# 目 次

# Python における GUI の構築法 III
## — Text ウィジェット —

有澤 健治

平成 17 年 3 月 9 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。

ここでは Python 2.0 に基づいて Python におけるテキストウィジェットを解説する。 Python におけるテキストウィジェットの簡単な解説、即ち、プレーンテキストの扱いは既に筆者は解説済である (文献 [14])。そこでの解説は Python1.5 に基づいていたが、2.0 においても基本的な修正点は存在しない。

テキストの扱いにおいて Python 1.5 から Python 2.0 への改訂の中で発生した大きな変化は、Python1.6 においてフォントの指定が

```
"Times-New-Roman 12 italic bold"
```

のように 1 個の文字列で表現するやりかたから

```
("Time New Roman",12,"italic bold")
```

のようにタプルで表現するやり方に変更された事と、Python 2.0 においてユニコードがサポートされた事である (文献 [16])。

Python 2.0 においても 1 個の文字列で表す旧来のフォントの表現形式がサポートされているので、フォント名に漢字を含まない限り 1.5 時代のプログラムの実行に支障はない[2]。

この 1 年の間に Python に関する本が新たに何冊か出版された (文献 [3],[4],[6])。この中で J.E.Grayson の “Python and Tkinter Programming” (文献 [6] ) が注目される。この本は Python の Tkinter に関する初めての本格的な解説書である。今回の筆者の `Text` ウィジェットの解説は当然ながらこの本が参考にされている。この本に載っている `Text` クラスのメソッドの個別解説には間違いが散見されるが、それでも、これまで手探りによる Tkinter の研究を強い

[1]「Python における GUI の構築法 III — Text ウィジェット —」( 「Com」Vol.12, No.1、愛知大学情報処理センター、2001 年 5 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

[2]Python 2.0 ではプログラムの中にユニコード以外の漢字コードを含んだ場合に問題が発生したが、この問題は Python 2.2 で解決している

られていた筆者にとっては、この本の出現は画期的な事である。読者には筆者の解説 (文献 [11],[13],[14]) と合わせてこの本を読む事を勧めたい。

筆者が文献 [11],[13],[14] で行ってきた解説および今回の解説と J.E.Grayson の解説の特徴を比較しておく。J.E.Grayson は Tkinter の全体をバランス良く纏めているように思える。そして、オプションやメソッドに関して網羅的に解説している。他方筆者は必ずしも網羅的な解説は行なわずに、メソッドの使い方をいろいろなプログラム例を挙げながら深く解説している。

Python は `http://www.python.org/` から無料で手に入る。Python の Windows への組み込みは Python2.0 では非常に簡単になったので特に説明する程でもないであろう。この解説に現れるサンプルプログラムは全て Windows2000 で動作の確認が行われている。またプログラムの実行図は全て Windows2000 でのものである。これらのプログラムは UNIX でも (フォントやウィンドウデザインの違いを別にすれば) 同じように動作するはずである。

# 1 サンプルプログラム

## 1.1 テキストウィジェットの表現能力

図 1 にテキストウィジェットの出力サンプルを載せる。プログラムコードは譜 1 に示されている。このプログラムではテキストの中に画像とマウスに反応するボタンが埋め込まれている。そしてボタンを押すたびにテキストの中の標題文字 “Tea Pot” の色が変化する。

図で分かる通り、Python のテキストウィジェットでは複数のフォントの混在が可能で、さらに文字列に対して色や下線などの属性を指定できる。図では画像とボタンが埋め込まれているが、それだけではなく任意のウィジェットを埋め込む事が可能である。この例では扱われていないがテキストウィジェットはハイパーテキストを扱う能力も備えている。即ち文字列に対してマウス操作で発生するイベントと、イベントが発生した時に実行される関数を指定できる。

## 1.2 実行上の注意点

図 1 に現れる画像情報はファイルに納められている。画像ファイルの名前は `teapot.ppm` である[3]。譜 1 のプログラムでは、`teapot.ppm` はプログラム `sample.py` と同じフォルダ (ディレクトリ) の中に置かれていると想定されている。そうする事によって、このプログラムは単にマウスのクリックによって実行できる。コマンドによってプログラムを実行する場合にはカレントディレクトリをこのディレクトリに移す必要がある。

[3] `teapot.ppm` は Tcl/Tk 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。読者は `teapot.ppm` の代わりに任意の GIF 形式のファイルを使用しても構わない。その場合にはファイル拡張子を `gif` にする。

**譜 1** sample.py

```
from Tkinter import *
title_color=("blue","red","green","yellow","black")
k=0

def doit():
    global k
    k=k+1
    if k == len(title_color): k=0
    t.tag_configure("title",foreground=title_color[k])

t = Text(width=45, height=40, wrap="word")
t.tag_configure('title', font=("Verdana", 24, "bold italic"),
    foreground=title_color[k],justify="center")
t.tag_configure('signature', font=("Times Roman", 12, "bold"))

t.insert(END, "Tea Pot\n","title")
s='''The Text widget is a versatile widget. Its primary purpose is\
 to display text, of course, but it is capable of multiple styles\
 and fonts, embedded images and windows, and localized event binding.
-- John E.Grayson --
'''
t.insert(END, s)

photo=PhotoImage(file='teapot.ppm')
t.image_create(END,image=photo)

t.insert(END,'''
This picture is stolen from Tk demos:
/usr/lib/th8.0jp/demos/images/teapot.ppm
in Linux for example
You can embed windows in the text like this: ''')

b = Button(text='Push', command=doit)
t.window_create(END, window=b)

t.insert(END,'''
Push the button, then you will observe the title color is changed.

This sample program is coded by: ''')
t.insert(END,"arisawa@aichi-u.ac.jp", 'signature')
t.pack()

mainloop()
```

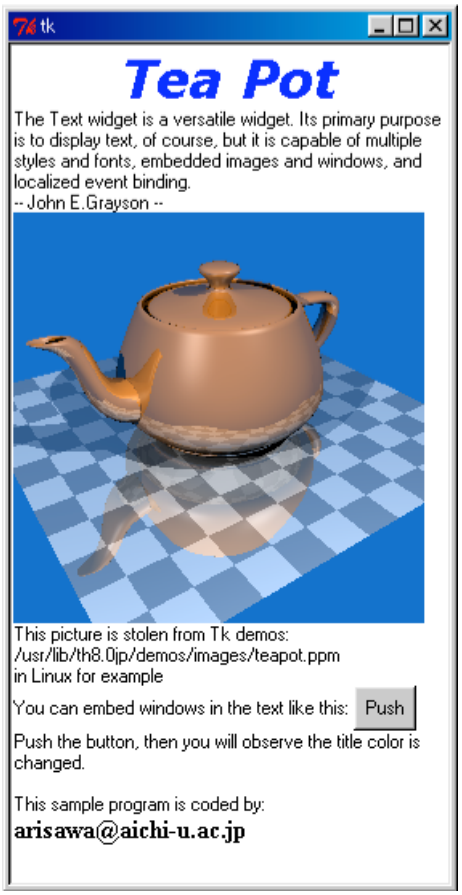



図 1: テキストウィジェット (sample.py の実行結果)

## 1.3 プログラムの解説

プログラムを眺めてみよう。Tkinter の Text ウィジェットを使用するには

```
from Tkinter import *
```

をあらかじめ実行しておく必要がある。

プログラムの中の

```
t = Text(width=45, height=40, wrap="word")
```

に注目しよう。ここに現れる Text はテキストウィジェットの生成子である。Python ではウィジェットの生成子[4]の名称はクラス名を使用し、関数のような使い方をする。即ち ( ) 内にオプションを与え、生成子が生成したウィジェットが変数に代入される。ここでは変数として t が使用されている。従ってこの代入によって t は Text によって生成されたテキストウィジェットを表す事になる。

生成されるテキストウィジェットは、横幅 45、高さ 40 のサイズである事が ( ) 内のオプションから読み取れる。ここに現れる数字は文字数および行数である。ここでは文字の大き

[4]「生成子」と言う言い方は一般的ではない。普通はクラスオブジェクトと言うが、ここでは分かりやすく生成子と言う。

さの基本となるフォント名が与えられていない。この場合には省略時に暗黙に仮定されるフォント名 ( default font ) が使用される。Windows ではこのフォント名として 8 ポイントの "MS Sans Serif" が選ばれている。

Text のオプション WRAP="word" は、テキストが行の右端で折り曲げられる時にワード単位で折り曲げられる事を意味している。このオプションが指定されていない場合には文字単位の折り曲げとなる。Text には以上の他に多数のオプションが存在するが詳しくは第 2 節を参照されたい。

```
t.tag_configure('title', font=("Verdana", 24, "bold italic"),
foreground=title_color[k], justify="center")
t.tag_configure('signature', font=("Times Roman", 12, "bold"))
```

ここではテキストウィジェット t におけるテキストタグ 'title' と 'signature' を定義している。タグとは付箋の意である。ここではタグ 'title' の属性として

```
font=("Verdana", 24, "bold italic")
foreground=title_color[k]
justify="center"
```

また 'signature' の属性として

```
font=("Times Roman", 12, "bold")
```

を定義している。ここでは、font によってフォントガ指定され、foreground によって文字の色が、justify によって行の中での文字列の配置が指定されている。このプログラムではこれらの 2 つの tag_configure は最初に 1 回だけ実行される。この段階では k==0 なのでタグ 'title' の foreground の値は "blue" で初期設定された事になる。(tag_configure で指定可能な属性に関しては第 3 節を見よ。)

このプログラムでは tag_configure で定義されたタグ "title" と "signature" は後に

```
t.insert(END, "Tea Pot\n","title")
```

と

```
t.insert(END,"arisawa@aichi-u.ac.jp\n", 'signature')
```

で使用している。ここに現れる

```
t.insert(....)
```

は指定された文字位置に文字列を挿入するメソッドである。ここでもテキストウィジェット t を指定し、それに対する作用として insert が使用されている。そして insert の最初の引数に現れる END はテキスト中の最後の文字位置を表している。insert の第 3 引数にはタグが指定される。即ちフォントや文字属性の指定はタグを通じて指定される仕組みになっているために文字を飾りたい場合には tag_configure を使用することになる。

```
s='''The Text widget is a versatile widget. Its primary purpose is\
 to display text, of course, but it is capable of multiple styles\
 and fonts, embedded images and windows, and localized event binding.
-- John E.Grayson --
'''
t.insert(END, s)
```

この insert の使用例ではタグが指定されていない。この場合には省略時に暗黙に仮定されるフォント ( default font ) で文字を表示する。変数 s への代入文の右辺の文字列はプログラムコードの中では複数の行に渡っている。ここでは最初の 2 つの行は逆スラッシュ(\) で終わるので、意味の上では行は次に継続される。

```
photo=PhotoImage(file='teapot.ppm')
t.image_create(END,image=photo)
```

で画像が張りつけられる。画像情報はファイル teapot.ppm から読み取られる。ここでは画像ファイルがこのサンプルプログラムと同一のディレクトに置かれていると想定されている。

ボタンは

```
b = Button(text='Push', command=doit)
t.window_create(END, window=b)
```

によって張り付けられている。ボタンが押されると command オプションで指定された関数 doit に制御が移る。そして関数 doit の中では

```
    k=k+1
    if k == len(title_color): k=0
```

が実行され k の値が 1 だけ増える。そして k の値が 4 になれば k=0 に修正される。即ちボタンをマウスクリックするたびに k は 0, 1, 2, 3 と変化する。そしてそのたびに

```
t.tag_configure("title",foreground=title_color[k])
```

が実行され、タグ "title"の foreground 属性だけが title_color[k] で指定された値 、即ち"blue","red","green","yellow","black" の 1 つに設定され、図 1 の標題文字 “Tea Pot” の色が変化する。

# 2 文字位置の表現

## 2.1 論理行と物理行

テキストは行の集まりである。Python が扱う行は改行記号 ('\n') で終わり、残りは空白文字と印字可能な文字から構成される。1 つの行の文字数には制約はない。このようにテキストの表示のされ方とは関係なく、テキストを構成する文字だけで行が定義される場合には「論理行」と言う。

他方、テキストの実際の見え方はウィンドウの大きさから発生する物理的な影響を受けている。そこでは行の大きさはウィンドウの横幅を超える事ができないので折り返して表示されるのが普通である。我々の日常的な行概念では、折り返された部分は次の行に属していると見做している。例えば本のあるページの行数を数える場合にはそのような考えに立っている。このように表示装置との関わりで行を捕らえる時「物理行」と言う。

論理行において行の先頭にあった文字は物理行でも行の先頭に位置付けられる。1 つの論理行はウィンドウの中では我々が段落と呼んでいるものに相当するのである。

## 2.2 テキストインデックス

テキストウィジェットが保持しているテキストの中の文字の位置を表現するのに「テキストインデックス」が使用される。テキストインデックスは

'1.0','1.1','1.2',…
'2.0','2.1','2.2',…

などのように行番号 (1, 2, 3, ...) と、その行の中での文字位置 (0, 1, 2, ...) をピリオドで結んだ文字列として表現される。ここで行番号とは論理行の番号である。例えば

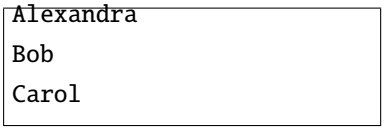



図 2: テキスト例

のように 3 つの行から構成されるテキストでは、
Alexandra の 'A' のテキストインデックスは '1.0'、 'e' のテキストインデックスは '1.2'
Bob の 'o' のテキストインデックスは '2.1'
Carol の 'r' のテキストインデックスは '3.2'
である。

行末には改行記号が存在する。この改行記号の位置もまたテキストインデックスで表される。この例では 1 行目、2 行目、3 行目の改行記号のテキストインデックスは各々 '1.9'、'2.3'、'3.5' である[5]。

逆にテキストインデックスから文字が定まる。例えば図 2 では '1.0' の文字は 'A' である。テキストインデックスはテキストウィジェットに含まれるテキストの実際の行数や、行中の文字数とは関わりなく形式的に定義できる。例えば図 2 のようなテキストに対しても形式的に '1.20' のようなテキストインデックスを考える事ができる。このようにテキストインデックスで指定された文字位置がその行の文字数を超える場合にはその行の改行記号を指すものと考える。従って図 2 では '1.20' は 1 行目の改行記号の位置を表し、'1.9' と同じ意味であると考える。

テキストインデックスで指定された行番号がテキストウィジェット中のテキストの行数を超える場合には、そのテキストインデックスは END (テキストの終わりを意味する) を指すものとする。 END のテキストインデックスは N をテキストの行数とするとき、'N.0' とする。例えば図 2 では END のテキストインデックスは '4.0' である。

テキストウィジェットの中に画像やボタン等のウィジェットが含まれる場合にはそれらのウィジェットはテキストインデックスの計算では文字の一種と看做される。

テキストウィジェットをプログラムで扱う場合には屡々ウィジェット中に文字列を挿入したり、削除したりする必要が発生する。そのような場合に、挿入や削除する場所を指定するのにテキストインデックスが使用されるのである。

## 2.3 拡張されたテキストインデックス

Python はここに述べたテキストインデックスの他に文字の位置を表す他の便宜的な表現法を持っている。

[5]埋め込んだテキストとその結果として生じるテキストウィジェット中の内部テキストの間にはテキスト末の改行記号を巡って微妙な違いが存在する。この問題に関しては次の節を見よ。

(A) ピクセルを単位とする座標 を $(x,y)$ とする時 '@$x$,$y$' で表現される文字列。例えば '@34,105' は座標点 (34, 105) に存在する文字位置を表す。

(B) 'end'、'insert'、'anchor'、'sel.first'、'sel.last' などの語。この内最初の 3 つは END, INSERT, ANCHOR で代用できる。

文字列'end' はテキストの末尾を意味する。ここには文字は存在しない。'insert' は入力カーソルの右の文字の位置を表す。入力カーソルが行末にある場合には、その右に改行記号が存在する事に注意せよ。'anchor' とはマウスの左ボタンを使用してテキストウィジェットの文字範囲を選択した時のドラッグの開始位置である。'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。例えばテキストウィジェットの 2 行目が文字列 “Bob” で始まるとして、“Bob” の中の “ob” が選択されている時には 'sel.first' のテキストインデックスは '2.1' 、 'sel.last' のテキストインデックスは '2.3' である。さらに Python はテキストインデックスおよび上記の文字位置の表現法 (A), (B) に続けて表 2 に示す修飾子を続ける事ができる。

| 修飾子 | 意味 |
|---|---|
| + *n* chars | *n* 文字後 |
| - *n* chars | *n* 文字前 |
| + *n* lines | *n* 行後 |
| - *n* lines | *n* 行前 |
| linestart | 行の先頭位置 |
| lineend | 行の末尾 |
| wordstart | 単語の先頭 |
| wordend | 単語の末尾 |

表 1: テキストインデックス修飾子

ここに $n$ は整数である。即ち、例えば

'2.3 + 5 chars'

や

'insert lineend'

のような表現が可能である。なお $n$ 文字後の文字とは、入力カーソルを指定されたテキストインデックスに位置付け、右向きのカーソルキーを $n$ 回叩いて移動した入力カーソルの位置である。特に、行末記号の 1 文字後の文字は (次の行が存在すれば) 次の行の先頭文字である。例えば図 2 では

'2.5 + 3 chars'

の意味は次のようになる。'2.5' の位置に文字は存在しない。この場合には '2.5' は 2 行目の行末である '2.3' を意味する。ここから 3 文字後の文字は Carol の r であり、このテキストインデックスは '3.2' である。

以下では (A) と (B) の表現や修飾子を追加した表現を含めて「拡張されたテキストインデックス」と呼ぶことにする[6]。

## 2.4 テキストインデックスを扱うメソッドの一般規則

メソッドの引数には常に拡張されたテキストインデックスが許される。他方メソッドが返すテキストインデックスは常に行番号とその中での文字位置で表現されている。従ってメソッドの引数の説明では単にテキストインデックスと言えば拡張されたテキストインデックスを意味ものとする。

## 2.5 テキストインデックスの比較関係

テキストインデックス $i$ が $j$ の前にあるとは、$i$ の指す文字が $j$ の指す文字の前にある事を言う。テキスト中の文字の前後関係は、英文を読む時の自然な時間順序で考えればよい。 また、$j$ が $i$ の後にあるとは、$i$ が $j$ の前にある事を言う。拡張されたテキストインデックスについても同様である。

## 2.6 テキストの範囲の表現

テキストの範囲は 2 つのインデックスの組で表現できる。 $i$ と $j$ をテキストインデックスとすると範囲 $(i, j)$ とは、$i$ 以降で $j$ より前にあるテキストインデックスの集まりを言う。(即ち $j$ を含まない。) Python は範囲を常にこの考え方で扱う。

## 2.7 テキストインデックスの値を調べるための簡単なプログラム

譜 2 にテキストインデックスの値を調べるための簡単なプログラムを載せる。
このプログラムを実行すると図 3 が現れる。


図 3: 譜 2 の出力

マウスカーソルをこのテキストウィジェットの中に持っていくとカーソルが十字型に変化し、マウスの右ボタンを押すと関数 `doit` が実行される。この動作はプログラムの中の

```
t = Text(width=15, height=4, cursor="tcross")
```

と

[6]J.E.Grason はここで言う「拡張されたテキストインデックス」を単に index と書いている (文献 [6])。また、Python のエラー出力では「拡張されたテキストインデックス」が Text Index と表示されている。

**譜 2** テキストインデックスの性質を調べる実験プログラム index1.py

```
from Tkinter import *

def doit(event):
    s="@%d,%d"%(event.x, event.y)
    print s
    print t.index(s)
    print t.get(s)
    print t.bbox(s)
    print t.index(END)
    print t.index(INSERT)
    print t.tag_ranges(SEL)
    if t.tag_ranges(SEL) != ():
        print t.index(ANCHOR)
        print t.index('sel.first')
        print t.index('sel.last')

t = Text(width=15, height=4, cursor="tcross")
t.pack()

t.insert(END, "Alexandra is a little girl.\n")
t.insert(END,"Bob")
b=Label(text="label", relief=GROOVE)
t.window_create(END,window=b)
t.insert(END,"Carol")
t.bind("<Button-3>",doit)
mainloop()
```

```
t.bind("<Button-3>",doit)
```

によって引き起こされている。関数 doit の引数 event にはイベント情報が渡される。この引数は構造体 (複数の変数のパッケージ) であり、その中の x と y がウィジェット中のマウスカーソルの中心位置を与えている[7]。このプログラムの実行画面でマウスカーソルの中心を Bob の b に位置付け、マウスの右ボタンを押すとコンソール画面 (コマンドを実行したウィンドウの画面) に

```
@18,39
2.2
b
(16, 32, 6, 13)
3.0
2.9
()
```

が表示されるであろう。この内、'@18,39' がピクセル座標で表されたマウスカーソルの位置であり、プログラムの

```
s="@%d,%d"%(event.x, event.y)
print s
```

[7]他にどのような変数を内部に持っているかに興味のある読者は print vars(event) をこの関数の中で実行させるがよい。

によって出力される。

2 行目の '2.2' は Bob の 'b' のテキストインデックスであり、これは

```
print t.index(s)
```

によって出力される。

次の 'b' はテキストインデックス '2.2' の位置 (マウスカーソルの位置) にある文字であり、

```
print t.get(s)
```

によって出力される。

その次の '(16, 32, 6, 13)' は

```
print t.bbox(s)
```

によって出力されており、テキストインデックスのバウンディングボックス ( bounding box ) と呼ばれる矩形領域の情報を与えている。即ち、ピクセル座標で表されたこの 4 つの数字のうち (16, 32) が矩形領域の左上の座標、(6, 13) が矩形領域の横幅と高さを表しており、この領域はマウスカーソルの指す文字のテキストインデックスが '2.2' となる領域である。

5 行目の '3.0' は

```
print t.index(END)
```

が表示している。譜 2 のプログラムでは "Carol" の後に改行コードを与えていない。それにも関わらず END の位置は次の行の先頭の文字の位置になっている事が分かる[8]。

6 行目の '2.9' は

```
print t.index(INSERT)
```

によって表示され、入力カーソルの位置を与えている。この位置はキーを打った時に文字が挿入されていく位置である。読者は入力カーソルの位置を変更してマウスの右ボタンを押せば、この蘭には他の値が表示されるのを観測できるであろう。

最後の '()' は

```
print t.tag_ranges(SEL)
```

によって出力されている。この関数は文字列がマウスによって選択されている範囲を表している。文字列をマウスで選択するにはマウスの左ボタンを使って選択したい範囲をドラッグすればよい。(すると選択された文字列は青色の背景で表示されるであろう。) ここで '()' が表示されたと言う事は、文字列が選択されていなかったと言う事を意味している。もしも文字列を選択した状態でマウスの右ボタンをクリックすれば以下の 3 つの関数がさらに実行される事になる。

```
print t.index(ANCHOR)
print t.index('sel.first')
print t.index('sel.last')
```

この中に現れる ANCHOR はマウスのドラッグの開始位置である。ここでは文字列を選択した時にどの位置からドラッグしたかを表している。このうち、

```
print t.index('sel.first')
print t.index('sel.last')
```

[8] この問題に関しては次節を見よ。

の出力を見ると 'sel.first' と 'sel.last' は必要ない事が分かる。なぜなら、これらの位置は tag_ranges(SEL) によっても与えられているのである。

さて、このプログラムでは文字だけでなくラベルが挿入されているのは、テキストインデックスの計算においてラベルのようなウィジェットも文字と同じ扱いを受けている事を読者に知って欲しいからである。Carol の C にマウスを合わせてマウスの右ボタンをクリックして、その時のテキストインデックスが '2.4' である事を確認して欲しい。

## 2.8 テキストインデックスに関するメソッド

□ index(*i*)

引数 *i*: テキストインデックス
戻り値: テキストインデックス
機能: 拡張されたテキストインデックスを基にテキストインデックスを返す。

□ compare(*i*,*o*,*j*)

引数 *i*,*j*: テキストインデックス
引数 *o*: "<",">","<=",">=","==","!=" のいずれかの文字列
戻り値: 0 または 1
機能: 2 つのテキストインデックス *i* と *j* の指す文字位置を比較し、関係 *o* が成立していれば 1 を、成立していなければ 0 を返す。

関係 "<" は *i* の指す文字が *j* の指す文字より前にある事を意味する。関係 ">" は *i* の指す文字が *j* の指す文字より後にある事を意味する。また "==" は *i* と *j* の指す文字位置が同じである事を意味する。さらに "!=" は "==" の否定を意味する。">=" は ">" または "==" である事を意味する。"<=" は "<" または "==" を意味する。

こうした関係はテキストの具体的な文字配置との関わりで定義されている事に注意しよう。例えば t をテキストウィジェットとし、そのテキストが図 2 で与えられているとすると t.compare('1.2','<','3.1') は 1 を返す。また t.compare('1.10','!=','1.12') は 0 を返す。

□ bbox(*i*)

引数 *i*: テキストインデックス
戻り値: 4 個の数字のタプル $(x_1, y_1, w, h)$ または None
機能: 文字位置 *i* の文字を囲む矩形エリアを左上の座標 $(x_1, y_1)$ と矩形の横幅 $w$ と高さ $h$ で返す。

この矩形は *i* の文字をマウスカーソルが指すエリアを表す。(このエリアを bounding box と言う。) 文字の一部が隠れている場合にはこの矩形は見えている部分を表す。文字が完全に隠れている場合には空の None を返す。

*t* をテキストウィジェット、*i* をテキストインデックスとする時、*t*.bbox(*i*) が返す矩形エリアの中の点を $(x, y)$ とする時、*t*.index("@*x*,*y*") は *i* を返す。

□ get(*i*)

引数 *i*: テキストインデックス

戻り値: 文字
機能: この関数は *i* の位置にある文字を返す。

□ get(*i*, *j*)
引数 *i,j*: テキストインデックス
戻り値: 文字列
機能: この関数は範囲 (*i*, *j*) の間にある文字列を返す。
注意: *j* が *i* の後にない場合、または *i* が END の前にない場合には長さ 0 の文字列が返される。また *i* と *j* の間にウィジェットが存在する場合には、返されるのはその間に存在する文字列だけであり、間に存在するウィジェットに関する情報は得られない。

# 3 END の問題

## 3.1 insert メソッドと get メソッドの関係

ここでは insert メソッドによってテキストウィジェットに挿入されたテキストと get メソッドによってテキストウィジェットから取り出されるテキストの関係を議論する。そしてこれらの 2 つの間には微妙な違いが存在する。譜 3 にその違いを示す例題を示す。

**譜 3** end1.py

```
from Tkinter import *

t = Text()
t.pack()

t.insert(END, "Alice\nBob")
print t.get('1.0',END)+"Carol"

mainloop()
```

このプログラムではテキストウィジェット t に文字列 "Alice\nBob" を挿入し、それを get メソッドで取り出し、その結果を "Carol" と繋ぎ合わせている。(\n は改行コードである事を思い起こそう。) 素朴に考えると、この場合には繋ぎ合わせた結果は "Alice\nBobCarol" で print による出力として

```
Alice
BobCarol
```

を期待したくなるのだが、実際には

```
Alice
Bob
Carol
```

が表示される。この事は t.get('1.0', END) が "Alice\nBob\n" を返した事を意味している。譜 4 はこの問題をさらに極端な形で示している。

このプログラムは

```
------
```

**譜 4** end2.py

```
from Tkinter import *

t = Text()
t.pack()

print "---"+t.get('1.0',END)+"---"

mainloop()
```

ではなく

```
---
---
```

をコンソールに出力する。即ちテキストウィジェットには最初から 1 個の改行コードが含まれているのである。

## 3.2 delete メソッドの動作

Tkinter のテキストウィジェットでは insert メソッドによって明示的に挿入されなかった改行コードが常に END の直前に存在する。そしてこの事が 文字列の挿入 (insert メソッド) と文字列の削除 ( delete メソッド) の動作の説明を複雑にする。

$t$ をテキストウィジェット、$i$ をテキストインデックス、$s$ を文字列とする時 $t$.insert($i$,$s$) は $i$ の位置の文字の直前に文字列 $s$ を挿入する。しかし $i$ が END の場合には文字列 $s$ は END の直前の改行コードの直前に挿入される。

他方 $t$ をテキストウィジェット、$i$ と $j$ をテキストインデックスとする時 $t$.delete($i$, $j$) は $i$ の位置の文字から $j$ の位置の文字の直前までの文字列 を削除する。しかし $j$ が END の場合には $j$ の直前の改行コードは削除されない。

## 3.3 問題の起源

Python のテキストウィジェットが insert メソッドによって挿入されない改行コードをテキストの末尾に含むのは、テキストは行の集まりであるとモデル化しているからである[9]。このモデルでは改行コードで終了しない行の存在を許さない。そのような行はテキストの末尾に発生し得るので END の直前に機械的に改行コードを付加しているのである。

[9]テキストの構造の捕らえ方について 2 つの考えがテキストエディタの中にも存在する。UNIX を例に採ると vi はテキストを行の集まりであると考えるが emacs は文字の集まりであると考えている。

# 4　テキストエディタの作成

ここではテキストウィジェットの編集に関するメソッドを紹介する。指定された位置に文字列を挿入する事、指定された範囲の文字列を消去する事ができればテキストの編集は可能である。文字列の挿入は `insert` メソッドによって、削除は `delete` メソッドによって可能である。これらのメソッドの詳細はこの節の最後に個別に解説されているので必要ならばそれを参照して頂きたい。

## 4.1　仕様

まず最初にテキストエディタでポピュラーな カットアンドペースト (Cut&Paste ) を実現するプログラムサンプルを採りあげる。このプログラムは以下の事を実現する。

1. メニューバーに Edit を表示し、これが選択されると cut, paste, copy の 3 つの選択欄を持つサブメニューが表示されるようにする事。
2. テキスト中の文字列を選択状態にし、サブメニュー cut を選択すると、選択された文字列が切り取られ、ペーストバッファに退避される事。
3. テキスト中の文字列を選択状態にし、サブメニュー copy を選択すると、選択された文字列がペーストバッファにコピーされる事。
4. サブメニュー paste を選択すると、ペーストバッファの文字列が入力カーソルの位置に挿入される事。テキスト中の文字列が選択状態になっている時に paste を選択した場合には選択状態の文字列が削除され、ペーストバッファの文字列に置き換えられる事。いずれの場合にもペーストバッファの文字列は変更されない事。
5. 初期状態で入力カーソルが見えるようにし、入力カーソルの位置はテキストの先頭位置とする事。

なお、編集のエッセンスだけを示すために、以下のようにプログラムを簡略化する。

A. 初期テキストはサンプルをプログラムの中で与える。ファイルから読み取るようにする事は容易であるが、ここでのテーマから外れるのでこのプログラムの中では扱わない。
B. 大きなテキストを扱うための縦方向のスクロールバーも取り付けない。取り付ける事も容易であるがこのプログラムの中では扱わない。
C. システムペーストバッファとのインターフェースは扱わない[10]。
D. 日本語の文字を扱わない[11]。

以上の A,B,C,D の問題に関しては後に議論する事とする。

[10] ショートカットキーを使ったシステムペーストバッファとのインターフェースは Python 2.3 では自動的に備わっている。
[11] Python 2.3 では日本語の文字を扱える。これに関してはプログラムの解説の中でコメントする。

## 4.2 初期画面

図 4 にプログラムを実行した時の初期画面とそのサブメニューを示す。この図で Edit と表示されているのはメニューバーの選択項目である。この選択項目をマウスで選択するとサブメニューが現れ、その中に 3 つの項目、cut, paste, copy が表示されている。またサブメニューの中の破線をクリックするとサブメニューが切り離される。


図 4: 譜 5 のプログラムの初期画面とサブメニュー

## 4.3 プログラムコード

譜 5 にプログラムコードを載せる。このコードには重要な行にはコメントを載せてあるのでプログラムコードを読む時の助けになるであろう。読者が実験をするときには、もちろん、これらのコメント部分を捨てても構わない[12]。

## 4.4 プログラムの解説

このプログラムの

```
w=Tk()
```

から

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

まではメニューを定義している[13]。

入力カーソルの位置制御は

```
t.mark_set(INSERT,'1.0')
```

で行っている。'1.0' が入力カーソルを位置付ける位置を示すテキストインデックスである。もしもこの文が無いなら、入力カーソルの初期位置はテキストの末尾になっているであろう。

プログラムを実行した時に入力カーソルがチカチカ光って見えるのは

```
t.focus()
```

のお陰である。これによって直ちにキーを打ち込める状態になっている。

選択状態の文字列の範囲は

[12] 日本語のコメントを添える場合には文字コードを指定する必要がある。SHIFT-JIS の場合には最初の行に

```
# coding:  shift-jis
```

と書けばよい。

[13] この部分に興味のある読者は詳しい解説が文献 [14] にあるのでそれを参照して頂きたい。

**譜 5** Cut&Paste を行うプログラム edit1.py

```
from Tkinter import *
pastebuf=None  # ペーストバッファを空にする
def cut():  # cut を選択した時に実行される関数
    global pastebuf
    r=t.tag_ranges(SEL)  # 選択状態の文字列の範囲を調べる
    if r != ():          # 選択状態が存在しない時...
        pastebuf=t.get(r[0],r[1]) # ペーストバッファに選択状態の文字列をコピーする
        t.delete(r[0],r[1])       # 選択状態の文字列を削除する
def paste():  # paste を選択した時に実行される関数
    if pastebuf == None: return   # ペーストバッファが空の時は何もしない
    r=t.tag_ranges(SEL)           # 選択状態の文字列の範囲を調べる
    if r != ():                   # 選択状態が存在しない時...
        t.delete(r[0],r[1])       # 選択状態の文字列を削除する
    t.insert(t.index(INSERT),pastebuf)  # ペーストバッファの文字列を挿入する
def copy():  # copy を選択した時に実行される関数
    global pastebuf
    r=t.tag_ranges(SEL)  # 選択状態の文字列の範囲を調べる
    if r != ():          # 選択状態が存在しない時...
        pastebuf=t.get(r[0],r[1]) # ペーストバッファに選択状態の文字列をコピーする
w=Tk()
Frame(w).pack()
m1=Menu(w)
m1.add_command(label="cut",command=cut)     # Edit のサブメニューに "cut" を追加する
m1.add_command(label="paste",command=paste) # Edit のサブメニューに "paste" を追加する
m1.add_command(label="copy",command=copy)   # Edit のサブメニューに "copy" を追加する
m=Menu(w,type="tearoff")                  # Edit のサブメニューを切り離せるようにする
m.add_cascade(label='Edit',menu=m1)       # メニューバーに "Edit" を追加する
w.configure(menu=m)
t=Text(w,width=40, height=5)
t.pack()
t.insert(END,'Alice\nBob\nCarol')   # サンプルテキストの挿入
t.mark_set(INSERT,'1.0')  # 入力カーソルの初期位置をテキストの先頭位置にする
t.focus()                 # 入力フォーカスを t に設定する
mainloop()
```

`r=t.tag_ranges(SEL)`

によって判明する。選択状態の文字列が存在しない場合には r は空のタプルになっている。存在する場合には r は 2 つのテキストインデックスから構成されるタプルであり、r[2] は r[1] の後にある。

テキスト中の選択範囲の文字列は r[0]、r[1] をテキストインデックスとするとき

`t.get(r[0],r[1])`

によって得られる。( get メソッドに関しては第 2 節に解説済である。) 文字列の削除は

`t.delete(r[0],r[1])`

で実現している。r[0] と r[1] は選択状態の文字列の範囲であるから、これによって選択状態にある文字列が削除される。

## 4.5 機能の追加

実際にテキストエディタを作る場合にはファイルの入出力ルーチンを持つ必要がある。Windows 用の Python の配布ファイルの Lib/lib-tk/ の中 にはディレクトリの一覧を表示し、ファイルからの読み取りとファイルへの書き込みをサポートするモジュールとして FileDialog.py と tkFileDialog.py が添えられている。後者は Windows の「メモ帳」とそっくりなダイアログボックスを表示する。

テキストウィジェットにスクロールバーを追加するには 1 行目の

```
from Tkinter import *
```

を

```
from ScrolledText import *
```

に置き換え、そして

```
t=Text(w,width=40, height=5)
```

を

```
t=ScrolledText(w,width=40, height=5)
```

に置き換えればよい。スクロールバー付きのテキストウィジェットは、やはり配布ファイルの Lib/lib-tk/ の中の ScrolledText.py で定義されている。

残念ながらここで扱ったプログラムはせっかく Cut&Paste のメニューを持ちながら、このメニューはシステムペーストバッファ(クリップボード) とのインターフェースを持っていない[14]。この問題に関してはまた別の機会に扱いたい。

日本語が扱えるようにするためには文字列をユニコード文字列にする必要がある。プログラム中の

```
t.insert(END,'Alice\nBob\nCarol')
```

を

```
t.insert(END,u'Alice\nBob\n ボブ\nCarol')
```

と置き換えるとこのプログラムは日本語も扱える事が分かる[15]。

## 4.6 編集に関するメソッド

□ insert(*i*,*s*)

引数 *i*: テキストインデックス

引数 *s*: 文字列

戻り値: None

機能: テキストインデックス *i* の直前に文字列 *s* を挿入する。但し *i* が END の場合には、END の直前の改行コードの直前に s を挿入する。

---

[14]ショートカットキーによるインターフェースは持っている。

[15]この他に、プログラムの第一行目に

```
# coding:  shift-jis
```

のようにソースプログラムで使用されている文字コードを書く必要がある (Python 2.2)。

□ `insert(`$i$`,`$s_1$`,`$t_1$`,`$s_2$`,`$t_2$`,...)`

引数 $i$: テキストインデックス
引数 $s_1$,$s_2$,... : 文字列
引数 $t_1$,$t_2$,... : タグ
戻り値: `None`
機能: テキストインデックス i の直前に文字列 $s_1$,$s_2$,... を挿入し、かつそれらのタグを $t_1$,$t_2$, ... とする。但し $i$ が `END` の場合には、`END` の直前の改行コードの直前に $s_1$,$s_2$,... を挿入する。

□ `delete(`$i$`,`$j$`)`

引数 $i$,$j$: テキストインデックス
戻り値: `None`
機能: テキス範囲 ($i$, $j$) の文字列 を削除する。但し $j$ が `END` の場合には、`END` の直前の改行コードは削除されない。
注意: $j$ が $i$ の後にない場合には削除されない。$i$ が `END` の前にない場合も同様である。

# 5 文字列の探索

## 5.1 基本仕様

ここではテキストエディタを作成する時に発生する問題のうち、テキスト中の文字列の探索を行う `search` メソッドを含むプログラム例題を取り扱う。このプログラムでは以下の基本仕様を実現するものとする。

#### ウィンドウの構成

1. 主ウィンドウはメニューバーとテキストウィジェットで構成される。
2. メニューバーに ”Search” を置く。
3. ”Search” を選択するとサブウィンドウが生成される。
4. サブウィンドウは探索文字列の入力欄と OK ボタンから構成される。

#### 探索の方法

1. 単純な探索、即ち正規表現を使用しない探索を行う。
2. 探索にあたっては英字の大文字と小文字を区別しない。
3. 探索はテキストの下方に向かって行われる。

#### 保護

1. 文字列探索のサブウィンドウを多重に生成しない。

**告知**

1. 探索に失敗した時には、その旨を利用者に告知する。
2. 探索文字列を指定しないで探索を行った場合にも告知を行う。

**初期状態**

1. 初期状態で入力カーソルが見えるようにする。
2. 入力カーソルの初期位置をテキストの先頭位置とする。

**制限事項**　なお、編集のエッセンスだけを示すために、以下のようにプログラムを簡略化する。

A. 初期テキストはサンプルをプログラムの中で与える。ファイルから読み取るようにする事は容易であるが、ここでのテーマから外れるのでこのプログラムの中では扱わない。

B. 大きなテキストを扱うための縦方向のスクロールバーも取り付けない。取り付ける事も容易であるがこのプログラムの中では扱わない。

## 5.2　初期画面

図 5 にプログラムを実行した時の初期画面とそのサブウィンドウを示す。この図で “Search” と表示されているのはメニューバーの選択項目である。この選択項目をマウスで選択するとサブウィンドウが現れ、その中に文字列の入力欄と OK ボタンが置かれている。このウィンドウの入力欄に探索すべき文字列を入れ、OK ボタンを押すと、その文字列の探索を開始し、もしも探索に成功すれば見つけた文字列を選択状態 (背景が青色で示される) にする。さらに OK ボタンを押すと、選択された次の文字から探索する。探索に失敗するとダイアログウィンドウが開き、ユーザに注意を喚起する。


図 5: 譜**??**のプログラムの初期画面とサブウィンドウ

## 5.3　プログラムコード

譜 6 にプログラムコードを載せる。プログラムの解説において、読者は第 4 節の譜 5 の解説を既に読んでいると想定されている。従って、この解説と重複する部分は解説しない。

**譜 6** 文字列探索を行うプログラム search.py

```
from Tkinter import *
from Dialog import Dialog
e=None
def clean(event): # 探索ウィンドウが消去された時に実行される関数
    global e
    if e != None: e=None
def search(): # 探索を開始する
    pattern=e.get()   # 探索文字列を入力欄から読み取る
    if pattern == '': # 探索文字列の指定がない時は...
        Dialog(title='sample', text='no string is given',
        bitmap='', default=0, strings=('OK',))
        return
    s=t.tag_ranges(SEL)   # 選択文字列の範囲を調べる
    if s != (): t.tag_remove(SEL,s[0],s[1]) # 選択文字列があれば選択状態を解除する
    i=t.search(pattern,t.index(INSERT),END, nocase=YES, count='size') # 探索する
    if i == '':  # 探索に失敗した場合には...
        Dialog(title='sample', text='not found the string',
        bitmap='',default=0, strings=('OK',))
    else:
        m=t.getvar('size') # マッチした文字列のサイズを調べる
        j="%s + %s chars"%(i,m) # 見つけた文字列の末尾の位置を j とする
        t.tag_add(SEL,i,j)    # 見つけた文字列を選択状態にする
        t.mark_set(INSERT,j)  # 見つけた文字列の末尾に入力カーソルを設定する
        t.focus()             # 入力フォーカスを t に設定する
def cmd():  # メニューの search が選択された時に実行される関数
    global e
    if e != None: return # 探索文字列入力ウィンドウを多重に生成しない
    w1=Toplevel()     # 新しいウィンドウを生成する
    w1.title("search") # その標題
    w1.bind("<Destroy>",clean) # 探索ウィンドウが消去された時に clean を実行させる
    e=Entry(w1)
    e.pack()
    Button(w1,text="OK",command=search).pack() # OK ボタンで関数 search を実行する
Button(text='Search',command=cmd,relief='flat').pack(anchor='w')
t=Text(width=40, height=5)
t.pack()
t.insert(END,'Alice\nBob\nCarol')   # サンプルテキストの挿入
t.mark_set(INSERT,'1.0')  # 入力カーソルの初期位置をテキストの先頭位置にする
t.focus()                 # 入力フォーカスを t に設定する
mainloop()
```

## 5.4 プログラムの解説

メニューバーの ”Search” (実はこれはボタンである) を選択すると cmd が実行される。この関数の中で

```
w1=Toplevel()
```

が実行されるとウィンドウが新たに生成される[16]。このウィンドウの中には

```
e=Entry(w1)
e.pack()
```

[16] ウィンドウの生成に関しては例えば文献 [14] を見よ。

によって入力欄が、また

```
Button(w1,text="OK",command=search).pack()
```

によって OK ボタンが張りつけられる。そして OK ボタンをクリックすると関数 search に飛ぶ仕組みになっている。

関数 search が実行されると

```
pattern=e.get()
```

で探索文字列入力欄の文字列が読み取られる。

次に

```
s=t.tag_ranges(SEL)
```

でテキストウィジェットの中の選択文字列の範囲を調べている。もし選択文字列が存在すれば

```
t.tag_remove(SEL,s[0],s[1])
```

で選択状態を解除する。そして文字列を

```
i=t.search(pattern,t.index(INSERT),END, nocase=YES, count='size')
```

で探索する。ここでは入力カーソルの位置から END までを探索の範囲としている。ここに現れる search メソッドのオプション count='size' は見つかったテキスト中の文字列の長さを知りたい場合に指定する。この長さは

```
m=t.getvar('size')
```

によって得られる。但し返された値は整数型ではなく文字型である事に注意する。

ここでは count に対し 'size' を指定したが、この文字列は何でも構わない。単に count で指定した文字列と getvar の引数の文字列が一致していればよい。単純な探索 (正規表現でない探索) では pattern で指定された文字列の長さと、見つかった文字列の長さは必ず一致する。従って count オプションを指定する必要はないが、正規表現による探索への応用のためにここでは敢えて count オプションを使用している。

文字列が見つかった場合にはそれを選択状態にする必要がある。そのためには見つかった文字列の先頭と末尾のテキストインデックスを指定する必要がある。末尾のテキストインデックスは、先頭のテキストインデックス i を基にして chars による修飾子を付けて表現できる。即ち、

```
j="%s + %s chars"%(i,m)
t.tag_add(SEL,i,j)
```

そして最後に入力カーソルを再設定し、入力フォーカスを設定すれば 1 つの探索は終了する。

さて、このプログラムでは探索用のサブウィンドウが多重に生成されないようにガードが掛けられている。即ち、既に探索用のサブウィンドウが生成されていれば

```
w1=Toplevel()
...
```

を実行してはならない。そのために大域変数である e を利用して、探索用のサブウィンドウが生成されているか否かを判定している。これが

```
if e != None: return
```

が置かれている理由である。ところが探索用のサブウィンドウがマウスの操作によって消されてしまう事があり得る。消された場合には e=None が実行されないと、一旦消されたが最後、探索サブウィンドウが二度と生成できない事になる。そこでこのプログラムでは探索サブウィンドウが消された時に関数 clean が実行されるように、

```
w1.bind("<Destroy>",clean)
```

を実行している。

## 5.5 探索に関するメソッド

□ search(*s*,*i*)　search(*s*,*i*,*j*)

引数 *s*: 探索文字列

引数 *i*,*j*: テキストインデックス

戻り値: テキストインデックス または 長さ 0 の文字列

機能: *j* が与えられた第 2 の形式では文字列 *s* をテキストインデックス *i* の文字から、*j* の直前の文字までの範囲で探索する。*j* が省かれている第 1 の形式では、文字列 *s* をテキストインデックス *i* から探索し、そこに存在しない場合にはさらにテキストの先頭から *i* までを探索する。

search は以下のオプションを許す。( ) 内にはこれらのオプションの省略時の指定値を示す。

| | | |
|---|---|---|
| backwards | 後方探索 | (NO) |
| regexp | 正規表現によるマッチング | (NO) |
| nocase | 大文字と小文字の違いを無視する | (NO) |
| count | マッチした文字数を記憶する変数名 (文字列で指定する) | (None) |

regexp = YES を指定した場合には、マッチした文字列の長さは探索文字列 *s* の長さには必ずしも一致しない。マッチした文字列の実際の長さを知るには count='v' を指定し ( 'v' は任意の文字列で構わない)、 *t*.getvar('v') の値を調べる。(ここに *t* をテキストウィジェットとする。) このメソッドは実際にマッチした文字列の長さが、文字列の形式で返される。もしもマッチした文字列が存在しない時には、長さが 0 の文字列が返される。

以上の他に以下のオプションが存在するが、これらの省略時の指定値は YES なので、実際に使用される事はないであろう。

| | | |
|---|---|---|
| forwards | 前方参照 | (YES) |
| exact | 完全マッチング | (YES) |

# 6 タグ

## 6.1 タグの機能

テキストウィジェットのタグは、テキストウィジェットの中の文字列に付ける名札の事である。この名札は文字列の属性 (色や大きさやフォントなど) やイベントを指定するのに使用される。明示的にタグが指定されない場合には文字属性は Text ウィジェットの生成子のオプションによって定まる属性が採用される。Text ウィジェットの生成子のオプションでも明示的に指定されない属性はシステムの標準的な属性が採用される。

## 6.2 タグの優先順序

文字列に複数のタグが付けられた場合には、優先順序に従って文字列の属性が決定される。一般に後から付けられたタグの方が優先順位が高いが、タグの優先順位は tag_lower メソッドにより変更可能である。

## 6.3 タグの名前規則

タグに対してはかなり自由に名前を付けることができる。英数字、記号、空白で構成される任意の文字列がタグ名としての資格を持っている。

## 6.4 定義済タグ名

タグ名 SEL は定義済で、このタグ名は消去する事ができない。このタグはマウスの左ボタンをドラッグして文字列を選択した時の選択範囲を意味する。(選択範囲は背景が青色になる。)

## 6.5 タグで指定可能なオプション

文字列の属性は tag_configure メソッドのオプションで指定する。指定可能なオプションは以下の通りである (文献 [17])。

background, bgstipple, borderwidth, fgstipple, font, forground, justify
lmargine1, lmargine2, offset, overstrike, relief, rmargine, spacing1
spacing2, spacing3, tabs, underline, wrap

これらの多くは Text ウィジェットの生成子のオプションと、意味と指定の方法を共有している。それらについては第 5 節「Text で指定可能なオプションの一覧」を見ればたりる。従ってここではタグに特有なオプションについてのみ解説する。各々のオプションについての省略時の値が書かれていない事に注意しよう。省略時にはオプションは指定されなかったと看做され、次の優先順位のタグのオプションで属性が決定されるからである。

□ lmargin1
　　値: 非負整数

意味: 段落の出だしの左マージンをピクセル単位で指定する。段落の先頭の文字に対して指定する。

□ lmargin2
値: 非負整数
意味: 段落の 2 行目以降の行の左マージンをピクセル単位で指定する。段落の 2 行目の先頭の文字に対して指定する。

□ rmargin
値: 非負整数
意味: 段落の右マージンをピクセル単位で指定する。段落の先頭の文字に対して指定する。

□ justify
値: 'center', 'left', 'right'
意味: 行の中での文字列の寄せ方を指定する。行の先頭の文字に対して指定する。

□ offset
値: 整数
意味: ベースラインからのオフセットをピクセル単位で指定する。上付き文字、下付き文字を定義するのに有用である。

□ underline
値: YES, NO
意味: 文字列に対して下線を引くか否かを指定する。

□ oversrike
値: YES, NO
意味: 文字列に対して消去線を引くか否かを指定する。

□ bgstipple
値: 文字列
意味: stipple のパターンを指定する。パターンの名称は以下の通りである。'gray12', 'gray25', 'gray50', 'gray75'

□ fgstipple
値: 文字列
意味: stipple のパターンを指定する。パターンの名称は以下の通りである。
'gray12', 'gray25', 'gray50', 'gray75'
注意: Windows では機能していない。

## 6.6 タグを使用したプログラム例

既に第 1 節の譜 1 で tag_configure メソッドが、第 4 節の譜 5 で tag_ranges メソッドが、第 5 節の譜 6 で tag_remove と tag_add メソッドが使用されている。ここではタグを研究するために筆者が使用したプログラムを紹介する (図 6 および譜 7)。このプログラムでは、タグを自由に作成し、テキストの任意の文字列に張りつけ、またそれらのタグを文字列から削除し、あるいはタグの定義そのものを抹消できるようになっている。読者はプログラムを実行してみて、そうした操作の効果がテキストウィジェットにどのように反映されるかを観察するのがよい。

操作の説明譜 7 のプログラムを実行すると図 6 に示した左上のウィンドウ (ウィンドウの標題が Tk となっている) が現れる。但しテキストウィジェット中のテキストには字飾りはなく、

Alice
Bob
Carol

となっている。メニューバーには configure, add, delete, remove の 4 つのメニューが置かれている。これらをマウスで選択すると図の 4 つのサブウィンドウが表示され、各々の標題は configure, add, delete, remove となっている。(但し、表示される内容は図と同じではない。)

タグの定義は configure サブウィンドウで行う。tagname の蘭には必ずタグ名を書く。名前は自由に付ける事ができる。他の蘭は必要に応じて書けばよい。ここではタグ名として "title" を定義している。OK ボタンを押せばここに定義されたタグ名が有効になる。サブウィンドウの add, delete, remove には定義されたタグの一覧が載っている。タグを新たに定義した時に、この一覧が自動的に更新されるわけではない。一旦サブウィンドウを削除し、再度メインウィンドウ Tk のメニューを選択しサブウィンドウを生成すれば configure による更新が一覧に反映される。

文字列にタグを与えるには、対象とする文字列をマウスで選択し (マウスの左ボタンでドラッグすればよい)、サブウィンドウ add の一覧の中からタグ名をマウスの左ボタンでクリックする。図 6 の Tk はテキスト中の alice を選択し、次にサブウィンドウ add の一覧の中の title をマウスの左ボタンでクリックした結果である。

タグの定義を抹消するにはサブウィンドウ delete の一覧の中から抹消するタグ名をマウスの左ボタンで選択すればよい。

文字列に割り付けられたタグを消去するには文字列を選択して、サブウィンドウ remove の中のタグ名をマウスの左ボタンでクリックする。このプログラムでは一部の属性しか扱っていない。譜 7 のプログラムには容易に他の属性を追加できるので、必要とあれば読者の方で追加して頂きたい。

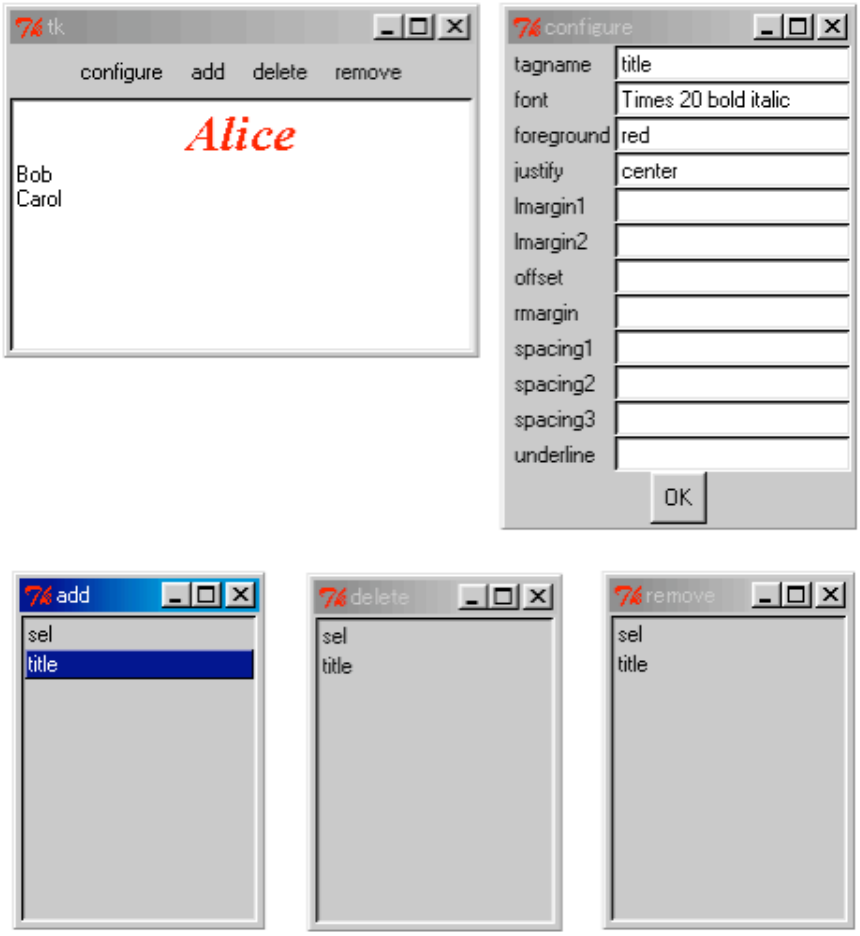



図 6: ウィンドウ

図 7: 譜 7 の初期画面とサブウィンドウ

**譜 7** タグの実験プログラム (次頁へ続く)

```
\small
from Tkinter import *
from Dialog import Dialog
opts={'foreground':None,'font':None,'underline':None,'rmargin':None,
    'lmargin1':None,'lmargin2':None,'justify':None,'offset':None,
    'spacing1':None,'spacing2':None,'spacing3':None}  # 属性の一覧
e=w1=w2=w3=w4=None
def warning(s):
    Dialog(title='warning',text=s,bitmap='',default=0,strings=('OK',))
def listbox(title,destroy,button1): # Listbox を備えたウィンドウの生成
    w=Toplevel()  # win: 親ウィンドウ
    w.title(title)   # title: ウィンドウの標題
    w.bind("<Destroy>",destroy) # destroy: 消滅時に実行される関数
    b = Listbox(w)
    b.pack()
    i = 0
    for n in t.tag_names(): # タグ名から一覧を構成する
        b.insert(i,n)
        i = i+1
    b.bind("<Button-1>",button1) # button1: 左ボタンを押した時に実行される関数
    return (w,b)
def add1(event): # add サブウィンドウの項目が選択された時に実行される関数
    tagname=b2.get(int(b2.index("@%d,%d"%(event.x,event.y))))
    s=t.tag_ranges(SEL)
    if s == None: warning('select a range'); return
    t.tag_add(tagname,s[0],s[1])
def delete1(event): # delete サブウィンドウの項目が選択された時に実行される関数
    tagname=b3.get(int(b3.index("@%d,%d"%(event.x,event.y))))
    t.tag_delete(tagname)
def remove1(event): # remove サブウィンドウの項目が選択された時に実行される関数
    tagname=b4.get(int(b4.index("@%d,%d"%(event.x,event.y))))
    s=t.tag_ranges(SEL)
    if s == None: warning('select a range'); return
    t.tag_remove(tagname,s[0],s[1])
def configure1(): # configure サブウィンドウの項目が選択された時に実行される関数
    tagname=e.get()
    if tagname == '': warning('tagname is required'); return
    op=opts.copy()
    for s in opts.keys():
        op[s]=opts[s].get()
    t.tag_configure(tagname,cnf=op)
```

**譜 8** タグの実験プログラム (前頁から続く)

```
def f(w,s):
    g=Frame(w)
    g.pack(expand=YES,fill=X)
    Label(g,text=s).pack(side=LEFT)
    e=Entry(g)
    e.pack(side=RIGHT)
    return e
def configure(event=None): # configure メニューが選択された時に実行される関数
    global w1,e,opts
    if event: w1=None;return
    if w1 != None: return
    w1=Toplevel()
    w1.title("configure")
    w1.bind("<Destroy>",configure)
    e=f(w1,"tagname")
    op=opts.keys()
    op.sort()
    for s in op:
        opts[s]=f(w1,s)
    Button(w1,text="OK",command=configure1).pack()
def add(event=None): # add メニューが選択された時に実行される関数
    global w2,b2
    if event: w2=None; return
    if w2 != None: return
    (w2,b2)=listbox("add",add,add1)
def delete(event=None): # delete メニューが選択された時に実行される関数
    global w3,b3
    if event: w3=None; return
    if w3 != None: return
    (w3,b3)=listbox("delete",delete,delete1)
def remove(event=None): # remove メニューが選択された時に実行される関数
    global w4,b4
    if event: w4=None; return
    if w4 != None: return
    (w4,b4)=listbox("remove",remove,remove1)
w=Frame()
w.pack()
Button(w,text='configure',command=configure,relief='flat').pack(side=LEFT)
Button(w,text='add',command=add,relief='flat').pack(side=LEFT)
Button(w,text='delete',command=delete,relief='flat').pack(side=LEFT)
Button(w,text='remove',command=remove,relief='flat').pack(side=LEFT)
t=Text(width=40,height=10)
t.pack()
t.insert(END,'Alice\nBob\nCarol')
t.mark_set(INSERT,'1.0')
t.focus()
mainloop()
```

## 6.7 プログラムの解説

今回のプログラム (譜 7) は 1 ページには納まらない程大きくなったのでプログラムは 2 つのページに渡って載せてある。さらにプログラムサイズを縮小するための技巧が含まれているので、プログラムは幾分分かりにくくなっている。

重要な技巧は、メインウィンドウのメニュー (これらは実際にはメニューである) が選択された時に実行される関数と、その時に生成されたサブウィンドウが消去された時に実行される関数とが同じになっている事である。

例えばメニューバーの configure を選択した時には関数 configure が実行される事が譜 7 の中の

```
Button(w,text='configure',command=configure,relief='flat').pack(side=LEFT)
```

で指定されているが、同じ関数が、関数 configure の定義文の中の

```
 w1.bind("<Destroy>",configure)
```

でも指定されている。

前者によって関数 configure が実行される時には configure には引数は渡されないが、後者による場合にはイベントを渡す 1 個の引数が渡される。この性質を利用して関数 configure はどちらの側からの実行であるかを判断している。即ち、関数 configure の定義文に現れる

```
    if event: w1=None;return
```

がそのために存在する。関数 add, delete, remove についても同様である。

メニューバーの add, delete, remove を選択した時に生成されるサブウィンドウはどれも同じである。そこでここでは関数 listbox を定義してこの部分を共通化している。Listbox は通常は OK ボタンと共に使用されるが、ここでは OK ボタンを使用しないで項目が選択されるといきなりアクションを起こすように設計されている。この問題ではこの仕様は便利なだけではなく、OK ボタンが適さない事による。即ち、OK ボタンによってアクションを起こすようにした場合には Listbox 中の項目を選択した段階でテキストウィジェットの選択項目が利用者からは消えてしまったかのように見える。この事は利用者に戸惑いをもたらすので採用する訳にはいかないのである。

項目を選択した時に Listbox がアクションを起こすようにするには、Listbox をマウスの左ボタンにバインドする。このことは関数 listbox の定義文の中の

```
b.bind("<Button-1>",button1)
```

によって行われている。ここに button1 は関数 listbox の第 3 引数である。この関数は例えば関数 add の中で次のように使用されている。

```
(w2,b2)=listbox("add",add,add1)
```

この場合には add サブウィンドウの項目を選択した場合には関数 add1 が実行される。関数 add1 では

```
tagname=b2.get(int(b2.index("@%d,%d"%(event.x,event.y))))
```

マウスカーソルの座標情報から Listbox のインデックスを構成し、選択されたタグ名を見つけているのである。

最後に関数 configure1 の定義文の中の

t.tag_configure(tagname,cnf=op)

に注目しよう。このプログラムは属性の追加に対して柔軟な構造を持っている。即ち譜 7 の 3 行目の opts に与えた属性の集合を変更するだけでオプションを追加できる。もしも tag_configure に対して通常の方法、即ち、

オプション名=オプションの値

の形式でオプションを与えたらそのような訳にはいかなかったであろう。オプションの与え方に柔軟性を持たせるために実は tag_configure はオプションを集合を通じて渡せるような隠し仕様を持っているのである (注 1)。

> 注 1: tag_configure だけではなく、オプションを許す Tkinter のどのメソッドも cnf でオプション集合を渡せるようにできている。

## 6.8 タグに関するメソッド

□ tag_configure(*t*,*o*=*v*,...)

引数 *t*: タグ名 (文字列)
引数 *o*=*v*: オプション
戻り値: None
機能: タグを定義する。
オプションは *o*=*v* の形式で指定する。ここに *o* はオプション名、 *v* はオプションの値で、*v* は文字列で指定する。

□ tag_configure(*t*,*o*)

引数 *t*: タグ名 (文字列)
引数 *o*: オプション名 (文字列)
戻り値: 5 個の文字列のタプル
機能: タグ *t* のオプションの値を知る。

オプションは文字列で指定する。戻り値を r とすると r[0] はオプションの名称、r[4] はタグ *t* に割りつけられたオプション *o* の値となっている。r[1],r[2],r[3] は常に長さ 0 の文字列になっているようである。

□ tag_bind(*t*)

引数 *t*: タグ名 (文字列)
戻り値: 文字列のタプル
機能: タグ *t* にバインドされている全てのイベント名の一覧を返す。

□ tag_bind(*t*,*e*,*f*)

引数 *t*: タグ名 (文字列)
引数 *e*: イベント名 (文字列)

引数 *f*: 関数
戻り値: None
機能: タグ *t* にイベントを割り付ける。

イベント名は例えば
"<Buttun-3>" — マウスの第 3 ボタン (マウスの右ボタン) が押された
"<KeyPress>" — キーが押された
"<KeyPress-a>" — キー a が押された
"<KeyRelease-Escape>" — Escape キーが離された
のように指定する。

イベント名に関する詳細の解説はここでのテーマではないので、必要ならば他の文献 (例えば文献 [6]、文献 [7]、文献 [8]) を参照せよ。
関数 f はイベントが発生した時に実行される関数名である。

□ tag_unbind(*t*,*e*)
引数 *t*: タグ名 (文字列)
引数 *e*: イベント名 (文字列)
戻り値: None
機能: タグ *t* にバインドされたイベント *e* を解除する。

□ tag_cget(*t*,*o*)
引数 *t*: タグ名 (文字列)
引数 *o*: オプション名 (文字列)
戻り値: 文字列のタプル
機能: タグ *t* のオプションの値を知る。

□ tag_delete($t_1$,$t_2$,...)
引数 $t_1$, $t_2$, ...: タグ名 (文字列)
戻り値: None
機能: $t_1$, $t_2$, ... はタグ名である。これらのタグに関する全ての情報を削除する。

□ tag_remove(*t*,*i*)
引数 *t*: タグ名 (文字列)
引数 *i*: テキストインデックス
戻り値: None
機能: テキストインデックス *i* の文字のタグ *t* を消去する。タグの定義は残る。*i* が END の前にない場合には効果を持たない。

□ tag_remove(*t*,*i*, *j*)
引数 *t*: タグ名 (文字列)
引数 *i*,*j*: テキストインデックス
戻り値: None
機能: テキスト範囲 (*i*, *j*) のタグ *t* を消去する。タグの定義は残る。*i* が END の前にない場合、または *i* が *j* の前にない場合には効果を持たない。

□ `tag_names()`

戻り値: 文字列のタプル

機能: テキストウィジェットの中の全てのタグの一覧を返す。返されたタグの一覧は優先度の低いタグから高いタグの順に整理されている。

□ `tag_names(`*i*`)`

引数 *i*: テキストインデックス

戻り値: 文字列のタプル

機能: テキストインデックス *i* に割りつけられている全てのタグの一覧を返す。返されたタグの一覧は優先度の低いタグから高いタグの順に整理されている。

□ `tag_ranges(`*t*`)`

引数 *t*: タグ名 (文字列)

戻り値: テキストインデックスから構成されるタプル

機能: タグ *t* が割り付けられている全ての文字列の範囲を返す。

範囲はテキストインデックスのペアで与えられる。即ち `r` を戻り値とすると `r[0]` と `r[1]` でタグ *t* が割り付けられた最初の文字列の範囲を、`r[2]` と `r[3]` で次の文字列の範囲を ... 返す。タグ *t* が割り付けられている文字列が存在しない場合には空のタプルが返される。

□ `tag_add(`*t*`,`*i*`)`

引数 *t*: タグ名 (文字列)

引数 *i*: テキストインデックス

戻り値: `None`

機能: テキストインデックス *i* にタグ *t* を割り付ける。*i* が `END` の前にない場合には効果を持たない。

□ `tag_add(`*t*`,`*i*`,` *j*`)`

引数 *t*: タグ名 (文字列)

引数 *i,j*: テキストインデックス

戻り値: `None`

機能: テキスト範囲 (*i,j*) の間の文字列にタグ *t* を割り付ける。 *i* が `END` の前にない場合、または *i* が *j* の前にない場合には効果を持たない。

□ `tag_lower(`*t*`)`

引数 *t*: タグ名 (文字列)

戻り値: `None`

機能: タグ *t* の優先順位を最下位にする。

□ `tag_lower(`*t*`,`$t_1$`)`

引数 *t*,$t_1$: タグ名 (文字列)

戻り値: `None`

機能: タグ *t* の優先順位をタグ $t_1$ の次にする。

□ `tag_raise(`*t*`)`

引数 *t*: タグ名 (文字列)

戻り値: None
機能: タグ $t$ の優先順位を最上位にする。

□ tag_raise($t$,$t_1$)
引数 $t$,$t_1$: タグ名 (文字列)
戻り値: None
機能: タグ $t$ の優先順位をタグ $t_1$ の 1 つ上にする。

□ tag_nextrange($t$,$i$)　tag_nextrange($t$,$i$,$j$)
引数 $t$: タグ名 (文字列)
引数 $i$,$j$: テキストインデックス
戻り値: 空のタプルまたは 2 つのテキストインデックスから構成されるタプル
機能: タグ $t$ を持つ文字列をテキストインデックス $i$ の後方に探す。

探索の限界は $j$ で与えられる。即ち、文字列の先頭位置のテキストインデックスを $x$ とするとき、$x$ は $i$ 以降でかつ $j$ の前である。$j$ が省略された形式では END までを探索する。見つかれば最初に見つかった文字列の範囲を返す。見つからない時には空のタプルを返す。

□ tag_prevrange($t$,$i$)　tag_prevrange($t$,$i$,$j$)
引数 $t$: タグ名 (文字列)
引数 $i$,$j$: テキストインデックス
戻り値: 空のタプルまたは 2 つのテキストインデックスから構成されるタプル
機能: タグ $t$ を持つ文字列をテキストインデックス $i$ の前方に探す。

探索の限界は $j$ で与えられる。(従って $j$ は $i$ の前にあるテキストインデックスである。) 即ち、文字列の先頭位置のテキストインデックスを $x$ とするとき、$x$ は $i$ 以前でかつ $j$ の後である。$j$ が省略された形式ではテキストの先頭位置までを探索する。見つかれば最初に見つかった文字列の範囲を返す。見つからない時には空のタプルを返す。

# 7 マーク

マークとはテキストウィジェットに付ける印の事である。タグと事なり、マークは文字と文字の間に付ける。またマークはテキストウィジェットの中の文字列を削除しても消える事はない。

マークは名前を持つ。名前規則はタグ名と同じである。同じ名前のマークがテキストの 2 ケ所に付けられる事はない。そしてテキストウィジェットには以下の定義済のマークが存在する。

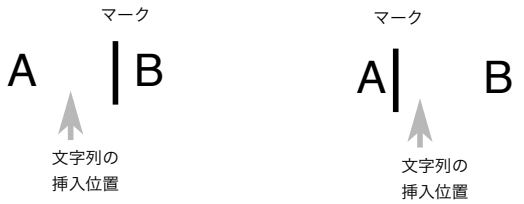
- INSERT (='insert') : このマークは入力カーソルの位置を表す。
- CURRENT (='current') :このマークはマウスカーソルが指している最寄りの文字の位置を表す。マウスボタンを押している間はこのマークは変化しない。
- ANCHOR(='anchor'): このマークはマウスをテキストウィジェット内でドラッグした時のドラッグの開始位置を表す。

従ってマークは名前と位置情報を持っているが、その他に gravity(=引力) と言う属性を持つ、この属性は LEFT または RIGHT の値を持つ。 gravity を理解するために、文字列を挿入したり削除した場合のマークの振る舞いを要約しよう。

1. マークの位置より前に文字列が挿入された時にはマークは挿入された文字数だけ後方にずれる。
2. マークの位置より前の文字列が削除された時にはマークは削除された文字数だけ前方にずれる。
3. マークの位置より後方で文字列が挿入または削除されてもマークは移動しない。

マークの位置に文字列が挿入された場合のマークの振る舞いには 2 通り考えられる。つまり、マークは挿入された文字数だけ後方にずれるのか、それとも移動しないのか? この振る舞いを指定しているのが gravity 属性である。

図 8 は隣り合う 2 つの文字 A と B の間のマークの位置を表している。左側の図に示すように、マークが右側の文字の方に寄り添っている (これを right gravity と言う) と文字列が挿入された時にマークは右方向 (後方) に移動する。右側の図ではマークが左側の文字に寄り添っている (これを left gravity と言う) と文字列が挿入されてもマークの位置は変化しない。




right gravity　　left gravity

図 8: 文字の挿入位置と gravity

マーク INSERT は right gravity である。

## 7.1 マークに関するメソッド

□ mark_names()
　戻り値: 文字列のタプル
　機能: 定義されているマーク の一覧を返す。

□ mark_gravity(*m*)
　引数 *m*: マーク名 (文字列)
　戻り値: 'right' または 'left'
　機能: マーク *m* の gravity を返す。

□ mark_gravity(*m*,*g*)
　引数 *m*: マーク名 (文字列)
　引数 *g*: 'right または 'left'

戻り値: None
機能: マーク *m* を定義する。

□ mark_set(*m*,*i*)
引数 *m*: マーク名 (文字列)
引数 *i*: テキストインデックス
戻り値: None
機能: テキストインデックス i の位置にマーク *m* を設定する。以前のマークの位置は消える。

□ mark_unset(*m*)
引数 *m*: マーク名 (文字列)
戻り値: None
機能: マークの定義を抹消する。INSERT と CURRRENT は抹消できない。

**注意** マークの現在位置は、t をテキストウィジェットとすると、
t.index(' マーク名')
で判明する。

# 8 埋め込みウィンドウ

ここではテキストウィジェットに埋め込まれたウィジェットを議論する。(以下、埋め込みウィジェットと言う。) 埋め込みウィジェットは通常のウィジェットと同じ視覚デザインを持ち、同じようにマウスに反応する。

埋め込みウィジェットの配置位置はテキストインデックスを通じて指定される。この事は埋め込みウィジェットはその配置に関しては文字と同じ扱いを受ける事を意味する。

埋め込みウィジェットは文字と同様にタグ付けが可能である。tag_configure のオプションの多くは埋め込みウィジェットに対しては意味を持たないが、マージンや行間調整や justify のような位置決めに関する指定は効力を持っている。

埋め込みウィジェットに対するタグ付けは文字列に対するタグ付けと事なり、挿入時に行う事はできないので、 tag_add メソッドを使用する。例えば行の中央にボタンを埋め込むには次のようにすればよいであろう。

```
w=Button(....)
t=Text(....)
t.{\tt tag\_configure}('ceter-tag',justify='center')
t.insert(END,".....\n")
t.window_create(END,window=w)
t.tag_add('center-tag','end - 2 chars')
```

ここに .... の部分は実際に則して適当に何かを書くべきである事を意味している。この最後の 'end - 2 chars' に着目すべきである。'end' の前には改行記号が存在するので、その 1 つ前を指定する必要があるのである。(window_create のオプションでタグを指定できればよいのだが、残念ながらそうはなっていない。)

## 8.1 埋め込み時のオプション

埋め込みウィジェットは window_create メソッドによってテキスト中に埋め込まれる。その際に指定可能なオプションは以下の通りである。これらのオプションの値は window_configure メソッドによって変更可能である。

□ window
　値: ウィジェット
　意味: 埋め込むウィジェットを指定する。

□ align
　値: 'top', 'center', 'bottom' (あるいは TOP,CENTER, BOTTOM)
　意味: 行間のどの位置に配置するかを指定する。

□ padx
　値: 非負整数
　意味: 両サイドに付ける余白の大きさをピクセル単位で指定する。

□ pady
　値: 非負整数
　意味: 上下に付ける余白の大きさをピクセル単位で指定する。

□ stretch
　値: YES,NO
　意味: 埋め込みウィジェットの高さが行の高さに満たない場合に、ウィジェットを行の高さに合わせて引き伸ばすか否かを指定する。

**注意**　他に create オプションが存在するが解説しない。

以下に padx, stretch, align オプションの効果を示すプログラムとその実行結果を示す。

**譜 9** padx, stretch, align の効果を示すプログラム

```
from Tkinter import *
t = Text(width=25, height=2, font='Times 20')
t.pack()
a1=Label(text="padx",relief=GROOVE)
a2=Label(text="stretch",relief=GROOVE)
b1=Label(text="top",relief=GROOVE)
b2=Label(text="center",relief=GROOVE)
b3=Label(text="bottom",relief=GROOVE)
t.insert(END, "Alice")
t.window_create(END,window=a1,padx=10)
t.insert(END,"Bob")
t.window_create(END,window=a2,stretch=YES)
t.window_create(END,window=b1,align='top')
t.window_create(END,window=b2,align='center')
t.window_create(END,window=b3,align='bottom')
t.insert(END,"Carol\n")
mainloop()
```

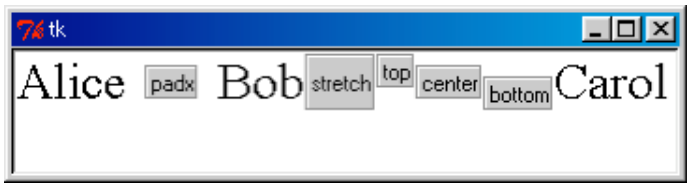



図 9: padx, stretch, align の効果

## 8.2 埋め込みウィンドウに関するメソッド

□ window_create(*i*,*o*=*v*,...)
引数 *i*: テキストインデックス
引数 *o*=*v*: オプション
戻り値: None
機能: テキストインデックス *i* の位置にウィジェットを埋め込む。埋め込むウィジェットは window オプションで指定する。指定可能なオプションに関しては「埋め込み時のオプション」を見よ。

□ window_configure(*i*,*o*)
引数 *i*: テキストインデックス
引数 *o*: オプション名 (文字列)
戻り値: 5 個の文字列のタプル
機能: 戻り値を r とすると、r[0] はオプション名、r[4] はオプションの現在値となっている。

□ window_configure(*i*,*o*=*v*,...)
引数 *i*: テキストインデックス
引数 *o*=*v*: オプション
戻り値: None
機能: テキストインデックス *i* の位置の埋め込みオプションを変更する。指定可能なオプションに関しては「埋め込み時のオプション」を見よ。

□ window_names()
戻り値: 文字列のタプル。
機能: 埋め込まれたウィジェットの一覧を返す。

□ window_cget(*i*,*o*)
引数 *i*: テキストインデックス
引数 *o*: オプション名 (文字列)
戻り値: 文字列
機能: テキストインデックス *i* に埋め込まれたウィジェットの、埋め込み時に指定可能なオプション *o* の現在値を文字列形式で返す。(この値は window_configure(*i*,*o*) が返す値と一致しているようである。)

# 9 埋め込み画像

ここではテキストウィジェットに埋め込まれた画像を議論する。(以下、埋め込み画像と言う。) 画像フォーマットは現在の所 PPM 形式、GIF 形式および BMP 形式のみが許される[17]。第 8 節で述べた埋め込みウィジェットの性質の多くが埋め込み画像に対しても適用される。埋め込み画像を含むプログラム例は既に第 1 節で扱われているので、ここでは改めて例題を採り挙げない。

埋め込み画像の配置位置はテキストインデックスを通じて指定される。この事は埋め込み画像はその配置に関しては文字と同じ扱いを受ける事を意味する。

埋め込み画像は文字と同様にタグ付けが可能である。tag_configure のオプションの多くは埋め込み画像に対しては意味を持たないが、マージンや行間調整や justify のような位置決めに関する指定は効力を持っている。

埋め込み画像に対するタグ付けは文字列に対するタグ付けと事なり、挿入時に行う事はできないので、 tag_add メソッドを使用する。やりかたは埋め込みウィジェットと同様なので第 8 節を参考にして頂きたい。

## 9.1 埋め込み時のオプション

埋め込み画像は image_create メソッドによってテキスト中に埋め込まれる。その際に指定可能なオプションは以下の通りである。これらのオプションの値は image_configure メソッドによって変更可能である。

□ image
　値: 画像 (PhotoImage メソッドによって返された値)
　意味: 埋め込む画像を指定する。

□ name
　値: 文字列
　意味: 埋め込む画像に付ける名前を指定する。name オプションを指定されていない場合には画像には埋め込まれた順に、pyimage1, pyimage2, ... の名前が与えられる。

□ align
　値: 'top', 'center', 'bottom' (あるいは TOP,CENTER, BOTTOM)
　意味: 行間のどの位置に配置するかを指定する。

□ padx
　値: 非負整数
　意味: 両サイドに付ける余白の大きさをピクセル単位で指定する。

□ pady
　値: 非負整数
　意味: 上下に付ける余白の大きさをピクセル単位で指定する。

[17] Python が扱う BMP 形式は Windows の BMP 形式とは異なる。

## 9.2 埋め込み画像に関するメソッド

□ image_create(*i*,*o*=*v*,...)

引数 *i*: テキストインデックス
引数 *o*=*v*: オプション
戻り値: None
機能: テキストインデックス *i* の位置に画像を埋め込む。埋め込む画像は image オプションで指定する。指定可能なオプションに関しては「埋め込み時のオプション」を見よ。

□ image_configure(*i*,*o*=*v*,...)

引数 *i*: テキストインデックス
引数 *o*=*v*: オプション
戻り値: None
機能: テキストインデックス *i* の位置の埋め込みオプションを変更する。指定可能なオプションに関しては「埋め込み時のオプション」を見よ。

□ image_names()

戻り値: 文字列のタプル。
機能: 埋め込まれた画像の名前の一覧を返す。

□ image_cget(*i*,*o*)

引数 *i*: テキストインデックス
引数 *o*: オプション名 (文字列)
戻り値: 文字列
機能: テキストインデックス *i* に埋め込まれた画像の、埋め込み時に指定可能なオプション *o* の現在値を文字列形式で返す。

# A Text で指定可能なオプションの一覧

□ background
値: 色名 ( 省略時の値は SystemWindow で白)
意味: テキストの背景色

□ bd
意味: borderwidth の別名。

□ borderwidth
値: 非負整数 ( 省略時の値は 2 )
意味: ウィジェットの輪郭の線幅。

□ cursor
値: カーソル名 ( 省略時の値は xterm )
意味: ウィジェットの中でのマウスカーソルの形状。カーソル名とその形状は文献 [6] および文献 [9] に載っている。また文献 [14] の付録 C にカーソル名とその形状を示すプログラムが載っている。xterm はワープロでポピュラーな縦棒のカーソルである。

□ 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。

□ font
値: フォント名 ( 省略時の値は Windows では ('MS Sans Serif',8) )
意味: text オプションで与えた文字列のフォント。

□ foreground
値: 色名 ( 省略時の値は SystemWindowText で黒)
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 ( 省略時の値は 24 )
意味: テキストウィジェットの高さ。単位は行数。1 つの行に要する高さは spacing1 と spacing3 だけが考慮されているようである。(文献 [17] では文字の高さを単位とすると書かれているが正しくない。)

□ highlightbackground
値: 色名 ( 省略時の値は SystemButtonFace でグレー)
意味: ウィジェットに入力フォーカスが無い時の強調枠の色。
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。highlightthickness も参照せよ。

□ highlightcolor
値: 色名 ( 省略時の値は SystemWindowFrame で黒)

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。 highlightthickness も参照せよ。

□ highlightthickness
値: 非負整数 (省略時の値は 0 )
意味: 強調枠の線幅。highlightcolor と highlightbackground も参照せよ。

□ insertbackground
値: 色名 ( 省略時の値は SystemWindowText で黒)
意味: カーソルの色

□ insertborderwidth
値: 非負整数 (省略時の値は 0 )
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 (省略時の値は 300 )
意味: カーソル点滅の OFF 時間 (単位はミリ秒)。

□ insertontime
値: 非負整数 (省略時の値は 600 )
意味: カーソル点滅の ON 時間 (単位はミリ秒)。

□ insertwidth
値: 非負整数 (省略時の値は 2 )
意味: カーソルの幅 (単位はピクセル)。

□ padx
値: 非負整数 (省略時の値は 1)
意味: テキスト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 1)
意味: テキスト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 'sunken')
意味: テキストウィジェットの輪郭デザイン。

□ selectbackground
値: 色名 (省略時の値は SystemHighlight)
意味: 選択状態の部分の背景色。

□ selectborderwidth

値: 非負整数 (省略時の値は 0)

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 (省略時の値は SystemHighlightText)

意味: 選択状態の部分の前景色。

□ setgrid

値: 非負整数 (省略時の値は 0)

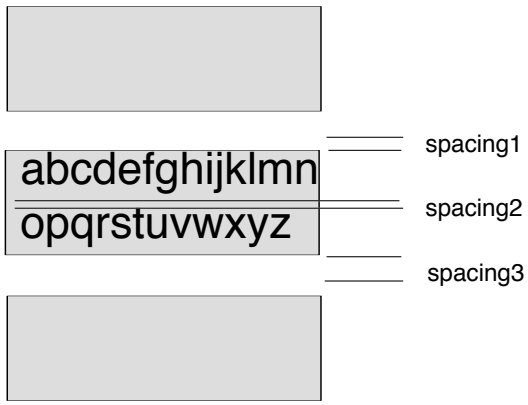
このオプションは必要が無いと思えるので解説しない。興味のある読者は文献 [7] を見よ。

□ spacing1　spacing2　spacing3

値: 非負整数 (いずれも省略時の値は 0 )

意味: 行間の空白を調整する。段落 (改行で終わる文字の列) と段落の間の行間は spacing1 と spacing3 で調整する。

spacing1 は段落の始まりに与える空白を、spacing3 は段落の終わりに与える空白を表す。spacing2 は段落の中での行間に与える空白である。いずれも単位はピクセルである。以下にその様子を図示する。但し段落を灰色で囲った矩形で代表している。




□ state ( 省略時の値は NORMAL )

値: NORMAL または DISABLED

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。さらに inser メソッドのようなテキストの状態を変更するメソッドも無効となる。テキストウィジェットの文字をユーザの操作によって変更されたくない場合には

```
mainloop()
```

の手前で

```
t.configure(state=DISABLED)
```

を実行しておくがよい。ここに t はテキストウィジェットである。

□ tabs

値: 文字列

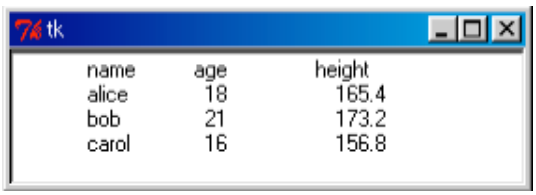
意味: タブストップの位置と方法を指定する。tabs の指定が無い場合には 8 文字幅の左揃えが仮定される。tabs を指定する場合には以下のように行う。

位置の指定は 40 あるいは 2c のように数字あるいは数字の後に単位を付ける。(単位に関しては文献 [10] あるいは文献 [11] を見よ。) 40 のように単位の無い数字はピクセルが単位である事を意味する。2c の c はセンチメートルを意味する。

位置を表す情報に続けて文字列の揃え方の方法を指定できる。方法は left, right, center, numeric のいずれかである。この内、numeric は数値を揃えるのに都合の良い方法である。即ち numeric で指定された蘭に小数点を含む数値が来る場合には小数点の位置で揃えられる。整数の場合には右揃えになる。数字でない文字列が来る場合には左揃えになる。方法が指定されていない場合には left が指定されたものと看做される。

以下に tabs を用いたプログラム例とその出力結果を示す。コーディングではデータを TAB で区切る必要がある。alice や bob や carol の前にも TAB が必要である。(TAB は TAB キーを打つ事によって挿入される空白である。)

```
from Tkinter import *
t = Text(width=45, height=5,tabs="1c left 3c numeric 5c numeric")
t.insert(END,
'''name age height
alice 18 165.4
bob 21 173.2
carol 16 156.8
'''
)
t.pack()
mainloop()
```




□ 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。

□ width
値: 非負整数 (省略時の値は 80)
意味: ウィジェットの横幅。単位は文字数である。固定幅フォントでないフォントを基に width が使用される場合には文字 0 (ゼロ) の横幅が基準に採られる (文献 [17])。

□ wrap
値: NONE, CHAR, WORD (省略時の値は CHAR)
意味: 行を折り返す時の方法。各意味は以下の通り。
NONE: 折り返さない
CHAR: 文字単位で折り返す
WORD: 単語単位で折り返す

□ 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。使い方は文献 [14] の第 14 節 Scrollbar を見よ。

# B Text のメソッド

Text に関するメソッドは本文の中に各節ごとに解説されている。以下にメソッド名と解説されている節を挙げるので、ここに挙げられているメソッドに関しては、その節を参照して頂きたい。ここでは本文の中で採り挙げられなかったメソッドを解説する。

## B.1 本文中に解説された Text のメソッド

- テキストインデックスに関するメソッド (第 2 節)

  index compare bbox get

- テキスト編集に関するメソッド (第 4 節)

  insert delete

- 文字列探索に関するメソッド (第 5 節)

  search

- タグに関するメソッド (第 6 節)

tag_configure tag_bind tag_unbind tag_cget tag_delete
tag_remove tag_names tag_ranges tag_add tag_lower tag_raise
tag_prevrange

- マークに関するメソッド (第 7 節)

  mark_names mark_gravity mark_set mark_unset

- 埋め込みウィンドウに関するメソッド (第 8 節)

  window_create window_configure window_names
  window_cget

- 埋め込み画像に関するメソッド (第 9 節)

  image_create image_configure image_names image_cget

## B.2 本文で採り挙げられなかった Text のメソッド

□ dlineinfo(*i*)

dlineinfo に関する解説は長くなるので第 B.3 節に回す。

□ see(*i*)

引数 *i*: テキストインデックス
戻り値: None
機能: *i* の位置の文字が見えるようにする。(既に見えている場合には何も行わない。)

□ scan_mark(*x*,*y*)

引数 *x*,*y*: 整数
戻り値: None
機能: scan_dragto を見よ。

□ scan_dragto(*x*,*y*)

引数 *x*,*y*: 整数
戻り値: None
機能: scan_markto で指定された座標 $(x, y)$ と scan_dragto で指定された座標 $(x, y)$ で指定した座標 $(x, y)$ の差の 10 倍だけテキストをスクロールさせる。

□ xview()

戻り値: 0 と 1 の間の 2 つの実数を要素とするタプル
機能: テキストウィジェットの中に見えているテキストの範囲を全体の割合で返す。このメソッドは NONE wrap オプションの時に意味を持っている。その場合 $x$ 方向のテキストの最大値は最大の長さの行の長さを意味する。xview に関しては詳しくは文献 [14] の第 10 節を参考にせよ。

□ xview('moveto',*f*)
引数 *f*: 0 と 1 の間の実数
戻り値: None
機能: テキストウィジェットの左側に隠れている部分を *f* で指定された割合にする。このメソッドは NONE wrap オプションの時に意味を持っている。その場合 *x* 方向のテキストの最大値は最大の長さの行の長さを意味する。xview に関しては詳しくは文献 [14] の第 10 節を参考にせよ。

□ xview('scroll',*n*,*s*)
引数 n: 整数
引数 s: 'units' または 'pages'
戻り値: None
機能: テキストウィジェットのビュー (テキストが見えている矩形領域) を *x* 方向に *n* で指定された大きさだけ移動する。'units' が指定された場合には文字単位で (平均的な文字幅で)、'pages' が指定された場合にはページ単位で (ビューの横幅で) 移動する。このメソッドは NONE wrap オプションの時に意味を持っている。

□ xview_moveto(*f*)
引数 *f*: 0 と 1 の間の 2 つの実数
戻り値: None
機能: xview('moveto',*f*) と同じ。

□ xview_scroll(*n*,*s*)
引数 *n*: 整数
引数 *s*: 'units' または 'pages'
戻り値: None
機能: xview('scroll',*n*,*s*) と同じ。

□ yview()
戻り値: 0 と 1 の間の 2 つの実数を要素とするタプル
機能: テキストウィジェットの中に見えているテキストの行範囲を全体の行の割合で返す。yview に関しては詳しくは文献 [14] の第 10 節を参考にせよ。

□ yview('moveto',*f*)
引数 *f*: 0 と 1 の間の実数
戻り値: None
機能: テキストウィジェットの上側に隠れている部分を *f* で指定された割合にする。yview に関しては詳しくは文献 [14] の第 10 節を参考にせよ。

□ yview('scroll',*n*,*s*)
引数 *n*: 整数
引数 *s*: 'units' または 'pages'
戻り値: None
機能: テキストウィジェットのビュー (テキストが見えている矩形領域) を *y* 方向に *n* で

指定された大きさだけ移動する。'units' が指定された場合には行単位で、'pages' が指定された場合にはページ単位で (ビューの高さで) 移動する。

□ yview_moveto(*f*)
引数 *f*: 0 と 1 の間の 2 つの実数
戻り値: None
機能: yview('moveto',*f*) と同じ。

□ yview_scroll(*n*,*s*)
引数 *n*: 整数
引数 *s*: 'units' または 'pages'
戻り値: None
機能: yview('scroll',*n*,*s*) と同じ。

□ yview_pickplace(*i*)
引数 *i*: テキストインデックス
戻り値: None
機能: このメソッドは現在では see(*i*) にとって代わられているので解説しない。

以上の他に

debug
tk_textBackspace
tk_textIndexCloser
tk_textResetAnchor
tk_textSelectTo

があるが、解説しない。

## B.3 dlineinfo メソッド

dlineinfo(*i*)

**引数** *i*: テキストインデックス

**戻り値** : 5 個の整数のタプルまたは None

**機能** : 戻り値を $(x, y, w, h, b)$ とすると、テキストインデックス i を含む行 (テキストウィジェットの中の行) を囲む矩形領域のうちテキストウィジェットの中で見える範囲が $(x, y, w, h)$ で、ベースライン (base line) の位置情報が $b$ で返される。但し、行の wrap オプションが NONE の場合には行は折り返されないので、行の一部が視界からはみ出す事があるが、この場合にははみ出した範囲も含まれている。行がテキストウィジェットの見える範囲に存在しない場合には None を返す。

以上の説明を図 10 に示す。図ではテキストウィジェットを灰色の矩形で、dlineinfo が返す矩形は黒色で示してある。また各行の右には説明が加えられている。この図で "A line of CHAR wrap" は CHAR wrap オプションの結果生じた行であり、テキストエィジェット中では

折り返されて 2 つの行に分かれている。この場合には dlineinfo は "A line CHAR w" の中の文字に対してと "rap" の文字に対して異なる矩形領域を返す。"A line of NONE wrap" は NONE wrap オプションが指定された行である。この場合には "A line of NONE wr" までがテキストウィジェットに表示されるが、dlineinfo が返す領域はテキストウィジェットからはみ出した "ap" をも含む。

矩形領域を表す $(x, y, w, h)$ の座標系はテキストウィジェットの左上を原点とし、右方向を $x$ 座標の正の方向、下方向を $y$ 座標の正の方向とする。単位はいずれもピクセルである。

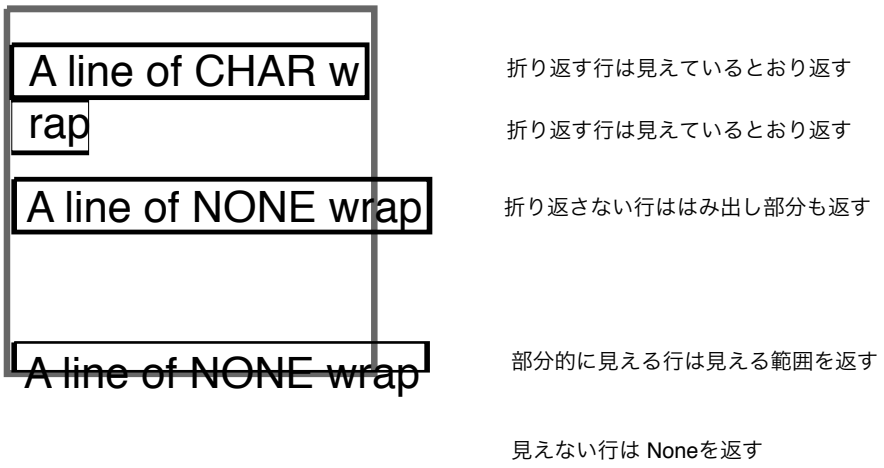



図 10: dlineinfo が返す矩形領域

灰色の線で描かれた矩形はテキストウィジェットである。他方黒色の線で描かれた矩形は dlineinfo が返す矩形領域である。ベースラインとは英字を行内で揃える時の基準線である (図 11)。dlineinfo が返すベースラインの位置情報は $y$ からの相対位置である。即ちベースライン $y$ 座標は $y + d$ で与えられる。


図 11: dlineinfo が返す矩形領域

# 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, David Ascher 著/紀太章訳: 「初めての Python 」(O'Reilly Japan, 2000)

[4] David M. Beazley 著/習志野弥治朗訳:
「Python テクニカルリファレンス — 言語仕様とライブラリ —」
(ピアソン・エデュケーション, 2000)

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

[6] John E.Grayson "Python and Tkinter Programming", (Manning, 2000)

[7] 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)

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

[9] 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)

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

[11] 有澤健治「Python によるグラフィックス」
(「Com」第 17 号,、愛知大学情報処理センター、1999 年 9 月)

[12] 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)

[13] 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)

[14] 有澤健治「Python における GUI の構築法 II — ウィジェット各論 —」
(「Com」第 19 号、 愛知大学情報処理センター、2001)

[15] "Python 2.0 Documentation" (配布ファイル Python/Doc/index.html)

[16] Python 2.0 配布ファイル Python20/NEWS.txt

[17] "Tcl/Tk Reference Manual" ( Tcl/Tk8.0 配布ファイル Tcl/doc/tcl80.hlp, v8.0.4)