目次

第 III 部　Python によるグラフィックス　125

第 1 章　グラフィックスの基礎　127

1.1 サンプルプログラム 127
1.2 キャンバス 130
1.3 座標系 131
1.4 基本図形 132
1.4.1 create_line 132
1.4.2 create_rectangle 134
1.4.3 create_polygon 135
1.4.4 create_oval 136
1.4.5 create_arc 137
1.4.6 create_text 138
1.5 オプション (補足) 140
1.5.1 font 140
1.5.2 color 142
1.5.3 stipple 143
1.5.4 capstyle 143
1.5.5 joinstyle 144
1.5.6 tags 145
1.6 印刷 147

付録 A　colors1.py　151

付録 B　colors2.py　153

第 III 部

Python によるグラフィックス

第1章　グラフィックスの基礎

Python で線分や円等の図形を表示するには Canvas クラスを使用する。Python の Canvas クラスはキャンバスに線分や円等の幾何学図形を描くためのクラスである。キャンバスに描いた絵は eps 形式のファイルに出力できる。描画結果はプリンタの解像度で印刷され、美しい仕上がりを得る。　Python の Canvas クラスに関しては今のところ解説は皆無である[1]。しかしながら Python の Canvas クラスは Tcl/Tk を利用しているので Tcl/Tk の解説書 [4][5] から Python の Canvas クラスの使用法を推測する事ができる。また Python の Canvas の記述のシンタックスは以下の Python のソースライブラリを参照すれば判明する。

```
python/Lib/lib-tk/Canvas.py
python/Lib/lib-tk/Tkinter.py
```

さらに Python の Canvas の使用法は UNIX 版 Python のソースプログラムに付属する 2 つのデモプログラム:

```
Python/Demo/tkinter/Guido/
Python/Demo/tkinter/Matt/
```

からも窺い知ることができる。この論稿はこうした断片的な情報を基に Python におけるグラフィックスの使い方を体系的に解説している。

以下に参照を容易にする為に、この解説の構成を記す。

第 1 節 サンプルプログラム
第 2 節 キャンバス
第 3 節 座標系
第 4 節 基本図形
第 5 節 オプション (補足)
第 6 節 印刷
参考文献
付録

1.1　サンプルプログラム

Python における Canvas クラスの使用法を見るために Canvas を使用したサンプルプログラムを 1 つ紹介し、それを解説する。

[1] この章は 1999 年の筆者の記事 [8] を基に最小限の修正を加えている。現在ではこの言い方は当たらない。

```
from Tkinter import *
from Canvas import *
import sys
def pr():
    d=c.postscript(file="a.eps")
def quit():
    sys.exit()
f=Frame()
b1=Button(f,text='write canvas to a.eps', command=pr)
b1.pack(side='left',expand=YES,fill='x')
b2=Button(f,text='quit',command=quit)
b2.pack(side='right',expand=YES,fill='x')
f.pack(fill='x')

# --- Canvas starts from this line ---
c=Canvas(width=400,height=300,background='cyan')
c.pack()

c.create_rectangle(2,2,401,301);# The maximum rectangle we can draw.

c.create_line(230,170,350,170)
c.create_rectangle(75, 30, 175,80,outline='blue')
c.create_rectangle(100, 50, 200,100,fill='#F00')
c.create_polygon(300,200,350,250,250,250,fill='yellow',width=2,outline='black')
c.create_oval(100, 200, 200,250,outline='green',width=20)
c.create_arc(250,30,350,80,start=90,extent=145,width=5,fill='pink')
c.create_text(250, 100,text='ABC',font='Times 30 bold italic',anchor='nw')
for n in range(0,35):
    c.create_line(30,100+5*n,70,100+5*n)
c.create_text(10,10,text='0',font='Courier 12')
c.create_line(20,10,360,10,arrow='last')
c.create_text(370,10,text='x',font='Courier 12')
c.create_line(10,20,10,270,arrow='last')
c.create_text(10,280,text='y',font='Courier 12')
c.create_line(100,150,130,180,160,150,190,180,smooth=YES,width=5,fill='blue',
    arrow='last')

mainloop()
```

図 1.1: プログラム sample.py

Python のプログラムは任意のテキストエディタで作成し、ファイル拡張子を 'py' とする。以下の説明では上のプログラムはファイル `sample.py` に保存されているとする。`sample.py` を単にマウスでダブルクリックするだけで、この内容が Python インタープリタに渡され、ファイルに書かれたプログラムが実行される。

しかしながら Python のプログラムの開発過程でこの実行方法をとるのは賢明ではない。プログラムにエラーが含まれている場合には、そのエラーの所在と性質を発見できないからである。開発中のプログラムはコマンドによって実行するのがよい。Windows であれば「DOS プロンプト」を立ち上げ、cd コマンドを使って `sample.py` の置かれたディレクトリに移り、

```
python sample.py
```

を実行する。

さて、このプログラムを実行すると次の図形が表示される。

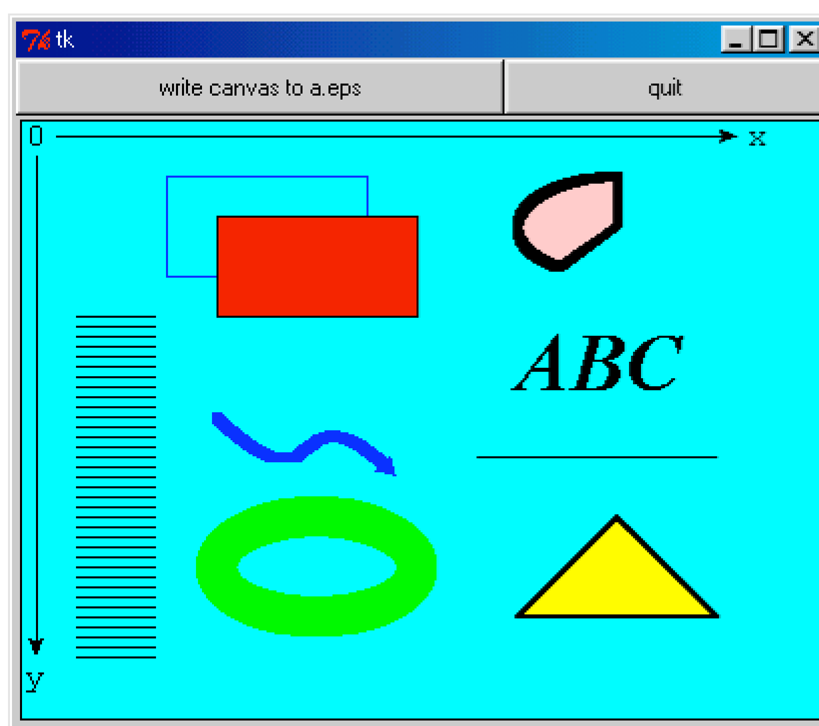

図 1.2: sample.py の実行結果

このプログラムの 19 行目

```
# --- Canvas starts from this line ---
```

まではボタンの構成と、ボタンが押された時の動作を記述している。このプログラムでは 'write canvas to a.eps' と書かれたボタンをマウスでクリックすればキャンバスの描画結果をファイル a.eps に書き出される。(このファイルは印刷に使用される予定である。) さらに 'quit' と書かれたボタンをクリックするとプログラムの実行が終了する。

しかしながらここではこの問題に関しては解説しない。従ってこの行から上の部分は所与のものとみなして欲しい。(ボタン等のユーザーズ・インターフェースの構築に関しては参考文献 [1],[2],[3] が詳しく解説している。)

1.2　キャンバス

解説は

```
# --- Canvas starts from this line ---
```

以下から始まる。Python では記号 ‘#’ で始まる行はコメントである。コメントはプログラムの実行に関しては影響を与えない。この次の行、

```
c=Canvas(width=400,height=300,background='cyan')
```

はキャンバスを生成する。width と height によってキャンバスのサイズを指定する。サイズの単位は画素 (pixcel) である。即ち画像を構成する色を持った点の個数である。キャンバスの背景色は backgroud によって指定する。これらの指定は省略する事もできる。その場合には暗黙に仮定された値が使用される。

生成されたキャンバスは変数 c によって参照される。キャンバス c に対する操作は

```
c.pack()
```

のように、c の次にピリオドを書き、その後に操作名を書く。ここでは c を pack している。この操作によって c が張りつけられる。(従って見えるようになる。) ここでは追加の方法を指定していない。この場合にはキャンバスを (ボタンの) 下に追加する。

後に見る様にキャンバスに描画する際には必ず Canvas によって生成された変数を指定する事になる。この変数はキャンバスの名前なのである。名前を指定してキャンバスに描画するという事は、Python では一つのプログラムの中で複数のキャンバスを扱えるという事である。

Canvas には width, height, background 以外にも多数のオプションが指定できる。ここではキャンバスの見栄えを変更する為のオプション relief だけを紹介する。relief は borderwidth と一緒に使用される。

```
c=Canvas(width=400,height=300,background='cyan')
```

を

```
c=Canvas(width=400,height=300,background='cyan',relief='raised', borderwidth=6)
```

に置き換えてプログラムを実行してみよう。キャンバスが浮き上がったように見えるはずである。relief には ’raised’ の他に、’sunken’, ’groove’, ’ridge’, ’flat’ が指定できる。次の図は、これらの指定よってキャンバスの境界がどのように見えるかを示す。

relief を指定しない場合には ’flat’ が仮定される。borderwidth は適度に調整する。

サンプルプログラムの最後の行の

```
mainloop()
```

は Canvas を使用したプログラムでは必須である。もしもこの行が存在しないならプログラムは直ちに終了し、結果を鑑賞する暇は無いであろう。このプログラムはこの行が存在する事によってマウスによるユーザ側のアクションを待っているのである。

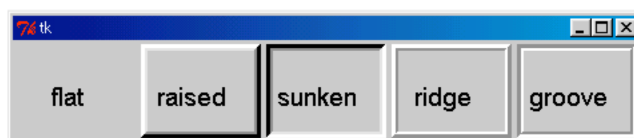

図 1.3: いろいろなレリーフ
レリーフ (relief) の指定によるキャンバスの境界の違いが図に示されている。
(順に、flat, raised, sunken, groove,ridge)

1.3　座標系

キャンバスに図形を描く場合には描画位置を指定する必要がある。座標は左から右方向に x 軸、上から下方向に y 軸である。座標原点はキャンバスの左上隅である。しかしこの言い方は厳密ではない。実際にはキャンバスの左上隅の座標は (2,2) となっている。プログラム 1 では、

```
c.create_rectangle(2,2,401,301)
```

によってこの事実を確認している。

rectangle とは矩形 (くけい) のことである。矩形とは水平線と垂直線によって囲まれた特殊な長方形の事である。そして Python では `create_rectangle` が矩形を描く命令文となっている。この例では `create_rectangle` の引数に現われる (2,2) は矩形の左上隅の座標、(401,301) は矩形の右下隅の座標である。そしてこれがこのキャンバスに描ける一番大きな矩形である。

読者は何故キャンバスの左上隅を座標 (0,0) として定義しないのか不思議に思うであろう。そもそもここに現われる 2 と言う数字は何を意味しているのか? この疑問に解決するには

```
c=Canvas(width=400,height=300,background='cyan')
```

を

```
c=Canvas(width=400,height=300,background='cyan',highlightthickness=10)
```

に置き換えてプログラムを実行してみるがよい。2 の数字の意味が分かるであろう。

紙に印刷する場合には画素をサイズの単位にするのは不便である。画素の間隔はディスプレイに依存しており、実際の印刷イメージを掴みにくい。画素の代わりにセンチメートルを単位として指定する事もできる。その場合には、

```
width="8c"
```

の様に、数字の後に c を付け、引用符で括る。同様にミリメートル、インチ、ポイントで指定する事もできる。各々

```
width="80m", width="3.15i", width="234p"
```

のように指定する。なお 1 インチは 2.54 センチメートルである。また 1 ポイント は 1/72 インチであり、この単位は文字フォントを指定する場面で使用されている。

Python ではサイズを単に数字で指定する事が多い。即ち画素の個数で指定する事が多いのである。 Python で指定した 1 インチは約 96 画素であると考えて構わない。Canvas で width をインチ単位で与えて、

```
print c.cget('width')
```

を実行させれば 画素単位のキャンバスの幅を出力してくれる。ここに c はキャンバスである。ディスプレイの基本的な 3 つの解像度 640 x 480 、800 x 600 及び 1024 x 768 の下でこの実験を行えば共に 96 ± 1 の値を得る。(これは Windows の下での実験である) 不思議な事に Python には長さの単位を変換する関数が見当たらない。1 インチの画素数を知る為に実験に頼ったのはこのためである。

Python で指定したサイズが実際にディスプレイ上でどのような大きさに見えるかはディスプレイ毎に異なる。従って 5 センチメートルの長さを指定してもディスプレイ上の長さは 5 センチメートルから大きく離れている場合が多いのである。(但し印刷の場合には正確に 5 センチメートルで描く。)

数学的な問題にはキャンバスの左上隅を座標原点とする座標系は扱いにくい。実は Python は長さの単位、座標原点、座標の正の方向を変更する機能を持っている。これに関しては第 5 節オプション (補足) の tags で述べる。

1.4　基本図形

キャンバスには以下の基本図形を描くことができる。

`create_line`	線分、折れ線、曲線
`create_rectangle`	矩形 (くけい)
`create_polygon`	多角形
`create_oval`	楕円
`create_arc`	円弧
`create_text`	文字列
`create_bitmap`	画像 (2 色)
`create_image`	画像 (カラー)
`create_window`	窓

キャンバスに図形を描くにはキャンバスの名称、図形の位置、さらに大きさ等の形状を特徴付ける情報を与える必要がある。以下にこれらの図形を描く命令を個別に解説する。(但しここでは `create_bitmap` と `create_image` と `create_window` に関しては解説を省略する。)

1.4.1　create_line

`create_line` は線分、折れ線、曲線を描く。単に 2 つの座標点を指定すると `create_line` は線分を描く。例えば、

```
c.create_line(230,170,350,170)
```

は座標 (230,170) と (350,170) を結ぶ線分をキャンバス c に描く。色を指定していないので黒色で描くことになる。描画オプションを付ける事もできる。例えば、

```
c.create_line(20,10,360,10,arrow='last')
```

は座標 (20,10) と (360,10) を結ぶ線分に矢印を付けて描く。矢印の指定は

```
'none', 'first', 'last', 'both'
```

の 4 つが可能である。'none' の場合には矢印を付けない。arrow の指定が無い場合には 'none' が仮定される。次の図は arrow の指定の意味を示している。何れも左から右に向かって描いた時の直線である。

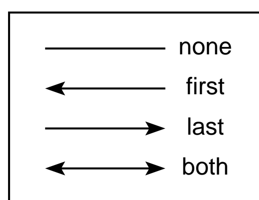

図 1.4: arrow の指定

`create_line` は線分を描くだけではない。折れ線や曲線も描く事ができる。例えば

```
c.create_line(100,150,130,180,160,150,190,180,smooth=YES,width=5,
    fill='blue', arrow='last')
```

は `create_line` の複雑な使い方である。もしも単に

```
c.create_line(100,150,130,180,160,150,190,180)
```

となっていれば、これは 4 つの座標 (100,150) と (130,180) と (160,150) と (190,180) とを結ぶ折れ線を表している。width=5 によってこの折れ線の線幅は 5 となる。fill='blue' によってこの折れ線は青色に描かれる。smooth=YES とすることによって線分を滑らかにした図形、即ち曲線を描く。

次の図は smooth の効果を示している。この図の曲線図形は単に `create_line` で描いた折れ線図形に smooth を指定しただけである。基になった折れ線も参考の為に合わせて示してある。

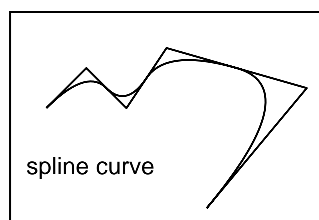

図 1.5: smooth 指定による図形の変化

smooth の指定によって生成される曲線は

1. 端点は通る
2. 中間の折れ線に関しては、その中点を通る

3. 折れ線には接する
4. 接点での曲率 (曲がり具合) は 0 である。
の性質を持っている事が分る。

smooth=YES を指定した場合には平滑の程度を splinesteps で指定できる。しかしながら splinesteps の指定は印刷結果には反映されない。従ってこの解説は省略する。

まとめ　`create_line` で指定可能なオプションの一覧

```
arrow        'none', 'first', 'last', 'both'
capstyle     'butt', 'projecting', 'round'
fill         'red', 'green', ...  などの色
joinstyle    'bevel', 'miter', 'round'
smooth       YES, NO
splinestep   1, 2, 3, ..
stipple      'gray12', 'gray25', 'gray50', 'gray75'
width        1, 2, 3, ..
tags         (第 5 節の tags を参照せよ)
```

1.4.2 create_rectangle

`create_rectangle` は矩形 (くけい) を描く。矩形とは、2 本の水平線と 2 本の垂直線によって囲まれる長方形の事である。矩形を描くには対角の 2 つの座標点を指定する。例えば、

```
c.create_rectangle(75, 30, 175,80,outline='blue')
```

によって座標 (75,30) と (175,80) を対角座標とする矩形が描かれる。

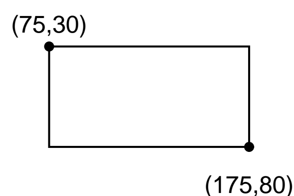

図 1.6: `c.create_rectangle(75,30,175,80)`

`outline='blue'` によってこの矩形は青色の輪郭で囲まれる。

```
c.create_rectangle(100, 50, 200,100,fill='#F00')
```

は内部を赤で塗り潰している。fill の指定が塗り潰し指定である。`#F00` は赤を意味している。

大抵の印刷機は現状では色を出せない。その場合でも印刷機は色を明暗に従って適度なグレーで表現してくれる。塗り潰しを色で指定する代わりに塗り潰しのパターンを指定することもできる。その為には、

```
c.create_rectangle(100,50,200,100,fill='black',stipple='gray12')
```

などとする。stipple によって塗り潰しパターンが指定されている。指定可能なパターンとそれらの見え方については付録に解説されている。

まとめ `create_rectangle` で指定可能なオプションの一覧

`fill`	`'red', 'green', ...` などの色 (塗り潰しの色)
`outline`	`'red', 'green', ...` などの色 (外枠の色)
`stipple`	`'gray12', 'gray25', 'gray50', 'gray75'`
`width`	`1, 2, 3, ...` (外枠の幅)
`tags`	第 5 節の `tags` を参照せよ

1.4.3 create_polygon

`create_polygon` は多角形あるいは閉曲線を描く。多角形を描くには頂点の座標を指定する。例えば、

```
c.create_polygon(300,200,350,250,250,250,fill='yellow',width=2,
outline='black')
```

によって 3 つの座標点 (300,200) と (350,250) と (250,250) を結ぶ多角形が描かれる。ここでは輪郭の太さが width=2 で指定されている。

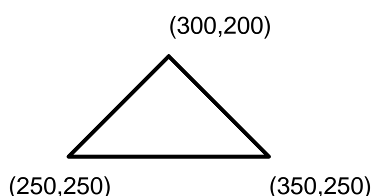

図 1.7: `c.create_polygon(300,200,350,250,250,250,fill='yellow',width=2,outline='black')`

5 角形を描きたい場合には座標点の個数を 5 つに増やす。この様にして任意の多角形を描くことができる。

閉曲線を描くには `create_line` で行った様に smooth=YES を指定する。(この下で `splinestep` も有効になる。) 次の図は先の三角形に対して smooth=YES を指定したものである。基になった三角形も参考の為に合わせて示してある。

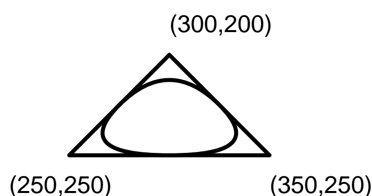

図 1.8: smooth の効果

まとめ　create_polygon で指定可能なオプションの一覧

fill	'red', 'green', ...　などの色 (塗り潰しの色)
outline	'red', 'green', ...　などの色 (外枠の色)
smooth	YES, NO
splinestep	1, 2, 3, ..
stipple	'gray12', 'gray25', 'gray50', 'gray75'
width	1, 2, 3, ...　(外枠の幅)
tags	第 5 節の tags を参照せよ

1.4.4　create_oval

create_oval は楕円を描く。円は楕円の特殊な場合として扱われる。create_oval で描かれる楕円は長径と短径が x 軸あるいは y 軸に平行である。楕円の大きさと位置を定める情報は楕円に外接する矩形を指定する事によって与える。例えば、

```
c.create_oval(100, 200, 200,250,outline='green',width=20)
```

によって楕円が描かれる。この楕円は

```
c.create_rectangle(100,200,200,250)
```

に内接する。

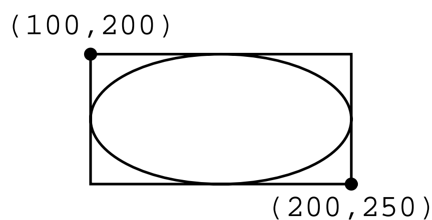

図 1.9: c.create_oval(100,200,200,250)

まとめ　create_oval で指定可能なオプションの一覧)

fill	'red', 'green', ... などの色 (塗り潰しの色)
outline	'red', 'green', ... などの色 (外枠の色)
stipple	'gray12', 'gray25', 'gray50', 'gray75'
width	1, 2, 3, ... (外枠の幅)
tags	第 5 節の tags を参照せよ

1.4.5 create_arc

create_arc は扇形または扇形の弧を描く。create_arc が描く扇形は Oval を基にしている。即ち、Oval の中心を通る 2 つの直線で Oval を切り取った図形が Arc である。例えば、

```
c.create_arc(250,30,350,80,start=90,extent=145,width=5,fill='pink')
```

によって扇形が描かれる。この扇形の弧は

```
c.create_oval(100, 200, 200,250)
```

の上にあり、弧の開始角と弧の開き角は start と extent で与えられている。角度の単位は度である。

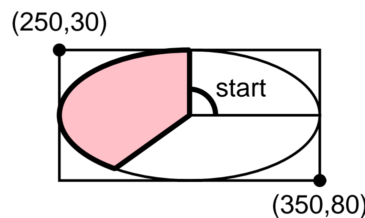

図 1.10: c.create_arc(250,30,350,80,start=90,extent=145,width=5,fill='pink')

図をよく見て見よう。注意深い読者は、この図で弧の開きが本当に 145 度にはなってはいない事に気付くであろう。弧の開始角は正しく 90 度になっている。実は開始角が正しく表示されたのは偶然なのである。この図の扇型は楕円を基礎にして作成されている。この楕円を横方向に一様に圧縮すると図 1.11 に示す円ができ上がる。図 1.11 ではこの円を上側に示している。

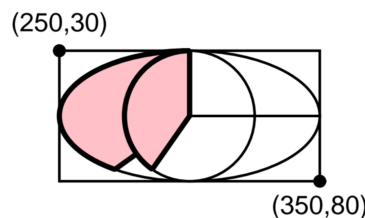

図 1.11: 円との比較で見る角度の指定
c.create_arc(250,30,350,80,start=90,extent=145) と
c.create_arc(275,30,325,80,start=90,extent=145) との比較

この図を見て分るように、create_arc の start と extent で示される開始角と弧の開き角は、円に矯正された扇型に対して、正しい数値を表す。

create_arc は style を指定する事によって次の図に示す図形を生成する。

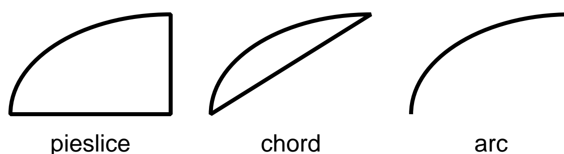

図 1.12: style の効果

まとめ　create_arc で指定可能なオプションの一覧

extent	数字 (開きの角度)
fill	'red', 'green', ...　などの色 (塗り潰しの色)
outline	'red', 'green', ...　などの色 (外枠の色)
start	数字 (開始角)
stipple	'gray12', 'gray25', 'gray50', 'gray75'
style	'pieslice', 'chord', 'arc'
width	1, 2, 3, ...　(外枠の幅)
tags	第 5 節の tags を参照せよ

1.4.6　create_text

create_text は文字列を表示する。例えば、

```
c.create_text(250, 100,text='ABC',font='Times 30 bold italic',
anchor='nw')
```

は文字列 ABC を座標 (250,100) に書く。font は文字の書体と大きさを指定する。ここでは anchor='nw' に注意しよう。文字列 ABC を (250,100) の位置に書くといっても色々な意味に解釈できる。例えば点 (250,100) を表示文字列 ABC の中央に来る様に書くのか、左端に来る様に書くのか等の解釈の多様性である。Python ではこの問題を anchor の指定によって解決する。anchor には

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

を指定できる。指定が無い場合には 'center' が採用される。'center' 以外に現われる e, w, s, n は東西南北の意味である。これらは座標が表示される文字列に対してどの位置なのかを指定している。その際、地図と同様に、上方向を北、右方向を東と考えている。同一の座標を指

定し、 anchor の指定を変化させた場合に結果がどのように変化するかを図 1.13 に示す。ディスプレイ上での結果と、eps ファイルを通じて印刷した結果とは微妙に異なっている。図では左がディスプレイでの結果であり、右が印刷結果である。指定した座標点は各々の図の中央部の太い点で示されている。但しこれは Windows に固有な問題であり、MacOSX や UNIX ではこのような問題は発生しない。

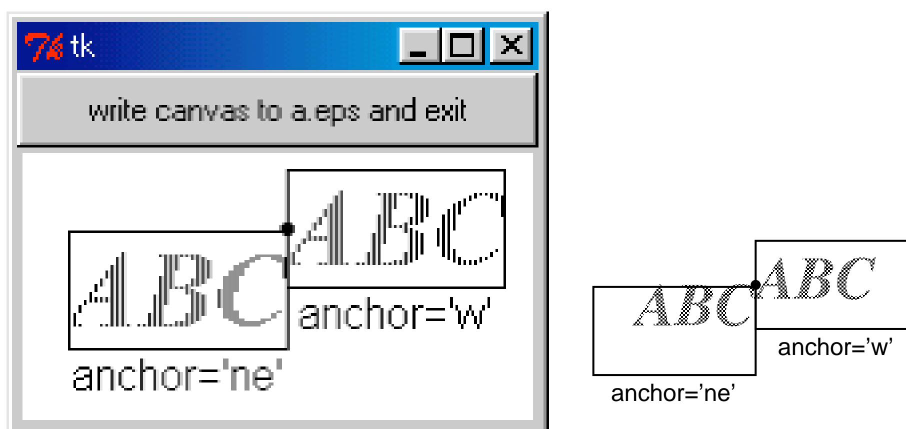

図 1.13: anchor の指定による文字列の表示位置の違い

`anchor=*ne'` と `anchor='w'` で同一の座標が指定されている。見え方はディスプレイでの表示と eps ファイルを通じて印刷した場合では異なる。右が左のディスプレイの表示を印刷した結果である。

キャンバスに複数行に渡る文字列を書きたい場合がある。その場合にはその文字列を一旦変数に格納した方が分りやすい。Python では複数行に渡る文字列は普通の引用符 (')、または二重引用符 (") を 3 つ並べて実現する事ができる。例えば、

```
s='''My name is Alice.
I am waiting for your mail.
Thank you.'''
c.create_text(250,100,text=s,font='Times 12',anchor=nw,justify='center')
```

等のようにすればよい。ここに現われている justify は行揃えを指定している。justify の値は 'center' の他、'left' と 'right' を指定できる。justify による指定を省略した場合には 'left' が仮定される。次の図は justify の効果を示している。

My name is Alice.
I am waiting for your mail.
Thank you.

My name is Alice.
I am waiting for your mail.
Thank you.

My name is Alice.
I am waiting for your mail.
Thank you.

図 1.14: justify の効果
上から順に left, center, right

まとめ create_text で指定可能なオプションの一覧

anchor	'nw', 'n', 'ne', 'w', 'center', 'e', 'sw', 's', 'se'
fill	'red', 'green', ... などの色 (塗り潰しの色)
font	'Times 12', ... など書体と大きさの指定
justify	'left', 'right', 'center'
stipple	'gray12', 'gray25', 'gray50', 'gray75'
text	'ABC' など表示したい文字列
width	100 など、一行のサイズ (このサイズで折り畳まれる。0 で折畳なし)
tags	第 5 節の tags を参照せよ

1.5 オプション (補足)

1.5.1 font

さてプログラム 1 の

```
c.create_text(250, 100,text='ABC',font='Times 30 bold italic',anchor='nw')
```

では書体が

```
font='Times 30 bold italic'
```

によって指定されている。実はこのフォントの指定の仕方はちょっと古いのである。Python の最近の配布ファイルに付属のプログラム例では

```
font=('Times', 30, 'bold', 'italic')
```

のようにタプルを使う書き方になっている。でも筆者はこの書き方は面倒で嫌いである。

font の指定には、書体の名称、文字サイズをこの順に指定し、その後に太字なら bold を、斜体なら italic を指定する。文字サイズの数値はポイントである。1 ポイントは 1/72 インチであり、キャンバスを構成する場合に使用される画素を単位としたピクセル値とは異なることに注意する。使用可能なフォントはシステム毎に異なる。実際にどのようなフォントが使用可能であるかはワープロなどのフォントメニューを調べれば分かる。例えば Windows であれば以下のようなフォントが含まれている。

Times　Helvetica　Bookman　Courier　Lucida　Script　Century

この他に日本語フォントを含め、多数のフォントが含まれている。

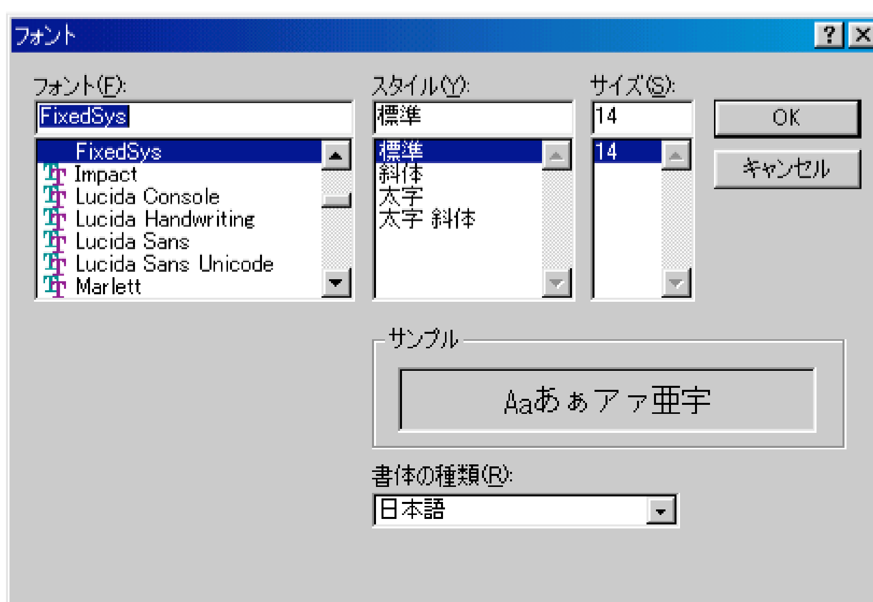

図 1.15: Window95 のフォントメニュー

Windows のフォントメニューを見るとフォント名称の中に空白文字を含むものがある。例えば

Times New Roman
ＭＳ 明朝

などである。このような場合に

font='ＭＳ 明朝'

と指定すると実行に失敗するであろう。何故なら Python は「ＭＳ」をフォント名称であると解釈し、その次には文字サイズを表す数字が来ると期待するが、その期待が裏切られるからである。フォント名称に空白が含まれる場合には空白を

```
font='Times-New-Roman'
font='ＭＳ-明朝'
```

のようにハイフンで置き換えれば認識してくれる。

フォントはビットマップフォントとアウトラインフォントに分類される。ビットマップフォントはディスプレイに小さな文字を表示させたい場合には良くデザインされているが、大きな文字を表示する場合やプリンタへの出力では印字品質が落ちる。従って可能な限りアウトラインフォントを指定した方が良い。Windows ではアウトラインフォントは TrueType フォントと呼ばれている。TrueType フォントは図 1.15 では TT で示されている。

印刷を必要とする場合にはフォントの選択にさらに注意が必要である。Python 自体は印刷機能を持っていない。しかし表示結果を EPS 形式のファイルに落とす事ができる。EPS 形式は PostScript 形式の親戚である。従って PostScript 形式のフォントが使用され、これは TrueType フォントと同じではない。Windows に備わっているフォントが必ずしも効力を持っているとは限らないのである。

印刷をも視野に入れた時に安全なフォントは限られてくる。現状では Times、Helvetica、Courier だけであると割り切った方がよい。その場合には残念ながら日本語は使えない。

1.5.2 color

Python では以下の色名称が使用できる。

```
black   darkgray   gray   lightgray   white   snow   seashell
AntiqueWhite   bisque   PeachPuff   NavajoWhite   LemonChiffon
cornsilk   ivory   honeydew   LavenderBlush   MistyRose   azure
SlateBlue   RoyalBlue   blue   DodgerBlue   SteelBlue   DeepSkyBlue
SkyBlue   LightSkyBlue   LightSteelBlue   LightBlue   LightCyan
PaleTurquoise   CadetBlue   turquoise   cyan   SlateGray
DarkSlateGray   aquamarine   DarkSeaGreen   SeaGreen   PaleGreen
SpringGreengreen   chartreuse   OliveDrab   DarkOliveGreen   khaki
LightGoldenrod   LightYellow   yellow   gold   goldenrod
DarkGoldenrod   RosyBrown   IndianRed   sienna   burlywood   wheat
tan   chocolate   firebrick   brown   salmon   LightSalmon
orange   DarkOrange   coral   tomato   OrangeRed   red   DeepPink
HotPinkpink   LightPink   PaleVioletRed   maroon   VioletRed
magentaorchid   plum   MediumOrchid   DarkOrchid   purple
MediumPurple   thistle
```

`gray` に関しては `gray30` のように数字で暗さを指定できる。`gray0` で黒、`gray100` で白である。gray 系列ではない他の色に関しては `1,2,3,4` の数字を色名の後に付ける事が可能である。

さらに Pyhton では 16 進数文字 (`0,1,2,···,9,A,B,···,F`) で色を指定する事もできる。光の三原色は赤、緑、青である。次の、

```
c.create_rectangle(100, 50, 200,100,fill='#F00')
```

の `fill='#F00'` に見られる `F` と `0` と `0` はこの順に色を構成する赤と緑と青の光の強さを表している。ここでは各々の色の強さが 1 桁の 16 進数で表されている。`F` が一番強く、`0` は一

番弱い。一番弱いと言う事はその色が含まれていないことを意味している。従って `#F00` の色は赤色であり、しかも一番明るい赤色である。ここでは各々の色が 16 諧調で表現されている。キャンバスに図形を描く場合にはこれで十分な事も多いが、実は Python は微妙な色合いを出すのにもっと木目細かく諧調を表現する事もできる。例えばオレンジ色は `#FFA400` である。ここでは 6 個の 16 進文字が並んでいる事に注意する。この場合には各々の色の強度は 2 桁の 16 進数で表現されており、ここでは赤が `FF`、緑が `A4`、青が `00` である。Python は各々の色を 3 桁の 16 進数で表現する事もできる。その場合には # の後に 9 個の 16 進文字が並ぶ。

指定した色が実際にどのように見えるかを知る為には Python のプログラムを作って実際に色を表示してみるのがよい。付録の `color1.py` あるいは `color2.py` を実行すれば色名称と実際の見え方の対応が分かる。

Python の色指定は Tk に従っている。 Tk はまた UNIX の X ウィンドウに従っている。X ウィンドウの色指定は文献 [6] に解説されている。

1.5.3 stipple

Python では塗り潰しのパターンを `stipple` によって指定できる。パターンとしては

```
'gray12'
'gray25'
'gray50'
'gray75'
```

が指定できる。パターンは

```
c.create_rectangle(100,50,200,100,fill='black',stipple='gray12')
```

の様に、文字列として与える。

次の図は各々のパターンの印刷結果である。

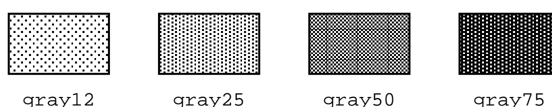

図 1.16: stipple の指定による塗り潰しパターンの違い

1.5.4 capstyle

capstyle は線の端点の形状を表す。capstyle は `create_line` におけるオプションであり、

```
c.create_line(230,170,350,170,width=20,capstyle='round')
```

の様に、文字列として指定する。形状としては次のものが指定できる。

'butt'
'projecting'
'round'

以下のそれらの違いを示す。

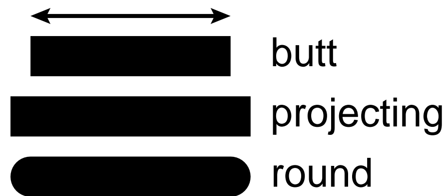

図 1.17: capstyle の指定による線端の違い

butt と projecting はよく似ている。butt の切り取り位置は projecting よりも線幅の半分だけ短い。capstyle は太い線でのみ実際上の意味を持つ。

1.5.5 joinstyle

joinstyle は線の接続点での形状を表す。capstyle と同様に形状を文字列として指定する。指定できる文字列は

'bevel'
'miter'
'round'

の 3 つである。次の図は指定の違いによる形状の違いを示している。

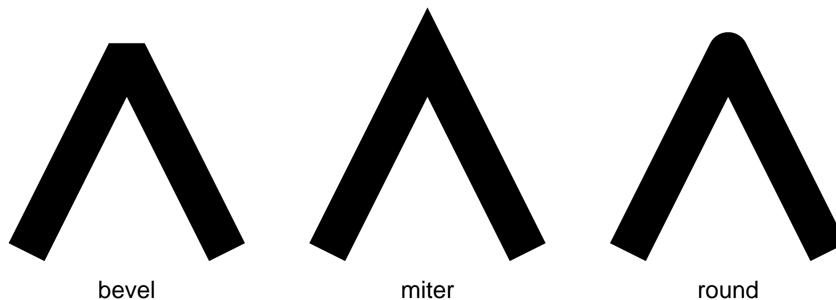

図 1.18: joinstyle の指定による接合点の違い

1.5.6 tags

Python ではキャンバス上の図形の幾つかを纏めてグループ化する機能を持っている。グループ化された図形は、あたかも一個の図形のように、大きさを変えたり、移動したり、削除したりできる。Python のグループ化は構成要素にタグ (札) を付けることによって行われる。同じタグを付けられた図形は同一のグループに属していると見なされるのである。例えば次のドラム形の図形は、 1 個の oval 、 1 個の arc 、それと 2 個の line から構成されている。これらが纏まって 1 個の図形が構成されているのである。

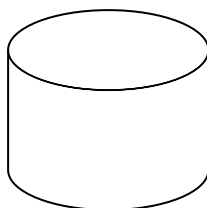

図 1.19: 描きたいドラム図形

この図は例えば次のように作成できる。(これはプログラムの全てではない。sample.py の「# --- Canvas starts from this line ---」以下の行である。)

```
c=Canvas(width=200,height=150,background='cyan')
c.pack()
c.create_oval(-1,1,1,0.2,tags='drum')
c.create_arc(-1,-1,1,-0.2,start=180,extent=180,style='arc',tags='drum')
c.create_line(-1,-0.6,-1,0.6,tags='drum')
c.create_line(1,-0.6,1,0.6,tags='drum')
c.scale('drum',0,0,50,-50)
c.move('drum',100,75)
mainloop()
```

ここではこの図形には分りやすく 'drum' と言うタグを付けているが、この名称に拘る必要はない。このプログラムではドラムをデザインするに当たって、次の図 1.20 に示す図案を頭の中に描いている。

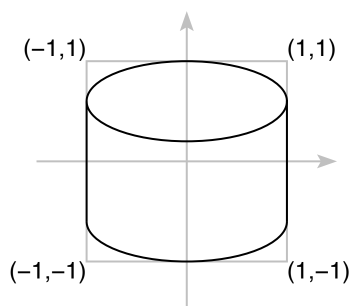

図 1.20: ドラム図形の図案

注目して欲しい点が幾つかある。この図案では

1. 上向きの方向が y の正の方向である。
2. 原点 (0, 0) を中心にデザインされている。
3. 長さの単位が画素数ではなく、抽象化された大きさになっている。

即ち、キャンバス本来の原始的な座標系よりも洗練され、扱い易くなっているのである。このトリックを可能にしたのは

```
c.scale('drum',0,0,50,-50)     --- [a]
c.move('drum',100,75)          --- [b]
```

の 2 つである。

scale は座標系を定める。抽象的な 1 の長さは [a] によって 50 画素 のサイズであると定められている。[a] に現われる 4 つの数字の内、最後の 2 つ (50, −50) が各々 x 方向、y 方向の拡大率を表しているのである。y 方向の拡大率を負の値に設定する事によって y 方向の向きが反転する。従って上方向を y の正の方向と考えてデザインする事ができる。

[a] の最初の 2 つの数字 (0, 0) は座標の原点を定めている。そして (0, 0) の場合には原点の移動は発生しない。(0, 0) でない場合の解説は後に回す。

原点を中心に描かれた図形は [b] によってキャンバス上 (100, 75) だけ移動される。この時の座標はキャンバス本来の座標である。それ故に下方向が y の正の方向となっている。

今の問題では [a] と [b] を次の 1 個に纏める事が可能である。

```
c.scale('drum',-2,1.5,50,-50)     --- [c]
```

ここに現われる scale の最初の 2 つの数字 (−2, 1.5) はキャンバスの左上隅の座標を表している。この座標の数字は図案 (図 1.20) を基にしている。

上のプログラム例では個々の図形にタグを付けている。そのためにコーディングが面倒になっている。Python では様々な方法でタグを付けることができる。ここでは二つの方法を紹介する。

一つは矩形 (くけい) 領域を指定しその中に含まれる全ての図形にタグを付ける方法である。それには addtag_enclosed を使用する。

```
c.create_oval(-1,1,1,0.2)
c.create_arc(-1,-1,1,-0.2,start=180,extent=180,style='arc')
c.create_line(-1,-0.6,-1,0.6)
c.create_line(1,-0.6,1,0.6)
c.addtag_enclosed('drum', -1.1,-1.1,1.1,1.1)
```

ここではドラムが完全に含まれる様に矩形領域を少し大きめに採っている。

定義済タグ 'all' が存在する。このタグは既に全ての図形に対して付けられている。タグ 'all' は関数のグラフのような数学的な問題を扱う場合に便利である。

1.6 印刷

Python では `background` で指定されたキャンバスの色は印刷されない。また、印刷に Ghostscript を借用しているために[2]、ディスプレイに表示できるフォントが必ずしも使用できない。さらに文字サイズの考え方が、ディスプレイに表示する時と、印刷する時とでは異なっている[3]。この違いを図に示す。

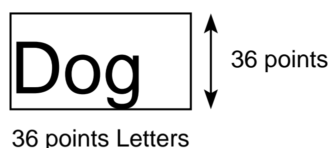

図 1.21: フォントで指定した文字の大きさと印刷結果の比較
ディスプレイ上では図に示した長方形の大きさに見える。
(図 1.13 も同様な問題を扱っているので参照せよ。)

この図は印刷結果である。OS に Windows を使用した場合には、ディスプレイでの表示では Dog の文字は丁度図の四角の枠ぐらいの大きさだったのである。

サンプルプログラムの描画結果を印刷するには以下の手順をとる。

1. ‘write to a.eps’ と書かれたボタンをマウスでクリックする。するとサンプルプログラムが置かれているディレクトリに、ファイル a.eps が生成される。
2. ファイル a.eps を Ghostview にドラッグする。すると Ghostview にキャンバスの図形が再び表示されるであろう。
3. Ghostview から印刷メニューを選ぶ。するとプリンタ一覧が表示されるであろう。ここで実装されているプリンタに合わせてプリンタを選ぶ。もしも希望するプリンタ名称がこの一覧に載っていない場合には、そのプリンタと同じ系列のプリンタ名称を捜すのがよい。印刷可能な場合がある。例えば Canon の BJC-80v は bjc800 で印刷可能である。

[2] MacOSX の印刷環境はとてもよい。Ghostscript 使わずに済むのだ。
[3] これは Windows での現象である。MacOSX や UNIX ではこのような問題は発生しない。

関連図書

[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] 江口庄英『Ghostscript Another Manual』(ソフトバンク、1997)

[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)

付 録A　colors1.py

次のプログラムは Python の色名称とディスプレイ上での実際の見え方を比較するためのものである。プログラムを打ち込むのが面倒であれば筆者のサーバから採ってくるがよい。http://ar.aichi-u.ac.jp/netlib/Python/samples/ に置かれている。

```
--------------------------- colors1.py -------------------------
#
# colors
# coded by Kenar
# REF: Tcl/lib/tk8.0/demos/colors.tcl
#
from Canvas import *
import sys, string
from Tkinter import *
colors='''
black darkgray gray lightgray
white snow seashell AntiqueWhite bisque
PeachPuff NavajoWhite LemonChiffon cornsilk
ivory honeydew LavenderBlush MistyRose azure
SlateBlue RoyalBlue blue DodgerBlue SteelBlue
DeepSkyBlue SkyBlue LightSkyBlue LightSteelBlue
LightBlue LightCyan PaleTurquoise CadetBlue
turquoise cyan SlateGray DarkSlateGray aquamarine
DarkSeaGreen SeaGreen PaleGreen SpringGreen
green chartreuse OliveDrab DarkOliveGreen
khaki LightGoldenrod LightYellow yellow
gold goldenrod DarkGoldenrod RosyBrown
IndianRed sienna burlywood wheat tan
chocolate firebrick brown salmon LightSalmon
orange DarkOrange coral tomato
OrangeRed red DeepPink HotPink pink LightPink
PaleVioletRed maroon VioletRed magenta
orchid plum MediumOrchid DarkOrchid
purple MediumPurple thistle
'''

colorlist=string.split(colors)
numcolor=len(colorlist)

def pr(event):
global t
n,m=event.x/20,event.y/20
num=16*m+n
if num >= numcolor: return
t.delete()
color=colorlist[num]
```

```
red,green,blue=c.winfo_rgb(color)
red=red/256
green=green/256
blue=blue/256
t=c.create_text(150,120,
text="%s     #%02X%02X%02X"%(color,red,green,blue),
anchor="w")

c=Canvas(width=320,height=140,background='white')
c.pack()
c.bind("<Button-1>",pr)

n,m=0,0
for color in colorlist:
c.create_rectangle(20*n, 20*m, 20*n+20, 20*m+20, fill=color)
n=n+1
if n == 16: n=0; m=m+1
t=c.create_text(150,420,text="")
c.mainloop()
--------------------------------------------------------
```

付 録B　colors2.py

次のプログラムは Python の色名称とディスプレイ上での実際の見え方を比較するためのものである。プログラムを打ち込むのが面倒であれば筆者のサーバから採ってくるがよい。http://ar.aichi-u.ac.jp/netlib/Python/samples/ に置かれている。

```
------------------------- colors2.py ---------------------
#
# colors
# coded by Kenar
# REF: Tcl/lib/tk8.0/demos/colors.tcl
#
from Canvas import *
import sys, string
from Tkinter import *
colors='''
    gray60 gray70 gray80 gray85 gray90 gray95
    snow1 snow2 snow3 snow4 seashell1 seashell2
    seashell3 seashell4 AntiqueWhite1 AntiqueWhite2 AntiqueWhite3
    AntiqueWhite4 bisque1 bisque2 bisque3 bisque4 PeachPuff1
    PeachPuff2 PeachPuff3 PeachPuff4 NavajoWhite1 NavajoWhite2
    NavajoWhite3 NavajoWhite4 LemonChiffon1 LemonChiffon2
    LemonChiffon3 LemonChiffon4 cornsilk1 cornsilk2 cornsilk3
    cornsilk4 ivory1 ivory2 ivory3 ivory4 honeydew1 honeydew2
    honeydew3 honeydew4 LavenderBlush1 LavenderBlush2
    LavenderBlush3 LavenderBlush4 MistyRose1 MistyRose2
    MistyRose3 MistyRose4 azure1 azure2 azure3 azure4
    SlateBlue1 SlateBlue2 SlateBlue3 SlateBlue4 RoyalBlue1
    RoyalBlue2 RoyalBlue3 RoyalBlue4 blue1 blue2 blue3 blue4
    DodgerBlue1 DodgerBlue2 DodgerBlue3 DodgerBlue4 SteelBlue1
    SteelBlue2 SteelBlue3 SteelBlue4 DeepSkyBlue1 DeepSkyBlue2
    DeepSkyBlue3 DeepSkyBlue4 SkyBlue1 SkyBlue2 SkyBlue3
    SkyBlue4 LightSkyBlue1 LightSkyBlue2 LightSkyBlue3
    LightSkyBlue4 SlateGray1 SlateGray2 SlateGray3 SlateGray4
    LightSteelBlue1 LightSteelBlue2 LightSteelBlue3
    LightSteelBlue4 LightBlue1 LightBlue2 LightBlue3
    LightBlue4 LightCyan1 LightCyan2 LightCyan3 LightCyan4
    PaleTurquoise1 PaleTurquoise2 PaleTurquoise3 PaleTurquoise4
    CadetBlue1 CadetBlue2 CadetBlue3 CadetBlue4 turquoise1
    turquoise2 turquoise3 turquoise4 cyan1 cyan2 cyan3 cyan4
    DarkSlateGray1 DarkSlateGray2 DarkSlateGray3
    DarkSlateGray4 aquamarine1 aquamarine2 aquamarine3
    aquamarine4 DarkSeaGreen1 DarkSeaGreen2 DarkSeaGreen3
    DarkSeaGreen4 SeaGreen1 SeaGreen2 SeaGreen3 SeaGreen4
    PaleGreen1 PaleGreen2 PaleGreen3 PaleGreen4 SpringGreen1
    SpringGreen2 SpringGreen3 SpringGreen4 green1 green2
    green3 green4 chartreuse1 chartreuse2 chartreuse3
```

```
    chartreuse4 OliveDrab1 OliveDrab2 OliveDrab3 OliveDrab4
    DarkOliveGreen1 DarkOliveGreen2 DarkOliveGreen3
    DarkOliveGreen4 khaki1 khaki2 khaki3 khaki4
    LightGoldenrod1 LightGoldenrod2 LightGoldenrod3
    LightGoldenrod4 LightYellow1 LightYellow2 LightYellow3
    LightYellow4 yellow1 yellow2 yellow3 yellow4 gold1 gold2
    gold3 gold4 goldenrod1 goldenrod2 goldenrod3 goldenrod4
    DarkGoldenrod1 DarkGoldenrod2 DarkGoldenrod3 DarkGoldenrod4
    RosyBrown1 RosyBrown2 RosyBrown3 RosyBrown4 IndianRed1
    IndianRed2 IndianRed3 IndianRed4 sienna1 sienna2 sienna3
    sienna4 burlywood1 burlywood2 burlywood3 burlywood4 wheat1
    wheat2 wheat3 wheat4 tan1 tan2 tan3 tan4 chocolate1
    chocolate2 chocolate3 chocolate4 firebrick1 firebrick2
    firebrick3 firebrick4 brown1 brown2 brown3 brown4 salmon1
    salmon2 salmon3 salmon4 LightSalmon1 LightSalmon2
    LightSalmon3 LightSalmon4 orange1 orange2 orange3 orange4
    DarkOrange1 DarkOrange2 DarkOrange3 DarkOrange4 coral1
    coral2 coral3 coral4 tomato1 tomato2 tomato3 tomato4
    OrangeRed1 OrangeRed2 OrangeRed3 OrangeRed4 red1 red2 red3
    red4 DeepPink1 DeepPink2 DeepPink3 DeepPink4 HotPink1
    HotPink2 HotPink3 HotPink4 pink1 pink2 pink3 pink4
    LightPink1 LightPink2 LightPink3 LightPink4 PaleVioletRed1
    PaleVioletRed2 PaleVioletRed3 PaleVioletRed4 maroon1
    maroon2 maroon3 maroon4 VioletRed1 VioletRed2 VioletRed3
    VioletRed4 magenta1 magenta2 magenta3 magenta4 orchid1
    orchid2 orchid3 orchid4 plum1 plum2 plum3 plum4
    MediumOrchid1 MediumOrchid2 MediumOrchid3 MediumOrchid4
    DarkOrchid1 DarkOrchid2 DarkOrchid3 DarkOrchid4 purple1
    purple2 purple3 purple4 MediumPurple1 MediumPurple2
    MediumPurple3 MediumPurple4 thistle1 thistle2 thistle3
    thistle4
'''

colorlist=string.split(colors)
numcolor=len(colorlist)

def pr(event):
global t
n,m=event.x/20,event.y/20
num=16*m+n
if num >= numcolor: return
t.delete()
color=colorlist[num]
red,green,blue=c.winfo_rgb(color)
red=red/256
green=green/256
blue=blue/256
t=c.create_text(150,420,
text="%s     #%02X%02X%02X"%(color,red,green,blue))

c=Canvas(width=320,height=440,background='white')
c.pack()
c.bind("<Button-1>",pr)
```

```
n,m=0,0
for color in colorlist:
c.create_rectangle(20*n, 20*m, 20*n+20, 20*m+20, fill=color)
n=n+1
if n == 16: n=0; m=m+1
t=c.create_text(150,420,text="")
c.mainloop()
----------------------------------------------------------------
```

なお、このプログラムに現われる色名称は Tk 付属のサンプルプログラムから借用されている。