

M110: Python Programming

Meeting #11

GUI Programming



AOU

الجامعة العربية المفتوحة
Arab Open University
Lebanon



Prepared by Dr. Ahmad Mikati

Topics

- Graphical User Interfaces
- Using the tkinter Module
- Displaying 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

Graphical User Interfaces (1 of 3)

A computer's *user interface* is the part of the computer with which the user interacts. One part of the user interface consists of hardware devices, such as the keyboard and the video display. Another part of the user interface lies in the way that the computer's operating system accepts commands from the user.

For many years, the only way that the user could interact with an operating system was through a **command line interface**.

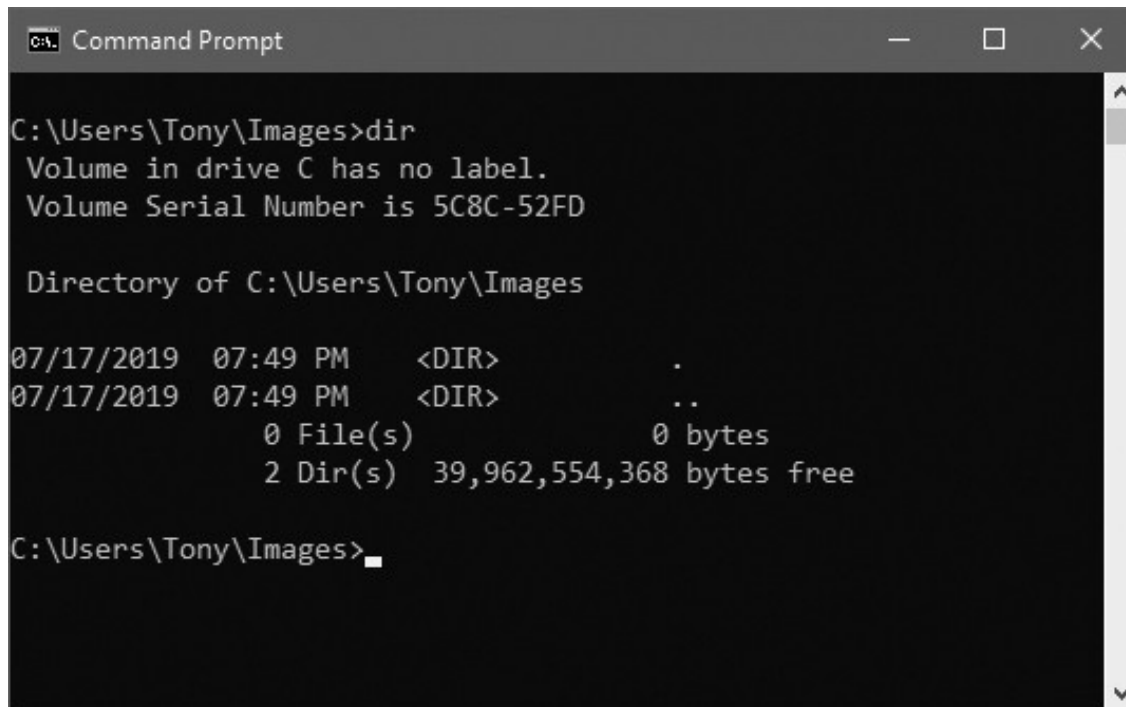
In the 1980s, a new type of interface known as a graphical user interface came into use in commercial operating systems, called the

graphical user interface of the computer with which the user interacts

- Command line interface: it 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.

Graphical User Interfaces (2 of 3)

- **Command line interface**: displays a prompt and the user types a command that is then executed



```
C:\Users\Tony\Images>dir
Volume in drive C has no label.
Volume Serial Number is 5C8C-52FD

Directory of C:\Users\Tony\Images

07/17/2019  07:49 PM    <DIR>          .
07/17/2019  07:49 PM    <DIR>          ..
               0 File(s)                0 bytes
               2 Dir(s)  39,962,554,368 bytes free

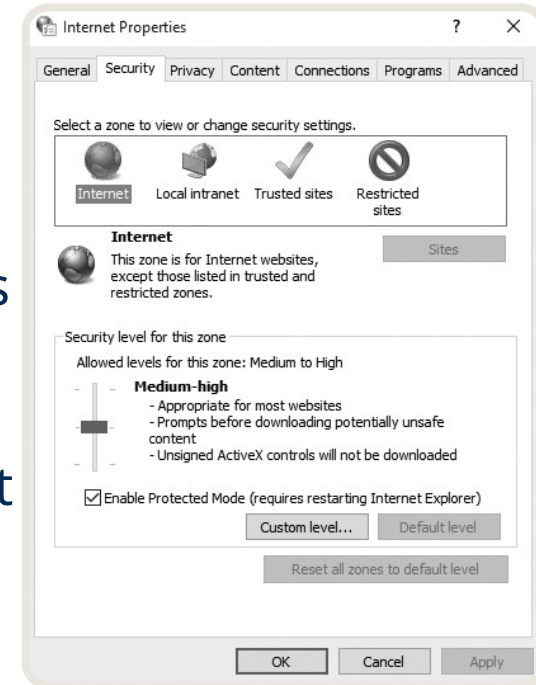
C:\Users\Tony\Images>
```

Figure 1 A command line interface

Graphical User Interfaces (3 of 3)

Graphical User Interface (GUI): (GUI; pronounced “gooey”), allows the user to interact with the operating system and other programs through graphical elements on the screen.

- GUIs also popularized the use of the mouse as an input device.
- GUIs allow the user to point at graphical elements and click the mouse button to activate them.
- Much of the interaction with a GUI is done through *dialog boxes*, which are 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 (1 of 3)

- Python does not have GUI programming features built into the language itself. However, it comes with a module named tkinter that allows you to create simple GUI programs.
- The name “tkinter” is short for “Tk interface.”
- **tkinter module** allows you to create simple GUI programs
- A GUI program presents a window with various graphical widgets with which the user can interact or view. The tkinter module provides 15 widgets, which are described in Table 1.
- We won't cover all the tkinter widgets in this chapter, but we will demonstrate how to create simple GUI programs that gather input and display data
- **Widget**: A graphical element that the user can interact with or view (Known also as control)
 - Presented by a GUI program

Using the tkinter Module (2 of 3)

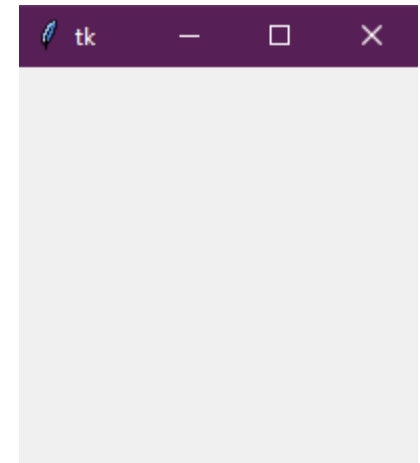
| Widget | Description |
|--------------------|--|
| Button | A button that can cause an action to occur when it is clicked. |
| Canvas | A rectangular area that can be used to display graphics. |
| Checkbutton | A button that may be in either the “on” or “off” position. |
| Entry | An area in which 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 displayed when the user clicks a <i>Menubutton</i> 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. <i>Radiobutton</i> 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 |

Table 1 tkinter widgets

Using the tkinter Module

The simplest GUI program that we can demonstrate is one that displays an empty window. The below program shows how we can do this using the tkinter module. When the program runs, the window shown in the figure is displayed. To exit the program, simply click the standard Windows close button (×) in the upper right corner of the window.

```
# This program displays an empty window.
import tkinter
def main():
    # Create the main window widget.
    main_window = Tk()
    # Enter the tkinter main loop.
    tkinter.mainloop()
# Call the main function.
main()
```



main_window = Tk() creates an instance of the tkinter module's **Tk class** and assigns it to the main_window variable. This object is the root widget, which is the main window in the program.

tkinter.mainloop() calls the tkinter module's mainloop function. This function runs in an indefinite loop until you close the main window.

import <module> vs. from <module> import *

In general, we use **from <module>import *** when we want to save ourselves from typing the module name repeatedly.

In other words, use **from <module> import *** when referring to a member of the module many times in the code.

- **Example:** in order to avoid writing **tkinter.** before each command, you can use **from tkinter import *** instead of **import tkinter** at the beginning of your program.

```
import tkinter  
tkinter.mainloop()
```

OR

```
from tkinter  
import *  
mainloop()
```

Using the tkinter Module (OOP)

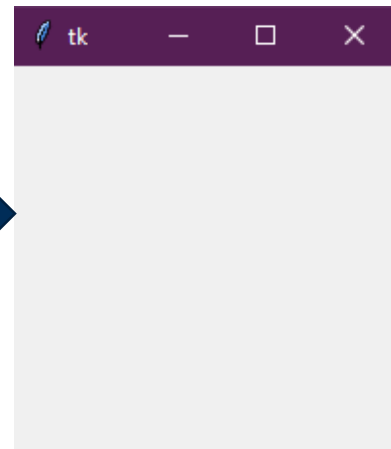
- Programs that use tkinter do not always run reliably under IDLE
 - For best results run them from operating system command prompt
- Most programmers take an **object-oriented** approach when writing GUI programs.
 - **`__init__`** method builds the GUI
 - When an instance is created the GUI appears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

```
# This program displays an empty window.  
from tkinter import *  
def main():  
    # Create the main window widget.  
    main_window = Tk()  
    # Enter the tkinter main loop.  
    mainloop()  
# Call the main function.  
main()
```

object-oriented approach

```
# This program displays a window with a title.  
from tkinter import *  
class MyGUI:  
    def __init__(self):  
        # Create the main window widget.  
        self.main_window = Tk()  
        # Enter the tkinter main loop.  
        mainloop()  
# Create an instance of the MyGUI class.  
my_gui = MyGUI()
```

Output



Changing the Tkinter GUI Window Title

To change the title of the GUI window in Tkinter, use the `title()` method to set the title of the window:

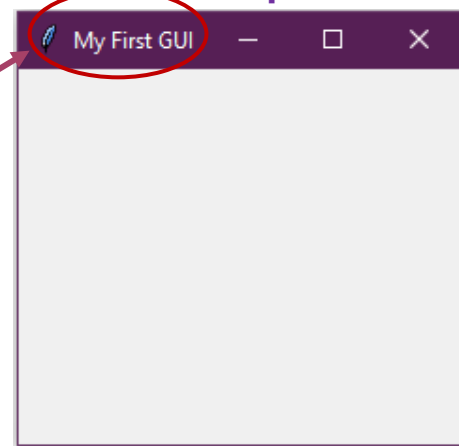
Format: `self.root.title("Your Title")`

Example 1

```
# This program displays a window with a title.

from tkinter import *
class MyGUI:
    def __init__(self):
        # Create the main window widget.
        self.main_window = Tk()
        # Display a title.
        self.main_window.title('My First GUI')
        # Enter the tkinter main loop.
        mainloop()
# Create an instance of the MyGUI class.
my_gui = MyGUI()
```

Output



Display Text with **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: **Label(self.main_window,text='my text')**
 - First argument references the root widget, second argument shows text that should appear in label
 - **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='left', side='right', side='bottom'
(or side=TOP, side= LEFT, side= RIGHT, side= BOTTOM).
- N.B: if no argument included, then TOP, or “top”, is considered the default case.**

Example 2

This program displays a title and a label with text.

```
from tkinter import *
```

```
class MyGUI:
```

```
    def __init__(self):
```

```
        # Create the main window widget.
```

```
        self.main_window = Tk()
```

```
        # Display a title.
```

```
        self.main_window.title('My First GUI')
```

```
        # Create a Label widget containing the text 'Hello World!'
```

```
        self.label = Label(self.main_window, text='Hello World!')
```

```
        # Call the Label widget's pack method.
```

```
        self.label.pack()
```

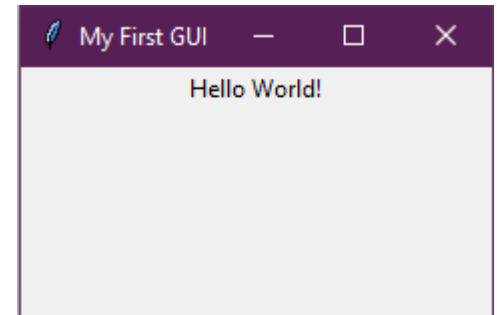
```
        # Enter the tkinter main loop.
```

```
        mainloop()
```

```
# Create an instance of the MyGUI class.
```

```
my_gui = MyGUI()
```

Output



The **pack** method determines where a widget should be positioned and makes the widget visible when the main window is displayed. **(You call the pack method for each widget in a window.)**

Example 3

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

from tkinter import *

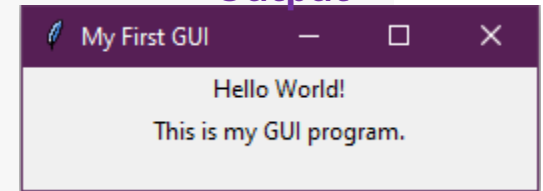
class MyGUI:
    def __init__(self):
        # Create the main window widget.
        self.main_window = Tk()
        # Display a title.
        self.main_window.title('My First GUI')
        # Create two Label widget.
        self.label1 = Label(self.main_window, text='Hello World!')
        self.label2 = Label(self.main_window, text='This is my GUI program.')

        # Call both Label widgets' pack method.
        self.label1.pack()
        self.label2.pack()

        # Enter the tkinter main loop.
        mainloop()

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

Output

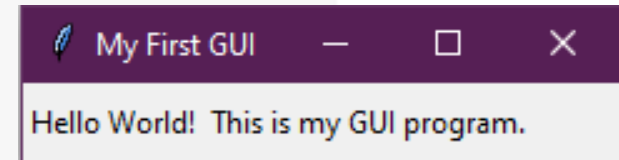


Notice the two Label widgets are displayed with one stacked on top of the other. We can change this layout by specifying an argument to pack method

Example 4

```
1 # using the side='left' argument with the pack method
2
3 from tkinter import *
4
5 class MyGUI:
6     def __init__(self):
7         # Create the main window widget.
8         self.main_window = Tk()
9         # Display a title.
10        self.main_window.title('My First GUI')
11        # Create two Label widget.
12        self.label1 = Label(self.main_window, text='Hello World!')
13        self.label2 = Label(self.main_window, text='This is my GUI program.')
14
15        # Call both Label widgets' pack method.
16        self.label1.pack(side='left')
17        self.label2.pack(side='left')
18
19        # Enter the tkinter main loop.
20        mainloop()
21
22 # Create an instance of the MyGUI class.
23 my_gui = MyGUI()
```

Output



In lines 16 and 17, label1 widget was added to the main window first, it will appear at the leftmost edge. The label2 widget was added next, so it appears next to the label1 widget. As a result, the labels appear side by side.

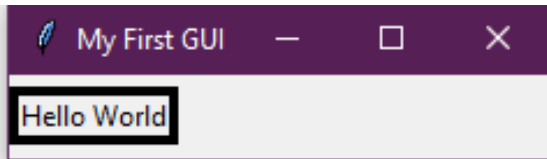
Adding Borders to Labels

- When creating a Label widget, you can use the **borderwidth** and **relief** arguments to display a border around the label
- The **borderwidth** argument specifies the width of the border, in pixels
- The **relief** argument specifies the border style

```
self.label = Label(self.main_window,  
                    text='Hello World',  
                    borderwidth=1,  
                    relief='solid')
```

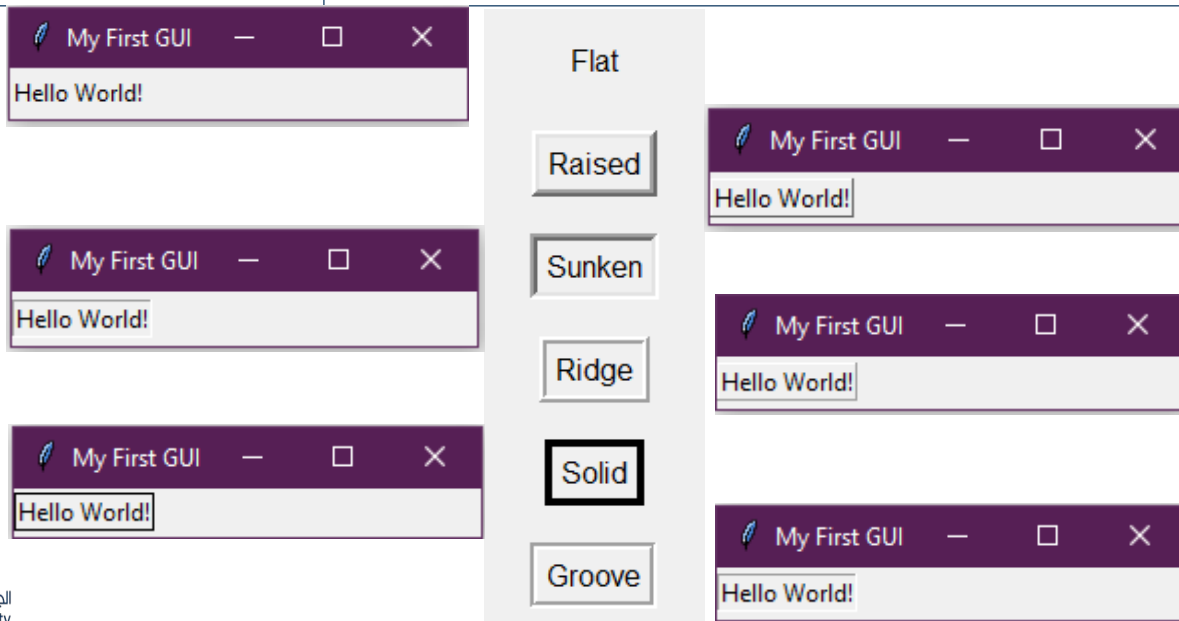


```
self.label = Label(self.main_window,  
                    text='Hello World',  
                    borderwidth=4,  
                    relief='solid')
```



Values for relief Argument (1 of 2)

| relief Argument | Description |
|-----------------|--|
| relief='flat' | The border is hidden and there is no 3D effect. |
| relief='raised' | The widget has a raised 3D appearance. |
| relief='sunken' | The widget has a sunken 3D appearance. |
| relief='ridge' | The border around the widget has a raised 3D appearance. |
| relief='solid' | The border appears as a solid line with no 3D effect. |
| relief='groove' | The border around the widget appears as a groove. |



Example 5

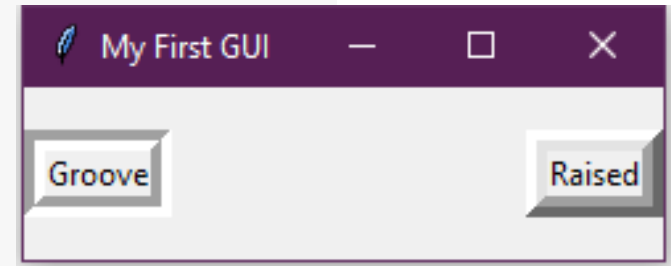
```
# This program displays 2 labels (with groove & raised releif arguments)
from tkinter import *
```

```
class MyGUI:
    def __init__(self):
        # Create the main window widget.
        self.main_window = Tk()
        # Display a title.
        self.main_window.title('My First GUI')
        # Create two Label widgets.
        self.label1 = Label(self.main_window,
                             text='Groove',
                             borderwidth=8, relief='groove')
        self.label2 = Label(self.main_window,
                             text='Raised',
                             borderwidth=8, relief='raised')
        # Call both Label widgets' pack method.
        self.label1.pack(side='left')
        self.label2.pack(side='right')

        # Enter the tkinter main loop.
        mainloop()

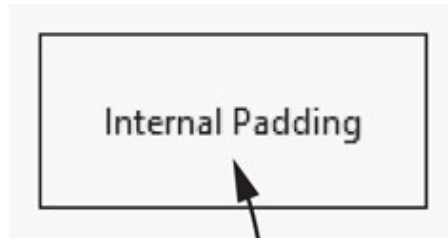
# Create an instance of the MyGUI class.
my_gui = MyGUI()
```

Output

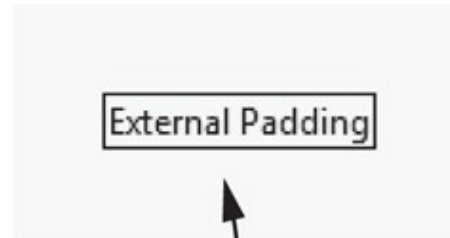


Padding

- **Padding:** space that appears around a widget
 - **Internal padding** appears around the inside edge of a widget
 - **External padding** appears around the outside edge of a widget



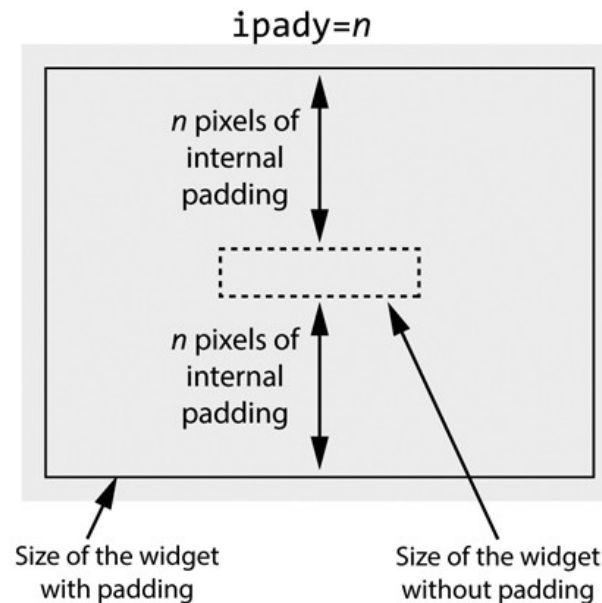
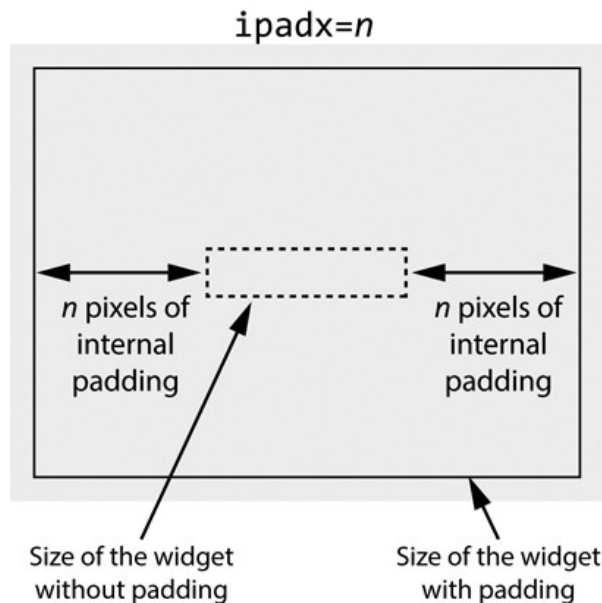
Internal padding is placed inside the widget.



External padding is placed outside the widget.

Internal Padding (1 of 2)

- To add horizontal internal padding to a widget, pass the argument **ipadx**= n to the widget's pack method
- To add vertical internal padding to a widget, pass the argument **ipady**= n to the widget's pack method



Internal Padding- Example (2 of 2)

```
# This program demonstrates internal padding.
import tkinter

class MyGUI:
    def __init__(self):
        # Create the main window widget.
        self.main_window = tkinter.Tk()

        # Create two Label widgets with solid borders.
        self.label1 = tkinter.Label(self.main_window,
                                     text='Hello World!',
                                     borderwidth=1,
                                     relief='solid')

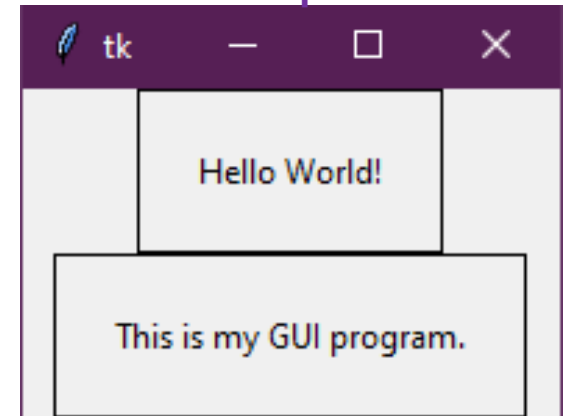
        self.label2 = tkinter.Label(self.main_window,
                                     text='This is my GUI program.',
                                     borderwidth=1,
                                     relief='solid')

        # Display the Labels with 20 pixels of horizontal
        # and vertical internal padding.
        self.label1.pack(ipadx=20, ipady=20)
        self.label2.pack(ipadx=20, ipady=20)

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

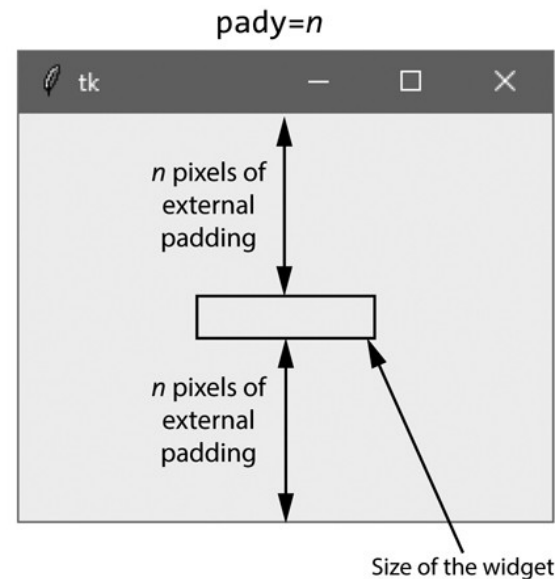
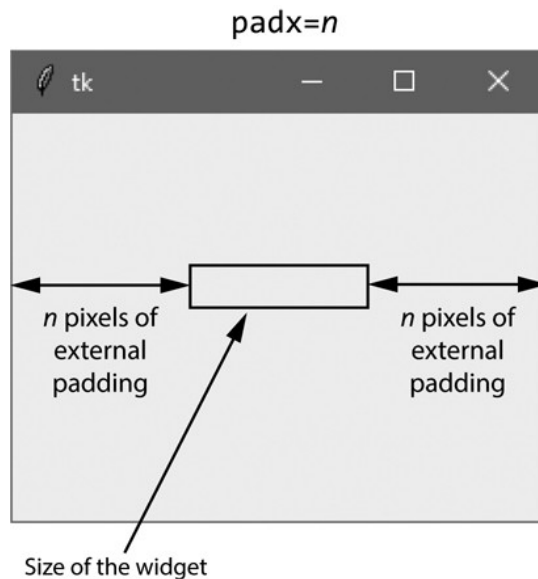
# Create an instance of the MyGUI class.
my_gui = MyGUI()
```

Notice that when used **import tkinter** at the beginning of the program we had to add **tkinter.** before each command



External Padding (1 of 2)

- To add horizontal external padding to a widget, pass the argument **padx**= n to the widget's pack method
- To add vertical external padding to a widget, pass the argument **pady**= n to the widget's pack method



External Padding- Example (2 of 2)

```
# This program demonstrates external padding.
import tkinter

class MyGUI:
    def __init__(self):
        # Create the main window widget.
        self.main_window = tkinter.Tk()

        # Create two Label widgets with solid borders.
        self.label1 = tkinter.Label(self.main_window,
                                     text='Hello World!',
                                     borderwidth=1,
                                     relief='solid')

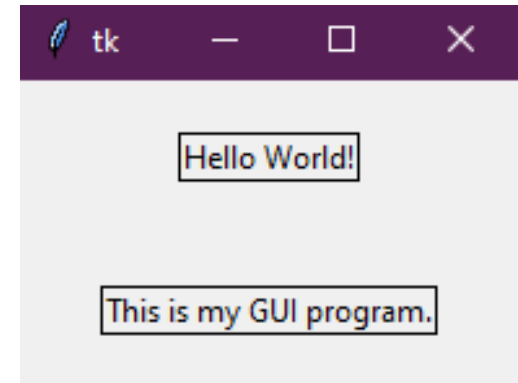
        self.label2 = tkinter.Label(self.main_window,
                                     text='This is my GUI program.',
                                     borderwidth=1,
                                     relief='solid')

        # Display the Labels with 20 pixels of horizontal
        # and vertical external padding.
        self.label1.pack(padx=20, pady=20)
        self.label2.pack(padx=20, pady=20)

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

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

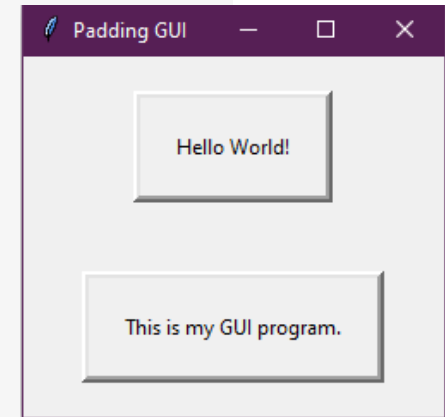
Output



Internal and External Padding- Example

```
1 # This program demonstrates internal and external padding.
2 from tkinter import *
3
4 class MyGUI:
5     def __init__(self):
6         # Create the main window widget.
7         self.main_window = Tk()
8         # Display a title.
9         self.main_window.title('Padding GUI')
10        # Create two Label widgets.
11        self.label1 = Label(self.main_window,
12                             text='Hello World!',
13                             borderwidth=4,
14                             relief='raised')
15
16        self.label2 = Label(self.main_window,
17                             text='This is my GUI program.',
18                             borderwidth=4,
19                             relief='raised')
20
21        # Display the Labels with 20 pixels of horizontal
22        # and vertical external padding.
23        self.label1.pack(side="left",ipadx=20, ipady=20, padx=20, pady=20)
24        self.label2.pack(side="left",ipadx=20, ipady=20, padx=20, pady=20)
25
26        # Enter the tkinter main loop.
27        mainloop()
28
29 # Create an instance of the MyGUI class.
30 my_gui = MyGUI()
```

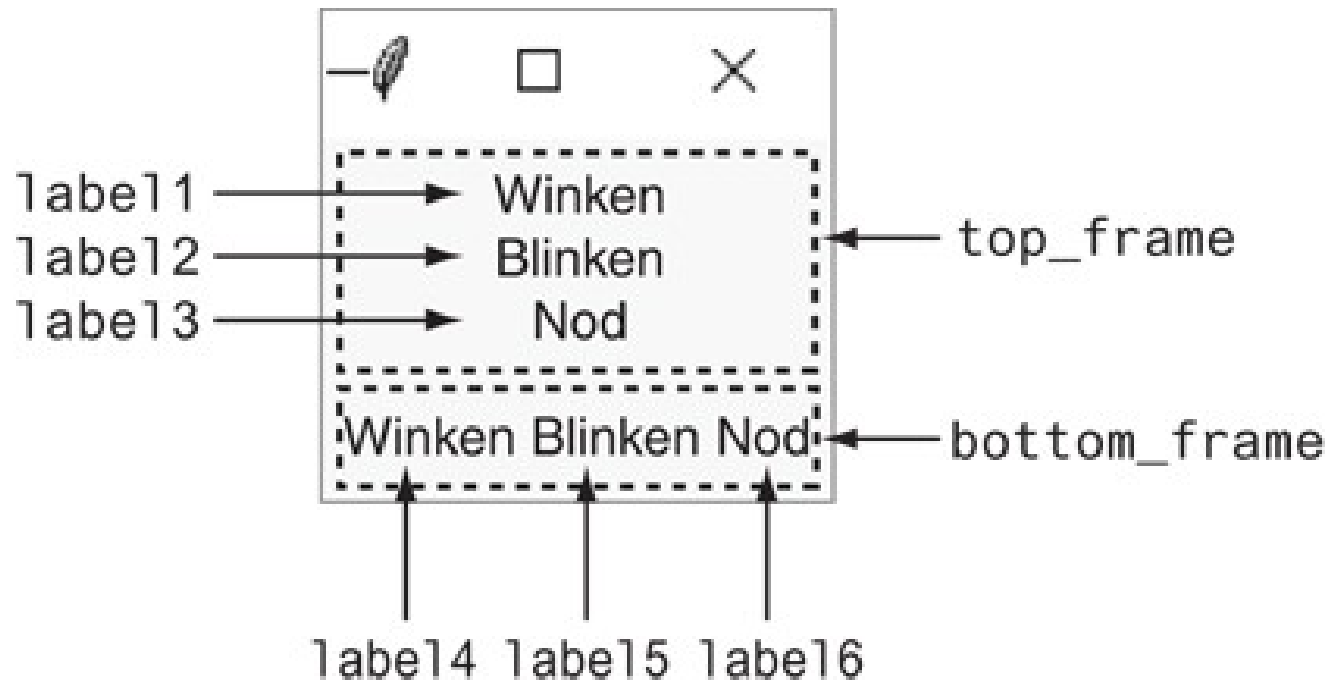
Output



Organizing Widgets with Frames (1 of 2)

- **Frame widget:** A container that holds other widgets
 - Useful for organizing and arranging groups of widgets in a window.
 - For example, you can place a set of widgets in one Frame and arrange them in a particular way, then place a set of widgets in another Frame and arrange them in a different way.
 - Example:
`Label(self.top_frame,text = 'hi')`

Organizing Widgets with Frames (2 of 2)



```
# This program creates labels in three different frames.

from tkinter import *

class MyGUI:
    def __init__(self):
        # Create the main window widget.
        self.main_window = Tk()

        # Create three frames, one for the top of the
        # window, and one for the bottom; and one more third frame
        self.top_frame = Frame(self.main_window)
        self.bottom_frame = Frame(self.main_window)
        self.bbb=Frame(self.main_window)
        # Create three Label widgets for the
        # top frame.
        self.label1 = Label(self.top_frame,
                             text='Location')
        self.label2 = Label(self.top_frame,
                             text='Traffic')
        self.label3 = Label(self.top_frame,
                             text='Date')

        # Pack the labels that are in the top frame.
        # Use the side='top' argument to stack them
        # one on top of the other.
        self.label1.pack(side='top')
        self.label2.pack(side='top')
        self.label3.pack(side='top')
```

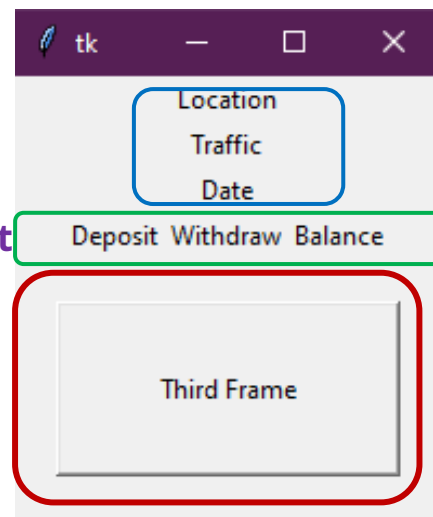
```
# Create three Label widgets for the
# bottom frame.
self.label4 = Label(self.bottom_frame,
                     text='Deposit')
self.label5 = Label(self.bottom_frame,
                     text='Withdraw')
self.label6 = Label(self.bottom_frame,
                     text='Balance')

# Pack the labels that are in the bottom frame.
# Use the side='left' argument to arrange them
# horizontally from the left of the frame.
self.label4.pack(side='left')
self.label5.pack(side='left')
self.label6.pack(side='left')
```

```
# My new frame
self.label99 = Label(self.bbb,text='Third Frame',relief="raised")
self.label99.pack(side='top',ipadx=45,ipady=30,padx=20,pady=20)
# Yes, we have to pack the frames too!
self.top_frame.pack()
self.bottom_frame.pack()
self.bbb.pack()
# Enter the tkinter main loop.
mainloop()
```

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

Output



Exercise

Create a GUI application using a class in Python that displays a **label** and a **frame**. The **label** should show a message, and the **frame** should contain some content with a solid relief and an external padding.

```
from tkinter import *
```

```
class MyGUIApp:
```

```
    def __init__(self):
```

```
        self.window = Tk()
```

```
        self.window.title("GUI App")
```

**Label
part**

```
        self.label = Label(self.window, text="Hello, World!")  
        self.label.pack()
```

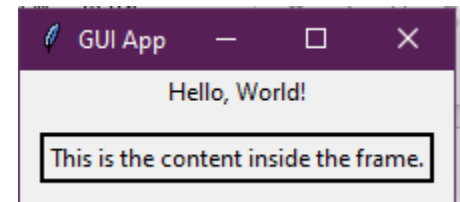
**Frame
with a
label
inside it**

```
        self.frame = Frame(self.window, borderwidth=2, relief="solid")  
        self.frame.pack(padx=10, pady=10)
```

```
        self.content_label = Label(self.frame, text="This is the content inside the  
frame.")
```

```
        self.content_label.pack()
```

```
        self.window.mainloop()
```



**Initialize
r
method**

```
# Create an instance of the class to start the GUI app  
my_app = MyGUIApp()
```