目 次

Python における GUI の構築法 I
—　ウィジェットの配置　—

有澤 健治

平成 17 年 3 月 1 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。

1　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` クラスの記述のシンタックスは `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。しかしながら Tcl/Tk を Python に翻訳する作業は機械的には進まない。Tcl と Python は文法体系が異なり、機械的な翻訳は必ずしも成功しないのである。また Tcl/Tk に関する知識が要求される事はプログ

[1]「Python による GUI の構築法 I — ウィジェットの配置 —」(「Com」Vol.11, No.1、愛知大学情報処理センター、2000 年 3 月) 但し、表現の微小な校正は行なっている

ラミングの初心者にとって余りにも辛い。Python の中で閉じたウィジェットの解説が要求されるのである。

ウィジェットの解説は 2 つの部分から構成される。1 つはウィジェットの配置法の解説である。他は個々のウィジェットの解説である。この論文ではウィジェットの配置法を解説する。個々のウィジェットの解説は「Python における GUI の構築法 II」で解説する。

この解説に現われるサンプルプログラムは 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 [6] や [7]) 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。さらに、この解説ではオブジェクト指向プログラミングの用語である「メソッド」が多数現われる。オブジェクト指向プログラミングに不慣れな読者の為に簡単に説明しておこう。メソッドとは変数の属性として定義された関数であり、その変数に作用する。例えば変数 `x` の属性として関数 `f()` が定義されている場合には `x.f()` で `x` の中の関数 `f()` を呼び出し、同時に変数 `x` に作用させる事ができる。普通の関数であれば `f(x)` と書く所であるがメソッドの場合には変数名の後にピリオドが来て、その後に関数名が来るのである。

最後に、本論に入る前に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。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 (Python はオブジェクト指向プログラミング言語であるから) 「b は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 `Button` 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 例題

Python における `Widget` クラスの使用法を見るために、まず最初に幾つかのウィジェットを使用したサンプルプログラムを図 1 に紹介する。

このプログラムを実行し、“Pet ” ボタンをマウスの左ボタンでクリックする (以下では単に「選択する」と言う) とプルダウンメニューが表示され、そのメニューの中に “cat” と “dog” 2 つの項目が現われる。そして “cat” を選択すると次の画面 (図 2) が表示される。

```
from Tkinter import *
from Canvas import *
import sys

names=("Alice","Bob","Carol")
def draw(c,s):
    c.addtag_all('t')
    c.delete('t')
        CanvasText(c, 100, 100,text=s,font='Times 30 bold italic',
        anchor='nw')
def cmd():
    t=a1.curselection()
    if t is (): return
    s = names[a1.index(t[0])]
    draw(c,s)
def cmd1():
    draw(c,"cat")
def cmd2():
    draw(c,"dog")
def quit():
    sys.exit()

a=Frame()
a.pack(side='left',fill='y')
a1 = Listbox(a,width=10)
a1.pack(expand=YES,fill='y')
for n in range(0,len(names)):
    a1.insert(n,names[n])
a2=Button(a,text="OK",command=cmd)
a2.pack()

b=Frame()
b.pack(fill='x')
b1=Label(b,text="Sample Program",font="times")
b1.pack()
b2 = Menubutton(b,text="Pet",relief='raised',width=30)
b2.pack(side=LEFT,fill='y')
m1 = Menu(b2)
m1.add_command(label="cat",command=cmd1)
m1.add_command(label="dog",command=cmd2)
b2["menu"]=m1

b3=Button(b,text='quit',command=quit)
b3.pack(side=RIGHT)

# --- Canvas starts from this line ---
c=Canvas(width=300,height=200,background='cyan')
c.pack()

mainloop()
```

図 1: プログラム 1.

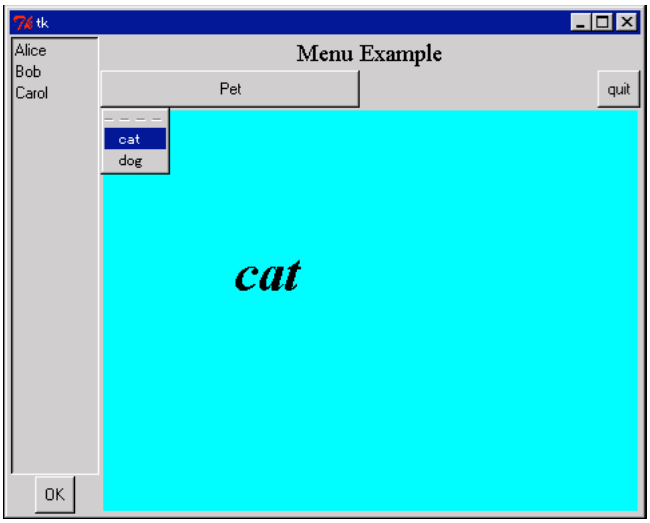

図 2: プログラム 1 の実行結果

図 2 は以下の部品要素 (ウィジェット) から構成されている。

ラベル (Label)　単に “Sample Program” と表示している。プログラムの中では

```
b1=Label(b,text="Sample Program",font="times")
b1.pack()
```

によってこの表示が行われている。

ボタン (Botton)　“quit” と表記されているのはボタンである。また “OK” もボタンである。プログラム 1 では “quit” を選択すると実行が終了する。プログラムの中ではボタン “quit” の表示とアクションは

```
b3=Button(b,text='quit',command=quit)
b3.pack(side=RIGHT)
```

および

```
def quit():
        sys.exit()
```

によって与えられている。他方 OK ボタンの表示とアクションは

```
a2=Button(a,text="OK",command=cmd)
a2.pack()
```

および

```
def cmd():
    t=a1.curselection()
```

```
    if t is (): return
    s = names[a1.index(t[0])]
    draw(c,s)
```

によって与えられている。

メニュー (Menu)　“Pet” ボタンとそれを選択した時のプルダウンメニューはプログラム中の

```
b2 = Menubutton(b,text="Pet",relief='raised',width=30)
b2.pack(side=LEFT,fill='y')
m1 = Menu(b2)
m1.add_command(label="cat",command=cmd1)
m1.add_command(label="dog",command=cmd2)
b2["menu"]=m1
```

による。またプルダウンメニューの項目を選択した時のアクションは

```
def cmd1():
          draw(c,"cat")
def cmd2():
    draw(c,"dog")
```

によって与えられている。

リストボックス (Listbox)　左側の “Alice”, “Bob”, “Carol” を表示しているボックスはリストボックスと呼ばれている。これは

```
names=("Alice","Bob","Carol")
```

及び

```
a1 = Listbox(a,width=10)
a1.pack(expand=YES,fill='y')
for n in range(0,len(names)):
    a1.insert(n,names[n])
```

によって与えられている。このボックスから項目を選択し、“OK” ボタンを選択すると Canvas に選択した項目の名称が表示される。このアクションは

```
def cmd():
    t=a1.curselection()
    if t is (): return
    s = names[a1.index(t[0])]
    draw(c,s)
```

によって与えられている。ここの

```
a1.curselection()
```

がボックス中の選択項目を表しているのである。

キャンバス (Canvas)　右下の水色の領域はキャンバスと呼ばれる描画領域である。この領域は

```
c=Canvas(width=300,height=200,background='cyan')
c.pack()
```

によって確保され、

```
def draw(c,s):
    c.addtag_all('t')
    c.delete('t')
    CanvasText(c, 100, 100,text=s,font='Times 30 bold italic',anchor='nw')
```

によって、このキャンバスへ文字列を描く関数 draw が定義されている。

ここに現われた部品は全て Widget クラスから派生したクラスである。これらが占める領域はどれも矩形 (2 つの垂直線と 2 つの水平線に囲まれた長方形) である事に注意する。全ての部品が矩形であるが為にそれらの配置規則は比較的簡単である。Python では 2 つの部品配置法を使用できる。一つは座標を使用する初等的な方法であり place メソッドを使用する。しかしながらこの方法は実用上の問題点を含むので普通は第二の方法である pack メソッドを使用する。この方法は部品を箱へ詰め込む方法を「上下左右」によって指示する。

3　ウィジェットの配置 (pack メソッド)

3.1　部品の詰め込み

図 3 の太い枠で囲った矩形の部分領域 (濃い灰色の矩形) にボタンを配置する事を考えよう。ここではボタンを例に採っているが、任意のウィジェットで構わない。太い枠は普通にはフレームと呼ばれる枠組みだけのウィジェットであるが、一般には任意のウィジェットと考えて差し支えない。一般的にはウィジェットの中にウィジェットを配置するので言葉使いが混乱しない様に、また話をあまり抽象的にしないために、フレームの中に部品を配置すると考えよう。pack メソッドでは部品の配置が許されるフレームの部分領域が存在する。そしてその領域は常に矩形である。この領域を「詰め込み可能領域」と呼ぶ事にする。ここでは詰め込み可能領域を濃い灰色で表しているのである。

pack メソッドでは部品の辺が詰め込み可能領域の境界に接する様に詰め込む。詰め込み方は 4 つのパターンに分類できる。即ち、上詰めにする、下詰めにする、左詰めにする、右詰めにするの 4 つである。

Python ではこれらの詰め込みの方法を各々 'top', 'bottom', 'left', 'right' であらわす。 これらを関数 pack の side オプションで与える事ができる。即ち、部品 A を表す変数を a とすると、左詰めにする場合には

```
a.pack(side='left')
```

と書き表すのである。('top', 'bottom', 'left', 'right' の代わりに TOP, BOTTOM, LEFT, RIGHT を使用してもよい。) 左詰めを例に解説する。部品 A を詰め込み可能領域に左詰めにすると部品 A は図 3 のように配置される。

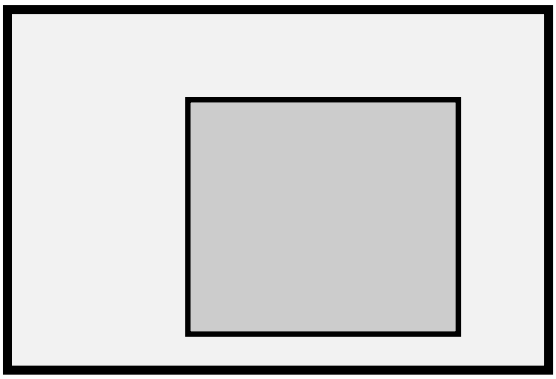

図 3: 詰め込み可能領域

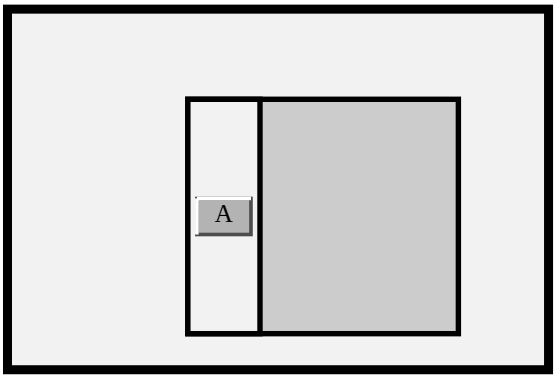

図 4: 図 3 の詰め込み可能領域に A を左詰めした図

部品 A を左詰めにした後に残る詰め込み可能領域は濃い灰色で示したある。部品 A の上下にできた隙間は A が自由に利用できる領域と考え、これを A の区画と言う。ここには他の部品は詰め込めない。この説明は他の詰め方に対しても同様に成立する。図 4 の詰め込み可能領域にさらに部品 B を上詰めすると図 5 に濃い灰色で示した詰め込み可能領域が残る。部品 B の左右にできた隙間は B の区画である。

詰め込みのどの段階でも残された詰め込み可能領域は矩形である事に注意する。部品のサイズが詰め込み可能領域より大きい場合にはフレームは自動的に必要なだけ拡大され、その結果詰め込み可能領域も拡大される。そしてフレームに詰め込まれる部品の配置位置が拡大されたフレームの下に計算される。このルールの下に詰め込み可能領域は常にどこかに残されている。残された矩形領域は高さまたは横幅が 0 かも知れないがどこかに残っているのである。従って部品は幾つでも、残された矩形領域の大きさを気にすることなく詰め込み可能である。

図 4 と図 5 では部品は各々の区画の中央に配置されている。この配置は関数 pack の anchor オプションによって変更可能である。部品はその区画いっぱいに引き伸ばす事が可能である。

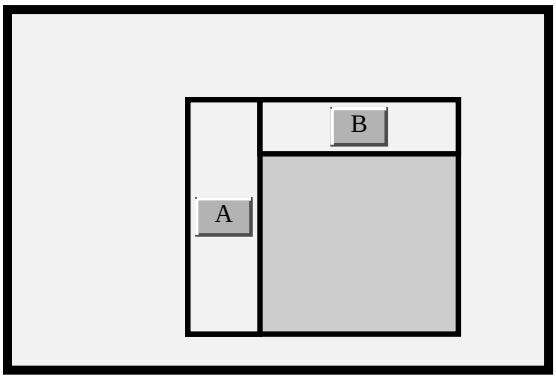

図 5: 図 4 の詰め込み可能領域に部品 B を上詰めした図

引き伸ばしの方向は pack 関数の fill オプションによって指定する。

プログラムの中では、部品 A を表す変数を a とすると

```
a.pack(fill='x')
```

で *x* 方向の引き伸ばしを表す。fill の値として 'x' の外に、'y', 'both', 'none' が許される。これらの値は、X, Y, BOTH, NONE と書いてもよい。

3.2 プログラム 1 による部品配置

さてプログラム 1 によって作成された図 2 はフレームと部品との関係が分る様に図示すると図 6 のようになっている。

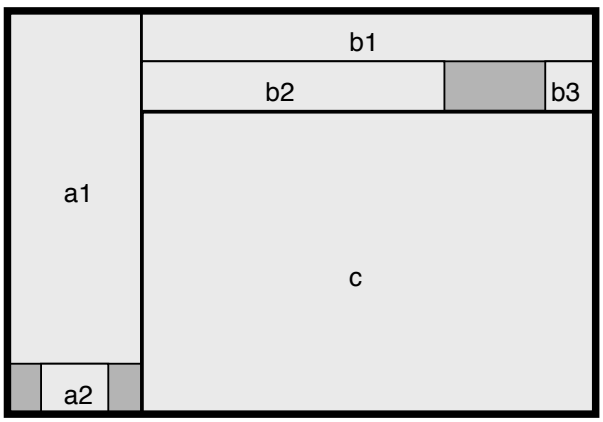

図 6: プログラム 1 によって生成される部品の入れ子関係

図の部品に表記された a1, a2, b1, b2, b3, c はプログラム 1 のウィジェットを表す変数に対応している。プログラムを観察するとこれらの変数は以下の様に生成されている。

```
a=Frame()
a1 = Listbox(a,...)
a2=Button(a,...)
b=Frame()
b1=Label(b,...)
b2 = Menubutton(b,...)
m1 = Menu(b2)
b3=Button(b,...)
c=Canvas(...)
```

最初の行

```
a=Frame()
```

により a はフレームであることが分る。第 2 の行

```
a1 = Listbox(a,...)
```

から a1 はリストボックスでありことが分る。関数 Listbox の最初の引数は親のウィジェットを表す。ここでは親として a が与えられている。この事はこのリストボックスは a の中に配置される事を意味する。 同様にして a2 も a の中に配置される事が分り、また、b1 と b2 と b3 は b の中に配置される事も分る。

3.3 Toplevel

このプログラムでは a, b, c の 3 つのウィジェットは親が指定されていない。親が指定されていないウィジェットは Toplevel と呼ばれるウィジェットの中に配置される。1 個のウィンドウについて 1 個の Toplevel が存在する。図 6 では一番外側の太い線で囲まれた矩形が Toplevel と呼ばれるウィジェットである。そして Toplevel の子ウィジェット a, b, c は 2 番目に太い線で囲んである。図 7 に a,b,c の占める 3 つの領域を示す。

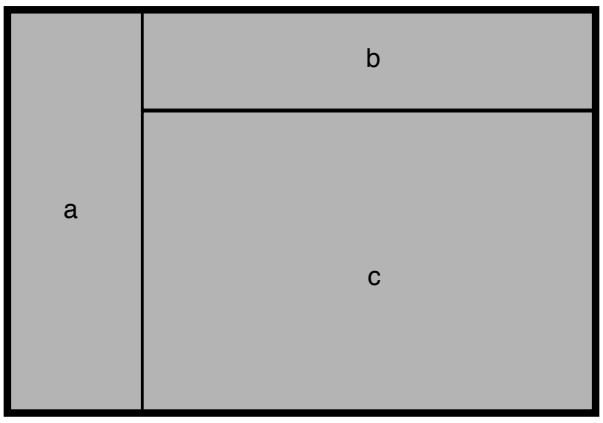

図 7: プログラム 1 による Toplevel の子ウィジェットの区画

3.4 ウィジェットの親子関係

ウィジェットの入れ子関係 (箱の中に箱を詰め込む時に発生する関係) はプログラムの中では親子関係として現われる。そしてこれらの関係は図 8 に整理して示す様に共に木構造として表現される。

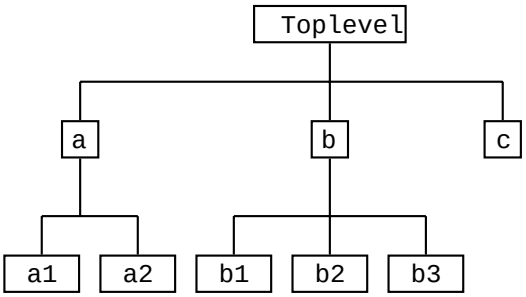

図 8: プログラム 1 のウィジェットの親子関係

3.5 Toplevel の大きさ

Toplevel やフレームは必要最小限度の大きさにとられる。プログラム 1 を見てみよう。ウィジェット c は大きさが

```
c=Canvas(width=300,height=200,background='cyan')
```

によって明示的に与えられている。またフレーム a の横幅はリストボックス a1 の横幅が

```
a1 = Listbox(a,width=10)
```

によって明示的に与えられることによって定まっている。フレーム b の中のウィジェット b1, b2, b3 の高さはプログラム中では明示的に指定されていないが、そのような場合には暗黙の仮定値が使用される。そして (何故か) メニューボタンの方がボタンよりも少し低い。そこで b2 の高さを b3 に合わせるために

```
b2.pack(side=LEFT,fill='y')
```

と fill オプションを使用して y 方向に引き伸ばしている。(読者はこの行を単に

```
b2.pack(side=LEFT)
```

としたプログラムを実行して結果を比較するのが良い。) 結局フレーム b の高さは b1 と b3 の和である。従って `Toplevel` の横幅は a1 の横幅と c の横幅の和、高さは b1 の高さと b3 の高さと c の高さの和として定まる。

3.6 残された詰め込み可能領域

プログラム 1 によって、残された詰め込み可能領域がどこに発生したかを理解する為に

```
a1.pack(expand=YES,fill='y')
```

の代わりに

```
a1.pack(fill='y')
```

と置き換えてプログラムを実行してみよう。この場合には部品は図 6 に代わって図 9 に示すように配置される。

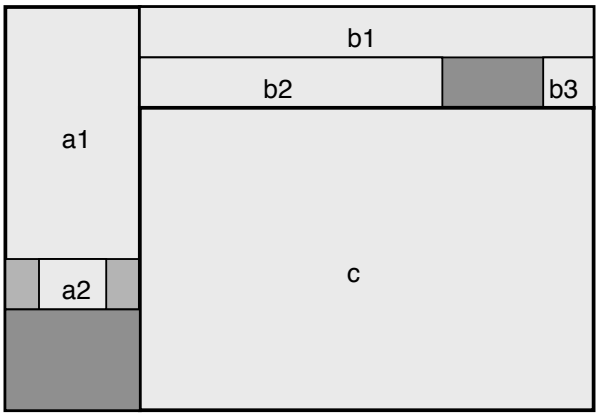

図 9: プログラム 1 による Toplevel の子ウィジェットの残された区画

ボタン a2 の下に示した濃い灰色の矩形領域がフレーム a の中に残された詰め込み可能領域であり、ボタン b3 の左の濃い灰色の矩形領域がフレーム b の中に残された詰め込み可能領域である。そして Toplevel に残された詰め込み可能領域は高さが 0 の矩形すなわち単なる線分としてウィジェット c の底辺と重なって存在する。

フレーム a の中に残された詰め込み可能領域はウィジェット a1 の区画ではない。従って

```
a1.pack(fill='y')
```

のように fill オプションを指定しても、この領域にまではウィジェット a1 は引き伸ばされる事はない。ウィジェット a1 を引き伸ばして a2 の下の濃い灰色の矩形領域をなくするには

```
a1.pack(expand=YES,fill='y')
```

のようにオプションとして expand=YES を指定する。

3.7 pack メソッドのオプション

3.7.1 after

値: ウィジェット意味: after で指定されたウィジェットの次に詰め込む。例: a, b, c をウィジェットとする。すると、

```
a.pack(); b.pack(); c.pack(after=a)
```

は

```
a.pack(); c.pack(); b.pack()
```

と同一の効果を持つ。

3.7.2 anchor

値: 'n', 's', 'w', 'e', 'nw', 'ne', 'sw', 'se', 'center' (省略時の値は 'center') 意味: ウィジェットをその区画のどこに配置するかを指定する。下図に区画を矩形でウィジェットをボタンで示し、anchor の値と配置位置との関係を示す。ボタンのラベルが anchor の値を表している。

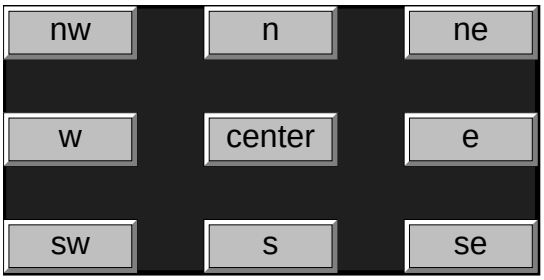

図 10: anchor の指定による配置位置の違い

例: anchor オプションの効果を調べるには次のプログラムを実験してみるが良い。

```
from Tkinter import *
a=Button(text='alice',width=30)
a.pack()
b=Button(text='bob', height=5)
b.pack(side=LEFT)
c=Button(text='carol')
c.pack(expand=YES, anchor='center')
mainloop()
```

このプログラムによって Toplevel ウィジェットに広い c の区画が発生する。anchor の指定によって 'carol' と表示されたボタンの位置が変化するであろう。

3.7.3 before

値: ウィジェット意味: before で指定されたウィジェットの前に詰め込む。例: a, b, c をウィジェットとする。すると、

```
a.pack(); b.pack(); c.pack(before=b)
```

は

```
a.pack(); c.pack(); b.pack()
```

と同一の効果を持つ。

3.7.4 expand

値: YES, NO または 'yes', 'no' (省略時の値は NO) 意味: 親に生じた余白 (詰め込み可能領域) にまで引き伸ばす事を指示する。本文を参照せよ。

3.7.5 fill

値: X, Y, BOTH, NONE または 'x', 'y', 'both', 'none' (省略時の値は NONE) 意味: ウィジェットをその区画いっぱいに引き伸ばすか否かを指定する。引き伸ばす場合にはその方向を X, Y, BOTH で指定する。

3.7.6 ipadx

値: 数値 (省略時の値は 0) 意味: ウィジェットの内側 (左右) に追加する余白の量をピクセル単位で指定する。

3.7.7 ipady

値: 数値 (省略時の値は 0) 意味: ウィジェットの内側 (上下) に追加する余白の量をピクセル単位で指定する。

3.7.8 padx

値: 数値 (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。

3.7.9 pady

値: 数値 (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。

3.7.10 side

値: LEFT, RIGHT, TOP, BOTTOM または 'left', 'right', 'top', 'bottom' (省略時の値は TOP) 意味: ウィジェットを親のどの辺に配置するかを指定する。

4　ウィジェットの配置 (place メソッド)

place メソッドは初等的ではあるが、実用的には問題点が含まれるのでは実際には殆ど使用されない。従ってこの節は読み飛ばして構わない。

4.1　基本な考え方

ボタン A をフレームの中に配置する事を考えよう。 ここではボタンを例に採っているが実際には任意のウィジェットで構わない。また、フレームの中に配置する事を考えているが実は任意のウィジェットの中に詰め込み可能である。(もっともフレーム以外に配置する事は実際問題としては無いであろう。) フレームは明示的にそのサイズが指定されているものとする。

place メソッドはフレームに座標系を定め、ウィジェットの配置位置をその座標によって指示する。この考え方は Canvas の中に図形を配置する時の考え方そのものである。フレーム内のウィジェットの座標は図 11 に示す様に、 Canvas の場合と同様にフレームの左上隅を原点 $(0,0)$ と考え、右方向を x 軸の正方向、下方向を y 軸の正方向と考える。図ではフレームの横幅を w 、高さを h としている。するとフレームの右上隅は $(w,0)$ 、左下隅は $(0,h)$ 、右下隅は (w,h) となる。

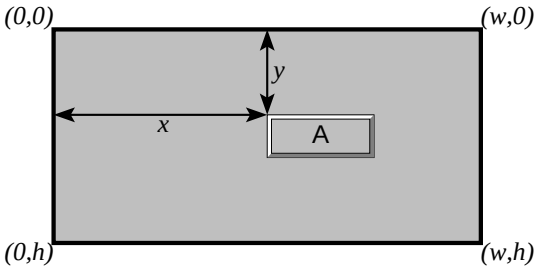

図 11: ウィジェットとフレームおよびフレームの座標系

ウィジェットが点ではなく大きさを持つ図形の為にウィジェットの座標なる概念は確定しない。そこで特に指定しない限り、ウィジェットの左上の角の位置でフレーム内でのウィジェットの座標を表す。図にはウィジェットの座標が (x,y) である事の意味も合わせて示されている。

ウィジェットの座標を指定するのに左上の角ではなく他の位置、例えばウィジェットの中心の方が都合が良い場合もあり得る。その場合には anchor オプションで座標の意味を変更できる。

4.2　例題

以下に座標を指定してラベルを配置するプログラムとその実行結果を紹介する。ウィジェットの輪郭が見える様に relief='raised' を指定してある。そのため実行結果はボタンが張り付けられた様に見えるが実際にはラベルである。

```
from Tkinter import *
# we must explicitly give the size of frame if 'place' method is used
f=Frame(width=150,height=60)
f.pack()
# 'relief' is specified in order to see the position of label
widget = Label(f, text='Hello GUI World!',relief='raised')
widget.place(x=60,y=30)
mainloop()
```

図 12: プログラム 2: place メソッドを使用したラベルの配置

図 13: プログラム 2 の実行結果

このプログラムでは Frame で矩形領域 f を生成する。f の大きさは横幅 150 高さ 60 であり、長さはピクセルを単位としている。ラベルが配置されるフレームは関数 Label の最初の引数によって与えられている。今の場合はフレーム f である。ラベルの配置位置は

```
widget.place(x=60,y=30)
```

によって与えられている。ここにはアンカーの指定がない。この場合には座標 $(60, 30)$ はウィジェットの左上角の位置を表していると解釈される。anchor オプションによって明示的にウィジェットの座標の意味を指定できる。

anchor は図 12 に示す 9 個の値をとる事ができる。図の矩形は今の場合ラベルであり、黒丸はラベル上のアンカーの位置を表している。'center' は中央を意味している。'e', 'w', 's', 'n' は何れも辺の中点で東西南北の意味である。また 'nw', 'ne', 'sw', 'se' は矩形の頂点である。

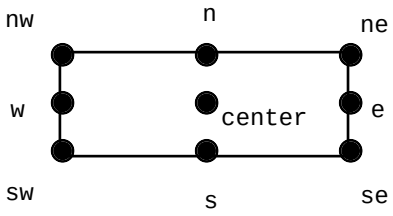

図 14: anchor の値の意味

place の引数 x=60,y=30 はフレームの座標系でのアンカーの座標を表している。単位はピクセルである。

フレームの座標系をピクセル単位ではなくフレームの大きさに対する相対値で与える事もできる。即ち、x 方向についてはフレームの横幅と比べた大きさで、y 方向についてはフレー

ムの高さと比べた大きさで座標系を定義する。そのような座標系ではフレームの 4 つの左上隅は $(0, 0)$ 、右上隅は $(1, 0)$ 、左下隅は $(0, 1)$ 、右下隅は $(1, 1)$ となる。

プログラム 2 の中の

```
widget.place(x=60,y=30)
```

を相対座標で表すと

```
widget.place(relx=0.4, rely=0.5)
```

となる。(`relx = 60/150, rely = 30/60`) ウィジェットの大きさをフレームの大きさとの比率で表す事も可能である。(`place` メソッドのオプションである `relwidth` 及び `relheight` を見よ。)

ウィジェットを `place` メソッドで配置するのは初等的ではあるが、プログラムを起動した後で `Toplevel` の大きさが変更された場合に困難が生じる。すなわち絶対座標を使用した場合にはウィジェットが相互に重なり合う事がある。相対座標を使用した場合にはウィジェットの大きさが不自然に変る。こうした困難のために `place` メソッドは殆ど使用されない。

4.3 place メソッドのオプション

4.3.1 anchor

値: `'n', 'w', 's', 'e', 'nw', ''ne', 'sw', 'se', 'center'` (省略時の値は `'nw'`)
意味: フレームの座標系におけるアンカーの x 座標。

4.3.2 x

値: 実数意味: フレームの座標系におけるアンカーの x 座標。

4.3.3 y

値: 実数意味: フレームの座標系におけるアンカーの y 座標。

4.3.4 relx

値: 0 から 1 の間の実数意味: フレームの相対座標系におけるアンカーの x 座標。

4.3.5 rely

値: 0 から 1 の間の実数意味: フレームの相対座標系におけるアンカーの y 座標。

4.3.6 width

値: 数値意味: ウィジェットの横幅。

4.3.7 height

値: 数値意味: ウィジェットの高さ。

4.3.8 relwidth

値: 0 から 1 の間の実数意味: フレームに対する相対的なウィジェットの横幅。

4.3.9 relheight

値: 0 から 1 の間の実数意味: フレームに対する相対的なウィジェットの高さ。

4.3.10 bordermode

値: `'inside'`, `'outside'`, `'ignore'` (省略時の値は `'inside'`) 意味: フレームの輪郭線には幅 (太さ) があるのでフレームの座標原点 (0, 0) の意味は曖昧性を持っている。`'inside'` の場合には輪郭の内側の左上隅で、`'outside'` の場合には輪郭の外側の左上隅で座標原点を定義する。`'ignore'` は輪郭線を無視する。その結果は `'outside'` と同じになる。

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] 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)

[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)