Appendix H: Complete Programs

20

20.1 The Draw Program

This is the sample drawing application from the first chapter. It illustrates the use of the tkinter library including many widgets and mouse handling. This program can be downloaded from the text's website.

```
# The imports include turtle graphics and tkinter modules.
# The colorchooser and filedialog modules let the user
# pick a color and a filename.
import turtle
import tkinter
import tkinter.colorchooser
import tkinter.filedialog
import xml.dom.minidom

# The following classes define the different commands that
# are supported by the drawing application.
class GoToCommand:
    def __init__(self,x,y,width=1,color="black"):
        self.x = x
        self.y = y
        self.width = width
        self.color = color

    # The draw method for each command draws the command
    # using the given turtle
    def draw(self,turtle):
        turtle.width(self.width)
        turtle.pencolor(self.color)
        turtle.goto(self.x,self.y)

    # The __str__ method is a special method that is called
    # when a command is converted to a string. The string
    # version of the command is how it appears in the graphics
    # file format.
    def __str__(self):
        return '<Command x="' + str(self.x) +'" y="' + str(self.y) + \
               '" width="' + str(self.width) \
               +'" color="' + self.color +'">GoTo</Command>'

class CircleCommand:
    def __init__(self,radius, width=1,color="black"):
        self.radius = radius
        self.width = width
```


K.D. Lee and S. Hubbard, *Data Structures and Algorithms with Python*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-13072-9_20

```
39          self.color = color
40
41      def draw(self,turtle):
42          turtle.width(self.width)
43          turtle.pencolor(self.color)
44          turtle.circle(self.radius)
45
46      def __str__(self):
47          return '<Command radius="' + str(self.radius) +'" width="' + \
48                 str(self.width) +'" color="' + self.color +'">Circle</Command>'
49
50  class BeginFillCommand:
51      def __init__(self,color):
52          self.color = color
53
54      def draw(self,turtle):
55          turtle.fillcolor(self.color)
56          turtle.begin_fill()
57
58      def __str__(self):
59          return '<Command color="' + self.color +'">BeginFill</Command>'
60
61  class EndFillCommand:
62      def __init__(self):
63          pass
64
65      def draw(self,turtle):
66          turtle.end_fill()
67
68      def __str__(self):
69          return "<Command>EndFill</Command>"
70
71  class PenUpCommand:
72      def __init__(self):
73          pass
74
75      def draw(self,turtle):
76          turtle.penup()
77
78      def __str__(self):
79          return "<Command>PenUp</Command>"
80
81  class PenDownCommand:
82      def __init__(self):
83          pass
84
85      def draw(self,turtle):
86          turtle.pendown()
87
88      def __str__(self):
89          return "<Command>PenDown</Command>"
90
91  # This is the PyList container object. It is meant to hold a
92  class PyList:
93      def __init__(self):
94          self.gcList = []
95
96      # The append method is used to add commands to the sequence.
97      def append(self,item):
98          self.gcList = self.gcList + [item]
99
100     # This method is used by the undo function. It slices the sequence
101     # to remove the last item
102     def removeLast(self):
103         self.gcList = self.gcList[:-1]
104
105     # This special method is called when iterating over the sequence.
106     # Each time yield is called another element of the sequence is returned
107     # to the iterator (i.e. the for loop that called this.)
```

```
108         def __iter__(self):
109             for c in self.gcList:
110                 yield c
111 
112         # This is called when the len function is called on the sequence.
113         def __len__(self):
114             return len(self.gcList)
115 
116     # This class defines the drawing application. The following line says that
117     # the DrawingApplication class inherits from the Frame class. This means
118     # that a DrawingApplication is like a Frame object except for the code
119     # written here which redefines/extends the behavior of a Frame.
120     class DrawingApplication(tkinter.Frame):
121         def __init__(self, master=None):
122             super().__init__(master)
123             self.pack()
124             self.buildWindow()
125             self.graphicsCommands = PyList()
126 
127         # This method is called to create all the widgets, place them in the GUI,
128         # and define the event handlers for the application.
129         def buildWindow(self):
130 
131             # The master is the root window. The title is set as below.
132             self.master.title("Draw")
133 
134             # Here is how to create a menu bar. The tearoff=0 means that menus
135             # can't be separated from the window which is a feature of tkinter.
136             bar = tkinter.Menu(self.master)
137             fileMenu = tkinter.Menu(bar,tearoff=0)
138 
139             # This code is called by the "New" menu item below when it is selected.
140             # The same applies for loadFile, addToFile, and saveFile below. The
141             # "Exit" menu item below calls quit on the "master" or root window.
142             def newWindow():
143                 # This sets up the turtle to be ready for a new picture to be
144                 # drawn. It also sets the sequence back to empty. It is necessary
145                 # for the graphicsCommands sequence to be in the object (i.e.
146                 # self.graphicsCommands) because otherwise the statement:
147                 # graphicsCommands = PyList()
148                 # would make this variable a local variable in the newWindow
149                 # method. If it were local, it would not be set anymore once the
150                 # newWindow method returned.
151                 theTurtle.clear()
152                 theTurtle.penup()
153                 theTurtle.goto(0,0)
154                 theTurtle.pendown()
155                 screen.update()
156                 screen.listen()
157                 self.graphicsCommands = PyList()
158 
159             fileMenu.add_command(label="New",command=newWindow)
160 
161             # The parse function adds the contents of an XML file to the sequence.
162             def parse(filename):
163                 xmldoc = xml.dom.minidom.parse(filename)
164 
165                 graphicsCommandsElement = xmldoc.getElementsByTagName("GraphicsCommands")[0]
166 
167                 graphicsCommands = graphicsCommandsElement.getElementsByTagName("Command")
168 
169                 for commandElement in graphicsCommands:
170                     print(type(commandElement))
171                     command = commandElement.firstChild.data.strip()
172                     attr = commandElement.attributes
173                     if command == "GoTo":
174                         x = float(attr["x"].value)
175                         y = float(attr["y"].value)
176                         width = float(attr["width"].value)
```

```
177                     color = attr["color"].value.strip()
178                     cmd = GoToCommand(x,y,width,color)
179
180                 elif command == "Circle":
181                     radius = float(attr["radius"].value)
182                     width = float(attr["width"].value)
183                     color = attr["color"].value.strip()
184                     cmd = CircleCommand(radius,width,color)
185
186                 elif command == "BeginFill":
187                     color = attr["color"].value.strip()
188                     cmd = BeginFillCommand(color)
189
190                 elif command == "EndFill":
191                     cmd = EndFillCommand()
192
193                 elif command == "PenUp":
194                     cmd = PenUpCommand()
195
196                 elif command == "PenDown":
197                     cmd = PenDownCommand()
198                 else:
199                     raise RuntimeError("Unknown Command: " + command)
200
201                 self.graphicsCommands.append(cmd)
202
203         def loadFile():
204
205             filename = tkinter.filedialog.askopenfilename(title="Select a Graphics File")
206
207             newWindow()
208
209             # This re-initializes the sequence for the new picture.
210             self.graphicsCommands = PyList()
211
212             # calling parse will read the graphics commands from the file.
213             parse(filename)
214
215             for cmd in self.graphicsCommands:
216                 cmd.draw(theTurtle)
217
218             # This line is necessary to update the window after the picture is drawn.
219             screen.update()
220
221
222         fileMenu.add_command(label="Load ...",command=loadFile)
223
224         def addToFile():
225             filename = tkinter.filedialog.askopenfilename(title="Select a Graphics File")
226
227             theTurtle.penup()
228             theTurtle.goto(0,0)
229             theTurtle.pendown()
230             theTurtle.pencolor("#000000")
231             theTurtle.fillcolor("#000000")
232             cmd = PenUpCommand()
233             self.graphicsCommands.append(cmd)
234             cmd = GoToCommand(0,0,1,"#000000")
235             self.graphicsCommands.append(cmd)
236             cmd = PenDownCommand()
237             self.graphicsCommands.append(cmd)
238             screen.update()
239             parse(filename)
240
241             for cmd in self.graphicsCommands:
242                 cmd.draw(theTurtle)
243
244
245
```

```
246             screen.update()
247
248         fileMenu.add_command(label="Load Into ...",command=addToFile)
249
250         # The write function writes an XML file to the given filename
251         def write(filename):
252             file = open(filename, "w")
253             file.write('<?xml version="1.0" encoding="UTF-8" standalone="no" ?>\n')
254             file.write('<GraphicsCommands>\n')
255             for cmd in self.graphicsCommands:
256                 file.write('    '+str(cmd)+"\n")
257
258             file.write('</GraphicsCommands>\n')
259
260             file.close()
261
262         def saveFile():
263             filename = tkinter.filedialog.asksaveasfilename(title="Save Picture As...")
264             write(filename)
265
266         fileMenu.add_command(label="Save As...",command=saveFile)
267
268
269         fileMenu.add_command(label="Exit",command=self.master.quit)
270
271         bar.add_cascade(label="File",menu=fileMenu)
272
273         # This tells the root window to display the newly created menu bar.
274         self.master.config(menu=bar)
275
276         # Here several widgets are created. The canvas is the drawing area on
277         # the left side of the window.
278         canvas = tkinter.Canvas(self,width=600,height=600)
279         canvas.pack(side=tkinter.LEFT)
280
281         # By creating a RawTurtle, we can have the turtle draw on this canvas.
282         # Otherwise, a RawTurtle and a Turtle are exactly the same.
283         theTurtle = turtle.RawTurtle(canvas)
284
285         # This makes the shape of the turtle a circle.
286         theTurtle.shape("circle")
287         screen = theTurtle.getscreen()
288
289         # This causes the application to not update the screen unless
290         # screen.update() is called. This is necessary for the ondrag event
291         # handler below. Without it, the program bombs after dragging the
292         # turtle around for a while.
293         screen.tracer(0)
294
295         # This is the area on the right side of the window where all the
296         # buttons, labels, and entry boxes are located. The pad creates some empty
297         # space around the side. The side puts the sideBar on the right side of the
298         # this frame. The fill tells it to fill in all space available on the right
299         # side.
300         sideBar = tkinter.Frame(self,padx=5,pady=5)
301         sideBar.pack(side=tkinter.RIGHT, fill=tkinter.BOTH)
302
303         # This is a label widget. Packing it puts it at the top of the sidebar.
304         pointLabel = tkinter.Label(sideBar,text="Width")
305         pointLabel.pack()
306
307         # This entry widget allows the user to pick a width for their lines.
308         # With the widthSize variable below you can write widthSize.get() to get
309         # the contents of the entry widget and widthSize.set(val) to set the value
310         # of the entry widget to val. Initially the widthSize is set to 1. str(1) is
311         # needed because the entry widget must be given a string.
312         widthSize = tkinter.StringVar()
313         widthEntry = tkinter.Entry(sideBar,textvariable=widthSize)
314         widthEntry.pack()
```

```
315         widthSize.set(str(1))
316
317         radiusLabel = tkinter.Label(sideBar,text="Radius")
318         radiusLabel.pack()
319         radiusSize = tkinter.StringVar()
320         radiusEntry = tkinter.Entry(sideBar,textvariable=radiusSize)
321         radiusSize.set(str(10))
322         radiusEntry.pack()
323
324         # A button widget calls an event handler when it is pressed. The circleHandler
325         # function below is the event handler when the Draw Circle button is pressed.
326         def circleHandler():
327             # When drawing, a command is created and then the command is drawn by calling
328             # the draw method. Adding the command to the graphicsCommands sequence means the
329             # application will remember the picture.
330             cmd = CircleCommand(float(radiusSize.get()), float(widthSize.get()), penColor.get())
331             cmd.draw(theTurtle)
332             self.graphicsCommands.append(cmd)
333
334             # These two lines are needed to update the screen and to put the focus back
335             # in the drawing canvas. This is necessary because when pressing "u" to undo,
336             # the screen must have focus to receive the key press.
337             screen.update()
338             screen.listen()
339
340         # This creates the button widget in the sideBar. The fill=tkinter.BOTH causes the button
341         # to expand to fill the entire width of the sideBar.
342         circleButton = tkinter.Button(sideBar, text = "Draw␣Circle", command=circleHandler)
343         circleButton.pack(fill=tkinter.BOTH)
344
345         # The color mode 255 below allows colors to be specified in RGB form (i.e. Red/
346         # Green/Blue). The mode allows the Red value to be set by a two digit hexadecimal
347         # number ranging from 00-FF. The same applies for Blue and Green values. The
348         # color choosers below return a string representing the selected color and a slice
349         # is taken to extract the #RRGGBB hexadecimal string that the color choosers return.
350         screen.colormode(255)
351         penLabel = tkinter.Label(sideBar,text="Pen␣Color")
352         penLabel.pack()
353         penColor = tkinter.StringVar()
354         penEntry = tkinter.Entry(sideBar,textvariable=penColor)
355         penEntry.pack()
356         # This is the color black.
357         penColor.set("#000000")
358
359         def getPenColor():
360             color = tkinter.colorchooser.askcolor()
361             if color != None:
362                 penColor.set(str(color)[-9:-2])
363
364         penColorButton = tkinter.Button(sideBar, text = "Pick␣Pen␣Color", command=getPenColor)
365         penColorButton.pack(fill=tkinter.BOTH)
366
367         fillLabel = tkinter.Label(sideBar,text="Fill␣Color")
368         fillLabel.pack()
369         fillColor = tkinter.StringVar()
370         fillEntry = tkinter.Entry(sideBar,textvariable=fillColor)
371         fillEntry.pack()
372         fillColor.set("#000000")
373
374         def getFillColor():
375             color = tkinter.colorchooser.askcolor()
376             if color != None:
377                 fillColor.set(str(color)[-9:-2])
378
379         fillColorButton = \
380             tkinter.Button(sideBar, text = "Pick␣Fill␣Color", command=getFillColor)
381         fillColorButton.pack(fill=tkinter.BOTH)
382
383
```

```
384         def beginFillHandler():
385             cmd = BeginFillCommand(fillColor.get())
386             cmd.draw(theTurtle)
387             self.graphicsCommands.append(cmd)
388 
389         beginFillButton = tkinter.Button(sideBar, text = "Begin Fill", command=beginFillHandler)
390         beginFillButton.pack(fill=tkinter.BOTH)
391 
392         def endFillHandler():
393             cmd = EndFillCommand()
394             cmd.draw(theTurtle)
395             self.graphicsCommands.append(cmd)
396 
397         endFillButton = tkinter.Button(sideBar, text = "End Fill", command=endFillHandler)
398         endFillButton.pack(fill=tkinter.BOTH)
399 
400         penLabel = tkinter.Label(sideBar,text="Pen Is Down")
401         penLabel.pack()
402 
403         def penUpHandler():
404             cmd = PenUpCommand()
405             cmd.draw(theTurtle)
406             penLabel.configure(text="Pen Is Up")
407             self.graphicsCommands.append(cmd)
408 
409         penUpButton = tkinter.Button(sideBar, text = "Pen Up", command=penUpHandler)
410         penUpButton.pack(fill=tkinter.BOTH)
411 
412         def penDownHandler():
413             cmd = PenDownCommand()
414             cmd.draw(theTurtle)
415             penLabel.configure(text="Pen Is Down")
416             self.graphicsCommands.append(cmd)
417 
418         penDownButton = tkinter.Button(sideBar, text = "Pen Down", command=penDownHandler)
419         penDownButton.pack(fill=tkinter.BOTH)
420 
421         # Here is another event handler. This one handles mouse clicks on the screen.
422         def clickHandler(x,y):
423             # When a mouse click occurs, get the widthSize entry value and set the width of the
424             # pen to the widthSize value. The float(widthSize.get()) is needed because
425             # the width is a float, but the entry widget stores it as a string.
426             cmd = GoToCommand(x,y,float(widthSize.get()),penColor.get())
427             cmd.draw(theTurtle)
428             self.graphicsCommands.append(cmd)
429             screen.update()
430             screen.listen()
431 
432         # Here is how we tie the clickHandler to mouse clicks.
433         screen.onclick(clickHandler)
434 
435         def dragHandler(x,y):
436             cmd = GoToCommand(x,y,float(widthSize.get()),penColor.get())
437             cmd.draw(theTurtle)
438             self.graphicsCommands.append(cmd)
439             screen.update()
440             screen.listen()
441 
442         theTurtle.ondrag(dragHandler)
443 
444         # the undoHandler undoes the last command by removing it from the
445         # sequence and then redrawing the entire picture.
446         def undoHandler():
447             if len(self.graphicsCommands) > 0:
448                 self.graphicsCommands.removeLast()
449                 theTurtle.clear()
450                 theTurtle.penup()
451                 theTurtle.goto(0,0)
452                 theTurtle.pendown()
```

```
453                 for cmd in self.graphicsCommands:
454                     cmd.draw(theTurtle)
455                 screen.update()
456                 screen.listen()
457 
458         screen.onkeypress(undoHandler, "u")
459         screen.listen()
460 
461 # The main function in our GUI program is very simple. It creates the
462 # root window. Then it creates the DrawingApplication frame which creates
463 # all the widgets and has the logic for the event handlers. Calling mainloop
464 # on the frames makes it start listening for events. The mainloop function will
465 # return when the application is exited.
466 def main():
467     root = tkinter.Tk()
468     drawingApp = DrawingApplication(root)
469 
470     drawingApp.mainloop()
471     print("Program Execution Completed.")
472 
473 if __name__ == "__main__":
474     main()
```

20.2 The Scope Program

This is the sample program from the first chapter that illustrates the use of scope within a program. This program can be downloaded from the text's website.

```
1  import math
2 
3 
4  PI = math.pi
5 
6  def area(radius):
7      global z
8      z = 6
9      theArea = PI * radius ** 2
10 
11     return theArea
12 
13 
14 def main():
15     global z
16 
17     historyOfPrompts = []
18     historyOfOutput = []
19 
20     def getInput(prompt):
21         x = input(prompt)
22         historyOfPrompts.append(prompt)
23 
24         return x
25 
26     def showOutput(val):
27         historyOfOutput.append(val)
28         print(val)
29 
30     rString = getInput("Please enter the radius of a circle:")
```