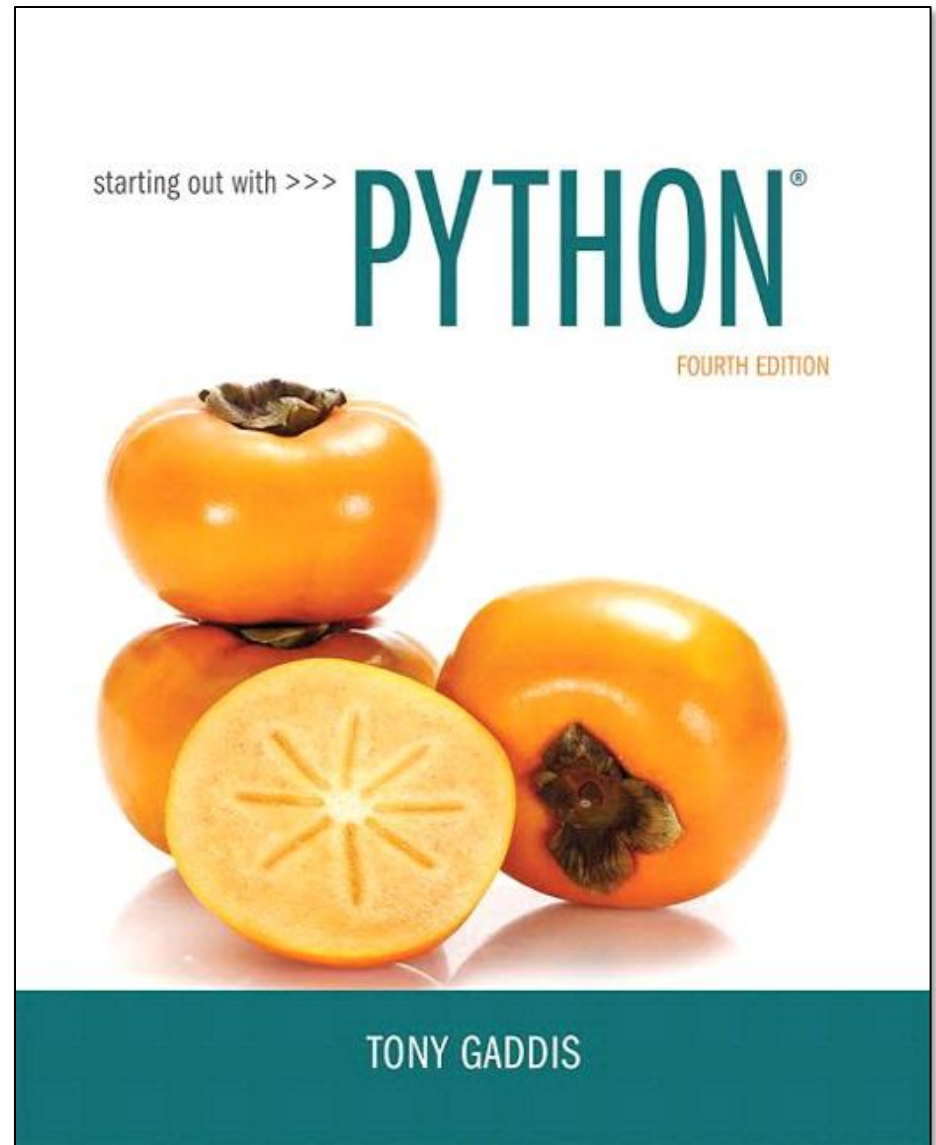


CHAPTER 13

GUI



Topics

- Graphical User Interfaces
- Using the `tkinter` Module
- Display Text with `Label` Widgets
- Organizing Widgets with Frames
- The `Button` Widgets and Info Dialog Boxes
- Getting Input with the `Entry` Widget
- Using Labels as Output Fields

Learning Outcomes

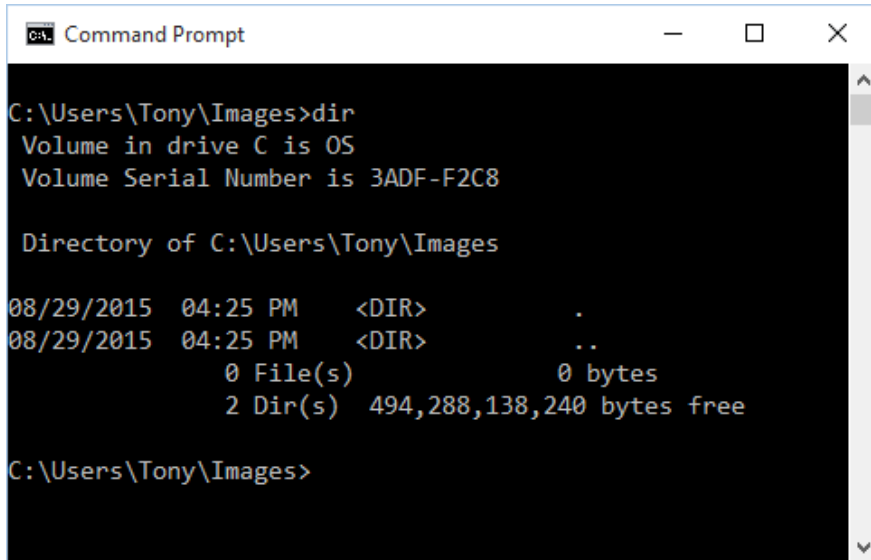
- By the end of this week a student must be able to:
 - Define terms such as user interface, GUI, command line interface, and widget
 - Design a GUI and select the correct widgets
 - Identify the events to handle
 - Define a GUI class using widgets like frame, label, entry and button
 - Use labels as an output fields
 - Define your `__init__` and callback Function
 - Recognize the scope of variables in the class.
 - Produce an output using the information dialog box

Graphical User Interfaces

- **User Interface:** the part of the computer with which the user interacts
- **Command line interface** displays a prompt, and the user types a command that is then executed
- **Graphical User Interface (GUI):** allows users to interact with a program through graphical elements on the screen

User Interfaces

- A command line interface
- A graphical user interface



```
C:\Users\Tony\Images>dir
Volume in drive C is OS
Volume Serial Number is 3ADF-F2C8

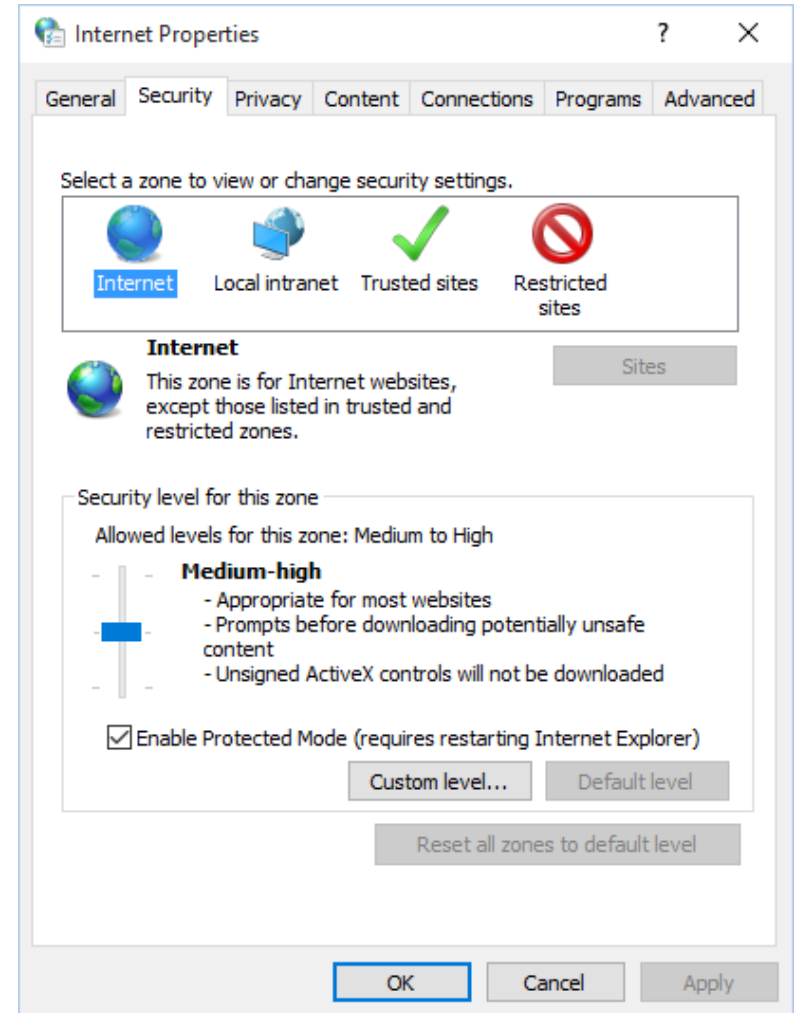
Directory of C:\Users\Tony\Images

08/29/2015  04:25 PM    <DIR>          .
08/29/2015  04:25 PM    <DIR>          ..
               0 File(s)                0 bytes
               2 Dir(s)  494,288,138,240 bytes free

C:\Users\Tony\Images>
```

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

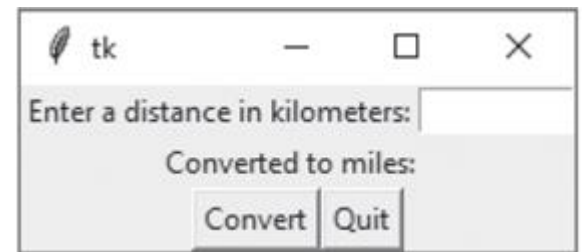


GUI Programs Are Event-Driven

- 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
- In GUI environments, the **user determines** the **order** in which things happen
 - User causes events to take place and the program responds to the events
 - These programs called **event-driven** programs

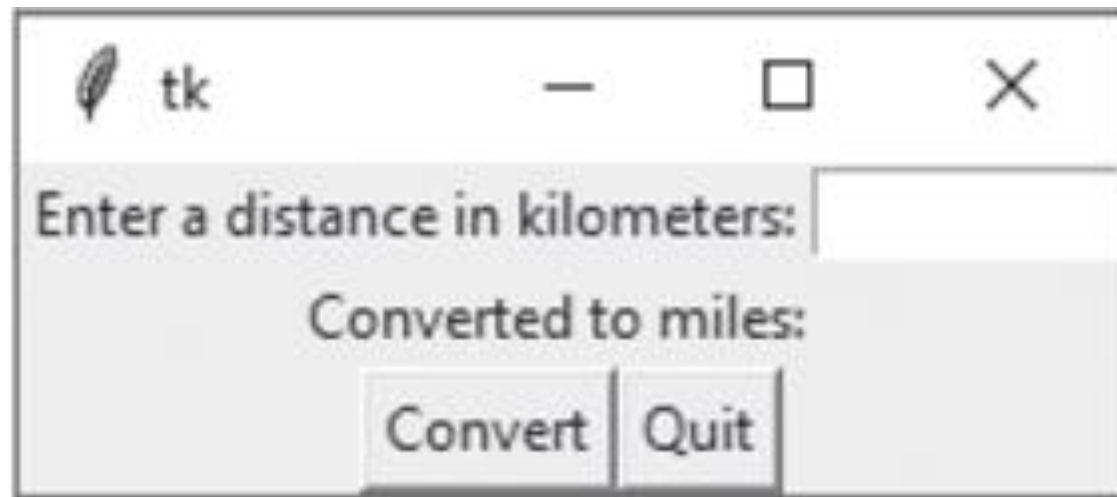
Using the `tkinter` Module

- No GUI programming features built into Python
- **`tkinter` module** (short for TK Interface): 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



Step 1 : Design GUI

- Design your GUI, and decide about the graphical widgets you want to display to the user.

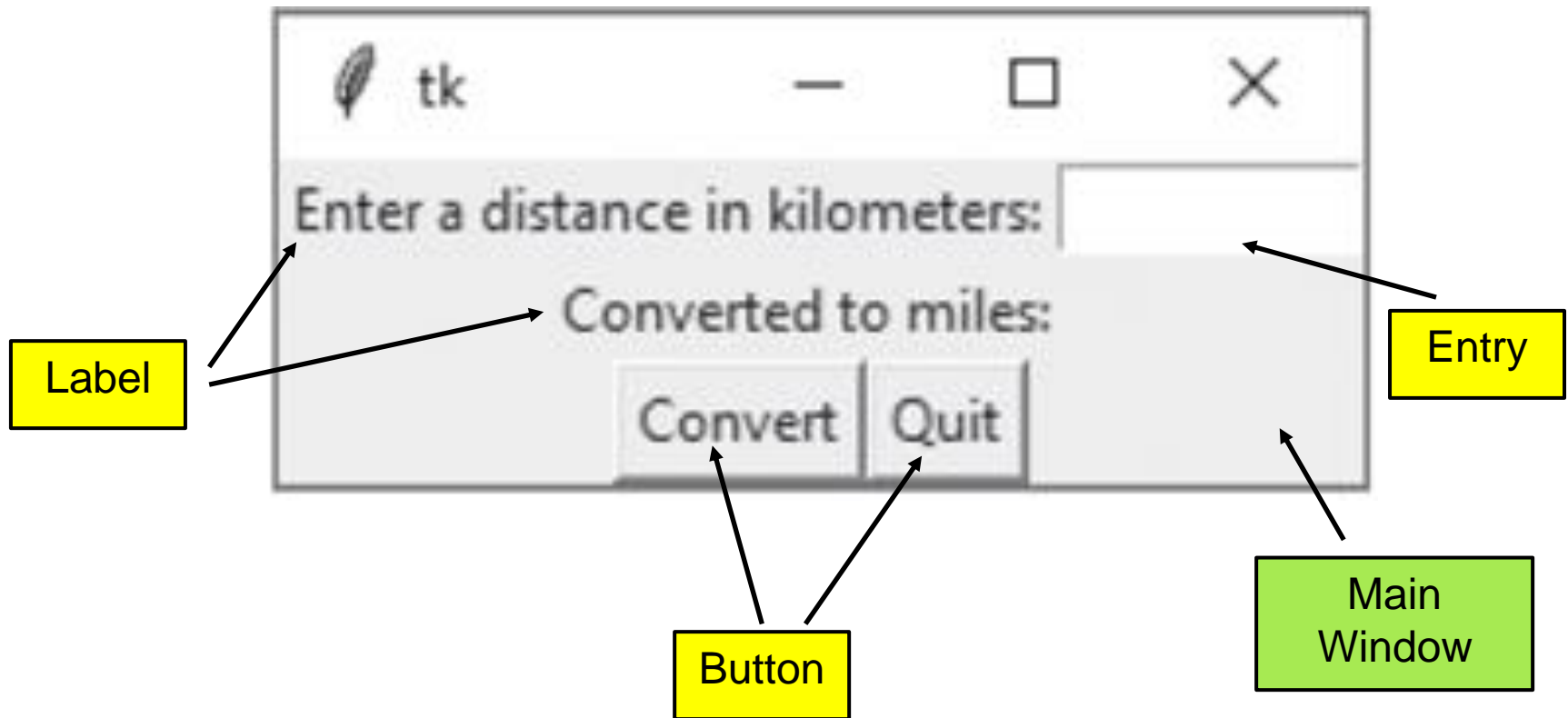


Step 2: Select Your Widgets

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.

Step 2: Select Your Widgets



Step 3: Identify the Events to Handle

- Identify all the events you would like to respond to in your GUI program.
 - **Clicking the Convert button:** Convert the entered number in kilometers to miles, and attach the result to the second label
 - **Clicking the Quit Button:** Close (destroy) the main window
- Button clicks are of the most popular events.

Step 4 : Define a GUI Class

- Create a .py file to include the definition of your GUI class.
- Include the following methods in your class:
 1. `__init__`: Initialize your GUI window along with all its components here.
 2. For each event identified in the previous step, define a method to handle the event (if required).

Define a GUI Class: `__init__` method

1. Create the main window which is the root widget. Instantiate from the Tk class defined in the tkinter module, and assign the created object to an attribute in your GUI class:

```
self.mainWindow = tkinter.Tk()
```

- Set the title of your window, if you like:

```
self.mainWindow.title("Kilometer to Mile  
Converter")
```

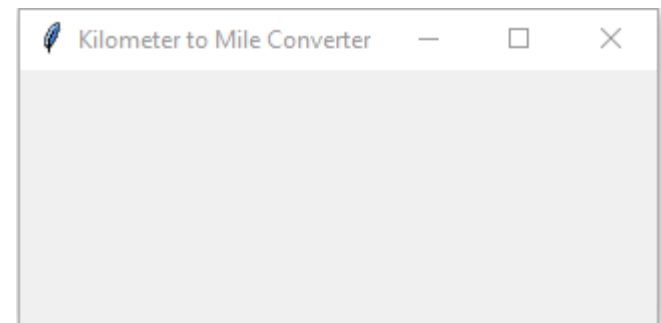
2. Enter the tkinter main loop. It runs like an infinite loop until you close the window

```
tkinter.mainloop()
```

Example – empty_window.py

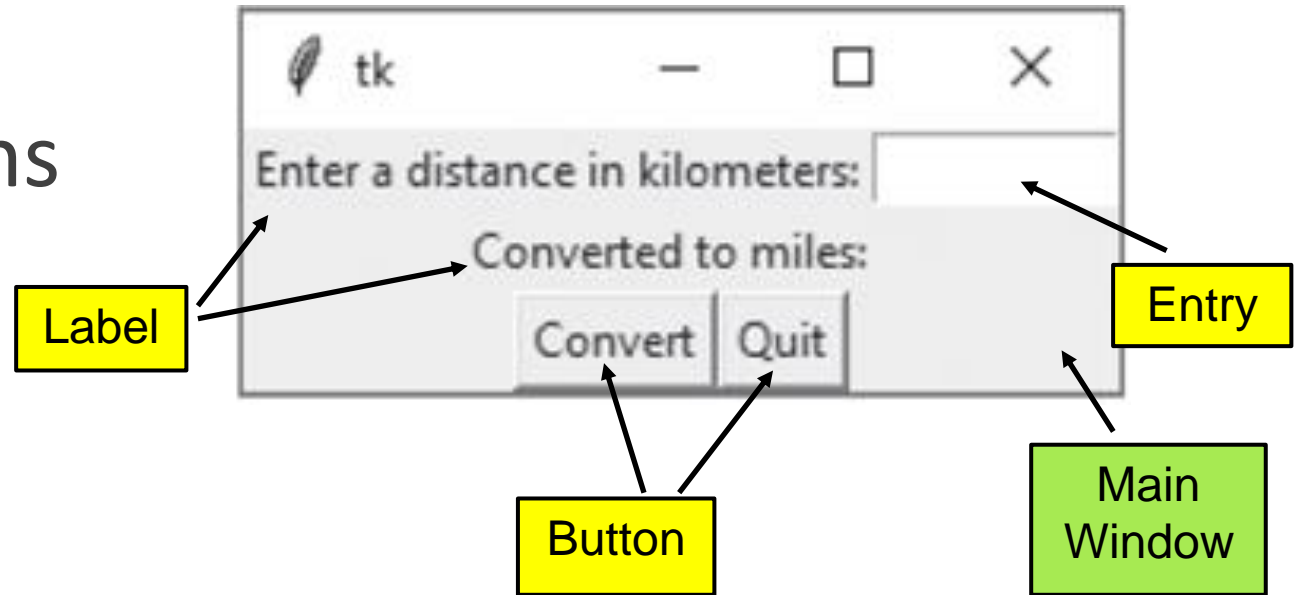
```
# This program displays an empty window.
import tkinter
class MyGUI:
    def __init__(self):
        # Create the main window widget, which is
        # the root widget
        self.main_window = tkinter.Tk()
        self.main_window.title("Kilometer to Mile Converter")
        # Enter the tkinter main loop. It runs like
        # an infinit loop until you close the window
        tkinter.mainloop()

# Create an instance of the MyGUI class.
my_gui = MyGUI()
```



Step 5: Add Widgets to Main Window

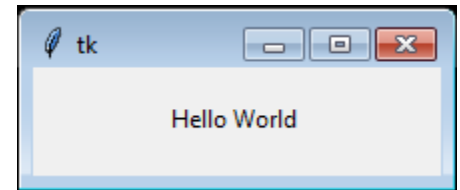
- Add labels
- Add entry
- Add buttons



Define Label Widgets

- **Label widget** 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 = 'Hello World')
```
 - First argument references the root widget, second argument shows text that should appear in label
 - Open `hello_world.py` and run



add_labels.py

```
# This program displays two labels with text.

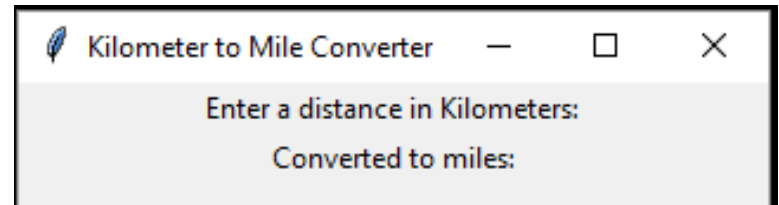
import tkinter

class MyGUI:
    def __init__(self):
        # Create the main window widget.
        self.main_window = tkinter.Tk()
        self.main_window.title("Kilometer to Mile Converter")
        # Create two Label widget.
        self.kilo_label1 = tkinter.Label(self.main_window,
                                         text='Enter a distance in Kilometers:')
        self.kilo_label2 = tkinter.Label(self.main_window,
                                         text='Converted to miles:')

        # Call both Label widgets' pack method.
        self.kilo_label1.pack()
        self.kilo_label2.pack()

        # Enter the tkinter main loop.
        tkinter.mainloop()

# Create an instance of the MyGUI class.
my_gui = MyGUI()
```



pack Method

- 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='bottom'`, `side='left'`, `side='right'`
`my_label.pack(side='left')`

Define Entry Widgets

- A 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

- Format :

```
self.kilo_entry = tkinter.Entry(self.main_window,  
                                width = 10)
```

- The width argument in the Entry class is used to set the size of the entry in characters; the default is 20.

add_entry.py

```
# This program displays two labels with text.

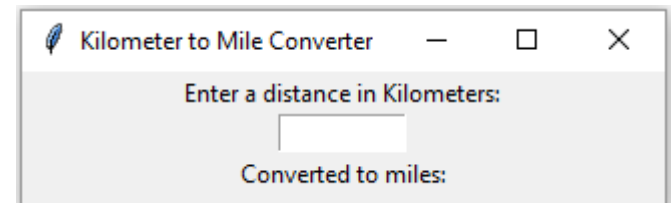
import tkinter

class MyGUI:
    def __init__(self):
        # Create the main window widget.
        self.main_window = tkinter.Tk()
        self.main_window.title("Kilometer to Mile Converter")
        # Create two Label widget.
        self.kilo_label1 = tkinter.Label(self.main_window,
                                         text='Enter a distance in Kilometers:')
        self.kilo_label2 = tkinter.Label(self.main_window,
                                         text='Converted to miles:')
        self.kilo_entry = tkinter.Entry(self.main_window, width = 10)

        # Call both Label widgets' pack method.
        self.kilo_label1.pack()
        self.kilo_entry.pack()
        self.kilo_label2.pack()

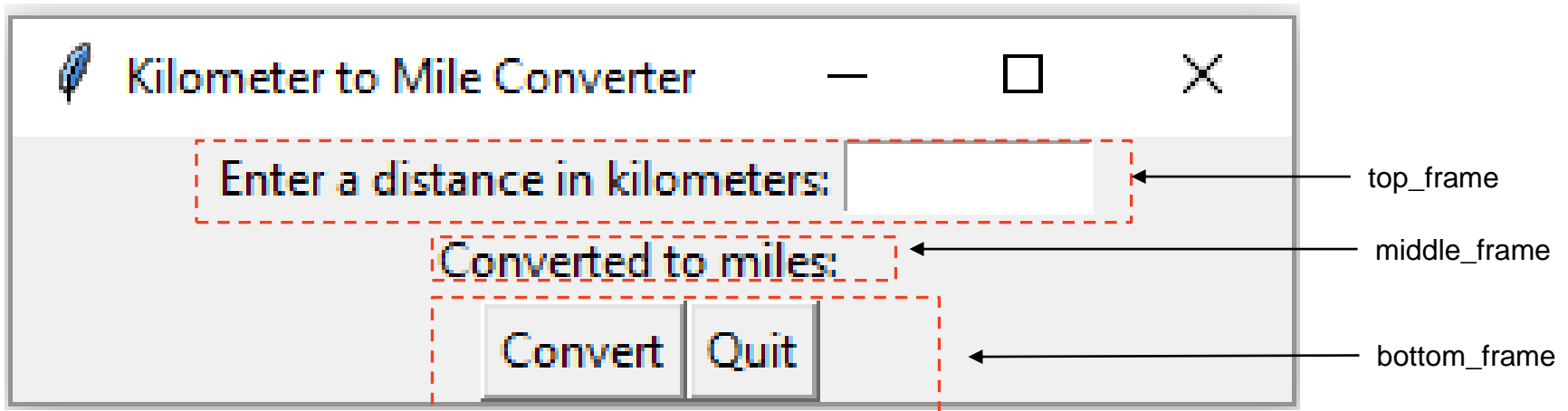
        # Enter the tkinter main loop.
        tkinter.mainloop()

# Create an instance of the MyGUI class.
my_gui = MyGUI()
```



Organizing Widgets with Frames

- Frame is a container to hold other widgets
- Frame objects are non-visible objects
- Useful for organizing and arranging groups of widgets in a window



Organizing Widgets with Frames

- The objects in a frame can be positioned independent from the objects in the other frames.
- The contained widgets are added to the frame widget which contains them

○ Example:

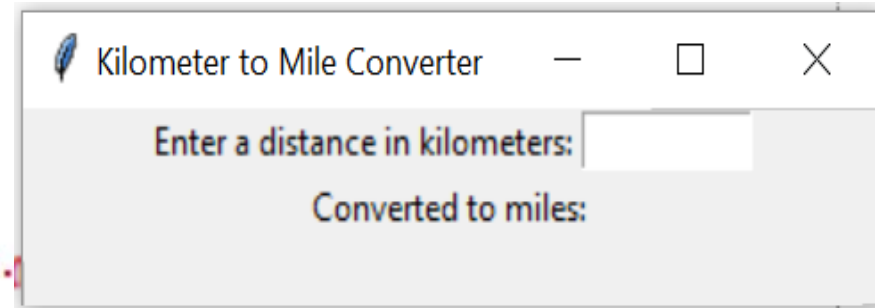
```
tkinter.Label(self.top_frame,  
               text = 'hi')
```

add_frame.py

```
import tkinter
class MyGUI:
    def __init__(self):
        # Create the main window.
        self.main_window = tkinter.Tk()
        self.main_window.title("Kilometer to Mile Converter")
        # Create three frames to group widgets.
        self.top_frame = tkinter.Frame()
        self.mid_frame = tkinter.Frame()
        # Create the widgets for the top frame.
        self.kilo_labell = tkinter.Label(self.top_frame,
                                         text='Enter a distance in kilometers:')
        self.kilo_entry = tkinter.Entry(self.top_frame, width=10)
        # Pack the top frame's widgets.
        self.kilo_labell.pack(side='left')
        self.kilo_entry.pack(side='left')
        # Create the widgets for the middle frame.
        self.kilo_label2 = tkinter.Label(self.mid_frame,
                                         text='Converted to miles:')

        self.kilo_label2.pack()
        # Pack the frames.
        self.top_frame.pack()
        self.mid_frame.pack()
        # Enter the tkinter main loop.
        tkinter.mainloop()

# Create an instance of the MyGUI class.
my_gui = MyGUI()
```



Define Button Widgets

- A widget that the user can click to cause an action to take place



- When creating a button can specify:

1. Text to appear on the face of the button
2. A callback function

- **Callback function:** function or method that executes when the user clicks the button

- Also known as an **event handler**

```
self.button = tkinter.Button(self.main_window,  
                             text = 'Click Me!', command = self.convertK2M)
```


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

```
self.quit_button = tkinter.Button(self.main_window,  
                                  text='Quit',  
                                  command=self.main_window.destroy)
```

add_buttons.py

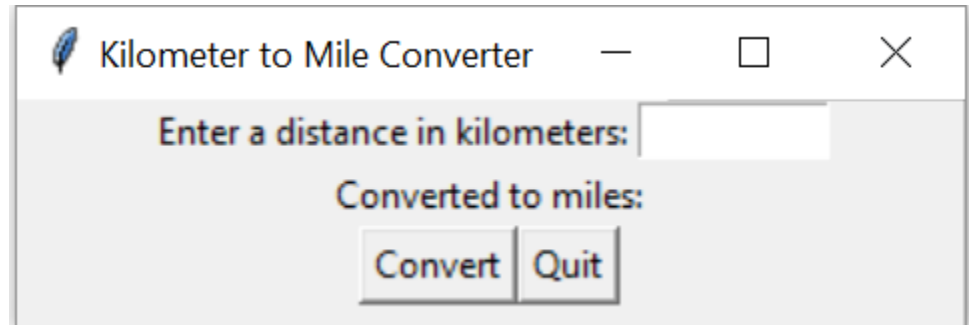
```
# add a new frame
self.bottom_frame = tkinter.Frame()

# Create a button widgets for the bottom frame
self.calc_button = tkinter.Button(self.bottom_frame,
                                   text='Convert')

self.quit_button = tkinter.Button(self.bottom_frame,
                                   text='Quit',
                                   command=self.main_window.destroy)

# Pack the buttons.
self.calc_button.pack(side='left')
self.quit_button.pack(side='left')

# Pack the frames.
self.bottom_frame.pack()
```



Using Labels as Output Fields

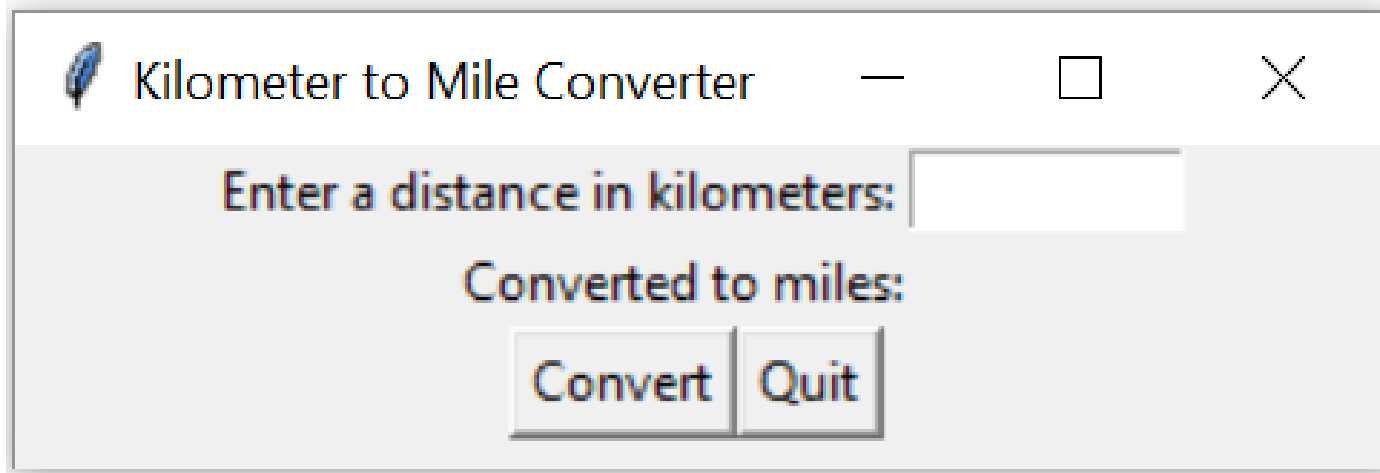
- **StringVar class:** `tkinter` module class that can be used along with `Label` widget to display data
 - Create `StringVar` object and then create `Label` widget and associate it with the `StringVar` object
 - Subsequently, any value stored in the `StringVar` object will automatically be displayed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

Define Your Own Callback Function

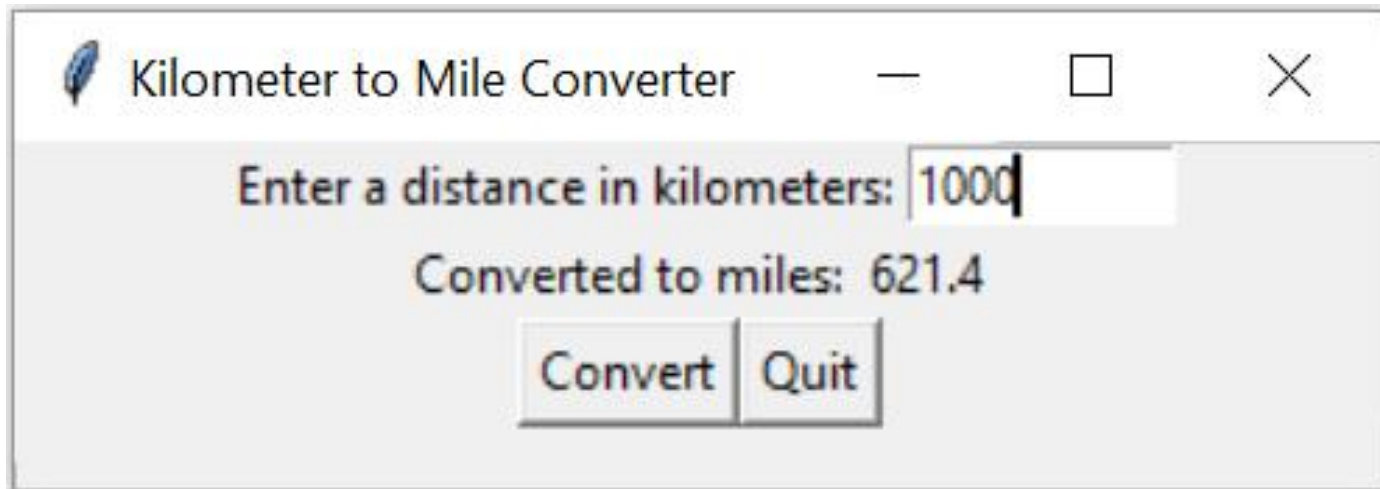
- Create empty `Label` widget in main window, and write code that displays desired data in the label when a button is clicked

```
self.value = tkinter.StringVar()
self.miles_label = tkinter.Label(self.mid_frame,
                                textvariable=self.value)
...
self.calc_button = tkinter.Button(self.main_window,
                                text='Convert',
                                command=self.convert)
...
def convert(self):
    kilo = float(self.kilo_entry.get())
    miles = kilo * 0.6214
    self.value.set(miles)
```

add_callback_function.py



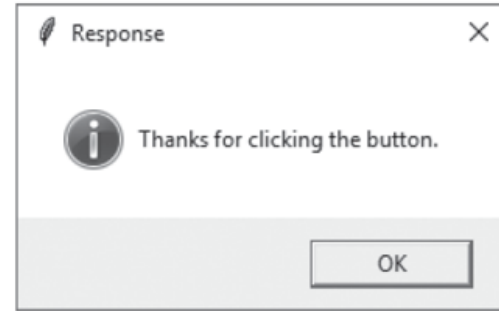
A screenshot of a Python GUI window titled "Kilometer to Mile Converter". The window has a standard title bar with a feather icon, a minus button, a maximize button, and a close button. The main content area has a light gray background. It contains two labels: "Enter a distance in kilometers:" and "Converted to miles:". Below the first label is a text input field that is currently empty. Below the second label are two buttons: "Convert" and "Quit".



A second screenshot of the same "Kilometer to Mile Converter" window. In this state, the text input field contains the value "1000". The "Converted to miles:" label now displays the result "621.4". The "Convert" and "Quit" buttons remain at the bottom.

Info Dialog Boxes

- A dialog box that shows information to the user



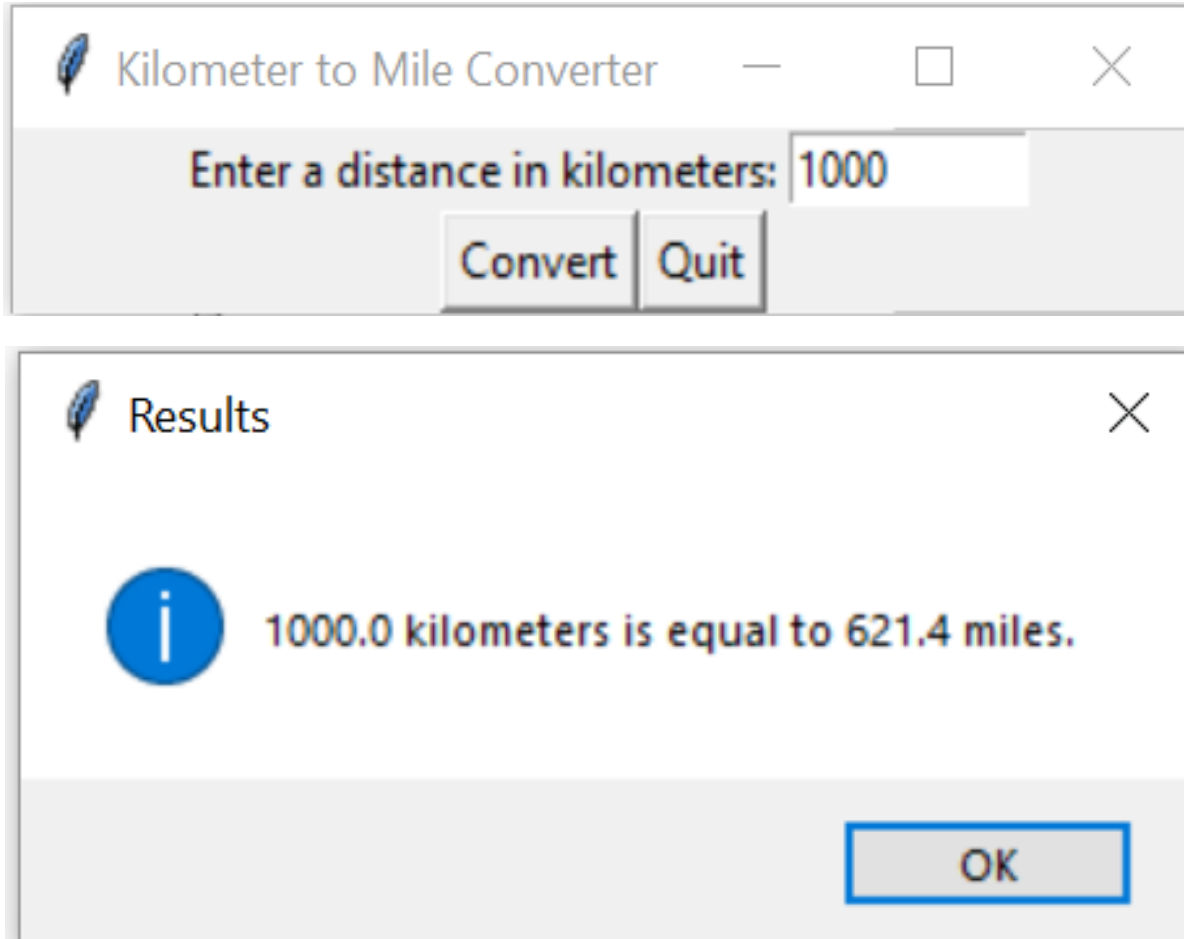
- **Format:**

- Import `tkinter.messagebox` module
- `tkinter.messagebox.showinfo(title,
message)`

- *title* is displayed in dialog box's title bar (e.g. Response)
- *message* is an informational string displayed in the main part of the dialog box (e.g: Thanks for clicking the button)

Getting Output with Info Dialog Box

- Open kilo_converter.py and run.



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

More Practice

- Check out review questions in chapter 13 of the textbook including :
 - Multiple Choices,
 - True or False
 - Short Answer
 - Algorithm WorkBench
 - Programming Exercises (only number 1, 2, 4, 6)