GUI Programming

Using Tkinter

Introduction

Tkinter enables you to develop GUI programs and is an excellent pedagogical tool for learning object-oriented programming.

- There are many GUI modules available for developing GUI programs in Python.
- We have already discussed the turtle module for drawing geometric shapes.
- Turtle is easy to use and is an effective pedagogical tool for introducing the fundamentals of programming to beginners.
- However, we cannot use turtle to create graphical user interfaces.

Tkinter

Tkinter (pronounced T-K-Inter) is short for "Tk interface." Tk is a GUI library used by many programming languages for developing GUI programs on Windows, Mac, and UNIX. Tkinter provides an interface for Python programmers to use the Tk GUI library, and it is the de-facto standard for developing GUI programs in Python.

The `tkinter` module contains the classes for creating GUIs. The `Tk` class creates a window for holding GUI widgets (i.e., visual components).

SimplyGUI.py

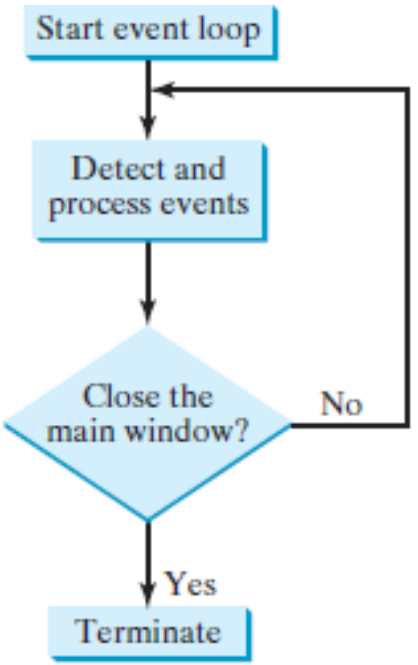
```
1  from tkinter import *  # Import all definitions from tkinter
2
3  window = Tk()  # Create a window
4  label = Label(window, text = "Welcome to Python")  # Create a label
5  button = Button(window, text = "Click Me")  # Create a button
6  label.pack()  # Place the label in the window
7  button.pack()  # Place the button in the window
8
9  window.mainloop()  # Create an event loop
```

- *Tk() creates an instance of a window.*
- *Label and Button are Python Tkinter widget classes for creating labels and buttons.*
- *The first argument of a widget class is always the parent container (i.e., the container in which the widget will be placed). The statement (line 4)*

```
label = Label(window, text = "Welcome to Python")
```

- Tkinter GUI programming is event driven.
- After the user interface is displayed, the program waits for user interactions such as mouse clicks and key presses. (line 9)

  window.mainloop()
- The statement creates an event loop. The event loop processes events continuously until you close the main window.



A Tkinter GUI program listens and processes events in a continuous loop.

Processing Events

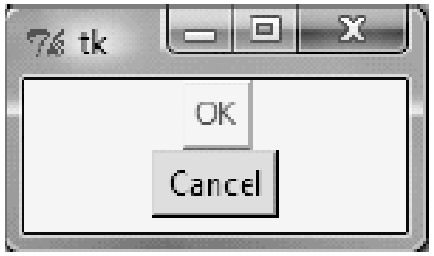
A Tkinter widget can be bound to a function, which is called when an event occurs.

- The Button widget is a good way to demonstrate the basics of event-driven programming.
- In this, when the user clicks a button, program should process the event.
- This action is enabled by defining a processing function and binding the function to the button and it is shown in next slide.

ProcessButtonEvent.py

```
1  from tkinter import * # Import all definitions from tkinter
2
3  def processOK():
4      print("OK button is clicked")
5
6  def processCancel():
7      print("Cancel button is clicked")
8
9  window = Tk() # Create a window
10 btOK = Button(window, text = "OK", fg = "red", command = processOK)
11 btCancel = Button(window, text = "Cancel", bg = "yellow",
12                    command = processCancel)
13 btOK.pack() # Place the OK button in the window
14 btCancel.pack() # Place the Cancel button in the window
15
16 window.mainloop() # Create an event loop
```

When you run the program, two buttons appear, & command window shown as:



- The program defines the functions processOK and processCancel (lines 3–7).
- These functions are bound to the buttons when the buttons are constructed.
- These functions are known as callback functions, or handlers.

```
btOK = Button(window, text = "OK", fg = "red", command = processOK)
```

- The above statement (line 10) binds the OK button to the processOK function, which will be called when the button is clicked.
- The fg option specifies the button's foreground color and the bg option specifies its background color.
- By default, fg is black and bg is gray for all widgets.

The Widget Classes

Tkinter's GUI classes define common GUI widgets such as buttons, labels, radio buttons, check buttons, entries, canvases, and others.

Following table describes the core widget classes Tkinter provides.

Tkinter Widget Classes

Widget Class	*Description*
`Button`	A simple button, used to execute a command.
`Canvas`	Structured graphics, used to draw graphs and plots, create graphics editors, and implement custom widgets.
`Checkbutton`	Clicking a check button toggles between the values.
`Entry`	A text entry field, also called a text field or a text box.
`Frame`	A container widget for containing other widgets.
`Label`	Displays text or an image.
`Menu`	A menu pane, used to implement pull-down and popup menus.
`Menubutton`	A menu button, used to implement pull-down menus.
`Message`	Displays a text. Similar to the label widget, but can automatically wrap text to a given width or aspect ratio.
`Radiobutton`	Clicking a radio button sets the variable to that value, and clears all other radio buttons associated with the same variable.
`Text`	Formatted text display. Allows you to display and edit text with various styles and attributes. Also supports embedded images and windows.

- There are many options for creating widgets from these classes.
- The first argument is always the parent container. We can specify a foreground color, background color, font, and cursor style when constructing a widget.

Color

- For a color, use either a color name (such as red, yellow, green, blue, white, black, purple) or explicitly specify RGB color components by using a string #RRGGBB.

Font

- A font includes , the font name, size, and style as follows:
  - *Times 10 bold*
  - *Helvetica 10 bold italic*
  - *CourierNew 20 bold italic*
  - *Courier 20 bold italic overstrike underline*

Text Formatting

- By default, the text in a label or a button is centered.
- We can change its alignment by using the justify option with the named constants LEFT, CENTER, or RIGHT.
- Multiple lines text can be inserted by the newline character \n.

Mouse Curser

- A particular style of mouse cursor is specified by using the cursor option with string values such as arrow (the default), circle, cross, plus, or some other shape.

Change Properties

- The property of widget can be specified during construct such as fg, bg, font, cursor, text, and command in the constructor. Later, widget's properties can be changed as follows:

```
widgetName["propertyName"] = newPropertyValue
```

Canvas

- We use the Canvas widget for displaying shapes.
- We can use the methods create_rectangle, create_oval, create_arc, create_polygon, or create_line to draw a rectangle, oval, arc, polygon, or line on a canvas.

- The next slide shows how to use the Canvas widget.

- The program displays a rectangle, an oval, an arc, a polygon, a line, and a text string.

- The objects are all controlled by buttons.

CanvasDemo.py

```
1  from tkinter import * # Import all definitions from tkinter
2
3  class CanvasDemo:
4      def __init__(self):
5          window = Tk() # Create a window
6          window.title("Canvas Demo") # Set title
7
8          # Place canvas in the window
9          self.canvas = Canvas(window, width = 200, height = 100,
10             bg = "white")
11         self.canvas.pack()
12
13         # Place buttons in frame
14         frame = Frame(window)
15         frame.pack()
16         btRectangle = Button(frame, text = "Rectangle",
17             command = self.displayRect)
```

```
18          btOval = Button(frame, text = "Oval",
19              command = self.displayOval)
20          btArc = Button(frame, text = "Arc",
21              command = self.displayArc)
22          btPolygon = Button(frame, text = "Polygon",
23              command = self.displayPolygon)
24          btLine = Button(frame, text = "Line",
25              command = self.displayLine)
26          btString = Button(frame, text = "String",
27              command = self.displayString)
28          btClear = Button(frame, text = "Clear",
29              command = self.clearCanvas)
30          btRectangle.grid(row = 1, column = 1)                    place buttons
31          btOval.grid(row = 1, column = 2)
32          btArc.grid(row = 1, column = 3)
33          btPolygon.grid(row = 1, column = 4)
34          btLine.grid(row = 1, column = 5)
35          btString.grid(row = 1, column = 6)
36          btClear.grid(row = 1, column = 7)
37
38          window.mainloop() # Create an event loop                 event loop
39
40      # Display a rectangle
```

```
41      def displayRect(self):
42          self.canvas.create_rectangle(10, 10, 190, 90, tags = "rect")    display rectangle
43
44      # Display an oval
45      def displayOval(self):
46          self.canvas.create_oval(10, 10, 190, 90, fill = "red",          display oval
47              tags = "oval")
48
49      # Display an arc
50      def displayArc(self):
51          self.canvas.create_arc(10, 10, 190, 90, start = 0,              display arc
52              extent = 90, width = 8, fill = "red", tags = "arc")
53
54      # Display a polygon
55      def displayPolygon(self):
56          self.canvas.create_polygon(10, 10, 190, 90, 30, 50,             display polygon
57              tags = "polygon")
58
59      # Display a line
60      def displayLine(self):
61          self.canvas.create_line(10, 10, 190, 90, fill = "red",          display line
62              tags = "line")
63          self.canvas.create_line(10, 90, 190, 10, width = 9,
64              arrow = "last", activefill = "blue", tags = "line")
```

```
65
66      # Display a string
67      def displayString(self):
68          self.canvas.create_text(60, 40, text = "Hi, I am a string",
69              font = "Times 10 bold underline", tags = "string")
70
71      # Clear drawings
72      def clearCanvas(self):
73          self.canvas.delete("rect", "oval", "arc", "polygon",
74              "line", "string")
75
76  CanvasDemo() # Create GUI
```

display string

clear canvas

create GUI

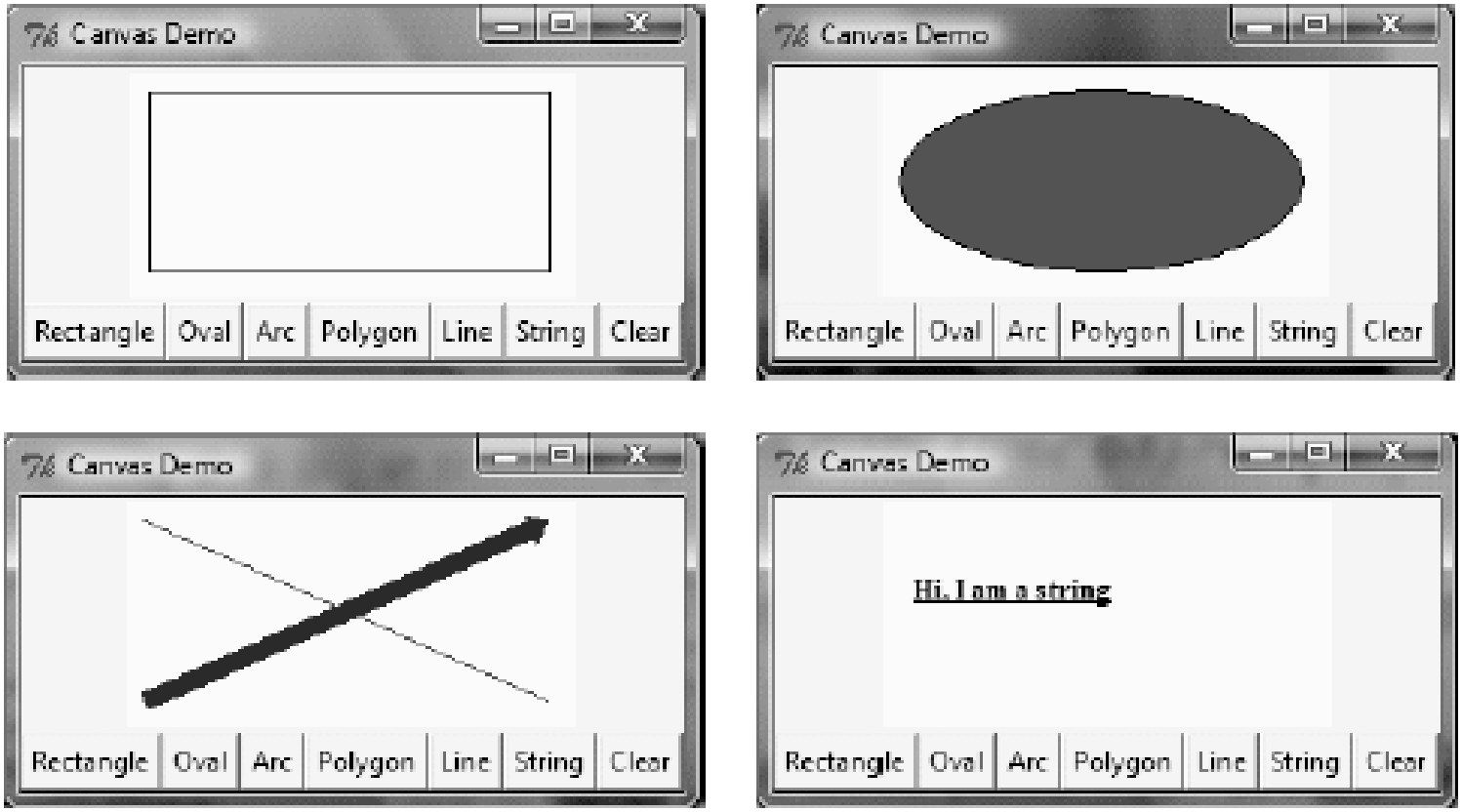
- The program creates a window (line 5) and sets its title (line 6).

- A Canvas widget is created within the window with a width of 200 pixels, a height of 100 pixels, and a background color of white (lines 9–10).

- Seven buttons—labeled with the text Rectangle, Oval, Arc, Polygon, Line, String, and Clear—are created (lines 16–29).

- The grid manager places the buttons in one row in a frame (lines 30–36).

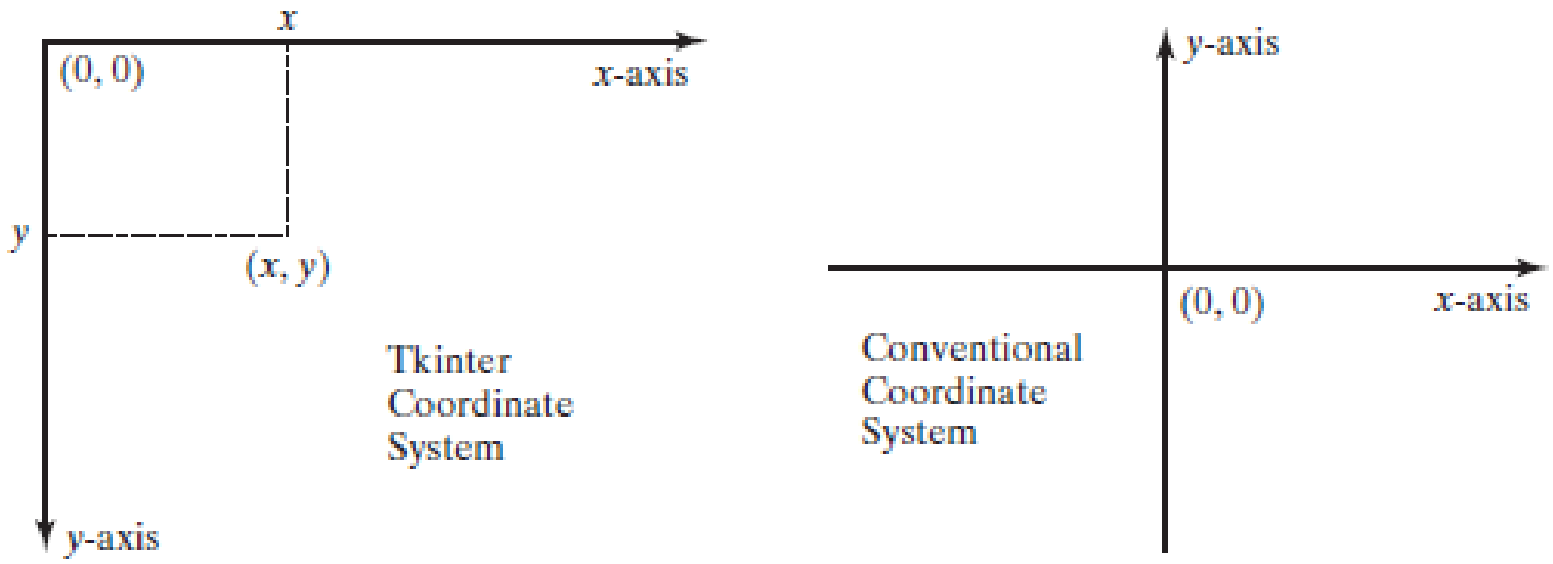
Output



The geometrical shapes and strings are drawn on the canvas.

Coordinate System

- To draw graphics, there is need to tell the widget where to draw.
- Each widget has its own coordinate system with the origin (0, 0) at the upper-left corner.
- The x-coordinate increases to the right, and the y-coordinate increases downward.
- Note that the Tkinter coordinate system differs from the conventional coordinate system, as shown in Figure in next slide.

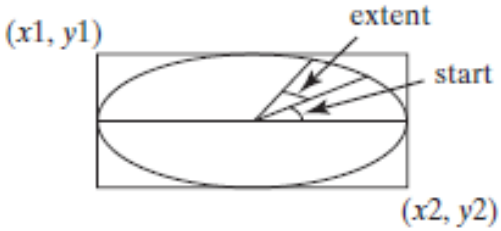


The Tkinter coordinate system is measured in pixels, with (0, 0) at its upper-left corner

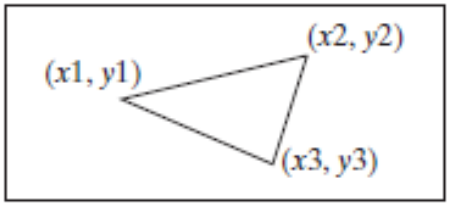
- The methods create_rectangle, create_oval, create_arc, create_polygon, and create_line (lines 42, 46, 51, 56, and 61) are used to draw rectangles, ovals, arcs, polygons, and lines, as illustrated in Figure at next slide.

canvas.create_rectangle(x1, y1, x2, y2)

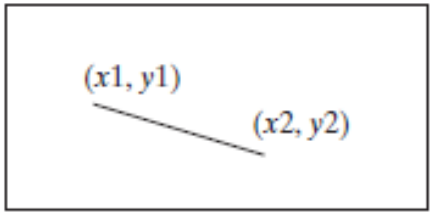
canvas.create_oval(x1, y1, x2, y2)



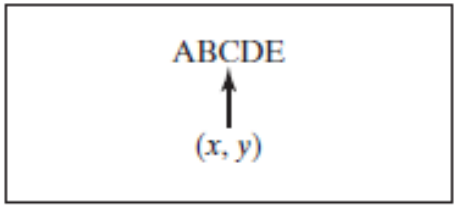
canvas.create_arc(x1, y1, x2, y2, start, extent)



canvas.create_polygon(x1, y1, x2, y2, x3, y3)



canvas.create_line(x1, y1, x2, y2)



canvas.create_text(x, y, text = "ABCDE")

The Canvas class contains the methods for drawing graphics.

- The create_text method is used to draw a text string (line 68).
- Note that the horizontal and vertical center of the text is displayed at (x, y) for create_text(x, y, text) as shown in previous Figure.
- All the drawing methods use the tags argument to identify the drawing.
- These tags are used in the delete method for clearing the drawing from the canvas (lines 73–74).

- The width argument can be used to specify the pen size in pixels for drawing the shapes (lines 52 and 63).
- The arrow argument can be used with create_line to draw a line with an arrowhead (line 64).
- The arrowhead can appear at the start, end, or both ends of the line with the argument value first, end, or both.
- The activefill argument makes the shape change color when you move the mouse over it (line 64).

Any Question

???