9.2 Widgets with Values

A variety of tk widgets allow the user to enter values that are accessible to the program. In this section we focus on two of these: scales, which have sliders that allow the user to choose a value out of a range of values, and entry boxes, that allow the use to type a value. These two classes of widgets will handle most of the data entry situations you are likely to encounter.

Both of these widgets make data available to the program via *control variables*. We used a control variable in section 8.1 when we wanted to make the text of a label change according to input from our program. The control variable mechanism provides a uniform way the system can pass data between your program and its user interface. There are three classes of control variables:

- IntVar( ), which holds an integer value
- DoubleVar( ), which holds a floating point value (*double* is an old C term for a certain kind of float point value)
- StringVar( ), which holds a string.

The datum held by these objects is stored in a variable called value. There are two methods for objects of each class: get( ) returns the value stored in the object, while **set**(v) stores v as the value of the object.

For example, we we want to make an integer control variable and give it the value 8, we would use the code

```
numVertices = IntVar( )
numVertices.set(8)
```

If at some future point we wanted to retrieve the value stored in numVertices, we would say

```
n = numVertices.get( )
```

The Scale class is used to make widgets that allow the user to select one value out of a range. The constructor for this class has many defaulted parameters; here is a typical call, with the only parameters we usually need to give values to

```
Scale( parent, \
          from_ = <low end of range>,\
          to = <high end of range>, \
          orient = <HORIZONTAL or VERTICAL>,  \
          variable = <control variable> )
```

As with all of the widgets, the first argument is the parent window in which the widget lives. We save the value returned by this constructor, since we need it for the grid( ) method; the widget is not visible until we place it in its grid location.

For example,

```
numVertices = IntVar()
vScale = Scale(MenuBar,from_=1,to=100, \
     orient=HORIZONTAL,variable= numVertices)
vScale.grid(row=0, column=2)
```

You may attach a callback function to a scale, but if you do it will be continuously called as the user drags the scales slider. This may be what you want, but in many situations you want to wait until the user has finished with the scale before taking action. If this is your choice, use the code above for the Scale widget and provide a button next to the scale for the user to click after the scale is adjusted. The buttons callback can refer to the scales control variable to see the value the scale is set to.

If you do wish to use a callback function, you should assign it to the command parameter in the Scale constructor. For example,

```
numVertices = IntVar()
vScale = Scale(MenuBar, from_=1, to=100, \
     orient=HORIZONTAL, variable= numVertices, \
     command=Draw)
```

The callback function for a scale must take one argument, which will be the scales current value. For technical reasons this argument is given a string value; if you want it to be an integer you must convert it.

For example, the following might be the start of the code for the Draw function in the Scale constructor above:

```
def Draw(string_n):
        n = str(string_n)
        < code to draw a polygon with n vertices>
```

Again, a callback function is called continuously as the user drags the Scale widget. If the widget is currently set to 6 and the user drags it to 10, this function will be called for values 7, 8, 9, and 10. In a larger range the function will be called continuously, but the system might not be fast enough to keep up with the users dragging. You are guaranteed, however, that it will be called for the value the dragging stops at.

An Entry widget serves a similar purpose, but for the entry of strings. This gives the user a text box into which to type. One difference between the Entry widget and other widgets is that nothing signals when the user is finished typing. We will later see a way to use the <Return>-key as a signal, but for now we will just use a button. The user can enter a string into the Entry box, and click the button as a signal that the entry is ready.

Because there is no callback function, the code to create an Entry box is particularly simple:

```
Entry( parent, variable =<control variable> )
```

As with all of the widgets, the first argument to this constructor is the window in which it will be placed. The only other argument is the control variable that will hold the text the user enters into the box. This must be of type StringVar.

Here is code to create an entry box and a button, followed by code for the buttons callback:

```
global numberOfVertices
numberOfVertices = StringVar()
t = Entry(MenuBar, textvar = numberOfVertices)
t.grid(row=0, column=1)

DrawButton =  Button(MenuBar, text = "Draw", \
     command = self.Draw)
DrawButton.grid(row=0, column = 2 )

def Draw(self)
     n = int(numberOfVertices.get())
     <code to draw a polygon with n vertices>
```

The following program illustrates these ideas. This is a complete program for drawing a number of circles given by the user. The user has two choices for input: either a Scale widget or an Entry box. Each is accompanied by a button whose callback function handles the actual drawing. Dont worry about the code for drawing circles; we will discuss that in section 8.3.

```
from tkinter import *
from random 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)

        self.numCircles = IntVar()
        s = Scale(MenuBar, from_ = 1, to=50, \
          orient=HORIZONTAL, variable = self.numCircles)
        s.grid(row=0, column = 1)
        lab = Label(MenuBar, text="Number of Circles" )
        lab.grid(row=1, column=1)

        sButton = Button(MenuBar, text="Draw", \
          command = self.DrawCircles)
        sButton.grid(row = 0, column = 2)

        self.numCircles2 = StringVar()
        e = Entry(MenuBar, textvar = self.numCircles2)
        e.grid(row = 0, column = 3)
        lab2 = Label(MenuBar, text = "Number of Circles" )
        lab2.grid(row = 1, column = 3)

        sButton2 = Button(MenuBar, text = "Draw", \
          command = self.DrawCircles2)
        sButton2.grid(row = 0, column = 4)

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

Program 9.2.1: Circle Drawer, first part

```
    def DrawCircles(self):
        canvas.delete("all")
        n = self.numCircles.get()
        for x in range(0, n):
            RandomCircle()

    def DrawCircles2(self):
        canvas.delete("all")
        n = int( self.numCircles2.get() )
        for x in range(0, n):
            RandomCircle()

class Circle:
    def __init__(self, x, y, radius, color):
        self.my_shape = canvas.create_oval(x-radius, y-radius,
          x+radius, y+radius, fill = color)

def RandomCircle():
    x = randint(1, 500)
    y = randint(1, 500)
    radius = randint(5, 50)
    colors = ["red", "green", "blue", "yellow", "purple"]
    color = colors[randint(0, 4)]
    Circle(x, y, radius, color)

def main():
    window = GUI()
    window.mainloop()

main()
```

Program 9.2.1: Circle Drawer, conclusion