Python Programming

Unit-IV

GUI Programming

❖ Introduction:

A graphical user interface is an application that has buttons, windows, and lots of other widgets that the user can use to interact with your application. A good example would be a web browser. It has buttons, tabs, and a main window where all the content loads.

In simple words setting up a GUI application is similar to an artist's producing a painting. Conventionally, there is a single canvas onto which the artist must put all the work. The way it works is like this: You start with a clean slate, a "top-level" windowing object on which you build the rest of your components.

In GUI programming, a top-level root windowing object contains all of the little windowing objects that will be part of your complete GUI application. These can be text labels, buttons, list boxes, etc. These individual little GUI components are known as widgets. So when we say create a top-level window, we just mean that you need such a thing as a place where you put all your widgets.

Usually, widgets have some associated behaviors, such as when a button is pressed, or text is filled into a text field. These types of user behaviors are called events, and the actions that the GUI takes to respond to such events are known as callbacks.

Python offers multiple options for developing GUI (Graphical User Interface). The most commonly used GUI methods are

- **JPython:** It is the Python platform for Java that is providing Python scripts seamless access o Java class Libraries for the local machine.
- **WxPython:** It is open-source, cross-platform GUI toolkit written in C++. It one of the alternatives to Tkinter, which is bundled with Python.
- **Tkinter:** It is the easiest among all to get started with. It is Python's standard GUI (Graphical User Interface) package. It is the most commonly used toolkit for GUI Programming in Python.

Seriously though, since Tkinter is the Python interface to Tk (Tea Kay), I've always pronounced it Tea-Kay-inter. i.e Tkinter = T K inter.

❖ Tkinter and Python Programming:

Python provides the standard library Tkinter for creating the graphical user interface for desktop based applications.

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, **tkinter** is most commonly used method. It is a standard Python interface to the **Tk GUI** toolkit shipped with Python.

Developing desktop based applications with python Tkinter is not a complex task. An empty Tkinter top-level window can be created by using the following steps.

1. Import the Tkinter module.
2. Create the main application window.
3. Add the widgets like labels, buttons, frames, etc. to the window.
4. Call the main event loop so that the actions can take place on the user's computer screen.

➢ Importing tkinter is same as importing any other module in the python code. Note that the name of the module in Python 2.x is 'Tkinter' and in Python 3.x is 'tkinter'.

  Usage:

  import tkinter (or) **from tkinter import ***

➢ After importing **tkinter** module we need to create a main window, tkinter offers a method **'Tk()'** to create main window. The basic code used to create the main window of the application is:

  top = tkinter.Tk() (or) **top=Tk()**

➢ After creating main window, we need to add components or widgets like labels, buttons, frames, etc.

➢ After adding widgets to main window, we need to run the application, tkinter offers a method **'mainloop ()'** to run application. The basic code used to run the application is:

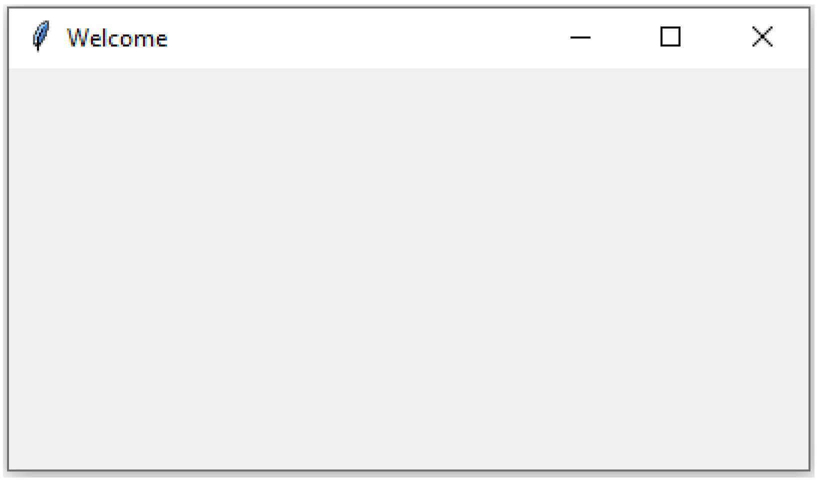
  top.mainloop ()

<u>Example:</u> **tkndemo.py**

```
import tkinter
top = tkinter.Tk()              #creating the application main window.
top.title("Welcome")            #title of main window
top.geometry("400x200")         #size of main window
top.mainloop()                  #calling the event main loop
```

<u>Output:</u>

```
>>> python tkndemo.py
```



- tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows.

Tkinter provides the following geometry methods

1. **pack () method:** It organizes the widgets in blocks before placing in the parent widget.
2. **grid () method:** It organizes the widgets in grid (table-like structure) before placing in the parent widget.
3. **place () method:** It organizes the widgets by placing them on specific positions directed by the programmer.

1. pack () method:

The pack() method is used to organize components or widgets in main window.

Syntax:

```
widget.pack (options)
```

A list of possible options that can be passed inside the pack() method is given below.

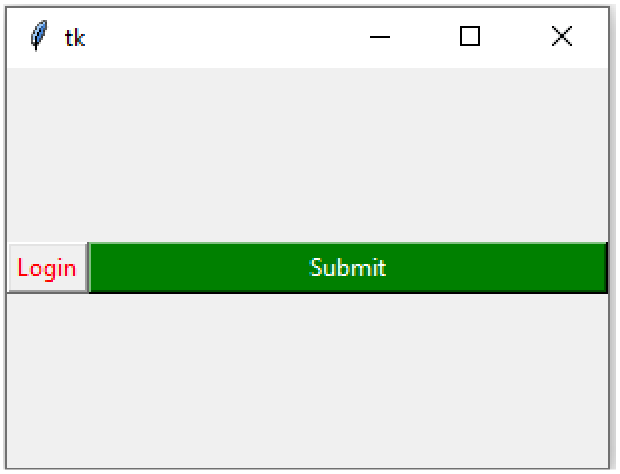
- **side:** it represents the side of the parent to which the widget is to be placed on the window. Side may be LEFT or RIGHT or TOP(default) or BOTTOM.
- **expand –** When set to true, widget expands to fill any space not otherwise used in widget's parent.
- **fill –** Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically).

Example: **tknpack.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
btn1 = Button(top, text = "Login", fg = "red")
btn1.pack( side = LEFT)
btn2 = Button(top, text = "Submit", fg = "white", background="green")
btn2.pack( side = TOP, fill = X , expand = True)
top.mainloop()
```

Output:

```
>>> python tknpack.py
```



2. grid() method:

The grid() method organizes the widgets in the tabular form. We can specify the rows and columns as the options in the method call. We can also specify the column span (width) or row span(height) of a widget.
This is a more organized way to place the widgets to the python application.

Syntax:

```
widget.grid (options)
```

A list of possible options that can be passed inside the grid() method is given below.

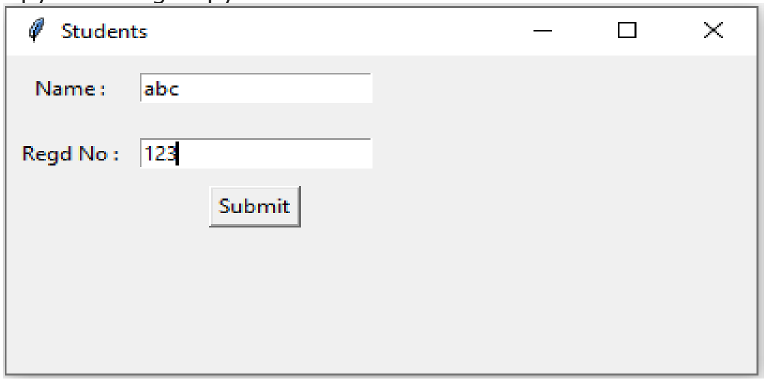
- **Column**
  The column number in which the widget is to be placed. The leftmost column is represented by 0.
- **Columnspan**
  The width of the widget. It represents the number of columns up to which, the column is expanded.
- **ipadx, ipady**
  It represents the number of pixels to pad the widget inside the widget's border.
- **padx, pady**
  It represents the number of pixels to pad the widget outside the widget's border.
- **row**
  The row number in which the widget is to be placed. The topmost row is represented by 0.
- **rowspan**
  The height of the widget, i.e. the number of the row up to which the widget is expanded.

Example: **tkngrid.py**

```
from tkinter import *
parent = Tk()
parent.title("Students")
parent.geometry("300x200")
name = Label(parent,text = "Name : ")
name.grid(row = 0, column = 0,pady=10,padx=5)
e1 = Entry(parent)
e1.grid(row = 0, column = 1)
regno = Label(parent,text = "Regd No : ")
regno.grid(row = 1, column = 0,pady=10,padx=5)
e2 = Entry(parent)
e2.grid(row = 1, column = 1)
btn = Button(parent, text = "Submit")
btn.grid(row = 3, column = 1)
parent.mainloop()
```

Output:

```
>>>python tkngrid.py
```



3. place() method:

The place() method organizes the widgets to the specific x and y coordinates.

Syntax:

```
widget.place(x,y)
```

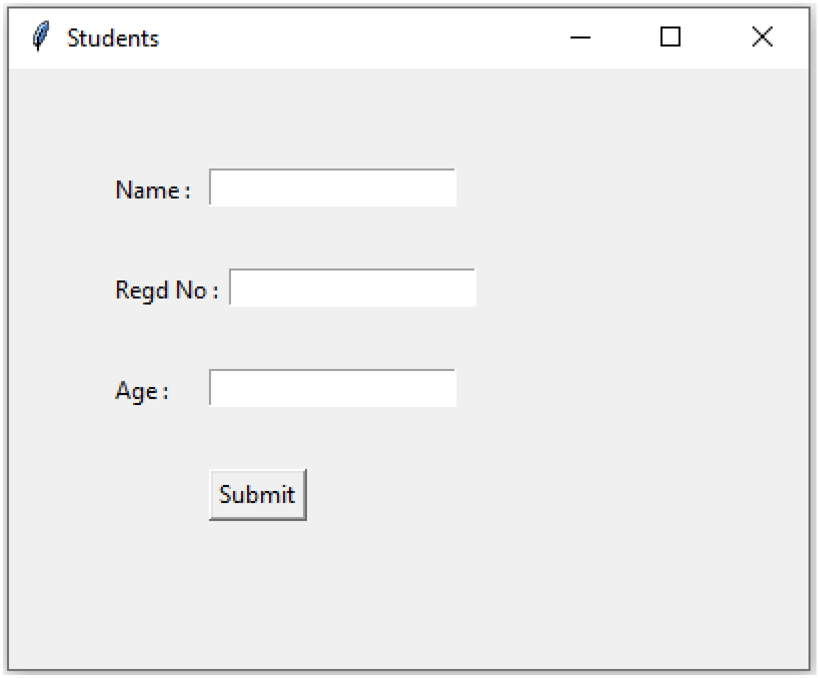
x, y: It refers to the horizontal and vertical offset in the pixels.

Example: **tknplace.py**

```
from tkinter import *
parent = Tk()
parent.title("Students")
parent.geometry("400x300")
name = Label(parent,text = "Name : ")
name.place(x=50,y=50)
e1 = Entry(parent)
e1.place(x=100,y=50)
regno = Label(parent,text = "Regd No : ")
regno.place(x=50,y=100)
e2 = Entry(parent)
e2.place(x=110,y=100)
age = Label(parent,text = "Age : ")
age.place(x=50,y=150)
e2 = Entry(parent)
e2.place(x=100,y=150)
btn = Button(parent, text = "Submit")
btn.place(x=100,y=200)
parent.mainloop()
```

Output:

```
>>>python tknplace.py
```



❖ **Tkinter widgets or components:**

Tkinter supports various widgets or components to build GUI application in python.

Sno	Widget	Description
1	Button	The Button is used to add various kinds of buttons to the python application.
2	Checkbutton	The Checkbutton is used to display the CheckButton on the window.
3	Entry	The entry widget is used to display the single-line text field to the user. It is commonly used to accept user values.
4	Frame	It can be defined as a container to which, another widget can be added and organized.
5	Label	A label is a text used to display some message or information about the other widgets.
6	ListBox	The ListBox widget is used to display a list of options to the user.
7	Radiobutton	The Radiobutton is different from a checkbutton. Here, the user is provided with various options and the user can select only one option among them.
8	Text	It is different from Entry because it provides a multi-line text field to the user so that the user can write the text and edit the text inside it.
9	Scale	It is used to provide the slider to the user.
10	Toplevel	It is used to create a separate window container.

1. Button Widget in Tkinter:

The Button is used to add various kinds of buttons to the python application. We can also associate a method or function with a button which is called when the button is pressed.

Syntax:

```
name = Button(parent, options)
```

Python allows us to configure the look of the button according to our requirements using various options.

A list of possible options is

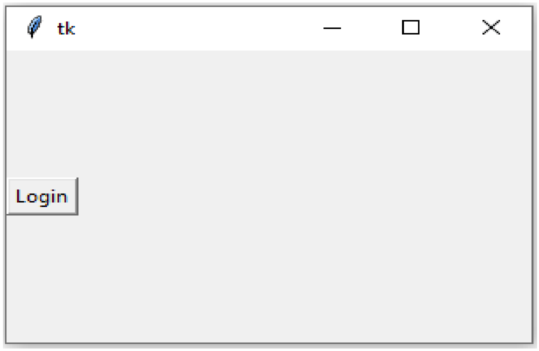
activebackground:	It represents the background of the button when it is active.
activeforeground:	It represents the font color of the button when it is active..
bd:	It represents the border width in pixels.
bg:	It represents the background color of the button.
command:	It is set to the function call which is scheduled when the function is called.
text:	It is set to the text displayed on the button.
fg:	Foreground color of the button.
height:	The height of the button.
image:	It is set to the image displayed on the button.
padx:	Additional padding to the button in the horizontal direction.
pady:	Additional padding to the button in the vertical direction.
underline:	Set this option to make the button text underlined.
width:	The width of the button.

Example: **btndemo.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
btn1 = Button(top, text = "Login")
btn1.pack( side = LEFT)
top.mainloop()
```

Output:

```
>>>python btndemo.py
```

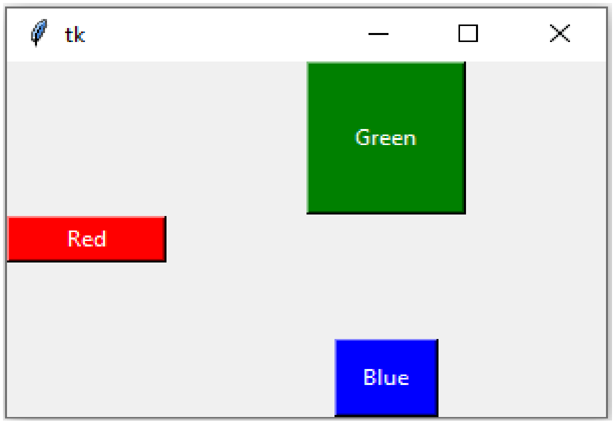


Example: **btndemo1.py**

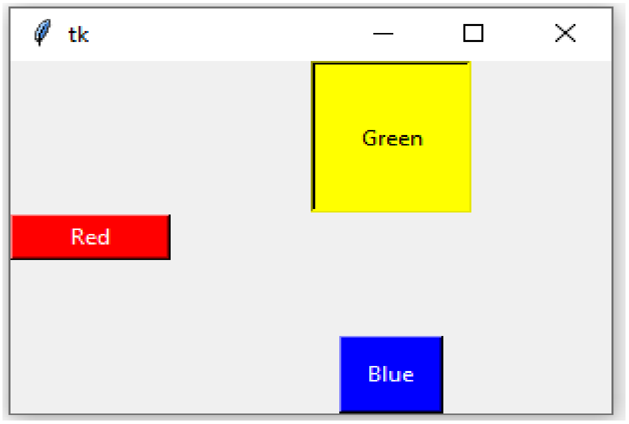
```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("300x200")
def fun():
    messagebox.showinfo("Hello", "Blue Button clicked")

btn1 = Button(top, text = "Red",bg="red",fg="white",width=10)
btn1.pack( side = LEFT)
btn2 = Button(top, text =
"Green",bg="green",fg="white",width=10,height=5,activebackground="yellow")
btn2.pack( side = TOP)
btn3 = Button(top, text = "Blue",bg="blue",fg="white",padx=10,pady=10,command=fun)
btn3.pack( side = BOTTOM)
top.mainloop()
```

Output:

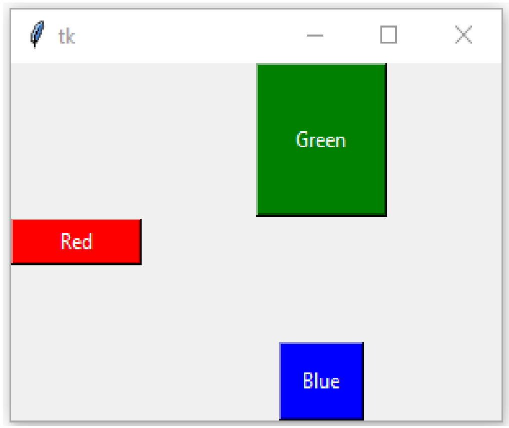
>>>python btndemo1.py

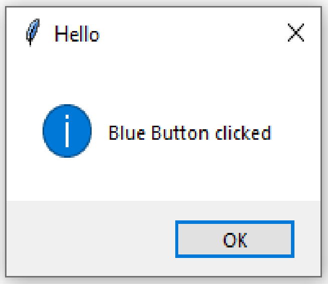


On Green Button active



On Button click





Blue Button clicked

2. Checkbutton Widget in Tkinter:

The Checkbutton is used to display the CheckButton on the window. The Checkbutton is mostly used to provide many choices to the user among which, the user needs to choose the one. It generally implements many of many selections.

Syntax:

```
name = Checkbutton(parent, options)
```

Python allows us to configure the look of the Checkbutton according to our requirements using various options. A list of possible options is

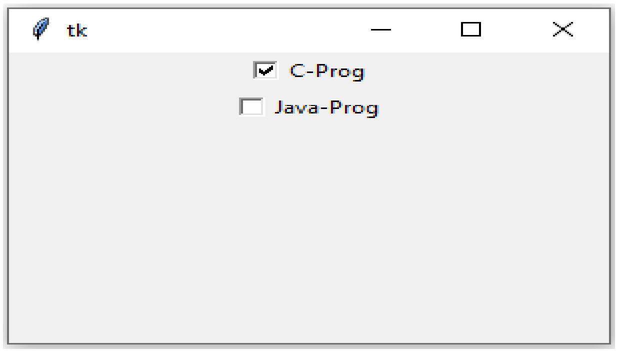
activebackground:	It represents the background of the Checkbutton when it is active.
activeforeground:	It represents the font color of the Checkbutton when when it is active.
bd:	It represents the border width in pixels.
bg:	It represents the background color of the Checkbutton.
command:	It is set to the function call which is scheduled when the function is called.
text:	It is set to the text displayed on the Checkbutton.
fg:	Foreground color of the Checkbutton.
height:	The height of the Checkbutton.
image:	It is set to the image displayed on the Checkbutton.
padx:	Additional padding to the Checkbutton in the horizontal direction.
pady:	Additional padding to the Checkbutton in the vertical direction.
underline:	Set this option to make the Checkbutton text underlined.
width:	The width of the Checkbutton.

Example: **chbutton.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
cbtn1 = Checkbutton(top, text = "C-Prog")
cbtn1.pack()
cbtn2 = Checkbutton(top, text = "Java-Prog")
cbtn2.pack()
top.mainloop()
```

Output:

```
>>>python chbutton.py
```

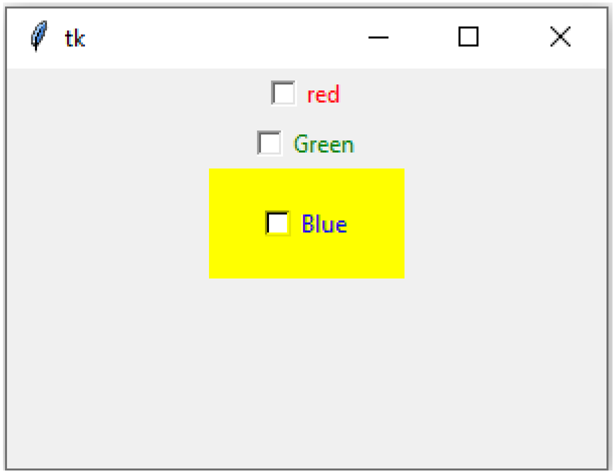


Example: **chbutton1.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
cbtn1 = Checkbutton(top, text="red",fg="red")
cbtn1.pack()
cbtn2 = Checkbutton(top, text="Green",fg="green",activebackground="orange")
cbtn2.pack()
cbtn3 = Checkbutton(top, text="Blue",fg="blue",bg="yellow",width=10,height=3)
cbtn3.pack()
top.mainloop()
```

Output:

```
>>>python chbutton1.py
```



3. Entry Widget in Tkinter:

The Entry widget is used to provide the single line text-box to the user to accept a value from the user. We can use the Entry widget to accept the text strings from the user.

Syntax:

```
name = Entry(parent, options)
```

Python allows us to configure the look of the Entry according to our requirements using various options. A list of possible options is

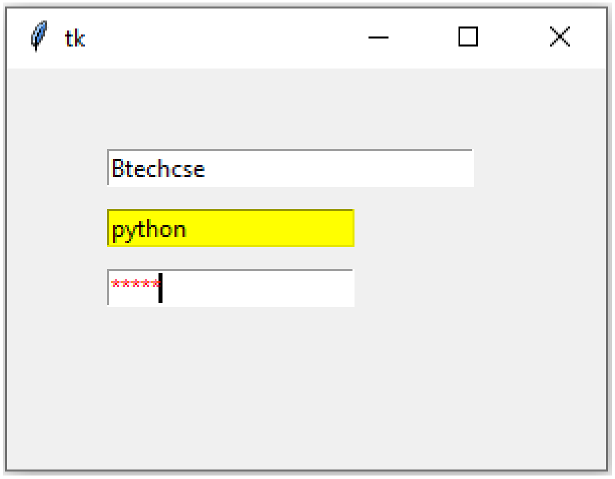
bd:	It represents the border width in pixels.
bg:	It represents the background color of the Entry.
show:	It is used to show the entry text of some other type instead of the string. For example, the password is typed using stars (*).
fg:	Foreground color of the Entry.
width:	The width of the Entry.

Example: **entrydemo.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
enty0 = Entry(top,width="30")
enty0.place(x=50,y=40)
enty1 = Entry(top,bg="yellow")
enty1.place(x=50,y=70)
enty2 = Entry(top,fg="red",show="*")
enty2.place(x=50,y=100)
top.mainloop()
```

Output:

```
>>>python entrydemo.py
```



4. Frame Widget in Tkinter:

Frame widget is used to organize the group of widgets. It acts like a container which can be used to hold the other widgets. The rectangular areas of the screen are used to organize the widgets to the python application.

Syntax:

```
name = Frame(parent, options)
```

Python allows us to configure the look of the Frame according to our requirements using various options. A list of possible options is

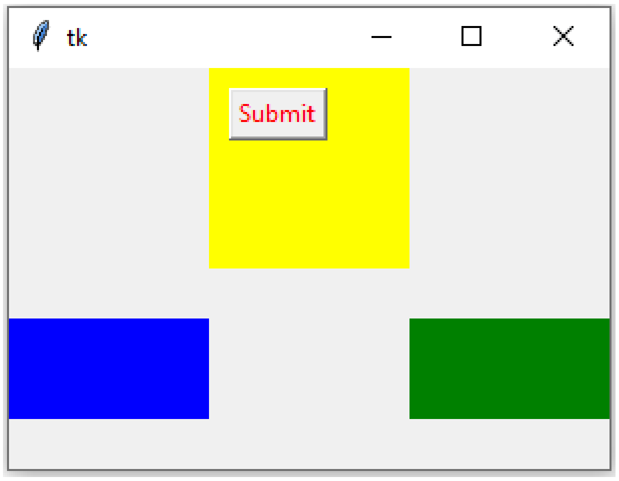
bd:	It represents the border width in pixels.
bg:	It represents the background color of the frame.
width:	The width of the frame.
height:	The height of the frame.

Example: **framedemo.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
tframe = Frame(top,width="100",height="100",bg="yellow")
tframe.pack()
lframe = Frame(top,width="100",height="50",bg="blue")
lframe.pack(side = LEFT)
rframe = Frame(top,width="100",height="50",bg="green")
rframe.pack(side = RIGHT)
btn1 = Button(tframe, text="Submit", fg="red")
btn1.place(x=10,y=10)
top.mainloop()
```

Output:

```
>>>python framedemo.py
```



5. Label Widget in Tkinter:

The Label is used to specify the container box where we can place the text or images.

Syntax:

```
name = Label(parent, options)
```

Python allows us to configure the look of the Label according to our requirements using various options.

A list of possible options is

bd:	It represents the border width in pixels.
bg:	It represents the background color of the label.
text:	It is set to the text displayed on the label.
fg:	Foreground color of the label.
height:	The height of the label.

image: It is set to the image displayed on the label.

padx: Additional padding to the label in the horizontal direction.

pady: Additional padding to the label in the vertical direction.

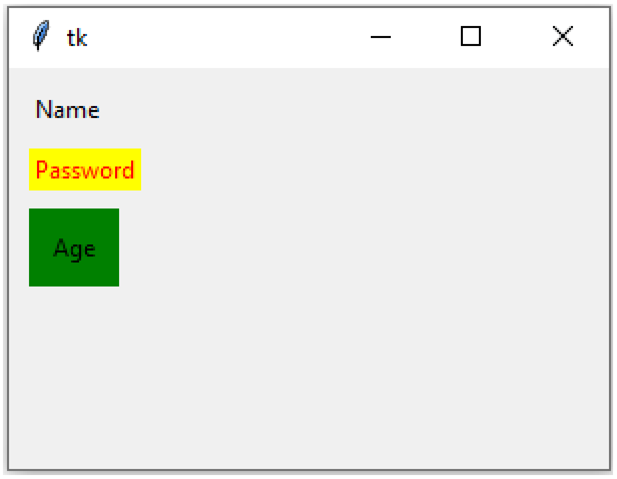
width: The width of the label.

<u>Example:</u> **labeldemo.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
lbl1 = Label(top, text="Name")
lbl1.place(x=10,y=10)
lbl2 = Label(top, text="Password", fg="red",bg="yellow")
lbl2.place(x=10,y=40)
lbl3 = Label(top, text="Age", padx=10,pady=10,bg="green")
lbl3.place(x=10,y=70)
top.mainloop()
```

<u>Output:</u>

```
>>>python labeldemo.py
```



6. Listbox Widget in Tkinter:

The Listbox widget is used to display the list items to the user. We can place only text items in the Listbox. The user can choose one or more items from the list.

<u>Syntax:</u>

```
name = Listbox(parent, options)
```

Python allows us to configure the look of the Listbox according to our requirements using various options. A list of possible options is

bd: It represents the border width in pixels.

bg: It represents the background color of the listbox.

fg: Foreground color of the listbox.

width: The width of the listbox.

height: The height of the listbox.

- The following method is associated with the Listbox to insert list item to listbox at specified index.i.e, **insert ().**

Syntax:

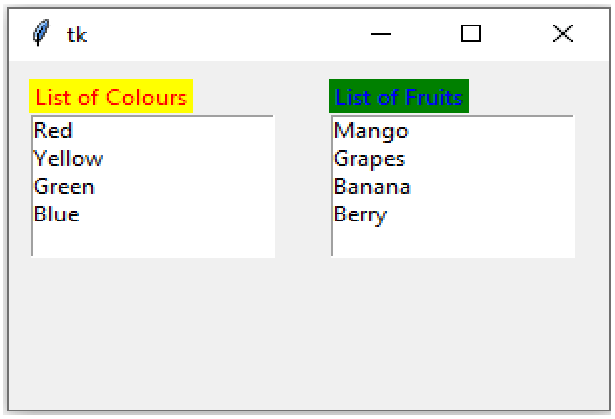
Listbox.insert (index, item)

Example: **listboxdemo.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
lbl1 = Label(top, text="List of Colours",fg="red",bg="yellow")
lbl1.place(x=10,y=10)
lb = Listbox(top,height=5)
lb.insert(1,"Red")
lb.insert(2, "Yellow")
lb.insert(3, "Green")
lb.insert(4, "Blue")
lb.place(x=10,y=30)
lbl2 = Label(top, text="List of Fruits",fg="blue",bg="green")
lbl2.place(x=160,y=10)
lb1 = Listbox(top,height=5)
lb1.insert(1,"Mango")
lb1.insert(2, "Grapes")
lb1.insert(3, "Banana")
lb1.insert(4, "Berry")
lb1.place(x=160,y=30)
top.mainloop()
```

Output:

```
>>>python listboxdemo.py
```



List of Colours

Red
Yellow
Green
Blue

List of Fruits

Mango
Grapes
Banana
Berry

7. Radiobutton Widget in Tkinter:

The Radiobutton widget is used to select one option among multiple options. The Radiobutton is different from a checkbutton. Here, the user is provided with various options and the user can select only one option among them.

Syntax:

```
name = Radiobutton(parent, options)
```

Python allows us to configure the look of the Radiobutton according to our requirements using various options. A list of possible options is

activebackground: It represents the background of the Radiobutton when it is active.

activeforeground: It represents the font color of the Radiobutton when when it is active.

bd: It represents the border width in pixels.

bg: It represents the background color of the Radiobutton.

command: It is set to the function call which is scheduled when the function is called.

text: It is set to the text displayed on the Radiobutton.

fg: Foreground color of the Radiobutton.

height: The height of the Radiobutton.

padx: Additional padding to the Radiobutton in the horizontal direction.

pady: Additional padding to the Radiobutton in the vertical direction.

width: The width of the Radiobutton.

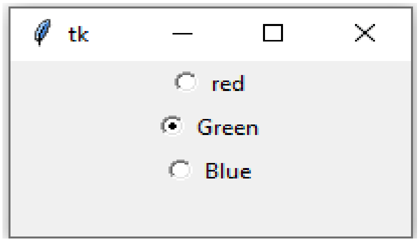
Variable: It is used to keep track of the user's choices. It is shared among all the radiobuttons.

Example: **rbtndemo.py**

```
from tkinter import *
top = Tk()
top.geometry("200x100")
radio = IntVar()
rbtn1 = Radiobutton(top, text="red",variable=radio,value="1")
rbtn1.pack()
rbtn2 = Radiobutton(top, text="Green",variable=radio,value="2")
rbtn2.pack()
rbtn3 = Radiobutton(top, text="Blue",variable=radio,value="3")
rbtn3.pack()
top.mainloop()
```

Output:

```
>>>python rbtndemo.py
```



8. Text Widget in Tkinter:

The Text widget allows the user to enter multiple lines of text.It is different from Entry because it provides a multi-line text field to the user so that the user can write the text and edit the text inside it.

Syntax:

```
name = Text(parent, options)
```

Python allows us to configure the look of the Text according to our requirements using various options. A list of possible options is

bd: It represents the border width in pixels.

bg: It represents the background color of the Text.

show: It is used to show the entry text of some other type instead of the string. For example, the password is typed using stars (*).

fg: Foreground color of the Text.

width: The width of the Text.

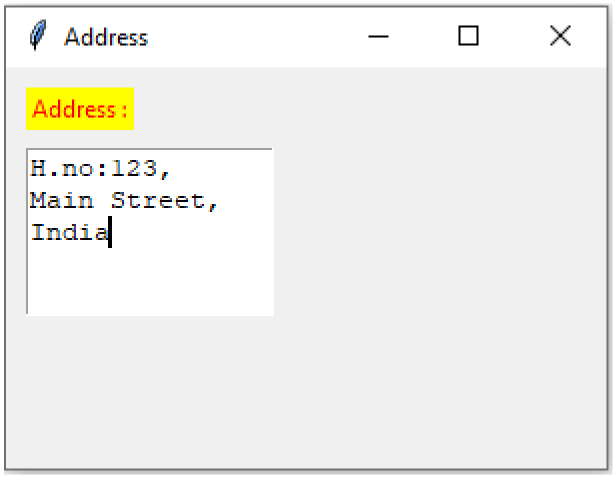
height: The vertical dimension of the widget in lines.

Example: **textdemo.py**

```
from tkinter import *
top = Tk()
top.title("Address")
top.geometry("300x200")
lbl=Label(top,text="Address :",fg="red",bg="yellow")
lbl.place(x=10,y=10)
txt=Text(top,width=15,height=5)
txt.place(x=10,y=40)
top.mainloop()
```

Output:

```
>>>python textdemo.py
```



```
H.no:123,
Main Street,
India
```

9. Scale Widget in Tkinter:

The Scale widget is used to implement the graphical slider to the python application so that the user can slide through the range of values shown on the slider and select the one among them.

Syntax:

```
name = Scale(parent, options)
```

Python allows us to configure the look of the Scale according to our requirements using various options. A list of possible options is

activebackground: It represents the background of the Scale when it is active.

bd: It represents the border width in pixels.

bg: It represents the background color of the Scale.

command: It is set to the function call which is scheduled when the function is called.

fg: Foreground color of the Scale.

from_: It is used to represent one end of the widget range.

to: It represents a float or integer value that specifies the other end of the range represented by the scale.

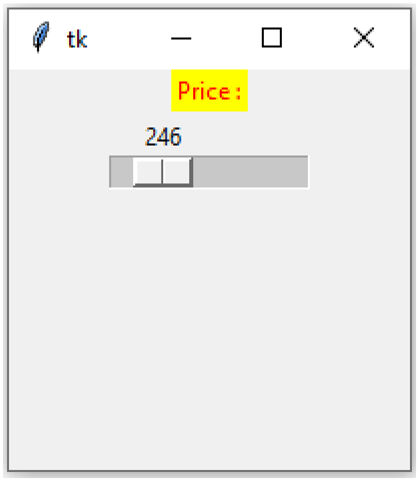
orient: It can be set to horizontal or vertical depending upon the type of the scale.

Example: **scaledemo.py**

```
from tkinter import *
top = Tk()
top.geometry("200x200")
lbl=Label(top,text="Price :",bg="yellow",fg="red")
lbl.pack()
scale = Scale( top, from_ = 100, to = 1000, orient = HORIZONTAL)
scale.pack(anchor=CENTER)
top.mainloop()
```

Output:

```
>>>python scaledemo.py
```



10. Toplevel Widget in Tkinter:

The Toplevel widget is used to create and display the toplevel windows which are directly managed by the window manager.

Syntax:

```
name = Toplevel(options)
```

Python allows us to configure the look of the Toplevel according to our requirements using various options. A list of possible options is

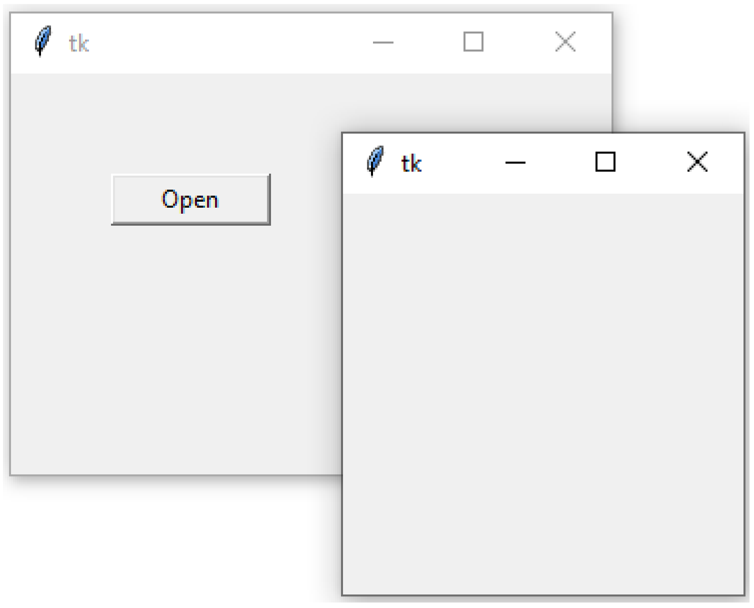
bd:	It represents the border width in pixels.
bg:	It represents the background color of the Toplevel.
fg:	Foreground color of the Toplevel.
width:	The width of the Toplevel.
height:	The vertical dimension of the widget in lines.

Example: **topleveldemo.py**

```
from tkinter import *
top = Tk()
top.geometry("300x200")
def fun():
        chld = Toplevel(top)
        chld.mainloop()
btn1 = Button(top, text = "Open",width=10,command=fun)
btn1.place(x=50,y=50)
top.mainloop()
```

Output:

```
>>>python topleveldemo.py
```



❖ Brief Tour of Other GUIs:

Python offers multiple options for developing GUI (Graphical User Interface). The most commonly used GUI methods are

Tix (Tk Interface eXtensions):
Tix, which stands for Tk Interface Extension, is an extension library for Tcl/Tk. Tix adds many new widgets, image types and other commands that allows you to create compelling Tcl/Tk-based GUI applications.

One advantage of Tix over other Tk widget libraries is many of the Tix standard widgets are implemented in native code. This enhances performance and provides native look-and-feel for your applications.

Tix includes the following standard widgets which, like their counterparts in Tk, are implemented in native code to achieve high performance and native look-and-feel.

tixGrid

The tixGrid widget displays items in a spread-sheet format.

tixHList

Hierarchical listbox widget. This widget display entries in a tree-like format.

tixInputOnly

A transparent window that can be used to cover another widget so as to disable mouse input.

tixNBFrame

The tixNBFrame widget is used internally by the tixNoteBook widget to display choices among a set of overlapping pages.

tixTList

Tabular listbox widget. This widget is similar to the built-in Tk listbox widget but provides more flexibility in displaying the list entries.

Pmw (Python MegaWidgets Tkinter extension), wxPython (Python binding to wxWidgets), and PyGTK (Python binding to GTK+).