CHAPTER 13

GUI Programming

Topics

- **Graphical User Interfaces**
- **Using the `tkinter` Module**
- **Display Text with `Label` Widgets**
- **Organizing Widgets with Frames**
- **`Button` Widgets and Info Dialog Boxes**
- **Getting Input with the `Entry` Widget**
- **Using Labels as Output Fields**
- **Radio Buttons and Check Buttons**
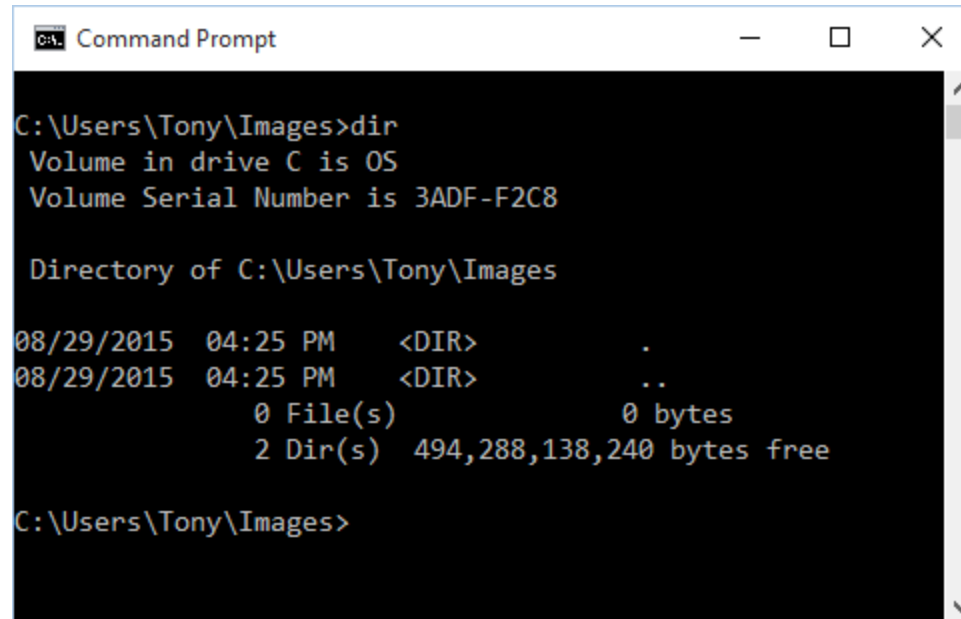- **Drawing Shapes with the `Canvas` Widget**

Graphical User Interfaces

- <u>User Interface</u>: the part of the computer with which the user interacts

- <u>Command line interface</u>: displays a prompt and the user types a command that is then executed

- <u>Graphical User Interface (GUI)</u>: allows users to interact with a program through graphical elements on the screen

Graphical User Interfaces (cont'd.)

- **A command line interface**

Graphical User Interfaces (cont'd.)

- **Dialog boxes**: **small windows that display information and allow the user to perform actions**
  - Responsible for most of the interaction through GUI
  - User interacts with graphical elements such as icons, buttons, and slider bars
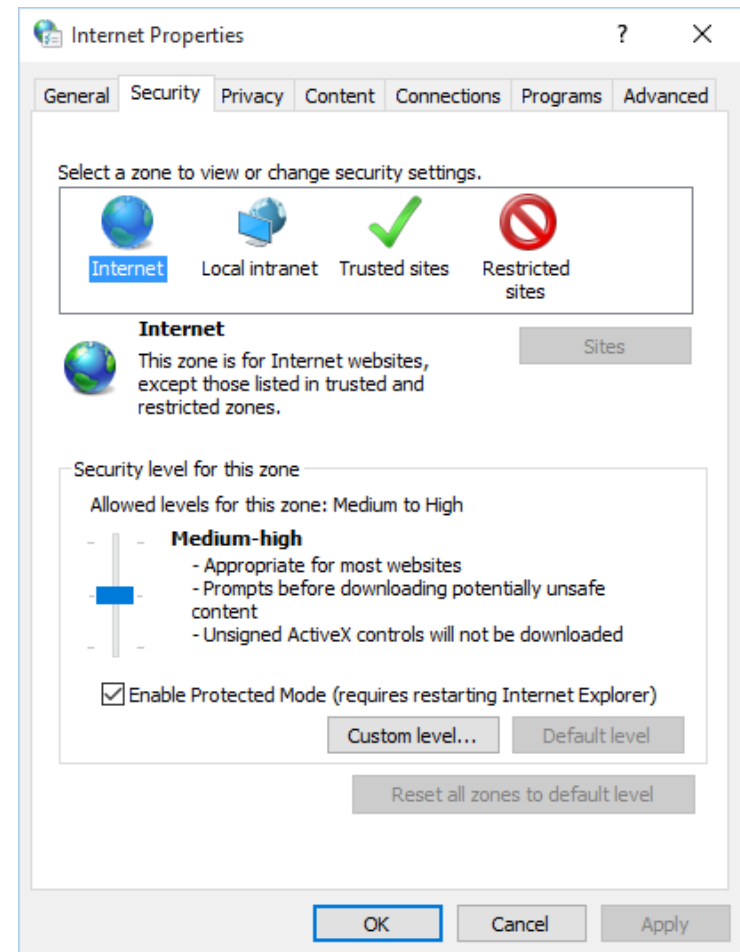
GUI Programs Are Event-Driven

- **In text-based environments, programs determine the order in which things happen**
  - The user can only enter data in the order requested by the program
- **GUI environment is event-driven**
  - The user determines the order in which things happen
    - User causes events to take place and the program responds to the events

Using the `tkinter` Module

- **No GUI programming features built into Python**
- **`tkinter` module: allows you to create simple GUI programs**
  - Comes with Python
- **Widget: graphical element that the user can interact with or view**
  - Presented by a GUI program

Table 13-1 tkinter Widgets

Widget	Description
Button	A button that can cause an action to occur when it is clicked.
Canvas	A rectangular area that can be used to display graphics.
Checkbutton	A button that may be in either the "on" or "off" position.
Entry	An area in which the user may type a single line of input from the keyboard.
Frame	A container that can hold other widgets.
Label	An area that displays one line of text or an image.
Listbox	A list from which the user may select an item
Menu	A list of menu choices that are displayed when the user clicks a Menubutton widget.
Menubutton	A menu that is displayed on the screen and may be clicked by the user
Message	Displays multiple lines of text.
Radiobutton	A widget that can be either selected or deselected. Radiobutton widgets usually appear in groups and allow the user to select one of several options.
Scale	A widget that allows the user to select a value by moving a slider along a track.
Scrollbar	Can be used with some other types of widgets to provide scrolling ability.
Text	A widget that allows the user to enter multiple lines of text input.
Toplevel	A container, like a Frame, but displayed in its own window.

Using the `tkinter` Module (cont'd.)

- **Programs that use `tkinter` do not always run reliably under IDLE**
  - For best results run them from operating system command prompt
- **Most programmers take an object-oriented approach when writing GUI programs**
  - `__init__` method builds the GUI
  - When an instance is created the GUI appears on the screen

Display Text with `Label` Widgets

- **<u>`Label` widget</u>: displays a single line of text in a window**
  - Made by creating an instance of `tkinter` module's `Label` class
  - Format:
    ```
    tkinter.Label(self.main_window,
                          text = 'my text')
    ```
    - First argument references the root widget, second argument shows text that should appear in label

Display Text with `Label` Widgets (cont'd.)

- <u>`pack` method</u>: determines where a widget should be positioned and makes it visible when the main window is displayed
  - Called for each widget in a window
  - Receives an argument to specify positioning
    - Positioning depends on the order in which widgets were added to the main window
    - Valid arguments: `side='top'`, `side='left'`, `side='right'`

Display Text with `Label` Widgets (cont'd.)

Figure 13-5 Window displayed by Program 13-3

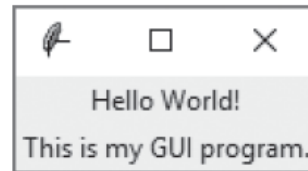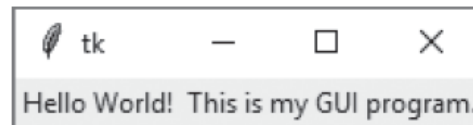**Figure 13-6** Window displayed by Program 13-4

Figure 13-7 Window displayed by Program 13-5
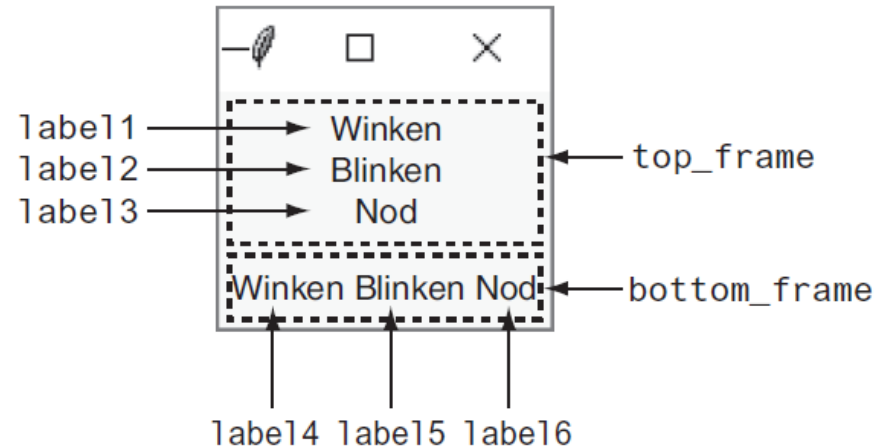
Organizing Widgets with Frames

- **`Frame` widget: container that holds other widgets**
  - Useful for organizing and arranging groups of widgets in a window
  - The contained widgets are added to the frame widget which contains them
    - Example:

```
tkinter.Label(self.top_frame,
              text = 'hi')
```

Organizing Widgets with Frames (cont'd.)

Figure 13-9 Arrangement of widgets

Button Widgets and Info Dialog Boxes

- **`Button` widget**: widget that the user can click to cause an action to take place
  - When creating a button can specify:
    - Text to appear on the face of the button
    - A callback function
- **Callback function**: function or method that executes when the user clicks the button
  - Also known as an event handler

Button Widgets and Info Dialog Boxes (cont'd.)

- **Info dialog box: a dialog box that shows information to the user**
  - Format for creating an info dialog box:
    - Import `tkinter.messagebox` module
    - `tkinter.messagebox.showinfo(`*`title`*`,` *`message`*`)`
      - *`title`* is displayed in dialog box's title bar
      - *`message`* is an informational string displayed in the main part of the dialog box

Button Widgets and Info Dialog Boxes (cont'd.)

Figure 13-10 The main window displayed by Program 13-7

Figure 13-11 The info dialog box displayed by Program 13-7
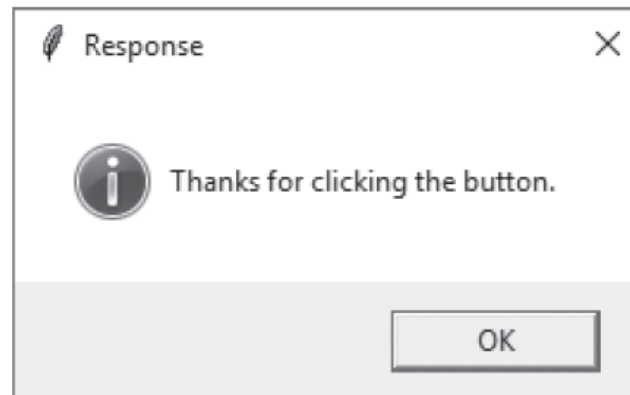
Creating a Quit Button

- **Quit button: closes the program when the user clicks it**

- **To create a quit button in Python:**
  - Create a `Button` widget
  - Set the root widget's `destroy` method as the callback function
    - When the user clicks the button the `destroy` method is called and the program ends

Getting Input with the `Entry` Widget

- **`Entry widget`: rectangular area that the user can type text into**
  - Used to gather input in a GUI program
  - Typically followed by a button for submitting the data
    - The button's callback function retrieves the data from the `Entry` widgets and processes it
  - `Entry` widget's `get` method: used to retrieve the data from an `Entry` widget
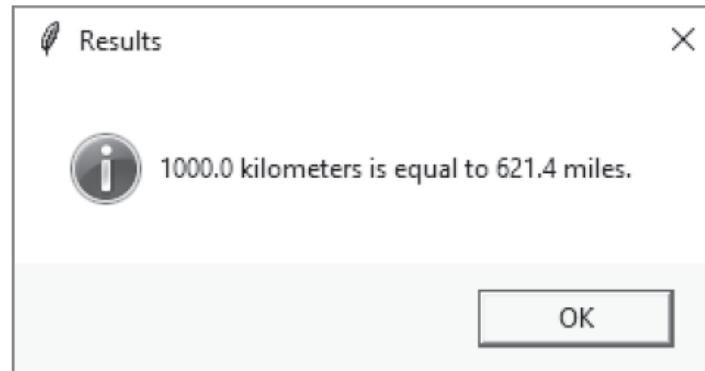   - Returns a string

Getting Input with the `Entry` Widget (cont'd.)

Figure 13-15   The info dialog box

① The user enters 1000 into the Entry widget and clicks the Convert button.
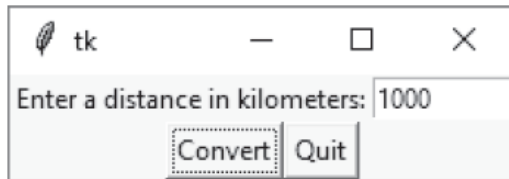
② This info dialog box is displayed.

Using Labels as Output Fields

- **Can use `Label` widgets to dynamically display output**
  - Used to replace info dialog box
  - Create empty `Label` widget in main window, and write code that displays desired data in the label when a button is clicked

Using Labels as Output Fields (cont'd.)

- **`StringVar` class: `tkinter` module class that can be used along with `Label` widget to display data**
  - Create `StringVar` object and then create `Label` widget and associate it with the `StringVar` object
  - Subsequently, any value stored in the `StringVar` object with automatically be displayed in the `Label` widget

Using Labels as Output Fields (cont'd.)

Figure 13-16 The window initially displayed
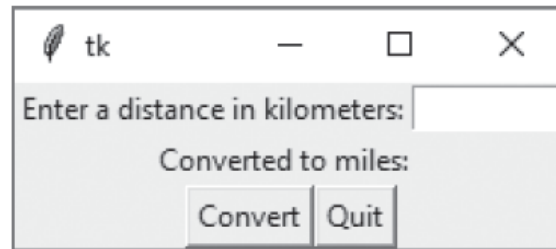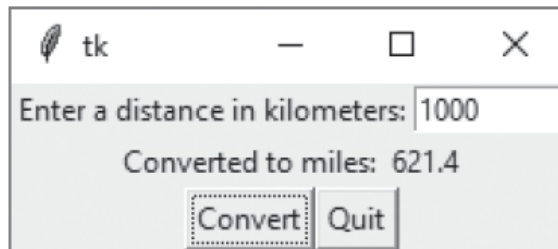

Figure 13-17 The window showing 1000 kilometers converted to miles

Radio Buttons and Check Buttons

- **Radio button: small circle that appears filled when it is selected and appears empty when it is deselected**
  - Useful when you want the user to select one choice from several possible options
- **Radiobutton widgets: created using tkinter module's Radiobutton class**
  - `Radiobutton` widgets are mutually exclusive
    - Only one radio button in a container may be selected at any given time

Radio Buttons and Check Buttons (cont'd)

- **`IntVar` class: a `tkinter` module class that can be used along with `Radiobutton` widgets**
  - Steps for use:
    - Associate group of `Radiobutton` widgets with the same `IntVar` object
    - Assign unique integer to each `Radiobutton`
    - When a `Radiobutton` widgets is selected, its unique integer is stored in the `IntVar` object
  - Can be used to select a default radio button

Using Callback Functions with `Radiobuttons`

- **You can specify a callback function with `Radiobutton` widgets**
  - Provide an argument `command=self.my_method` when creating the `Radiobutton` widget
  - The command will execute immediately when the radio button is selected
  - Replaces the need for a user to click OK or submit before determining which `Radiobutton` is selected

Check Buttons

- **Check button: small box with a label appearing next to it; check mark indicates when it is selected**
  - User is allowed to select any or all of the check buttons that are displayed in a group
    - Not mutually exclusive
- **Checkbutton widgets: created using `tkinter` module's `Checkbutton` class**
 - Associate different `IntVar` object with each `Checkbutton` widget

Drawing Shapes with the Canvas Widget

- **The `Canvas` widget is a blank, rectangular area that allows you to draw simple 2D shapes.**

- **You use the `Canvas` widget's *screen coordinate system* to specify the location of your graphics.**

- **The coordinates of the pixel in the upper-left corner of the screen are (0, 0).**
  - The *X* coordinates increase from left to right
  - The *Y* coordinates increase from top to bottom.

Drawing Shapes with the **Canvas** Widget

Figure 13-26 Various pixel locations in a 640 by 480 window
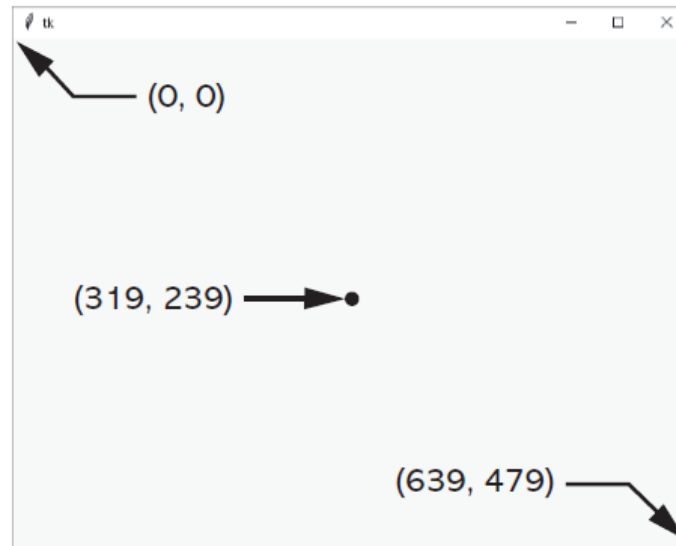
Drawing Shapes with the Canvas Widget

- **Creating a Canvas widget:**

```
# Create the main window.
self.main_window = tkinter.Tk()


# Create the Canvas widget.
self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
```

Drawing Shapes with the Canvas Widget

- **The `Canvas` widget has numerous methods for drawing graphical shapes on the surface of the widget.**

- **The methods that we will discuss are:**

  - `create_line`
  - `create_rectangle`
  - `create_oval`
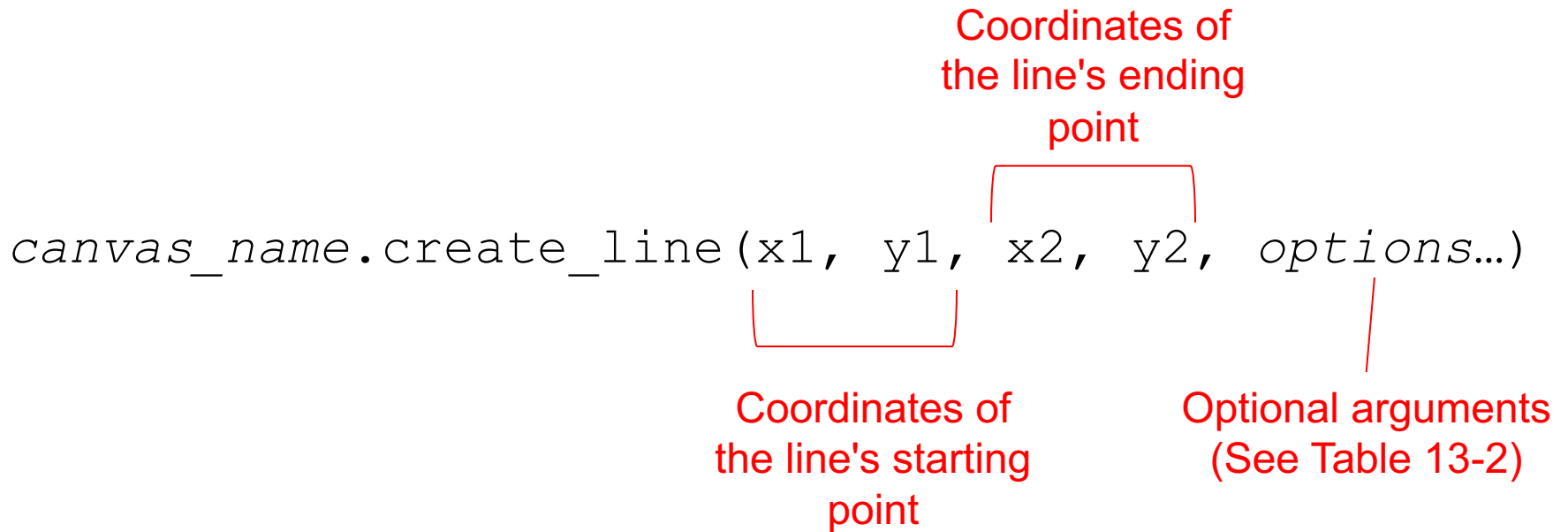 - `create_arc`
  - `create_polygon`
  - `create_text`

Drawing a Line

Coordinates of
the line's ending
point

$$canvas\_name.create\_line(x1, y1, x2, y2, options…)$$

Coordinates of
the line's starting
point

Optional arguments
(See Table 13-2)

Program 13-14 (draw_line.py)

```
1   # This program demonstrates the Canvas widget.
2   import tkinter
3
4   class MyGUI:
5       def __init__(self):
6           # Create the main window.
7           self.main_window = tkinter.Tk()
8
9           # Create the Canvas widget.
10          self.canvas = tkinter.Canvas(self.main_window, width=200,height=200)
11
12          # Draw two lines.
13          self.canvas.create_line(0, 0, 199, 199)
14          self.canvas.create_line(199, 0, 0, 199)
15
16          # Pack the canvas.
17          self.canvas.pack()
18
19          # Start the mainloop.
20          tkinter.mainloop()
21
22  # Create an instance of the MyGUI class.
23  my_gui = MyGUI()
```
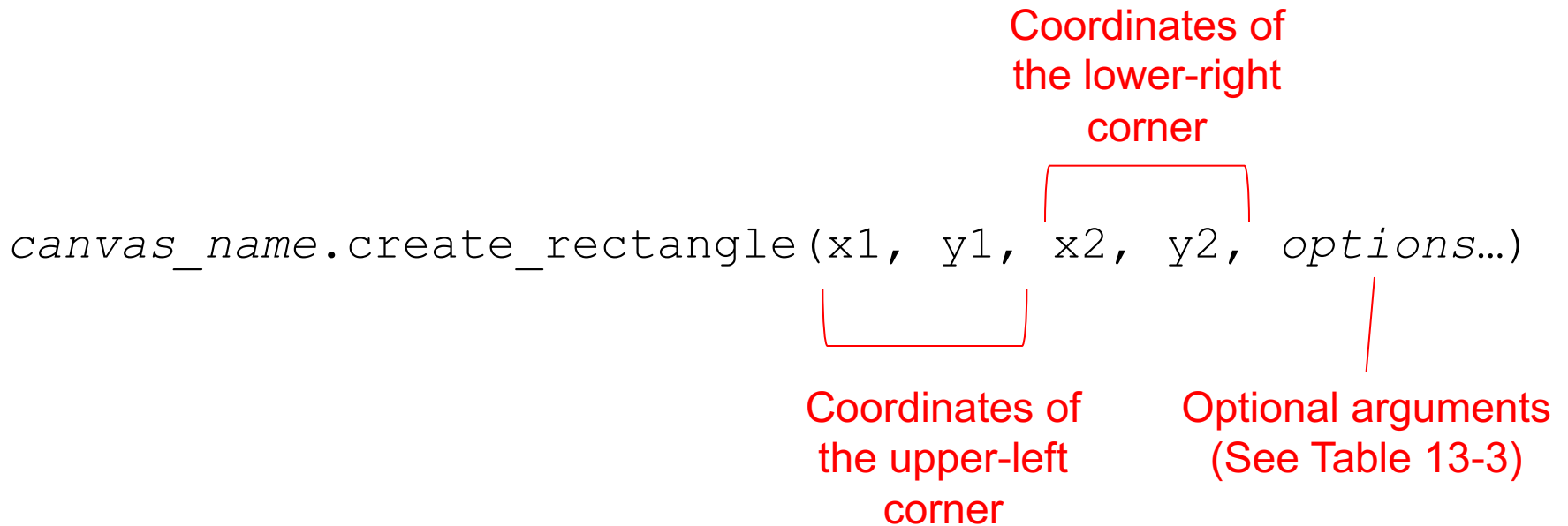
Drawing a Rectangle

Coordinates of
the lower-right
corner

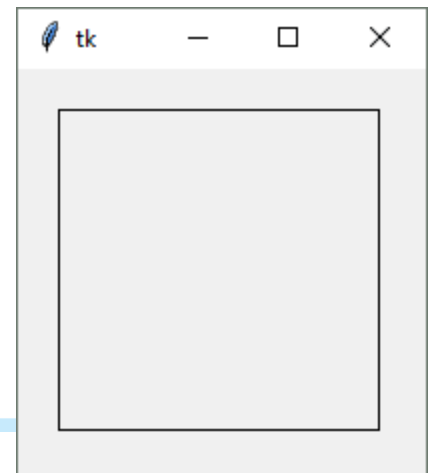`canvas_name.create_rectangle(x1, y1, x2, y2, options…)`

Coordinates of
the upper-left
corner

Optional arguments
(See Table 13-3)

Program 13-16 (draw_square.py)

```python
 1  # This program draws a rectangle on a Canvas.
 2  import tkinter
 3
 4  class MyGUI:
 5      def __init__(self):
 6          # Create the main window.
 7          self.main_window = tkinter.Tk()
 8
 9          # Create the Canvas widget.
10          self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
11
12          # Draw a rectangle.
13          self.canvas.create_rectangle(20, 20, 180, 180)
14
15          # Pack the canvas.
16          self.canvas.pack()
17
18          # Start the mainloop.
19          tkinter.mainloop()
20
21  # Create an instance of the MyGUI class.
22  my_gui = MyGUI()
```
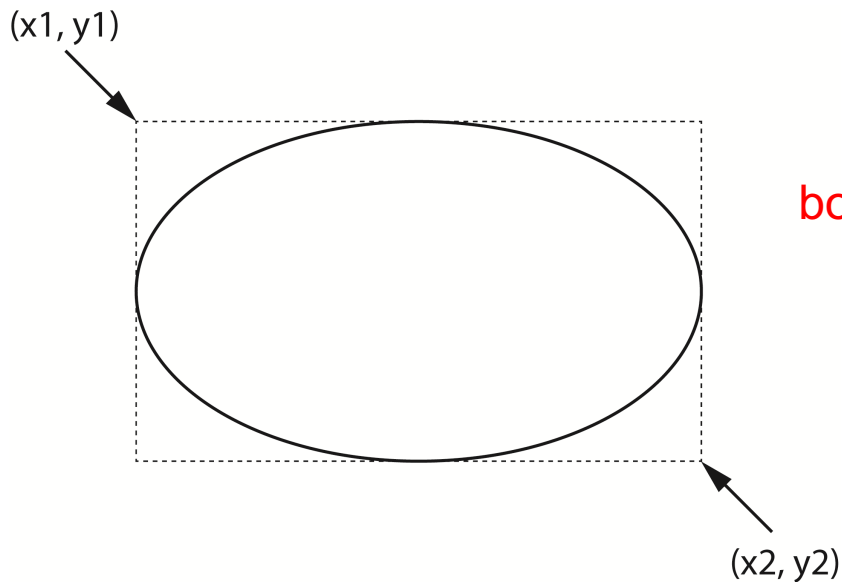
Drawing an Oval

Coordinates of the lower-right corner of bounding rectangle
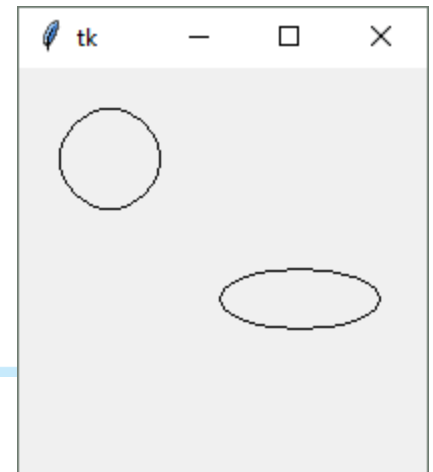
`canvas_name.create_oval(x1, y1, x2, y2, options…)`

Coordinates of the upper-left corner of bounding rectangle

Optional arguments (See Table 13-4)

(x1, y1)

(x2, y2)

Program 13-17 (draw_ovals.py)

```python
1  # This program draws two ovals Ion a Canvas.
2  import tkinter
3
4  class MyGUI:
5      def __init__(self):
6          # Create the main window.
7          self.main_window = tkinter.Tk()
8
9          # Create the Canvas widget.
10          self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
11
12          # Draw two ovals.
13          self.canvas.create_oval(20, 20, 70, 70)
14          self.canvas.create_oval(100, 100, 180, 130)
15
16          # Pack the canvas.
17          self.canvas.pack()
18
19          # Start the mainloop.
20          tkinter.mainloop()
21
22  # Create an instance of the MyGUI class.
23  my_gui = MyGUI()
```
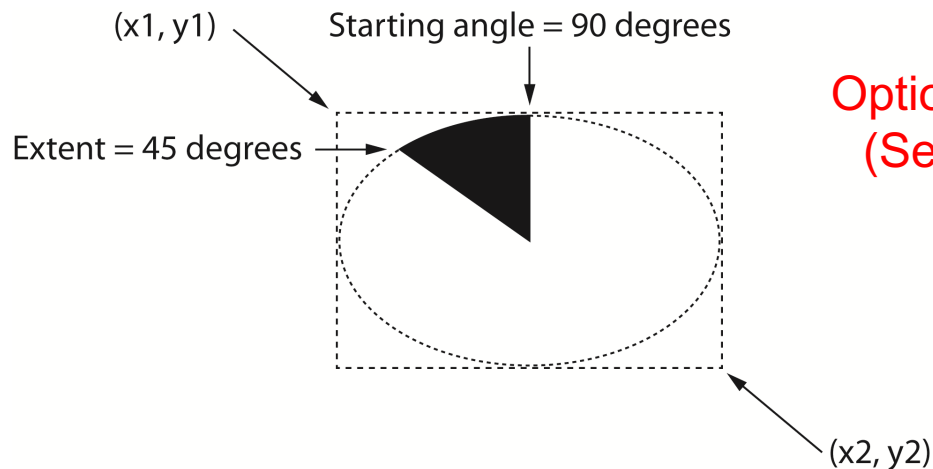
Drawing an Arc

Coordinates of the upper-left corner of bounding rectangle

Coordinates of the lower-right corner of bounding rectangle

```
canvas_name.create_arc(x1, y1, x2, y2,
              start=angle, extent=width,
              options…)
```

Starting angle

Counter clockwise extent of the arc

Optional arguments (See Table 13-5)

(x1, y1)

Starting angle = 90 degrees

Extent = 45 degrees

(x2, y2)

Program 13-18 (draw_arc.py)

```python
1   # This program draws an arc on a Canvas.
2   import tkinter
3
4   class MyGUI:
5       def __init__(self):
6           # Create the main window.
7           self.main_window = tkinter.Tk()
8
9           # Create the Canvas widget.
10          self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
11
12          # Draw an arc.
13          self.canvas.create_arc(10, 10, 190, 190, start=45, extent=30)
14
15          # Pack the canvas.
16          self.canvas.pack()
17
18          # Start the mainloop.
19          tkinter.mainloop()
20
21  # Create an instance of the MyGUI class.
22  my_gui = MyGUI()
```
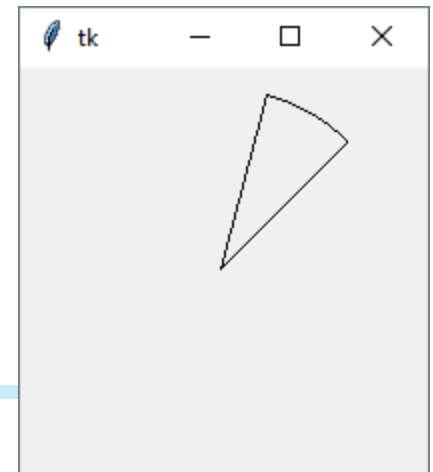
Drawing a Polygon

Coordinates of
the second vertex

`canvas_name.create_polygon(x1, y1, x2, y2, …,options…)`
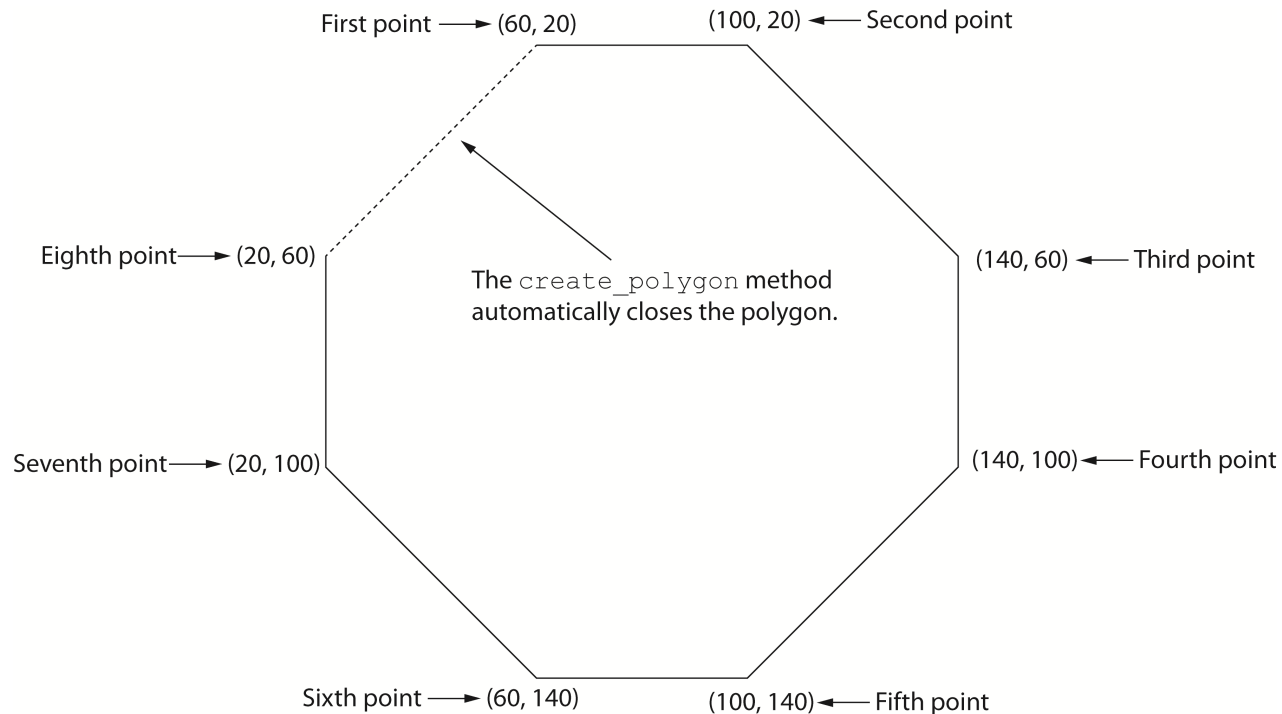
Coordinates of
the first vertex
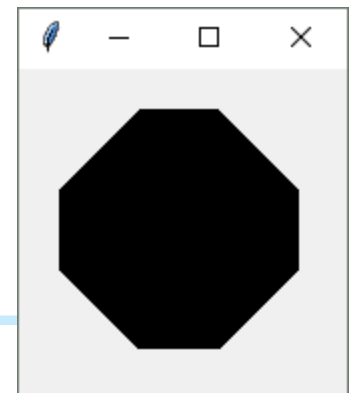
Optional arguments
(See Table 13-7)

Drawing a Polygon

```
self.canvas.create_polygon(60, 20, 100, 20, 140, 60, 140, 100,
                           100, 140, 60, 140, 20, 100, 20, 60)
```

First point ⟶ (60, 20)   (100, 20) ⟵ Second point

Eighth point ⟶ (20, 60)   (140, 60) ⟵ Third point

The `create_polygon` method automatically closes the polygon.

Seventh point ⟶ (20, 100)   (140, 100) ⟵ Fourth point

Sixth point ⟶ (60, 140)   (100, 140) ⟵ Fifth point

Program 13-20 (draw_polygon.py)

```python
1  # This program draws a polygon on a Canvas.
2  import tkinter
3
4  class MyGUI:
5      def __init__(self):
6          # Create the main window.
7          self.main_window = tkinter.Tk()
8
9          # Create the Canvas widget.
10         self.canvas = tkinter.Canvas(self.main_window, width=160, height=160)
11
12         # Draw a polygon.
13         self.canvas.create_polygon(60, 20, 100, 20, 140, 60, 140, 100,
14                                    100, 140, 60, 140, 20, 100, 20, 60)
15
16         # Pack the canvas.
17         self.canvas.pack()
18
19         # Start the mainloop.
20         tkinter.mainloop()
21
22 # Create an instance of the MyGUI class.
23 my_gui = MyGUI()
```

Displaying Text on the Canvas

Text to display

`canvas_name.create_text(x, y, text=text, options…)`

Coordinates of
the text's insertion
point

Optional arguments
(See Table 13-8)

Program 13-21 (draw_text.py)

```python
1   # This program draws text on a Canvas.
2   import tkinter
3
4   class MyGUI:
5       def __init__(self):
6           # Create the main window.
7           self.main_window = tkinter.Tk()
8
9           # Create the Canvas widget.
10          self.canvas = tkinter.Canvas(self.main_window, width=200, height=200)
11
12          # Display text in the center of the window.
13          self.canvas.create_text(100, 100, text='Hello World')
14
15          # Pack the canvas.
16          self.canvas.pack()
17
18          # Start the mainloop.
19          tkinter.mainloop()
20
21  # Create an instance of the MyGUI class.
22  my_gui = MyGUI()
```
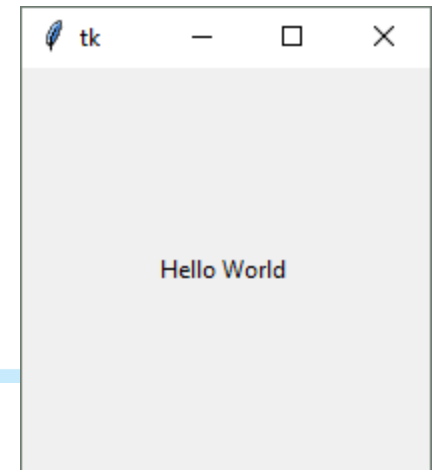
Summary

- **This chapter covered:**
  - Graphical user interfaces and their role as event-driven programs
  - The `tkinter` module, including:
    - Creating a GUI window
    - Adding widgets to a GUI window
    - Organizing widgets in frames
    - Receiving input and providing output using widgets
    - Creating buttons, check buttons, and radio buttons
    - Drawing simple shapes with the `Canvas` widget