## 9.3 Drawing

Tk provides a rich and easy-to-use set of primitive drawing operations. To use them we need to create a canvas widget, and then draw upon it. Since all of the drawing operations need access to the canvas widget, it is easiest to make this a global variable. The code for constructing a canvas is simple:

```
global canvas
canvas=Canvas(self,width=500,height=500,background="white")
canvas.grid(row=1, column=0)
```

The canvas is generally a child of the main GUI window. The only objects in this window typically are the MenuBar and the canvas. I usually put the MenuBar in row 0 and the canvas in row 1, below the MenuBar.

Canvases have a natural (x, y) coordinate system. The x-coordinate measures the distance in pixels from the left edge of the canvas; the y-coordinate measures the distance of the pixel from the top of the canvas. As we move to the right the x-coordinate increases; as we move DOWN the y-coordinate increases. This surprises some people who remember y-coordinates increasing as you move up in the traditional Cartesian coordinates in analytic geometry.

Canvas objects have methods for drawing various shapes. The most common shapes are rectangles (which include squares), ovals (including circles) and polygons. For rectangles you must give first the coordinates of the upper-left corner of the rectangle, then the coordinate of the lower-right corner. The default drawing color is black, but you can change this to any other color by giving the fill parameter of the drawing command the name of the color to draw.

For example, the following function draws a rectangle with upper-left corner at (x0, y0) and lower-right corner at (x1, y1):

```
def Rectangle(x0, y0, x1, y1, color):
    canvas.create_rectangle(x0, y0, x1, y1, fill=color)
```

We might call this with:

```
Rectangle(300, 300, 400, 450,    red    )
```

There is a corresponding canvas.create_oval( ) method that draws ellipses. This takes the same arguments as canvas.create_rectangle( ): the coordinates of the upper-left and lower-right corners, then fill −color. Since we are used to thinking of a circle as points a fixed distance from the center of the circle, we can write a function that takes the center and radius of the circle and builds the rectangle that surrounds the circle. Here is such a function:

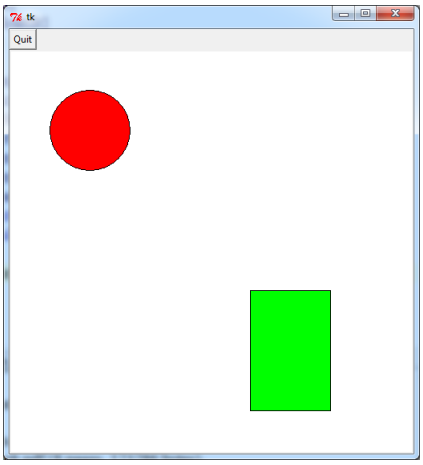
```
def Circle(x, y, radius, color):
    canvas.create_oval(x-radius, y-radius, \
                       x+radius, y+radius, fill=color)
```

Altogether, the following program will create a canvas and draw a circle and square on it:

```
from tkinter import *

class GUI(Frame):
    def __init__(self):
        Frame.__init__(self, None)
        self.grid()

        MenuBar = Frame(self)
        MenuBar.grid(row = 0, column = 0, sticky=W)

        QuitButton=Button(MenuBar,text="Quit",command=self.quit)
        QuitButton.grid(row = 0, column = 0)

        global canvas
        canvas=Canvas(self, width=500, height=500,\
          background="white")
        canvas.grid(row=1, column=0)


def Circle(x, y, radius, color):
    canvas.create_oval(x-radius, y-radius, \
          x+radius, y+radius, fill=color)
    canvas.update()

def Rectangle(x0, y0, x1, y1, color):
    canvas.create_rectangle(x0, y0, x1, y1, fill=color)

def main():
    window = GUI()
    Circle(100,100, 50, "red" )
    Rectangle(300, 300, 400, 450, "green" )
    window.mainloop()

main()
```

Program 9.3.1: Simple Drawing Program

Here is a picture of the canvas this creates:




The canvas.create_rectangle( ) and similar drawing methods return a value that can be used to manipulate the shape has been drawn. In particular, if we draw a 100x100 green rectangle with

```
r =canvas.create_rectangle(100,150,200,250, fill="green")
```

then we can move this rectangle 10 units horizontally and 30 vertically with

```
canvas.move(r, 10, 30)
```

We can change its color to red with

```
canvas.itemconfigure(r, fill="red")
```

Since this allows us to give functionality to the shapes, it makes sense to represent each category of shapes by a class. In the following code we make classes Rectangle, Square, Oval, Circle and Polygon. To avoid repeating the same code for each class, we make a top-level Shape class that contains all of the uniform code. Each of our basic shapes is a subclass of Shape, and each is constructed with only the code that differs from the uniform Shape code.

```
from tkinter import *

class GUI(Frame):
    def __init__(self):
        Frame.__init__(self, None)
        self.grid()

        MenuBar = Frame(self)
        MenuBar.grid(row = 0, column = 0, sticky=W)

        QuitButton=Button(MenuBar,text="Quit",command=self.quit)
        QuitButton.grid(row = 0, column = 0)

        global canvas
        canvas = Canvas(self, width=500, height=500, \
          background="white")
        canvas.grid(row=1, column=0)

class Shape:
    def __init__(self, vertices, color):
        self.color = color
        self.vertices = vertices
        self.my_shape = None

    def Moveto(self, a, b):
        # This moves the shape to point (a,b)
        [x, y] = self.vertices[0]
        d0 = a-x
        d1 = b-y
        self.Moveby(d0, d1)

    def Moveby(self, a, b):
        # This moves the shape a units horizontally
        # and b units vertically.
        canvas.move(self.my_shape, a, b)
        canvas.update()
        for v in self.vertices:
            v[0] = v[0] + a
            v[1] = v[1] + b
```

Program 9.3.2: Shape Classes

```
def ChangeColor(self, color):
        # This changes the shape's color.
        # Possible colors include "white", "black",
        # "red", "green", "blue", "yellow", "cyan",
        # "magenta", "light green", etc.
        canvas.itemconfigure(self.my_shape, fill = color )
        canvas.update()
        self.color = color

    def delete(self):
        canvas.delete(self.my_shape)

class Rectangle(Shape):
    def __init__(self, x, y, x1, y1, color):
        # This creates a square with corners (x, y) and (x1, y1)
        Shape.__init__(self, [[x, y], [x1, y1]], color)
        self.my_shape = canvas.create_rectangle(x, y, x1, y1, \
          fill = color)

    def pos(self):
        return (self.vertices[0][0], self.vertices[0][1])

class Oval(Shape):
    def __init__(self, x, y, hrad, vrad, color):
        # This creates an oval centered at (x, y)
        # with horizontal radius hrad and
        # vertical radius vrad
        Shape.__init__(self, [ [x-hrad, y-vrad], \
          [x+hrad, y+vrad] ], color)
        v0 = self.vertices[0]
        v1 = self.vertices[1]
        self.my_shape = canvas.create_oval(v0[0], v0[1], \
          v1[0], v1[1], fill = color)

    def pos(self):
        v0 = self.vertices[0]
        v1 = self.vertices[1]
        return ( (v0[0]+v1[0])/2, (v0[1]+v1[1])/2)

class Square(Rectangle):
    def __init__(self, x, y, side, color):
        # This creates a square with upper left corner at (x, y)
        # and length side
        Rectangle.__init__(self, x, y, x+side, y+side, color)
```

Program 9.3.2: Shape Classes, continued

```
class Circle(Oval):
    # This creates a circle centered at (x, y)
    # with the given radius
    def __init__(self, x, y, radius, color):
        Oval.__init__(self, x, y, radius, radius, color)

class Polygon(Shape):
# To create a polygon make a list of its vertices
# (each vertext itself is a list [x, y];
# you need a list of these)
# and call canvas.create_polygon().
# This returns a reference to the polygon
# that you will need if you ever want to
# change or delete it.
    def __init__(self,vertices, color):
        Shape.__init__(self, vertices, color)
        self.my_shape=canvas.create_polygon(vertices,fill=color)
        canvas.update()
        self.pos = (vertices[0][0], vertices[0][1])

def main():
    window = GUI()
    Circle(250,250, 100, "red" )
    window.mainloop()

main()
```

Program 9.3.2: Shape Classes, continued