()
f.pack(expand=YES,fill=X)
Label(f, text='father').pack(side=LEFT)
e1=Entry(f,width=20)
e1.pack(side=RIGHT)
e1.bind("<Key-Return>",do1)
g=Frame()
g.pack(expand=YES,fill=X)
Label(g, text='mother').pack(side=LEFT)
e2=Entry(g,width=20)
e2.pack(side=RIGHT)
e2.bind("<Key-Return>", do2)
e1.focus()
mainloop()
```

図 35: プログラム entry2.py

図 36: プログラム entry2.py の実行画面

この図のエントリウィジェットは e1=Entry(f) と e2=Entry(g) とによって生成される。このプログラムを起動した直後に上の入力欄にカーソルが位置付けられるのは

```
e1.focus()
```

が実行されるからである。

このプログラムでは欄の入力はリターンキーを使って完了する仕様になっている。“bob” を打ち込んでリターンキーを打ち、上の欄の入力が完了すると今度は下の入力欄にカーソルが移る。この動作は

```
e1.bind("<Key-Return>",do1)
```

と関数 do1 の中の

```
e2.focus()
```

によって引き起こされている。

このプログラムでは下の欄の入力が完了すると再び上の欄にカーソルが位置付けられる。この動作は関数 do2 の中の e1.focus() に由来する。

なお図 36 の入力欄は上下ともきっかりと揃っている。これはフレームの pack メソッドのオプション expand=YES,fill=X の御蔭である。

Entry の第三の例題として、入力欄にプログラムによって文字列を書き出したり、表示されている文字列を消したりする問題を採りあげる。例題に実用性を持たせるためにデータベースに検索キーとデータを登録する簡単なプログラムを考える。プログラムコードを図 38:プログラム entry3.py に、実行画面を図 38 に示す。図では検索キーとして “alice” を入力しデータとして 18 を入力した状態を表している。この後リターンキーを打てばキーとデータがデータベースに登録される。

このプログラムにはデータベースらしく次の 3 つのボタンが付いている。

- search ボタン: キーを与えてデータを探索する。
- register ボタン: キーとデータを登録する。
- delete ボタン: キーを削除する。(自動的にデータも削除される。)

このプログラムでは利用者側の操作の誤りを防止するための措置がとられ、その事がプログラムのコードの量を増やしている。利用者が犯す誤りとして次のようなものが考えられる。

1. キーを与えずにデータを登録したり、検索したり、削除したりする。
2. 存在しないキーを検索する。
3. 既に登録済のキーを登録する。
4. 存在しないキーを削除する。

```
from string import strip
from Tkinter import *
from Dialog import Dialog
db={}
def warning(text,*buttons):
    a=Dialog(None,title="warning",text=text,bitmap="",default=0,
        strings=buttons)
    return a.num
def clear(*e):
    for x in e: x.delete(0,"end")
def getkey(e):
    k=strip(e.get())
    if len(k) is 0:
        warning("Put a key","OK")
        e.focus()
        return ""
    return k
def do1(event):
    if getkey(e1) is "": return
    e2.focus()
def do2(event): cmd2()
def cmd1():
    k=getkey(e1)
    if k is "": return
    if db.has_key(k): clear(e2); e2.insert(0,db[k])
    else: warning("No such a key","OK")
    e1.focus()
def cmd2():
    k=getkey(e1)
    if k is "": return
    n=0
    if db.has_key(k): n=warning("The key is already present","Replace",
        "Cancel")
    if n is 0: db[k]=e2.get(); clear(e1,e2)
    e1.focus()
def cmd3():
    k=getkey(e1)
    if k is "": return
    if db.has_key(k): del db[k]; clear(e1,e2)
    else: warning("No such a key","OK")
    e1.focus()
f=Frame(); f.pack(expand=YES,fill=X)
Label(f,text='key').pack(side=LEFT)
e1=Entry(f,width=20)
e1.pack(side=RIGHT)
e1.bind("<Key-Return>",do1)
g=Frame()
g.pack(expand=YES,fill=X)
Label(g,text='data').pack(side=LEFT)
e2=Entry(g,width=20)
e2.pack(side=RIGHT)
e2.bind("<Key-Return>", do2)
e1.focus()
Frame(height=5).pack()
h=Frame(); h.pack()
Button(h,text="search",command=cmd1).pack(side=LEFT)
Button(h,text="register",command=cmd2).pack(side=LEFT)
Button(h,text="delete",command=cmd3).pack(side=LEFT)
mainloop()
```

図 37: プログラム entry3.py

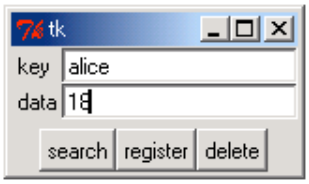

図 38: entry3.py の実行画面

但し、登録済のキーを登録する行為は必ずしも操作上の誤りとは言えず、再登録の可能性もあるので利用者自身の真意を確認する必要がある。操作の中で発生する様々な誤りの表示や確認を行う為に、このプログラムでは Dialog が使用されている。そして Dialog をそのまま使用するのは煩雑なので Dialog に代わる関数 warning を定義している。

```
def warning(text,*buttons):
    a=Dialog(None,title="warning",text=text,bitmap="",default=0,
        strings=buttons)
    return a.num
```

関数 warning は可変個数の引数を持つ。 warning の定義文の第 2 引数 *buttons に付いている記号 '*' が可変個数の引数を可能にしている。

さて、このプログラムにはエントリに作用する 2 つの新しいメソッドが使用されている。関数 clear の定義文に現われる x.delete(0,"end") はエントリ x の入力欄の文字列全体を削除する。関数 clear は delete をこの問題に即していくらか使いやすくしただけである。

search ボタンを選択した時に実行される関数 cmd1 の定義文の中には insert メソッドが使用されている。これはエントリに作用してその入力欄の指定された位置に指定した文字列を挿入するメソッドである。キーとデータは変数 db に登録される。この変数は辞書である[5]。

単にデザインの問題であるが、

```
Frame(height=5).pack()
```

は入力欄とボタンとの間に隙間を空けるために使われている。

このプログラムを完全に実用的なものにするためには、登録されたキーとデータをファイルに保存する機能と、それをファイルからロードする機能が必要になる。(読者の練習問題として残しておく。)

12.2 Entry のオプション

□ show

値: 文字

意味: 入力した文字をこのオプションでした文字で置き換えて表示する。パスワード入力の時に役に立つ。

□ state

値: NORMAL または DISABLED (省略時の値は NORMAL)

[5]辞書の使い方については例えば文献 [1], [7], [9] を見よ。

意味: DISABLED の場合には文字が入力できない。また文字入力用のカーソルも出てこない。

□ width
値: 非負整数
意味: 文字入力欄の横幅。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

background bd bg borderwidth cursor exportselection fg font foreground highlightbackground highlightcolor highlightthicknes insertbackground insertborderwidth insertofftime insertontime insertwidth justify relief selectbackground selectborderwidth selectforeground takefocus textvariable xscrollcommand

12.3 Entry のメソッド

メソッドの説明で「インデックス」とは 0,1,2,… などの非負整数を指す[6]。この数字は文字列の要素を指定している。他方「文字位置」とは

- 0,1,2,… などの非負整数
- '0', '1', '2',… などの非負整数を表す文字列
- 'anchor'、'end'、'insert'、'sel.first'、'sel.last'
- '@0', '@1', '@2', … など '@' 記号が非負整数の先頭に付いた文字列

の事を言う。

これらは入力欄に表示されている文字列の中の特定の文字位置を指定するのに使用される。'anchor' とはマウスで入力欄の文字範囲を選択した時のドラッグの開始位置である。'end' は入力欄の文字列の末尾を意味する。ここには文字は存在しない。'insert' はカーソルの右の文字の位置を表す。'sel.first' は選択されている文字範囲の先頭の文字位置を表す。また 'sel.last' は選択されている文字範囲の最後の文字の次の文字位置を表す。例えば文字列 "alice" の中の "li" が選択されている時には 'sel.first' のインデックスは 1、'sel.last' のインデックスは 3 である。'@100' は入力欄の x 座標 (単位はピクセル) が 100 の位置に最も近い文字位置を表す。

□ delete(*n*)
引数 *n*: 文字位置
戻り値: None
意味: エントリ内の文字位置 *n* の文字を削除する。

[6] ここでの「インデックス」の使い方は Tcl/Tk での使い方とは異なる。Tcl/Tk ではここで言う「文字位置」の事をインデックスと呼んでいる (文献 [4],[5])。Python においてはインデックスとは配列の添字を意味しており、非負整数の値をとる。従ってここでは「インデックス」を Python 本来の意味で用いる。なお Tcl の配列は連想配列であるために、その添字はもともと文字列を許すのである。

□ `delete(n,m)`

引数 *n*,*m*: 文字位置

戻り値: `None`

意味: エントリ内の文字位置 *n* の文字から、 *m* の直前の文字までを削除する。例えば入力欄の文字列が “alice” の下で `delete(0,2)` を実行すると結果は “ice” となる。

□ `get()`

戻り値: 文字列

意味: エントリ内に表示されている文字列を返す。

□ `icurser(n)`

引数 *n*: 文字位置

戻り値: `None`

意味: エントリ内の文字位置 *n* の文字の直前にカーソルを位置付ける。

□ `index(n)`

引数 *n*: 文字位置

戻り値: 整数

意味: エントリ内の文字位置 *n* で指定された文字のインデックスを返す。

□ `insert(n,s)`

引数 *n*: 文字位置

引数 *s*: 文字列

戻り値: `None`

意味: エントリ内の文字位置 n の文字の直前に s で指定された文字列を挿入する。

□ `scan_mark(x)`

引数 *x*: *x* 座標

戻り値: `None`

意味: エントリ内のピクセル座標 *x* の値を記憶する。このメソッドは `scan_dragto` と共に用いられる。

□ `scan_dragto(x)`

引数 *x*: *x* 座標

戻り値: `None`

意味: エントリ内のピクセル座標 *x* の値と `scan_mark(x)` の *x* の値との差の 10 倍分だけ入力欄の文字列を *x* 軸の方向にずらす。入力欄の文字列が入力欄の幅よりも大きくないと効果はない。このメソッドは `scan_mark` と共に用いられる。

□ `selection_adjust(n)`

引数 *n*: 文字位置

戻り値: `None`

意味: `selection_to` とよく似ているが、*n* で指定された文字がセレクションに含まれる。

□ `selection_clear(n)`

引数 *n*: 文字位置

戻り値: None
意味: セレクションを消去する。

□ selection_from(*n*)
引数 *n*: 文字位置
戻り値: None
意味: セレクションの開始位置を指定する。この場合 *n* で指定された文字とその直前の文字の間がセレクションの境界になる。

□ selection_present()
戻り値: 0 または 1
意味: セレクションが存在すれば 1 存在しなければ 0 を返す。

□ selection_range(*n*,*m*)
引数 *n*,*m*: 文字位置
戻り値: None
意味: セレクションの範囲を指定する。
注釈: このメソッドは $n < m$ の場合には selection_from(*n*) と selection_to(*m*) の組と等価である。例えば、入力欄の文字列 “alice” に対して selection_range(1,3) を実行すると “li” がセレクションになる。条件 $n < m$ が成立しない場合にはセレクションを消去する。

□ selection_to(*n*)
引数 *n*: 文字位置
戻り値: None
意味: セレクションの終了位置を指定する。この場合 *n* で指定された文字とその直前の文字との間がセレクションの境界になる。

□ xview()
戻り値: 0 と 1 の間の 2 つの実数を要素とするリスト。
機能: 入力欄に見えている文字の割合を返す。リストの先頭の要素は左端の、リストの第 2 の要素は右端の全体に対する割合である。例えば文字が全体で 100 個あるとし、入力欄にはインデックスが 10 から 20 までの文字が見えているとせよ。その場合には [0.1, 0.2] が返される。

□ xview(*n*)
引数 *n*: 文字位置
戻り値: None
機能: 入力欄に見えている文字の先頭の文字位置が *n* になるようにする。

□ xview('moveto', *f*)
引数 *f*: 0 から 1 の間の実数
戻り値: None
機能: 入力欄の左端側に隠れている部分を *f* で指定された割合にする。

□ xview('scroll', *n*, *s*)

引数 *n*: 整数

引数 *s*: 'units' または 'pages'

戻り値: None

機能: 入力欄を *n* で指定された分だけスクロールさせる。第 3 引数で 'units' を指定した場合には文字数を単位とし、'pages' を指定した場合には入力欄の横幅を単位とする。

13 Text

13.1 Text の機能

Text はテキストウィジェットを生成する。テキストとは一般に複数の行から構成される文字列である。そしてテキストウィジェットはテキストを表示したり編集したりする時に使用されるテキストの容器である。最初に極く簡単なプログラム例を挙げる。

```
from Tkinter import *
def cmd():
    print t.get("1.0","end")
t=Text(None,width=40, height=7)
t.pack()
s='''This is a sample program to handle a text widget.
It shows how we can get the text of this window.
Edit this text and then click OK button.
You will find the text in the console window.
'''
t.insert("end",s)
Button(None,text="OK",command=cmd).pack()
mainloop()
```

図 39: プログラム text1.py

次の図はこのプログラムの実行画面である。

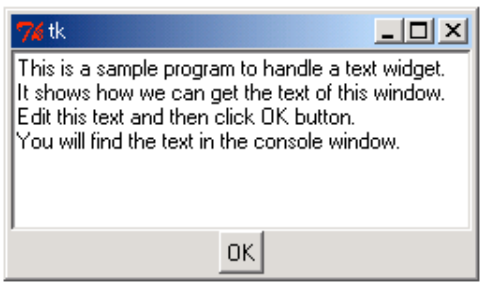

図 40: テキストウィジェット

この図に現われるテキストの文字はどれも同じフォントである。この例題では単一フォントのテキストを扱っているが、テキストウィジェットは複合フォントのテキストも扱うこと

が可能である。さらにテキストの一部をマウスに反応するようにもできるのでハイパーテキストを作成する能力をも持っている。

図 40 にはスクロールバーが備わっていない。スクロールバーが必要なら Scroll ウィジェットを併用すればよい[7]。この方法については第 15 節で解説する。

3 ボタンマウスの下では中央のボタンによってテキストを上下にスクロールさせる事ができる。またマウスを使用しなくても CTRL キーを押しながら上下のカーソルキーを打てばテキストをページ単位に上下に移動できる。 そもそも テキストウィジェットはマウスを使用しないでもある程度テキストを編集できるようになっているのである。SHIFT キーとカーソルキーを併用すればマウスのドラッグの代わりになる。(もっともキーボードだけでは長い長い行の編集はやりづらいであろう。)

このプログラムではテキストウィジェットは

```
t=Text(None,width=40, height=7)
t.pack()
```

によって生成されている。変数 s にはテキストが格納されており、insert メソッドを使った

```
t.insert("end",s)
```

が、このテキストウィジェットにテキストを書き出している。テキストウィジェットの中に表示されているテキストをプログラムの中で参照するには get メソッドを使用する。このメソッドは関数 cmd の定義文の中に次の様に現われている。

```
print t.get("1.0","end")
```

ここにおける "1.0" は最初の行の最初の文字位置を表している。さてこのプログラムでは OK ボタンを押すとテキストの内容がシェルウィンドウに出力される。これを可能にしているのは

```
Button(None,text="OK",command=cmd).pack()
```

であり、この中の command オプションがボタンを押された時に起動される関数を指定している。

テキストウィジェットは豊富な内容を持っており、単独に解説する予定である。従ってここではこれ以上の解説を加えない。

13.2 Text のオプション

Text には以下のオプションが存在する。

```
background bd bg borderwidth cursor exportselection
fg font foreground height
highlightbackground highlightcolor highlightthickness
insertbackground insertborderwidth insertofftime insertontime insertwidth
```

[7] スクロールバー付きのテキストウィジェットがモジュールとして既に存在する。
from ScrolledText import *
を実行して図 39 のプログラムの中の Text の代わりに ScrolledText を使用すればよい。

```
padx pady relief
selectbackground selectborderwidth selectforeground
setgrid spacing1 spacing2 spacing3 state,
tabs takefocus width wrap xscrollcommand yscrollcommand
```

14 Scale

14.1 Scale の機能

Scale はスケールを生成する。スケールとはある実数を 1 つだけ指定するためのウィジェットで、スライダーバーと目盛りから構成される。図 12 にスケールウィジェットを示す。メモリの上にはスライダーバーがあり、スライダーバーはスライダーと溝 (trough) から構成されている。この溝に沿ってスライダーをマウスでドラッグするとスライダーの上に表示された数字が変化する。この数字はスライダーの位置を表している。マウスカーソルを溝に持っていきマウスの左ボタンをクリックすると、マウスカーソルがスライダーの右にあれば右に、左にあれば左にスライダーが 1 単位だけ動く。もしも 3 ボタンマウスを使っていれば、中央のボタンが役に立つ。中央のボタンを溝の上でクリックすると、スライダーはマウスカーソルの位置に飛ぶ。

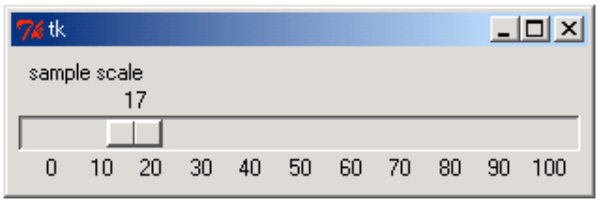

図 41: スケール

この図のスケールは次のプログラムによって生成される。

```
from Tkinter import *
def doit(event):
    print event
Scale(None,label="sample scale",to=100,tickinterval=10,orient="horizontal",
    length=300, command=doit).pack()
mainloop()
```

図 42: プログラム scale1.py

Scale がスケールを生成する。目盛りの最大値は Scale のオプション to によって与えられている。最小値は与えられていないので、0 が仮定されている。目盛りの間隔は tickinterval オプションで与えられる。スケールが横になっているのは orient オプションの指定による。スケールの大きさはピクセル値を単位として length オプションで与えられる。command オプションがスライダーを動かした時に実行される関数を指定する。このプログラムでは関数 doit が実行される。Scale の仕様ではこの時 doit にスライダーの位置情報を渡す。このプログラムでは doit の引数 event がその情報を受け取っている。

14.2 Scale のオプション

□ command 値: 関数
意味: スライダーの位置が変化した時に実行される関数を指定する。スライダーの位置情報が第 1 引数に渡される。

□ digits
値: 非負整数 (省略時の値は 0)
意味: スケールに数字を表示する時の最小文字幅。例えば digits=4 を指定すると、80.0 の様に表示する。

□ from
値: 実数 (省略時の値は 0)
意味: スケールの目盛りの最小値。

□ label
値: 文字列
意味: スケールに添える説明ラベルを指定する。

□ length
値: 非負整数
意味: スケールの長さ (単位はピクセル)。

□ orient
値: 'horizontal' または 'vertical' (省略時の値は 'vertical')
意味: スケールの向きを指定する。'horizontal' を指定するとスライダーを水平に動かす向きに、'vertical' を指定するとスライダーを垂直に動かす向きにスケールを配置する。

□ resolution
値: 実数 (省略時の値は 1)
意味: スケールの精度。この単位で丸められる。

□ showvalue
値: YES または NO (省略時の値は YES)
意味: スライダーにスケールの現在位置を表示するか否かを指定する。

□ sliderlength
値: 非負整数
意味: スライダーの長さをピクセル単位で指定する。

□ state
値: NORMAL または DISABLED (省略時の値は NORMAL)
意味: DISABLED の場合にはスケールは使用できない。

□ tickinterval
値: 実数 (省略時の値は 0)
意味: 目盛りの間隔を与える。0 が指定された場合には刻みの印は表示されない。

□ `to`
値: 実数 (省略時の値は 100)
意味: スケールの目盛りの最大値。

□ `troughcolor`
値: 色名
意味: 溝の色を指定する

□ `variable`
値: 文字列
意味: このスケールと連動する広域変数を指定する。例えば `Entry` ウィジェットの `textvariable` オプションと同じ名前を指定すると、スケールの値が自動的に `Entry` ウィジェットに表示される。

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

```
activebackground background bd bg borderwidth cursor
fg font foreground hightlightbackground highlightthickness
relief repeatdelay repeatinterval takefocus
```

14.3 Scale のメソッド

□ `get()`
戻り値: 実数
意味: スライダーの位置 (スケールの目盛りの値) を返す。

□ `set(v)`
引数 *v*: 実数
戻り値: `None` 意味: スライダーの位置を指定した値に設定する。

15 Scrollbar

15.1 Scrollbar の機能

`Scroolbar` はスクロールバーを生成する。スクロールバーの仕組みはスケールとよく似ているが、スクロールバーは単独では使用されずテキストウィジェットやリストボックスなどと組み合わせ、それらのウィジェットに表示されているものをスクロールさせるのに都合が良いように設計されている。まず最初に次の簡単なプログラムを実行してスクロールバーの働き方を理解しよう。このプログラムを実行するとスクロールバーだけが現われる。ウィンドウの右下隅を下向きにドラッグして、これを縦に引き伸ばすと図 14a に示す姿になる。

この図で分る様に、スクロールバーは三角印の付いた両端の 2 つのボタン (これをスクロールボタンと言う) とスライダーおよびスライダーが動く溝 (trough) から構成されている。このプログラムでは

```
Scrollbar(None,command=cmd).pack(side=RIGHT,fill='y')
```

```
from Tkinter import *
def cmd(*e):
    print e
Scrollbar(None,command=cmd).pack(side=RIGHT,fill='y')
mainloop()
```

図 43: プログラム scrollbar1.py

図 44: スクロールバー

がスクロールバーを生成している。マウスでスライダーを動かすと command オプションで指定された関数 cmd が実行される。

関数 cmd の定義文では任意個数の引数を受け取る事ができるよう になっている。プログラムでは cmd が実行されると受け取った引数を出力する。

スライダーにマウスカーソルを持って行き、ドラッグすると例えば

```
('moveto','0.4103')
```

のように 2 つの文字列が出力される。これは cmd が受け取った引数の e の値である。第一の文字列は 'moveto' であるが、数字を表している第二の文字列はスライダーの位置に応じて変化する。この数字は一番下までドラッグすると 1 になる。スライダーの位置を割合で表しているのである。

上のスクロールボタンをクリックすると今度は

```
('scroll','-1','units')
```

が出力される。下のスクロールボタンだと

```
('scroll','1','units')
```

である。

この奇妙な出力は Listbox の yview メソッドの奇妙な入力仕様に対応している。Scrollbar の command オプションで直接 yview を指定すればよい様になっているのである。

スクロールバーを備えたリストボックスを作成しよう。プログラム 13b に例を載せる。スクロールバーを備えたリストボックスは再利用の価値が高いので本来ならオブジェクトプロ

グラミングのスタイルでコードを書くべきである。しかしここではエッセンスだけを理解して貰うために、あえて初等的な方法を採っている。

```
from Tkinter import *
names=['mouse','cow','tiger','rabit','cat','dog','snake','monkey','lion',
       'elephant',
'sheep','goat','horse','giraffe','zebra','pig','chiken']
def cmd():
    for n in b.curselection(): print names[int(n)]
f=Frame()
f.pack()
b=Listbox(f,selectmode='multiple')
b.pack(side=LEFT)
for n in range(0,len(names)): b.insert(n,names[n])
s=Scrollbar(f)
s.pack(side=RIGHT,fill='y')
b.config(yscrollcommand=s.set)
s.config(command=b.yview)
Button(None,text="OK",command=cmd).pack()
mainloop()
```

図 45: プログラム scrollbar2.py

図 46 はこのプログラムの実行結果である 。

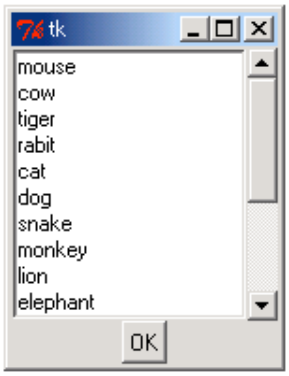

図 46: スクロールバー付きのリストボックス

スクロールバーを備えたリストボックスでは、スクロールバーはリストボックスに対してマウスでドラッグされたスライダーの位置情報を渡す必要がある。他方、リストボックスはスクロールバーに頼らなくてもリストボックスの中でマウスの操作やキーの操作だけで自らスクロールする事が可能である。この場合リストボックス内の一覧の動きに合わせてスクロールバーのスライダーが連動する必要があるので、リストボックスはスクロールバーに対してその動作に必要な情報を渡す必要がある。即ち、リストボックスはスクロールバーと相互に情報を渡し合う必要があるのである。

この問題は変数の相互参照を引き起こす。インタープリタ言語である Python に相互参照問題はやっかいな問題である。プログラム scrollbar2.py がこの問題をどのように解決しているかに注意を払いながらプログラムコードを見て行く事にしよう。

リストボックスは

```
b=Listbox(f,selectmode='multiple')
```

で生成されている。リストボックスが (スクロールバーを使わずに) スクロールした場合に実行される関数は yscroll オプションで指定される。しかしこのオプションはこの段階では使用されていない事に注意すべきである。スクロールバーは

```
s=Scrollbar(f)
```

で生成されている。スクロールバーのスライダーがマウスでドラッグされたり、スクロールボタンが押された場合に実行される関数は command オプションで指定されるが、これもこの段階では指定されていない。

どのウィジェットにも config メソッドが存在する。このメソッドは Listbox や Scrollbar のオプションを後で指定する事を可能にする。このプログラムの中の

```
b.config(yscrollcommand=s.set)
s.config(command=b.yview)
```

によって Listbox と Scrollbar のオプションが指定されている。そしてオプションの中には相互にクロスする形で変数 s や b が使用されている。スクロールバーに作用するメソッド set は s.set(v_1,v_2) のように 2 つの引数を持ち、スライダーを v_1 と v_2 で指定された位置に配置する。2 つの値が指定されるのはスライダーが長さを持っており、上端と下端の位置決めが必要だからである。v_1 と v_2 は 0 と 1 の間の実数で各々スライダーの上端と下端の位置をスライダーが動く溝の長さを基準として表している。他方では Listbox は yscrollcommand オプションで指定される関数に対して、リストボックス内の項目の見え方に関する情報を 2 つの引数で与える。1 つは上方に隠れている項目数の全体の項目数に対する比率であり、もう 1 つは下方に隠れている一番小さな項目番号の全体の項目数に対する比率である。例えば 100 個の項目の内、見えている項目番号が 10 から 19 であれば $v_1 = 0.1$ 、$v_2 = 0.2$ である。従って yscrollcommand には s.set を直接使用できる事になる。ここに述べた動作原理を理解していれば config メソッドを使用しなくてもプログラム scrollbar2.py と同じ動作をするプログラムは作る事ができる。具体的な方法は読者の練習問題としよう。

15.2 Scrollbar のオプション

□ command

値: 関数

意味: スライダーの位置が変化した時に実行される関数を指定する。スライダーの位置情報が第 1 引数に渡される。

□ orient

値: 'horizontal' または 'vertical' (省略時の値は 'vertical')

意味: スクロールバーの向きを指定する。'horizontal' を指定するとスライダーを水平に動かす向きに、'vertical' を指定するとスライダーを垂直に動かす向きにスクロールバーを配置する。

□ `troughcolor`
　値: 色名
　意味: 溝の色を指定する

以上の他に以下のオプションが存在する。これらについては付録 A を参照せよ。

`cursor`

15.3 Scrollbar のメソッド

□ `get()`
　戻り値: 0 と 1 の間の 2 つの実数を要素とするタプル
　意味: スライダーの上端と下端の位置情報を溝の長さを基準として 2 つの実数の組で返す。

□ `set(`v_1`,`v_2`)`
　引数 v_1,v_2: 実数
　戻り値: `None`
　意味: スライダーの位置を指定した値に設定する。ここに v_1 と v_2 は各々スライダーの上端と下端の位置を溝の長さを基準として 0 と 1 の間の実数で表した値である。

□ `fraction(`x`,`y`)`
　引数 x,y: 非負整数
　戻り値: 0 と 1 の間の実数
　意味: 溝の長さを L とせよ。また $(x.y)$ をスクロールバー内の座標点とせよ。垂直方向のスクロールバーであればこの関数は y/L を返す。水平方向の場合には x/L を返す。

□ `identify(`x`,`y`)`
　引数 x,y: 非負整数
　戻り値: `'arrow1'`, `'trough1'`, `'slider'`, `'trough2'`, `'arrow2'`
　意味: スクロールバー内の座標 $(x.y)$ に存在する部品の名前を返す。

16 Image

16.1 Image の機能

`Image` は画像ファイルを読み取り、その内容をウィジェットに張り付ける時に使用される。現在の所、画像ファイルの形式として、PPM 形式、GIF 形式、BMP 形式がサポートされている[8]。これらの形式のファイルは各々 `ppm`、`gif`、`bmp` をファイル拡張子として付ける習わしなのでファイル名を見ればその形式が判明するだろう。画像ファイルのサンプルは Tcl の配布ファイルのディレクトリ `Tcl/lib/tk8.0/demos/images/` の中に入っている[9]。ここではこの中の 3 つのファイル

[8]現在ではこれ以外にも多数のファイル形式が扱える。`http://www.pythonware.com/products/pil/` を見よ。
[9]現在では Windows 版であれば Python の配布ファイルの中に既に含まれている (`Python2.3/tcl/tk8.4/demos/images/`)。また Tk の配布アィルでは `tk8.4.3/library/demos/images/` に含まれている。

teapot.ppm earth.gif flagdown.bmp

を例に解説する。読者はこれらのファイルを以下のサンプルプログラムと同じディレクトリにコピーして例題を実行して欲しい[10]。以下に画像ファイルの扱い方を示すプログラムを示す。

```
from Tkinter import *
filename="teapot.ppm"
Frame().pack()
d=Image("photo",file=filename)   # same as d=PhotoImage(file=filename)
Label(image=d).pack()
mainloop()
```

図 47: プログラム image1.py

このプログラムの実行結果を次に示す。

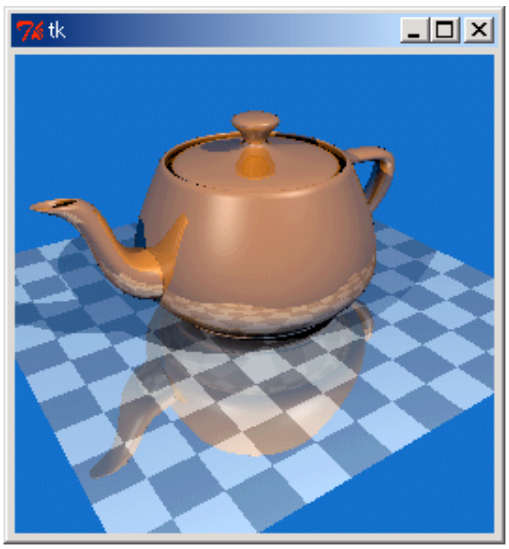

図 48: フォトイメージ (teapot.ppm)

ラベルにイメージを張り付けるには

Label(image=d).pack()

のように image オプションでイメージデータを指定する。イメージデータは

d=Image("photo",file=filename)

で生成される。あるいはプログラム中のコメントで注意されている様に

d=PhotoImage(file=filename)

[10]画像のサンプルは http://ar.aichi-u.ac.jp/netlib/Python/samples/ にも置かれている。

と書いてもよい。Image を使用する場合には、それが実行される前に何らかのウィジェットを生成する必要があるようだ (注 1)。そうしないと、実行時のエラーを引き起こす。このプログラムでは

```
Frame().pack()
```

でダミーのウィジェットが作られている。またこのプログラムを終了するにはウィンドウを消去するしかない。するとエラーメッセージがコンソールに表示される。これを避けるには品よく sys.exit() によってプログラムを終了させるればよい (注 2)。以下のプログラムでも同様である。

> 注 1: Tk() が実行される事が本質的。ウィジェットが生成される時に内部で Tk() が実行されている。第 1 節を見よ。
> 注 2: 例えば
>
> ```
> import sys
> Button(text="quit", command=sys.exit)
> ```
>
> を追加すればよい。

このプログラムは PPM 形式のファイルと GIF 形式のファイルに適用できる。"teapot.ppm" の代わりに "earth.gif" と書き換えれば今度は人工衛星から見た地球の写真が見えるはずである。BMP 形式のファイルはいくらか扱いが異なる。この場合には Image の第 1 引数には "photo" ではなく "bitmap" を指定する。(あるいは BitmapImage を使用する。)

```
from Tkinter import *
filename="flagdown.bmp"
Frame().pack()
d=Image("bitmap",file=filename)   # same as: d=BitmapImage(file=filename)
Label(image=d).pack()
mainloop()
```

図 49: プログラム image2.py

この実行結果を図 50 に示す。

図 50: ビットマップイメージ (flagdown.bmp)

この図で分る様に、BMP 形式の画像は写真画質ではない。二つの色だけで構成される画像である。中間色すら存在しない。ペン書きの絵を扱っているのだと考えればよい。白と黒以外の色で表示したい場合には backgroundcolor と foregroundcolor で色を指定する。

BMP 形式のファイルをラベルに表示するには次のように bitmap オプションでファイル名の前に '@' 記号を付ける事によっても簡便に実現できる。もしも '@' 記号を付けなければ

```
from Tkinter import *
Frame().pack()
Label(bitmap="@flagdown.bmp").pack()
mainloop()
```

図 51: プログラム image3.py

ファイルからではなく標準にサポートされているビットマップ画像の名前を指定した事になる。サポートされているビットマップ画像の名称に付いては付録 A の bitmap オプションを見るがよい。ファイル名である事を指定するのに bitmap オプションで '@' 記号を付ける仕様は Tcl/Tk から引き継いでいるが、いかにも便宜主義的な印象を受けるので、プログラム image2.py のスタイルでコードを書く事を勧める。

この例題では全てラベルに画像を張り付けているが image オプションや bitmap オプションを許すウィジェットであれば何にでも張り付けられる。なお Image, PhotoImage, BitmapImage は Tkinter のモジュールではあるがウィジェットではない。ウィジェットと共に用いられるのでここに紹介した。

17 補足

この節ではウィジェットに関する情報を読者自ら手にする為のノウハウを解説する。この方法は筆者がこの解説を書くにあたって実行した方法でもある。実際に実行して見る事によって筆者の解説が完全に網羅的なものではない事も分るであろう。さらに筆者の解説の誤り (せいぜいミスプリントであることを祈る) も発見できるかもしれない。

もっともこの方法で分る事はウィジェットを生成する時のオプションの一覧とそれが省略された時に設定される値、および内部で定義されている変数やメソッドの名前と値だけである。それでも Python の Tkinter モジュールに関する完全に整理された解説は存在しない事を考えると大いに役立つであろう。以下にプログラムを載せる。

```
from Tkinter import *
from Dialog import *

def attrib(w):
    print "------- %s ------" % w
    keys(w)
    inames(w)
def keys(w):
    t=w()
    for x in t.keys(): v=t.cget(x); print x, type(v), v
def inames(w):
    v=vars(w)   # v is a dictionary
    for x in v.keys(): print x,": ",type(v[x]), ": ", v[x]

attrib(Button)
```

図 52: プログラム default.py

このプログラムを実行すると次の出力が得られる。

```
------- Tkinter.Button ------
activebackground <type 'string'> SystemButtonFace
activeforeground <type 'string'> SystemButtonText
anchor <type 'string'> center
background <type 'string'> SystemButtonFace
bd <type 'string'> 2
bg <type 'string'> SystemButtonFace
bitmap <type 'string'>
borderwidth <type 'string'> 2
command <type 'string'>
cursor <type 'string'>
default <type 'string'> disabled
disabledforeground <type 'string'> SystemDisabledText
fg <type 'string'> SystemButtonText
font <type 'string'> {MS Sans Serif} 8
foreground <type 'string'> SystemButtonText
height <type 'string'> 0
highlightbackground <type 'string'> SystemButtonFace
highlightcolor <type 'string'> SystemWindowFrame
highlightthickness <type 'string'> 1
image <type 'string'>
justify <type 'string'> center
padx <type 'string'> 1
pady <type 'string'> 1
relief <type 'string'> raised
state <type 'string'> normal
takefocus <type 'string'>
text <type 'string'>
textvariable <type 'string'>
underline <type 'string'> -1
width <type 'string'> 0
wraplength <type 'string'> 0
__init__ :  <type 'function'> :  <function __init__ at 8ff790>
tkButtonInvoke :  <type 'function'> :  <function tkButtonInvoke at 8ff8e0>
flash :  <type 'function'> :  <function flash at 8ff900>
invoke :  <type 'function'> :  <function invoke at 8ff9e0>
tkButtonDown :  <type 'function'> :  <function tkButtonDown at 8ff840>
tkButtonUp :  <type 'function'> :  <function tkButtonUp at 8ff8c0>
__doc__ :  <type 'None'> :  None
__module__ :  <type 'string'> :  Tkinter
tkButtonLeave :  <type 'function'> :  <function tkButtonLeave at 8ff820>
tkButtonEnter :  <type 'function'> :  <function tkButtonEnter at 8ff800>
```

この出力の最初の行

```
------- Tkinter.Button ------
```

はタイトルであり、Tkinter の Button ウィジェットに関する情報である事を表している。残りの行は 2 つの部分に分かれる。

1. オプションの一覧とオプションに与えるデータ型および省略時に設定される値

   ```
   activebackground <type 'string'> SystemButtonFace
   ....
   wraplength <type 'string'> 0
   ```

読者はこの表示と筆者の解説 (第 6 節) を比較するがよい。筆者は可能な限り網羅的にオプションに関しては説明しているが省略時の値に関しては十分ではない。

2. 内部変数およびメソッドの一覧とそのデータ型およびその値

```
__init__ :  <type 'function'> :  <function __init__ at 8ff790>
....
tkButtonEnter :  <type 'function'> :  <function tkButtonEnter at 8ff800>
```

このうちアンダースコア (_) で始まる名前は無視してよい。ここに `<type 'function'>` と表示されているのがメソッドである。このこち `tk` で始まるメソッドは Tcl/Tk の付属ドキュメント (文献**??**) には解説されていない。おそらく低レベルのメソッドでプログラムの中では直接使用しないのであろう。筆者の解説では `Button` のメソッドに関しては単に `flush` と `invoke` を解説している。

ここでは `Button` を例に採ったが、これは筆者の解説の一般的な傾向を表している。

A　ウィジェット共通オプション

□ `activebackground`

値: 色名

意味: ウィジェットが選択された時の背景色。

□ `activeforeground`

値: 色名

意味: ウィジェットが選択された時の前景色 (文字の色)。

□ `anchor`

値: `'n'`, `'s'`, `'w'`, `'e'`, `'nw'`, `'ne'`, `'sw'`, `'se'`, `'center'` (省略時の値は `'center'`)

意味: ウィジェット内部の文字列などの配置の方法を指定する。例えば `Label` におけるラベル欄への文字列の配置位置である。値は次に示す配置位置を表す。(文字 e,w,s,n は東西南北に由来する。)

```
nw       n       ne
w      center     e
sw       s       se
```

`pack()` や `place()` での `anchor` はウィジェットの配置を指定している。これと混同してはならない。

□ `background`

値: 色名

意味: 背景色。

□ `bd`

意味: `borderwidth` の別名。

☐ bg
意味: background の別名。

☐ bitmap
値: 'error', 'hourglass', 'info', 'questhead', 'question', 'warning', 'gray25', 'gray50'
意味: ビットマップ画像を表示する。以下に実際の画像を示す。

☐ borderwidth
値: 非負整数
意味: ウィジェットの輪郭の線幅。

☐ cursor
値: カーソル名
意味: ウィジェットの中でのマウスカーソルの形状。

☐ diabledforeground
値: 色名
意味: state オプションで 'disabled' を指定した場合に、text オプションで指定された文字列の表示に使用される色。

☐ exportselection
値: YES または NO (省略時の値は YES)
意味: このオプションで YES が指定されている時には選択文字列をマウスの中央のボタンを使用して他の位置にコピーできる。

☐ fg
意味: foreground の別名。

☐ font
値: フォント名
意味: text オプションで与えた文字列のフォント。

☐ foreground
値: 色名
意味: text オプションで与えた文字列の文字の色。

☐ height
値: 非負整数
意味: ウィジェットの高さ。単位はウィジェット毎に異なる。

☐ highlightcolor
値: 色名
意味: ウィジェットに入力フォーカスがある時の強調枠の色。
注釈: ウィジェットに入力フォーカスがあるとは、キーボードから打ち込まれた文字がそのウィジェット内の入力フィールドに書き込まれる状態を言う。
例: 第 12 節 Entry を見よ。

☐ highlightthickness
値: 非負整数 (省略時の値は 0)
意味: 強調枠の線幅。highlightcolor を参照。

☐ image
値: イメージ
意味: イメージを表示する

☐ insertbackground
値: 色名
意味: カーソルの色

☐ insertborderwidth
値: 非負整数
意味: カーソルの周りの境界線の幅 (単位はピクセル)。

☐ insertofftime
値: 非負整数
意味: カーソル点滅の OFF 時間 (単位はミリ秒)。

☐ insertontime
値: 非負整数
意味: カーソル点滅の ON 時間 (単位はミリ秒)。

☐ insertwidth
値: 非負整数
意味: カーソルの幅 (単位はピクセル)。

☐ justify
値: LEFT, CENTER, RIGHT
意味: 文字列の寄せ方を指定。

☐ padx
値: 非負整数 (省略時の値は 0)
意味: ウィジェットの外側 (左右) に追加する余白の量をピクセル単位で指定する。

☐ pady
値: 非負整数 (省略時の値は 0)
意味: ウィジェットの外側 (上下) に追加する余白の量をピクセル単位で指定する。

☐ relief
値: 'flat', 'raised', 'sunken', 'groove', 'ridge' (省略時の値はウィジェット毎に異なる)
意味: ウィジェットの輪郭のデザインを指定する。以下に実際の画像を示す。

☐ repeatdelay
値: 非負整数
意味: オートリピートまでの時間 (ミリ秒)。

□ `repeatinterval`

値: 非負整数

意味: オートリピートの時間間隔 (ミリ秒)。

□ `selectbackground`

値: 色名

意味: 選択状態の部分の背景色。

□ `selectborderwidth`

値: 非負整数

意味: 選択状態の部分の境界線の幅 (単位はピクセル)。

□ `selectforeground`

値: 色名

意味: 選択状態の部分の前景色。

□ `setgrid`

このオプションは必要が無いと思えるので解説しない。興味のある読者は文献 [4] を見よ。

□ `takefocus`

値: 0 または 1 または長さが 0 の文字列

意味: TAB キーまたは SHIFT+TAB は入力フォーカスの対象となるウィジェットを順に切り替えて行くが、このオプションはその時に入力フォーカスを受け入れるか否かを指定する。`takefocus` の値が 0 ならば受け入れない、1 ならば受け入れる。長さが 0 の文字列を与えた場合には判断をウィジェットに任せる。なお入力フォーカスとは入力欄にキーカーソルを位置付ける事であり、ウィジェットの内部にキー入力欄が存在しない場合には `takefocus` を 1 にしても実際的な意味はないが、入力欄が存在しなくても強調枠は (TAB で入力フォーカスを切り替えた場合には) `highlightcolor` で指定された色に変化する。

□ `text`

値: 文字列

意味: `text` で指定された文字列を表示する。

□ `textvariable`

値: 文字列

意味: 表示されている文字列と連動する広域変数の名称を指定する。

□ `troughcolor`

値: 色名

意味: スライダーが動く溝の色を指定する。

□ `underline`

値: 自然数

意味: `text` で指定された文字列の中の文字に下線を付ける。

例:
`text="Person", underline=4` の場合、Person の s に下線が引かれる。

□ `width`
値: 非負整数
意味: ウィジェットの横幅。単位はウィジェット毎に異なる。

□ `wraplength`
値: 自然数
意味: 文字列を折り畳む時の 1 行の最大文字数。

□ `xscrollcommand`
意味: 横方向のスクロールを指定する。使い方は `yscrollcommand` と同じである。

□ `yscrollcommand`
意味: 縦方向のスクロールを指定する。使い方は第 15 節 `Scrollbar` を見よ。

B ウィジェット共通メソッド

以下に初等的なウィジェット共通メソッドを挙げる。

□ `cget(`*s*`)`
引数 *s*: 文字列
機能: ウィジェットのオプションの現在の値を返す。

□ `configure()`
機能: ウィジェットを指定してオプションを与える。
注意: `configure` に与えるオプションはウィジェット毎に異なる。

□ destroy()
機能: ウィジェットを消す。

□ `getvar(`*s*`)`
引数 *s*: ウィンドウ内のチェックボタンやラジオボタンの `variable` オプションで指定した文字列
機能: ウィンドウ内のチェックボタンやラジオボタンの状態を返す。

□ `pack()`
機能: フレーム内のウィジェットを指定された方法で詰め込む。
注意: `pack` には多数のオプションが存在する。詳しくは文献 [10] を見よ。

□ `place()`
機能: フレーム内のウィジェットを指定された方法で配置する。
注意: `place` には多数のオプションが存在する。詳しくは文献 [10] を見よ。

□ `setvar(`*s*`,`*v*`)`
引数 *s*: 文字列

引数 *v*: 任意の値
機能: ウィジェットの変数 *s* に値を設定する。

C　カーソル名

カーソル名の一覧は `Tcl/include/X11/cursorfont.h` から知る事ができる。以下はその一覧である。

```
arrow based_arrow_down based_arrow_up boat bogosity bottom_left_corner
bottom_right_corner bottom_side bottom_tee box_spiral center_ptr circle
clock coffee_mug cross cross_reverse crosshair diamond_cross dot dotbox
double_arrow draft_large draft_small draped_box exchange fleur gobbler
gumby hand1 hand2 heart icon iron_cross left_ptr left_side left_tee
leftbutton ll_angle lr_angle man middlebutton mouse pencil pirate plus
question_arrow right_ptr right_side right_tee rightbutton rtl_logo
sailboat sb_down_arrow sb_h_double_arrow sb_left_arrow sb_right_arrow
sb_up_arrow sb_v_double_arrow shuttle sizing spider spraycan star target
tcross top_left_arrow top_left_corner top_right_corner top_side top_tee
trek ul_angle umbrella ur_angle watch xterm
```

次の図 53 のプログラムを実行すれば実際のカーソルの形状を知る事ができる。

実行すると下図の画面が表示されるのでカーソル名を選んで OK ボタンをクリックする。そしてカーソルを ”Select a name ... cursor here.” と表示されているウィンドウへ持って行けば指定したカーソルが現れる。

プログラムを打ち込むのが面倒であれば筆者の ftp サーバ

`ftp://ar.aichi-u.ac.jp/Python/samples/`

から採ってくるがよい[11]。ファイル名は `cursor.py` である。

[11] `http://ar.aichi-u.ac.jp/netlib/Python/samples/` にも置かれている。

```
from Tkinter import *
from string import split, splitfields
names=['arrow', 'based_arrow_down', 'based_arrow_up','boat', 'bogosity',
'bottom_left_corner', 'bottom_right_corner', 'bottom_side', 'bottom_tee',
'box_spiral', 'center_ptr', 'circle', 'clock', 'coffee_mug', 'cross',
'cross_reverse','crosshair', 'diamond_cross', 'dot', 'dotbox',
'double_arrow', 'draft_large','draft_small', 'draped_box','exchange',
'fleur','gobbler', 'gumby', 'hand1','hand2', 'heart','icon','iron_cross',
'left_ptr', 'left_side', 'left_tee', 'leftbutton','ll_angle', 'lr_angle',
'man', 'middlebutton', 'mouse', 'pencil', 'pirate', 'plus',
'question_arrow', 'right_ptr', 'right_side', 'right_tee', 'rightbutton',
'rtl_logo','sailboat', 'sb_down_arrow', 'sb_h_double_arrow',
'sb_left_arrow', 'sb_right_arrow', 'sb_up_arrow', 'sb_v_double_arrow',
'shuttle', 'sizing', 'spider', 'spraycan', 'star','target', 'tcross',
'top_left_arrow', 'top_left_corner', 'top_right_corner', 'top_side',
'top_tee', 'trek', 'ul_angle', 'umbrella', 'ur_angle', 'watch', 'xterm']
def cmd():
    n=int(b.curselection()[0])
    g.config(cursor=names[n])
f=Frame()
f.pack()
b = Listbox(f,selectmode='browse')
b.pack(side=LEFT)
for n in range(0,len(names)): b.insert(n,names[n])
s=Scrollbar(f)
s.pack(side=RIGHT,fill='y')
b.config(yscrollcommand=s.set)
s.config(command=b.yview)
g=Message(None,text='''Select a name in the Listbox and push OK button,
and then move the cursor here.''',relief=SUNKEN,bg="white")
g.pack(side=LEFT)
Button(None,text="OK",command=cmd).pack(side=RIGHT)
mainloop()
```

図 53: プログラム cursor.py

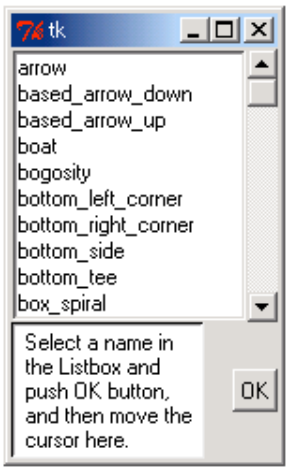

図 54: カーソル

参考文献

[1] Mark Lutz 著/飯坂剛一監訳、村山敏夫、戸田英子共訳: 「Python 入門 (O'Reilly Japan, 1998)

[2] Mark Lutz 著/飯坂剛一監訳、村山敏夫、戸田英子共訳: 「Python プログラミング (O'Reilly Japan, 1998)

[3] Mark Lutz:"Programming Python",(O'Reilly & Associates,Inc. 1996)

[4] John K.Ousterhout 著/ 西中芳幸、石曾根信共訳: 「Tcl&Tk ツールキット」 (ソフトバンク、1995)

[5] 宮田重明、芳賀敏彦『Tcl/Tk プログラミング入門』(オーム社、1995)

[6] V.Quercia,T.O'Reily/ 大木敦雄監訳: 「X ウィンドウ・システム・ユーザ・ガイド」 (ソフトバンク、1993)

[7] 有澤健治「Python によるプログラミング入門」(講義テキスト, 1999)

[8] 有澤健治「Python によるグラフィックス」 (「Com」Vol.10,No.2、愛知大学情報処理センター、1999 年 9 月)

[9] 有澤健治「Python のすすめ」 (「Com」第 17 号, 愛知大学情報処理センター、1999)

[10] 有澤健治「Python における GUI の構築法 I — ウィジェットの配置 —」 (「Com」第 18 号、 愛知大学情報処理センター、2000)

[11] "Python 1.5 Documentation" (配布ファイル Python/Doc/index.html)

[12] "Tcl/Tk Reference Manual" (配布ファイル Tcl/doc/tcl80.hlp, v8.0.4)