

# GUI Programming

**Graphical User Interface (GUI)** is nothing but a desktop application which helps you to interact with the computers. They are used to perform different tasks in the desktops, laptops and other electronic devices.

- **GUI** apps like **Text-Editors** are used to create, read, update and delete different types of files.
- **GUI** apps like **Sudoku, Chess and Solitaire** are games which you can play.
- **GUI** apps like **Google Chrome, Firefox and Microsoft Edge** are used to browse through the **Internet**.

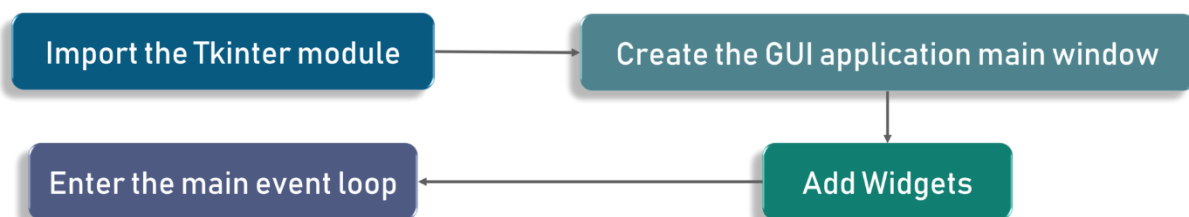
They are some different types of **GUI** apps which we daily use on the laptops or desktops.

## What Is Tkinter?

**Tkinter** is actually an inbuilt **Python** module used to create simple **GUI** apps. It is the most commonly used module for **GUI** apps in the **Python**.

## Fundamentals Of Tkinter

Consider the following diagram, it shows how an application actually executes in Tkinter:



To start out with, we first import the Tkinter model. Followed by that, we create the main window. It is in this window that we are performing operations and displaying visuals and everything basically. Later, we add the widgets and lastly we enter the main event loop.

These are the 2 keywords:

- Widgets
- Main Event Loop

An event loop is basically telling the code to keep displaying the window until we manually close it. It runs in an infinite loop in the back-end.

Check out the following code for better clarity:

```
1  import tkinter
2
3  window = tkinter.Tk()
4
5  # to rename the title of the window window.title("GUI")
6
7  # pack is used to show the object in the window
8
9  label = tkinter.Label(window, text = "Hello World!").pack()
10
11 window.mainloop()
```

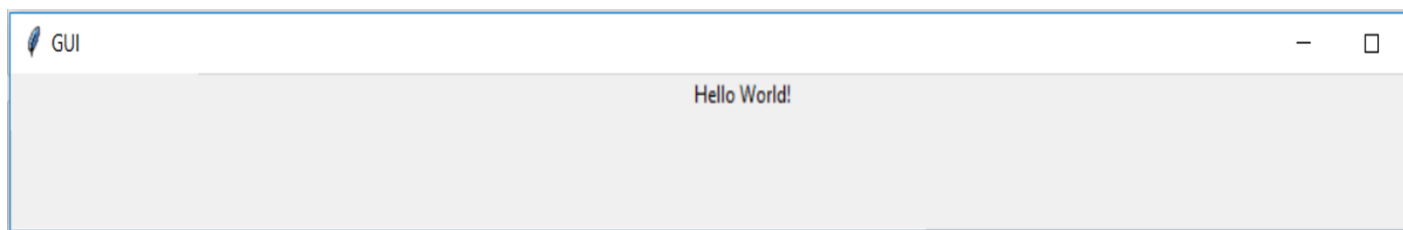
As you can see, we are importing the Tkinter package and defining a window. Followed by that, we are giving a window title

which is shown on the title tab whenever you open an application.

For example, Microsoft Word is shown on the title tab when you open a word application. Similarly here we call it GUI. We can call it anything we want based on the requirement.

Lastly, we have a label. A label is nothing is but what output needs to be shown on the window. In this case as you can already see, it is hello world.

Check out the output for the above code:

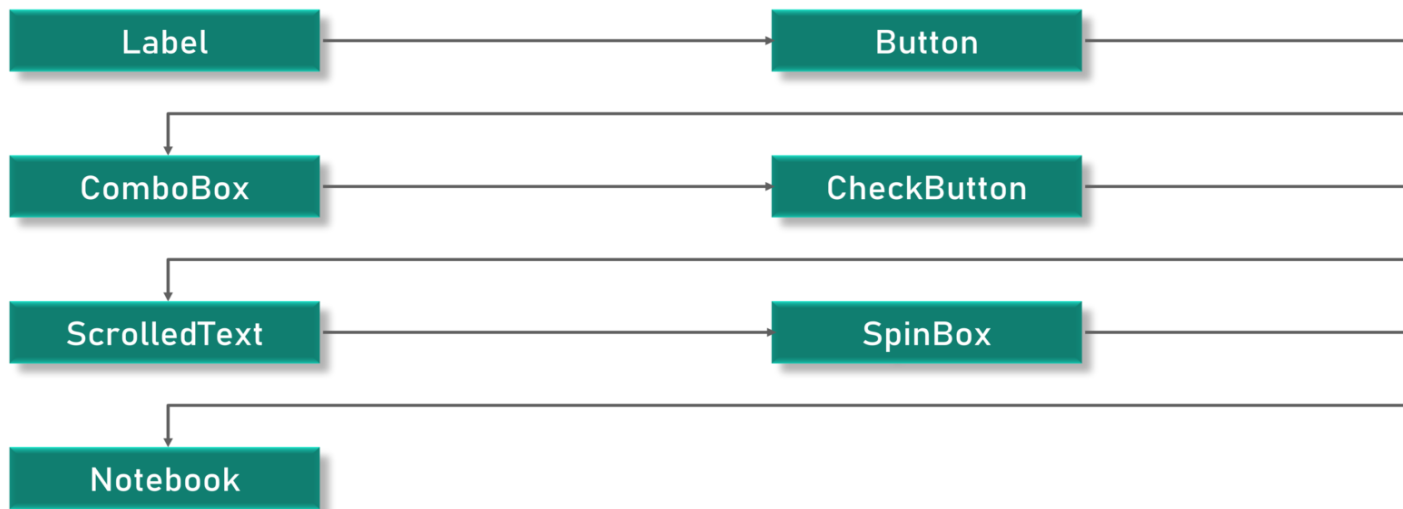


## Tkinter Widgets

**Widgets** are something like elements in the **HTML**. You will find different types of **widgets** to the different types of elements in the **Tkinter**.

Check out this diagram for the list of the majorly used Tkinter widgets:

Check out this diagram for the list of the majorly used Tkinter widgets:



- **Canvas** – Canvas is used to draw shapes in your GUI.
  - **Button** – Button widget is used to place the buttons in the Tkinter.
  - **Checkbutton** – Checkbutton is used to create the check buttons in your application. Note that you can select more than one option at a time.
  - **Radiobutton** The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
- 
- **Entry** – Entry widget is used to create input fields in the GUI.
  - **Frame** – Frame is used as containers in the Tkinter.
  - **Label** – Label is used to create a single line widgets like text, images etc.
  - **Menu** – Menu is used to create menus in the GUI.
  - **tkMessageBox** This module is used to display message boxes in your applications
  - **Text** The Text widget is used to display text in multiple lines.

- **Scrollbar** The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.

These widgets are the reason that Tkinter is so popular. It makes it really easy to understand and use practically.

## Geometry Management

All widgets in the **Tkinter** will have some geometry measurements. These measurements give you to organize the widgets and their parent frames, windows and so on.

**Tkinter** has the following three Geometry Manager classes.

- **pack()**:- It organizes the widgets in the block, which mean it occupies the entire available width. It's a standard method to show the widgets in the window.
- **grid()**:- It organizes the widgets in table-like structure.
- **place()**:- It's used to place the widgets at a specific position you want.

## PACK

Pack is the easiest to use of the three geometry managers of Tk and Tkinter. Instead of having to declare precisely where a widget should appear on the display screen, we can declare the positions of widgets with the pack command relative to each other. The pack command takes care of the details. Though the pack command is easier to use, this layout managers is limited

in its possibilities compared to the grid and place managers. For simple applications it is definitely the manager of choice. For example simple applications like placing a number of widgets side by side, or on top of each other.

Example:

```
from Tkinter import *  
  
root = Tk()  
  
Label(root, text="Red Sun", bg="red", fg="white").pack()  
Label(root, text="Green Grass", bg="green", fg="black").pack()  
Label(root, text="Blue Sky", bg="blue", fg="white").pack()  
  
mainloop()
```



## FILL

In our example, we have packed three labels into the parent widget "root". We used pack() without any options. So pack had to decide which way to arrange the labels. As you can see, it has chosen to place the label widgets on top of each other and centre them. Furthermore, we can see that each label has been given the size of the text. If you want to make the widgets as wide as the parent widget, you have to use the

fill=X option:

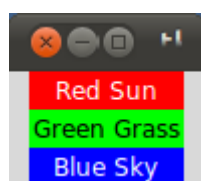
```
from Tkinter import *  
  
root = Tk()  
  
w = Label(root, text="Red Sun", bg="red", fg="white")  
w.pack(fill=X)  
  
w = Label(root, text="Green Grass", bg="green",  
fg="black")  
w.pack(fill=X)  
  
w = Label(root, text="Blue Sky", bg="blue", fg="white")  
w.pack(fill=X)  
  
mainloop()
```



## PADDING

The pack() manager knows four padding options, i.e. internal and external padding and padding in x and y direction:

padx                  External padding, horizontally

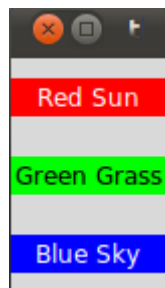


The code for the window above:

```
from Tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red",
fg="white")
w.pack(fill=X,padx=10)
w = Label(root, text="Green Grass",
bg="green", fg="black")
w.pack(fill=X,padx=10)
w = Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack(fill=X,padx=10)
mainloop()
```

pady

External padding, vertically



The code for the window above:

```
from Tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red",
fg="white")
w.pack(fill=X,pady=10)
w = Label(root, text="Green Grass",
bg="green", fg="black")
```



```
w.pack(fill=X,pady=10)
w = Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack(fill=X,pady=10)
mainloop()
```

ipadx      Internal padding, horizontally.

In the following example, we change only the label with the text "Green Grass", so that the result can be easier recognized. We have also taken out the fill option.



```
from Tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red",
fg="white")
w.pack()
w = Label(root, text="Green Grass",
bg="green", fg="black")
w.pack(ipadx=10)
w = Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack()
mainloop()
```

ipady      Internal padding, vertically

We will change the last label of our previous

example to ipady=10.

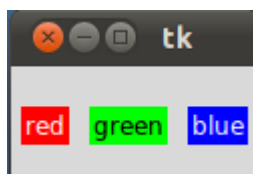


```
from Tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red",
fg="white")
w.pack()
w = Label(root, text="Green Grass",
bg="green", fg="black")
w.pack(ipadx=10)
w = Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack(ipady=10)
mainloop()
```

The default value in all cases is 0.

## PLACING WIDGETS SIDE BY SIDE

We want to place the three label side by side now and shorten the text slightly:



The corresponding code looks like this:

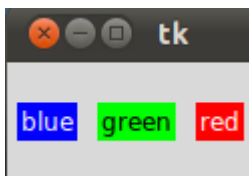
```
from Tkinter import *

root = Tk()

w = Label(root, text="red", bg="red", fg="white")
w.pack(padx=5, pady=10, side=LEFT)
w = Label(root, text="green", bg="green", fg="black")
w.pack(padx=5, pady=20, side=LEFT)
w = Label(root, text="blue", bg="blue", fg="white")
w.pack(padx=5, pady=20, side=LEFT)

mainloop()
```

If we change LEFT to RIGHT in the previous example, we get the colours in reverse order:



## **Grid**

The **Grid** geometry manager puts the widgets in a 2-dimensional table. The master widget is split into a number of rows and columns, and each "cell" in the resulting table can hold a widget.

Using the grid manager is easy. Just create the widgets, and use the **grid** method to tell the manager in which row and

column to place them. You don't have to specify the size of the grid beforehand; the manager automatically determines that from the widgets in it.

```
Label(master, text="First").grid(row=0)
```

```
Label(master, text="Second").grid(row=1)
```

```
e1 = Entry(master)
```

```
e2 = Entry(master)
```

```
e1.grid(row=0, column=1)
```

```
e2.grid(row=1, column=1)
```

Note that the column number defaults to 0 if not given.

Running the above example produces the following window:

### Simple grid example

Empty rows and columns are ignored. The result would have been the same if you had placed the widgets in row 10 and 20 instead.

Note that the widgets are centered in their cells. You can use the **sticky** option to change this; this option takes one or more values from the set **N**, **S**, **E**, **W**. To align the labels to the left border, you could use **W** (west):

```
Label(master, text="First").grid(row=0, sticky=W)
```

```
Label(master, text="Second").grid(row=1, sticky=W)
```

```
e1 = Entry(master)
```

```
e2 = Entry(master)
```

```
e1.grid(row=0, column=1)
```

```
e2.grid(row=1, column=1)
```

## Using the sticky option

You can also have the widgets span more than one cell.

The **columnspan** option is used to let a widget span more than one column, and the **rowspan** option lets it span more than one row. The following code creates the layout shown in the previous section:

```
label1.grid(sticky=E)
```

```
label2.grid(sticky=E)
```

```
entry1.grid(row=0, column=1)
```

```
entry2.grid(row=1, column=1)
```

```
checkboxbutton.grid(columnspan=2, sticky=W)
```

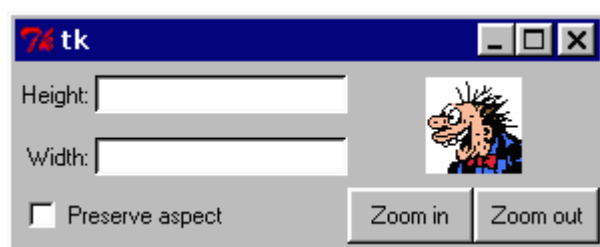
```
image.grid(row=0, column=2, columnspan=2, rowspan=2,  
            sticky=W+E+N+S, padx=5, pady=5)
```

```
button1.grid(row=2, column=2)
```

```
button2.grid(row=2, column=3)
```

There are plenty of things to note in this example. First, no position is specified for the label widgets. In this case, the column defaults to 0, and the row to the *first unused row in the grid*. Next, the entry widgets are positioned as usual, but the checkbutton widget is placed on the next empty row (row 2, in this case), and is configured to span two columns. The resulting cell will be as wide as the label and entry columns combined. The image widget is configured to span both columns and rows at the same time. The buttons, finally, is packed each in a single cell:

### Using column and row spans



## **FRAME**

To arrange the layout in the **window**, we will use **Frame**, class. Let's create a simple program to see how the **Frame** works.

### Steps:-

- **Frame** is used to create the divisions in the window. You can align the frames as you like with **side** parameter of **pack()** method.

- **Button** is used to create a button in the window. It takes several parameters like **text** (Value of the Button), **fg** (Color of the text), **bg** (Background color)

**Note** – The parameter of any **widget** method must be where to place the widget.

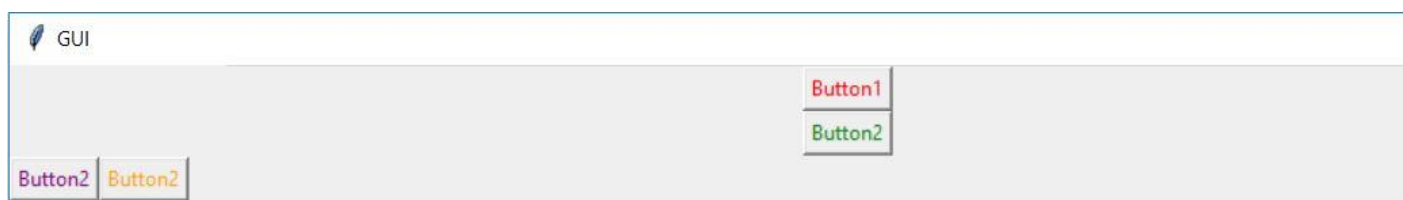
In the below code, we use to place in the **window**, **top\_frame**, **bottom\_frame**.

```

1 import tkinter
2 window = tkinter.Tk()
3 window.title("GUI")
4 # creating 2 frames TOP and BOTTOM
5 top_frame = tkinter.Frame(window).pack()
6 bottom_frame = tkinter.Frame(window).pack(side = "bottom")
7 # now, create some widgets in the top_frame and bottom_frame
8 btn1 = tkinter.Button(top_frame, text = "Button1", fg = "red").pack()
9 btn2 = tkinter.Button(top_frame, text = "Button2", fg = "green").pack()
10 btn3 = tkinter.Button(bottom_frame, text = "Button2", fg = "purple")
11 widgets
12 btn4 = tkinter.Button(bottom_frame, text = "Button2", fg = "orange")
13 window.mainloop()

```

Above code produces the following **window**, if you didn't change the above code.



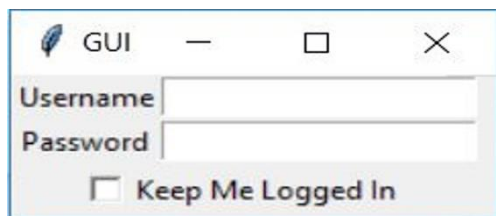
See the below example to get an idea of how it works.

```

1 import tkinter
2 window = tkinter.Tk()
3 window.title("GUI")
4 # creating 2 text labels and input labels
5 tkinter.Label(window, text = "Username").grid(row = 0) # this is placed in 0
6 # 'Entry' is used to display the input-field
7 tkinter.Entry(window).grid(row = 0, column = 1) # this is placed in 1
8 tkinter.Label(window, text = "Password").grid(row = 1) # this is placed in 0
9 tkinter.Entry(window).grid(row = 1, column = 1) # this is placed in 1
10 # 'Checkbutton' is used to create the check buttons
11 tkinter.Checkbutton(window, text = "Keep Me Logged In").grid(column = 1, row = 2)
12 # you can also use 'rowspan' in the similar manner
13 window.mainloop()

```

You will get the following output:



## LABEL

To add a label to our previous example, we will create a label using the label class like this:

```
lbl = Label(window, text="Hello")
```

Then we will set its position on the form using the grid function and give it the location like this:

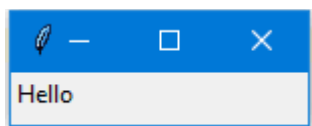
```
lbl.grid(column=0, row=0)
```



So the complete code will be like this:

```
1 from tkinter import *
2
3 window = Tk()
4
5 window.title("Welcome to LikeGeeks app")
6
7 lbl = Label(window, text="Hello")
8
9 lbl.grid(column=0, row=0)
10
11 window.mainloop()
```

And this is the result:



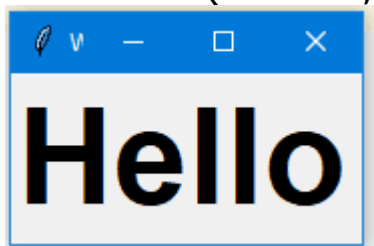
Without calling the grid function for the label, it won't show up.

Set label font size

You can set the label font so you can make it bigger and maybe bold. You can also change the font style.

To do so, you can pass the font parameter like this:

```
lbl = Label(window, text="Hello", font=("Arial Bold", 50))
```



Note that the font parameter can be passed to any widget to change its font not labels only.

Great, but the window is so small, we can even see the title, what about setting the window size?

Setting window size

We can set the default window size using geometry function like this:

```
window.geometry('350x200')
```

The above line sets the window width to 350 pixels and the height to 200 pixels.

Let's try adding more GUI widgets like buttons and see how to handle button click event.

## **BUTTON**

Let's start by adding the button to the window, the button is created and added to the window the same as the label:

```
1 btn = Button(window, text="Click Me")
```

```
2
```

```
3 btn.grid(column=1, row=0)
```

So our window will be like this:

```
1 from tkinter import *
```

```
2
```

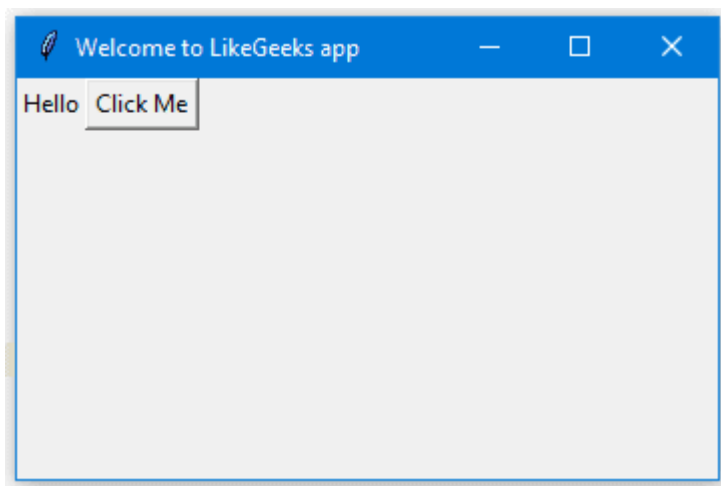
```
3 window = Tk()
```

```
4
```

```
5 window.title("Welcome to LikeGeeks app")
```

```
6 |  
7 | window.geometry('350x200')  
8 |  
9 | lbl = Label(window, text="Hello")  
10 |  
11 | lbl.grid(column=0, row=0)  
12 |  
13 | btn = Button(window, text="Click Me")  
14 |  
15 | btn.grid(column=1, row=0)  
16 |  
17 | window.mainloop()
```

The result looks like this:



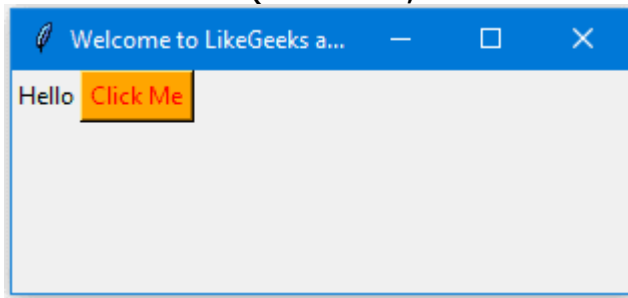
Note that we place the button on the second column of the window which is 1. If you forget and place the button on the same column which is 0, it will show the button only, since the button will be on the top of the label.

Change button foreground and background colors

You can change foreground for a button or any other widget using **fg** property.

Also, you can change the background color for any widget using **bg** property.

```
btn = Button(window, text="Click Me", bg="orange", fg="red")
```



Now, if you tried to click on the button, nothing happens because the click event of the button isn't written yet.

Handle button click event

First, we will write the function that we need to execute when the button is clicked:

```
1 def clicked():
```

```
2
```

```
3     lbl.configure(text="Button was clicked !!")
```

Then we will wire it with the button by specifying the function like this:

```
btn = Button(window, text="Click Me", command=clicked)
```

**Note that**, we typed clicked only not clicked() with parentheses.

Now the full code will be like this:

```
1 from tkinter import *
```

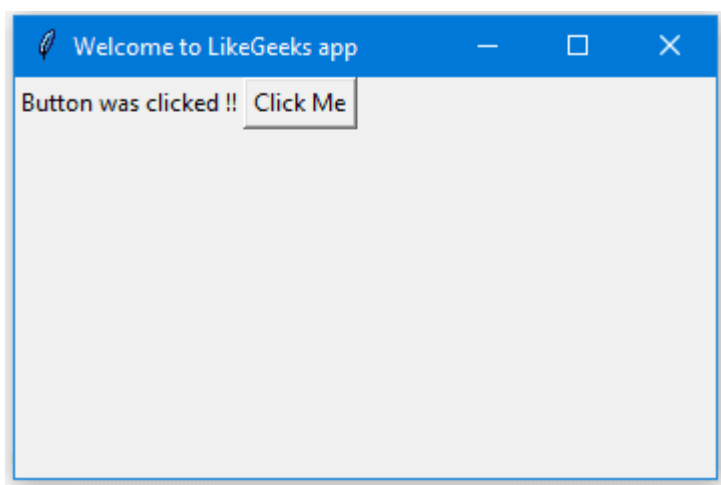
```
2
```

```
3 window = Tk()
```

```
4
```

```
5 window.title("Welcome to LikeGeeks app")
6
7 window.geometry('350x200')
8
9 lbl = Label(window, text="Hello")
10
11 lbl.grid(column=0, row=0)
12
13 def clicked():
14
15     lbl.configure(text="Button was clicked !!")
16
17 btn = Button(window, text="Click Me", command=clicked)
18
19 btn.grid(column=1, row=0)
20
21 window.mainloop()
```

And when we click the button, the result as expected:



## ENTRY BOX

In the previous Python GUI examples, we saw how to add simple widgets, now let's try getting the user input using Tkinter Entry class (Tkinter textbox).

You can create a textbox using Tkinter Entry class like this:

```
txt = Entry(window,width=10)
```

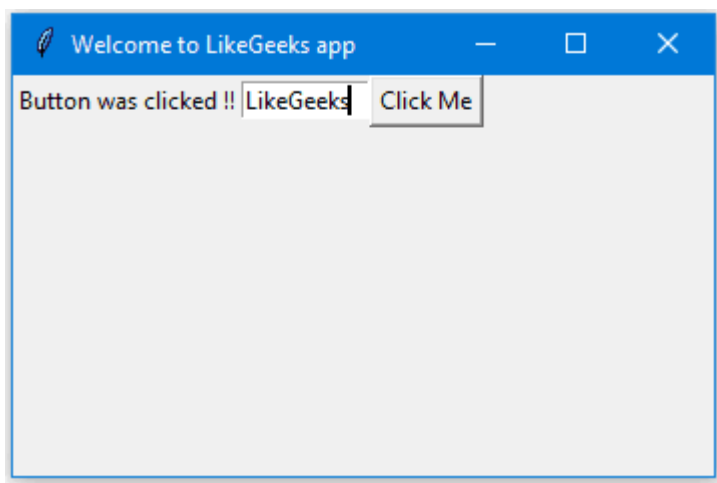
Then you can add it to the window using grid function as usual

So our window will be like this:

```
1 from tkinter import *
2
3 window = Tk()
4
5 window.title("Welcome to LikeGeeks app")
6
7 window.geometry('350x200')
8
9 lbl = Label(window, text="Hello")
10
11 lbl.grid(column=0, row=0)
12
13 txt = Entry(window,width=10)
14
```

```
15 txt.grid(column=1, row=0)
16
17 def clicked():
18
19     lbl.configure(text="Button was clicked !!")
20
21 btn = Button(window, text="Click Me", command=clicked)
22
23 btn.grid(column=2, row=0)
24
25 window.mainloop()
```

And the result will be like this:



Now, if you click the button, it will show the same old message, what about showing the entered text on the Entry widget?

First, you can get entry text using get function. So we can write this code to our clicked function like this:

```
1 def clicked():
```

```
2 |  
3 res = "Welcome to " + txt.get()  
4 |  
5 lbl.configure(text= res)
```

If you click the button and there is a text on the entry widget, it will show "Welcome to" concatenated with the entered text.

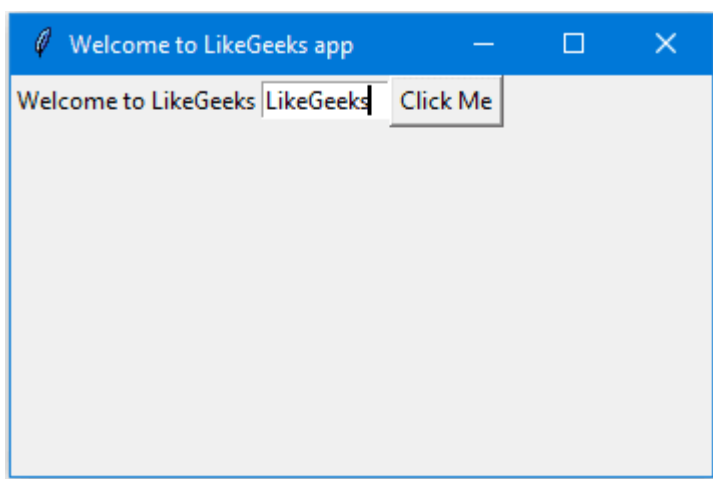
And this is the complete code:

```
1 from tkinter import *  
2 |  
3 window = Tk()  
4 |  
5 window.title("Welcome to LikeGeeks app")  
6 |  
7 window.geometry('350x200')  
8 |  
9 lbl = Label(window, text="Hello")  
10 |  
11 lbl.grid(column=0, row=0)  
12 |  
13 txt = Entry(window,width=10)  
14 |  
15 txt.grid(column=1, row=0)  
16 |  
17 def clicked():  
18 |
```



```
19 res = "Welcome to " + txt.get()
20
21 lbl.configure(text= res)
22
23 btn = Button(window, text="Click Me", command=clicked)
24
25 btn.grid(column=2, row=0)
26
27 window.mainloop()
```

Run the above code and check the result:



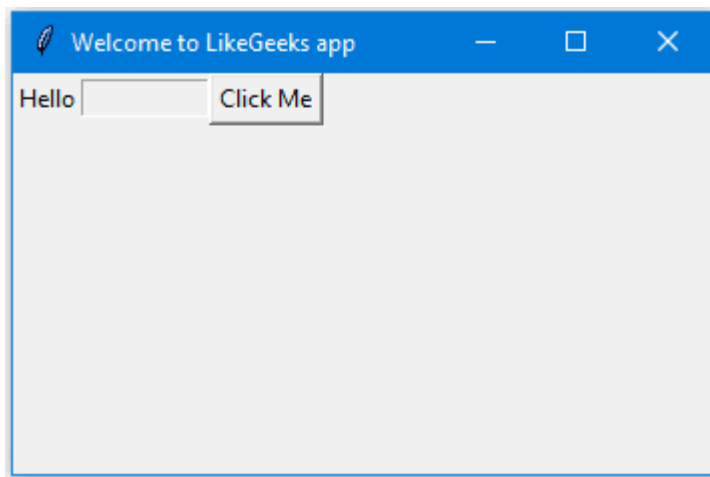
Every time we run the code, we need to click on the entry widget to set focus to write the text, what about setting the focus automatically?

.

Disable entry widget

To disable entry widget, you can set the state property to disabled:

```
txt = Entry(window,width=10, state='disabled')
```



Now, you won't be able to enter any text.

## COMBOBOX

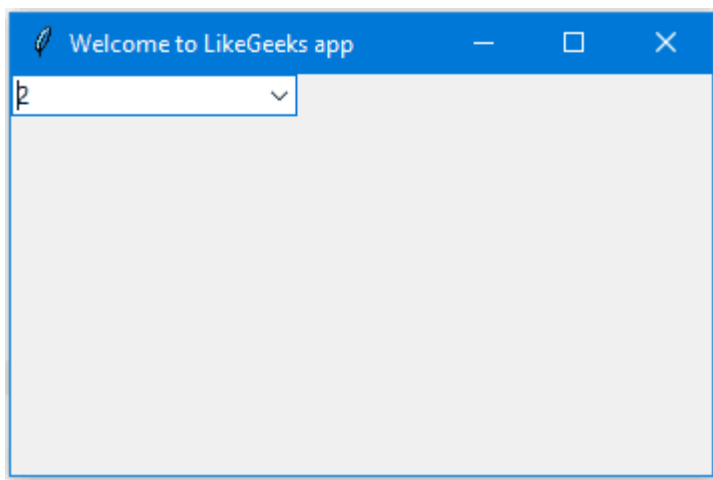
To add a combobox widget, you can use the Combobox class from ttk library like this:

```
1 from tkinter.ttk import *  
2  
3 combo = Combobox(window)
```

Then you can add your values to the combobox.

```
1 from tkinter import *  
2  
3 from tkinter.ttk import *  
4  
5 window = Tk()  
6
```

```
7 window.title("Welcome to LikeGeeks app")
8
9 window.geometry('350x200')
10
11 combo = Combobox(window)
12
13 combo['values'] = (1, 2, 3, 4, 5, "Text")
14
15 combo.current(1) #set the selected item
16
17 combo.grid(column=0, row=0)
18
19 window.mainloop()
```



As you can see, we add the combobox items using the tuple.

To set the selected item, you can pass the index of the desired item to the current function.

To get the select item, you can use the get function like this:

```
combo.get()
```

# CHECKBUTTON

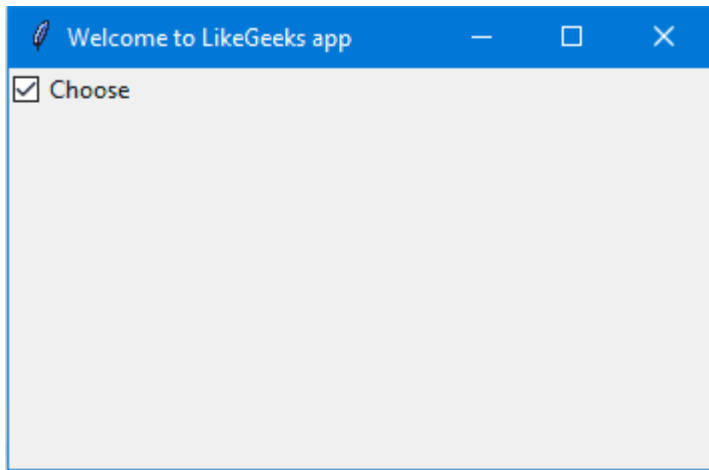
To create a checkbutton widget, you can use Checkbutton class like this:

```
chk = Checkbutton(window, text='Choose')
```

Also, you can set the checked state by passing the check value to the Checkbutton like this:

```
1 from tkinter import *
2
3 from tkinter.ttk import *
4
5 window = Tk()
6
7 window.title("Welcome to LikeGeeks app")
8
9 window.geometry('350x200')
10
11 chk_state = BooleanVar()
12
13 chk_state.set(True) #set check state
14
15 chk = Checkbutton(window, text='Choose', var=chk_state)
16
17 chk.grid(column=0, row=0)
18
19 window.mainloop()
```

Check the result:



Set check state of a Checkbutton

Here we create a variable of type BooleanVar which is not a standard Python variable, it's a Tkinter variable, and then we pass it to the Checkbutton class to set the check state as the highlighted line in the above example.

You can set the Boolean value to false to make it unchecked.

Also, you can use IntVar instead of BooleanVar and set the value to 0 or 1.

```
1 chk_state = IntVar()  
2  
3 chk_state.set(0) #unchecked  
4  
5 chk_state.set(1) #check
```

These examples give the same result as the BooleanVar.

## **RADIOBUTTONS**

To add radio buttons, simply you can use RadioButton class like this:

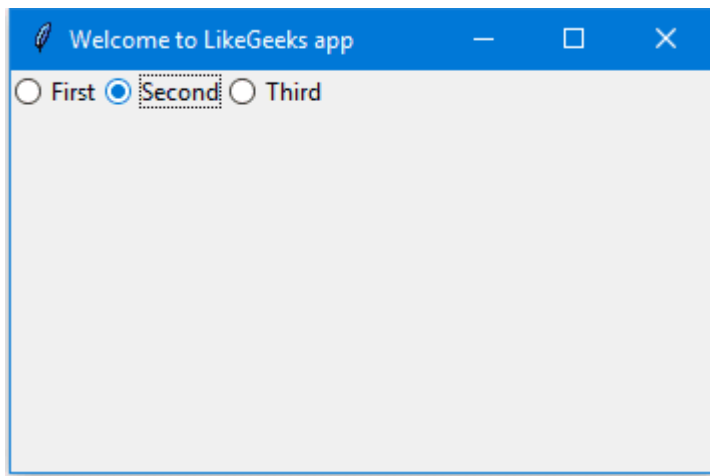
```
rad1 = Radiobutton(window,text='First', value=1)
```

Note that you should set the value for every radio button with a different value, otherwise, they won't work.

```
1 from tkinter import *
2
3 from tkinter.ttk import *
4
5 window = Tk()
6
7 window.title("Welcome to LikeGeeks app")
8
9 window.geometry('350x200')
10
11 rad1 = Radiobutton(window,text='First', value=1)
12
13 rad2 = Radiobutton(window,text='Second', value=2)
14
15 rad3 = Radiobutton(window,text='Third', value=3)
16
17 rad1.grid(column=0, row=0)
18
19 rad2.grid(column=1, row=0)
20
21 rad3.grid(column=2, row=0)
22
```

## 23 window.mainloop()

The result of the above code looks like this:



Also, you can set the command of any of these radio buttons to a specific function, so if the user clicks on any one of them, it runs the function code.

This is an example:

```
1 rad1 = Radiobutton(window, text='First', value=1,  
2 command=clicked)  
3  
4 def clicked():  
5     # Do what you need
```

Get radio button value (selected radio button)

To get the currently selected radio button or the radio button value, you can pass the variable parameter to the radio buttons and later you can get its value.

```
1 from tkinter import *
2
3 from tkinter.ttk import *
4
5 window = Tk()
6
7 window.title("Welcome to LikeGeeks app")
8
9 selected = IntVar()
10
11 rad1 = Radiobutton(window,text='First', value=1,
12 variable=selected)
13
14 rad2 = Radiobutton(window,text='Second', value=2,
15 variable=selected)
16
17 rad3 = Radiobutton(window,text='Third', value=3,
18 variable=selected)
19
20 def clicked():
21     print(selected.get())
22
23 btn = Button(window, text="Click Me", command=clicked)
24
25 rad1.grid(column=0, row=0)
26
```



```
27 rad2.grid(column=1, row=0)
```

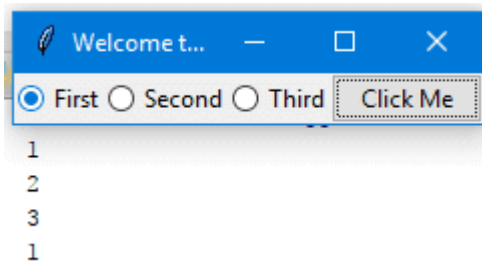
```
28
```

```
29 rad3.grid(column=2, row=0)
```

```
30
```

```
31 btn.grid(column=3, row=0)
```

```
window.mainloop()
```



Every time you select a radio button, the value of the variable will be changed to the value of the selected radio button.

## MessageBox

To show a message box using Tkinter, you can use messagebox library like this:

```
1 from tkinter import messagebox
```

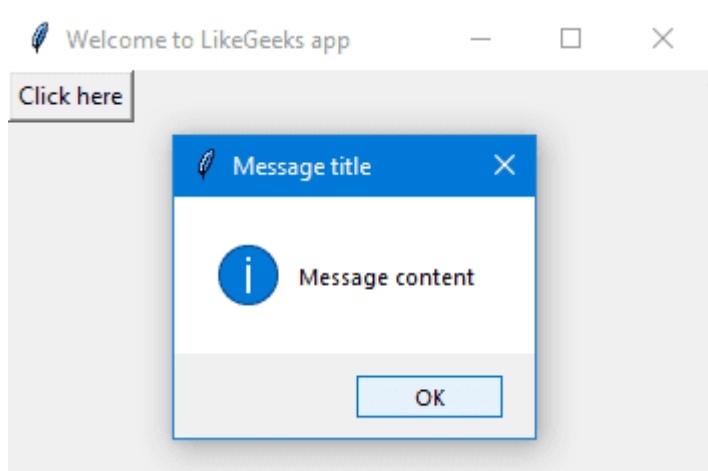
```
2
```

```
3 messagebox.showinfo('Message title','Message content')
```

Let's show a message box when the user clicks a button.

```
1 from tkinter import *
```

```
2 from tkinter import messagebox
3
4 window = Tk()
5
6 window.title("Welcome to LikeGeeks app")
7
8 window.geometry('350x200')
9
10 def clicked():
11
12     messagebox.showinfo('Message title', 'Message content')
13
14 btn = Button(window, text='Click here', command=clicked)
15
16 btn.grid(column=0, row=0)
17
18 window.mainloop()
19
```



When you click the button, an info messagebox will appear.

Show warning and error messages

You can show a warning message or error message the same way. The only thing that needs to be changed is the message function

```
1 messagebox.showwarning('Message title', 'Message  
2 content') #shows warning message  
3  
4 messagebox.showerror('Message title', 'Message  
5 content') #shows error message
```

## Show askquestion dialogs

To show a yes no message box to the user, you can use one of the following messagebox functions:

```
from tkinter import messagebox  
  
1  
2 res = messagebox.askquestion('Message title','Message  
3 content')  
4  
5 res = messagebox.askyesno('Message title','Message content')  
6  
7 res = messagebox.askyesnocancel('Message title','Message  
8 content')  
9  
10 res = messagebox.askokcancel('Message title','Message  
11 content')  
12  
13 res = messagebox.askretrycancel('Message title','Message  
14 content')
```

You can choose the appropriate message style according to your needs. Just replace the `showinfo` function line from the previous line and run it.

Also, you can check what button was clicked using the result variable

If you click **OK** or **yes** or **retry**, it will return **True** value, but if you choose **no** or **cancel**, it will return **False**.

The only function that returns one of three values is **askyesnocancel** function, it returns **True or False or None**.

## Images And Icons

You can add **Images** and **Icons** using **PhotoImage** method.

Let's how it works:

```
1 import tkinter
2 window = tkinter.Tk()
3 window.title("GUI")
4 # taking image from the directory and storing the source in a variable
5 icon = tkinter.PhotoImage(file = "images/edureka.png")
6 # displaying the picture using a 'Label' by passing the 'picture' variriab
7 parameter
8 label = tkinter.Label(window, image = icon)
9 label.pack()
10 window.mainloop()
```

## Menu bar

You will create drop-down menus in **tkinter** using the class **Menu**. Follow the below steps to create drop-down menus.

## Steps:-

- Create a **root menu** to insert different types of **menu options** using **tkinter.Menu(para)** and it takes a parameter where to place the **Menu**
- You have to tell the **tkinter** to initiate **Menu** using **window\_variable.config(menu = para)** and it takes a parameter called **menu** which is the **root menu** you previously defined.
- Now, creating **sub menus** using same method **tkinter.Menu(para)** and it takes the parameter **root menu**.
- **root menu.add\_cascade(para1, menu = para2)** creates the name of the **sub menu**, and it takes 2 parameters one is **label** which is the name of the **sub menu**, and another one is **menu** which is **sub menu**.
- **sub menu.add\_command()** adds an option to the **sub menu**.
- **sub menu.add\_separator()** adds a separator

Let's see the example to understand it fully.

```
import tkinter
```

```
window = tkinter.Tk()
```

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

```
def function():
```

```
    pass
```

```
# creating a root menu to insert all the sub menus
```

```
root_menu = tkinter.Menu(window)
```

```
window.config(menu = root_menu)
```

```
# creating sub menus in the root menu
```

```
file_menu = tkinter.Menu(root_menu) # it initializes a new sub menu in the root menu
```

```
root_menu.add_cascade(label = "File", menu = file_menu) # it creates the name of the sub menu
```

```
file_menu.add_command(label = "New file.....", command = function) # it adds a option to the sub menu 'command' parameter is used to do some action
```

```
file_menu.add_command(label = "Open files", command = function)
```

```
file_menu.add_separator() # it adds a line after the 'Open files' option
```

```
file_menu.add_command(label = "Exit", command = window.quit)
```

```
# creting another sub menu
```

```
edit_menu = tkinter.Menu(root_menu)
```

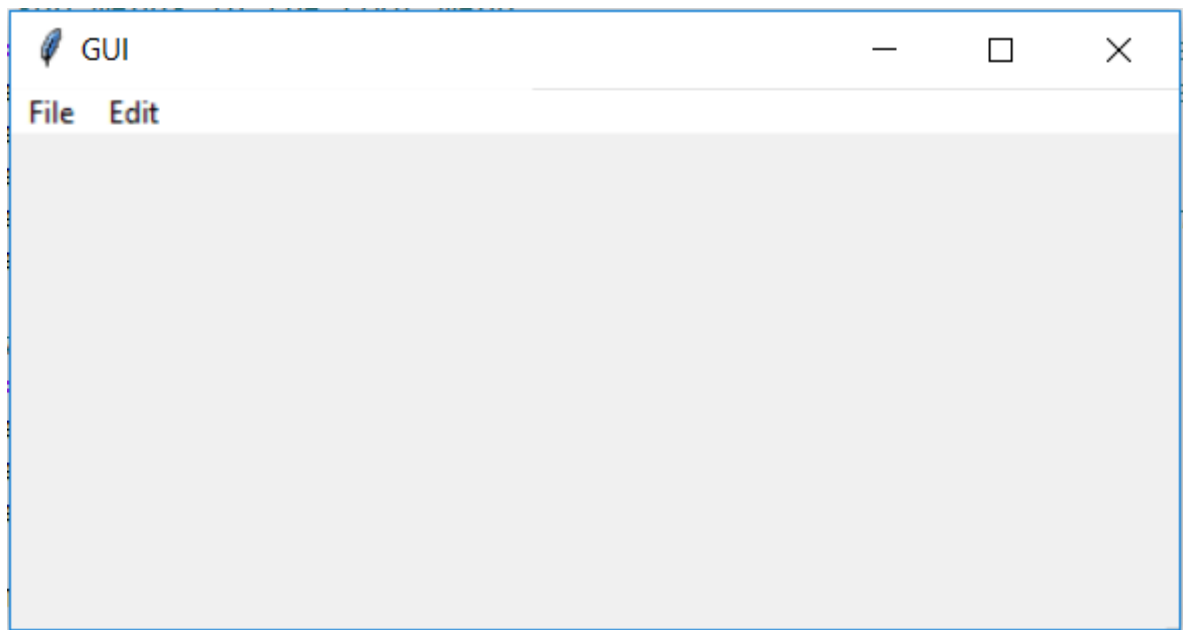
```
root_menu.add_cascade(label = "Edit", menu = edit_menu)
```

```
edit_menu.add_command(label = "Undo", command = function)
```

```
edit_menu.add_command(label = "Redo", command = function)
```

```
window.mainloop()
```

You will see the output something similar to the following. Click to the **File** and **Edit** menus to look at the options.



## **TEXT Widget**

We create a text widget by using the `Text()` method. We set the height to 2, i.e. two lines and the width to 30, i.e. 30 characters. We can apply the method `insert()` on the object `T`, which the `Text()` method had returned, to include text. We add two lines of text.

```
from Tkinter import *  
  
root = Tk()  
T = Text(root, height=2, width=30)  
T.pack()  
T.insert(END, "Just a text Widget\nin two lines\n")  
mainloop()
```



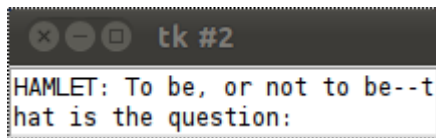
Let's change our little example a tiny little bit. We add another text, the beginning of the famous monologue from Hamlet:

```
from Tkinter import *

root = Tk()
T = Text(root, height=2, width=30)
T.pack()
quote = """HAMLET: To be, or not to be--that is the
question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune
Or to take arms against a sea of troubles
And by opposing end them. To die, to sleep--
No more--and by a sleep to say we end
The heartache, and the thousand natural shocks
That flesh is heir to. 'Tis a consummation
Devoutly to be wished."""
T.insert(END, quote)
mainloop()
```

If we start our little script, we get a very unsatisfying result. We can see in the window only the first line of the monologue and this line is even broken into two lines. We can see only two lines in our window, because we set the height to the value 2. Furthermore, the width is set to 30, so Tkinter has to break the first line of the monologue after 30 characters.





One solution to our problem consists in setting the height to the number of lines of our monologue and set width wide enough to display the widest line completely.

But there is a better technique, which you are well acquainted with from your browser and other applications: scrolling

## SCROLLBARS

So let's add a scrollbar to our window. To this purpose, Tkinter provides the `Scrollbar()` method. We call it with the root object as the only parameter.

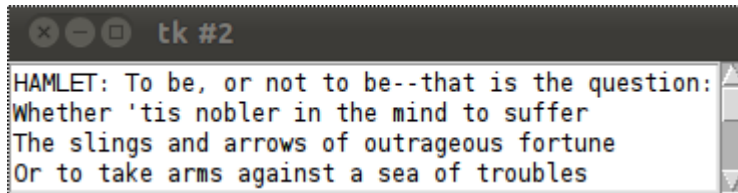
```
from Tkinter import *

root = Tk()
S = Scrollbar(root)
T = Text(root, height=4, width=50)
S.pack(side=RIGHT, fill=Y)
T.pack(side=LEFT, fill=Y)
S.config(command=T.yview)
T.config(yscrollcommand=S.set)

quote = """HAMLET: To be, or not to be--that is the
question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune
Or to take arms against a sea of troubles
And by opposing end them. To die, to sleep--
No more--and by a sleep to say we end
```

```
The heartache, and the thousand natural shocks
That flesh is heir to. 'Tis a consummation
Devoutly to be wished.""
T.insert(END, quote)
mainloop( )
```

The result is a lot better. We have now always 4 lines in view, but all lines can be viewed by using the scrollbar on the right side of the window:



## CANVAS WIDGET

The Canvas widget can support the following standard items –

**arc** – Creates an arc item, which can be a chord, a pieslice or a simple arc.

```
coord = 10, 50, 240, 210

arc = canvas.create_arc(coord, start=0, extent=150,
fill="blue")
```

**image** – Creates an image item, which can be an instance of either the BitmapImage or the PhotoImage classes.

```
filename = PhotoImage(file = "sunshine.gif")

image = canvas.create_image(50, 50, anchor=NE,
image=filename)
```

**line** – Creates a line item.

```
line = canvas.create_line(x0, y0, x1, y1, ..., xn, yn, options)
```

**oval** – Creates a circle or an ellipse at the given coordinates. It takes two pairs of coordinates; the top left and bottom right corners of the bounding rectangle for the oval.

```
oval = canvas.create_oval(x0, y0, x1, y1, options)
```

**polygon** – Creates a polygon item that must have at least three vertices.

```
oval = canvas.create_polygon(x0, y0, x1, y1,...xn, yn, options)
```

Example

Try the following example yourself –

```
import Tkinter

top = Tkinter.Tk()

C = Tkinter.Canvas(top, bg="blue", height=250, width=300)

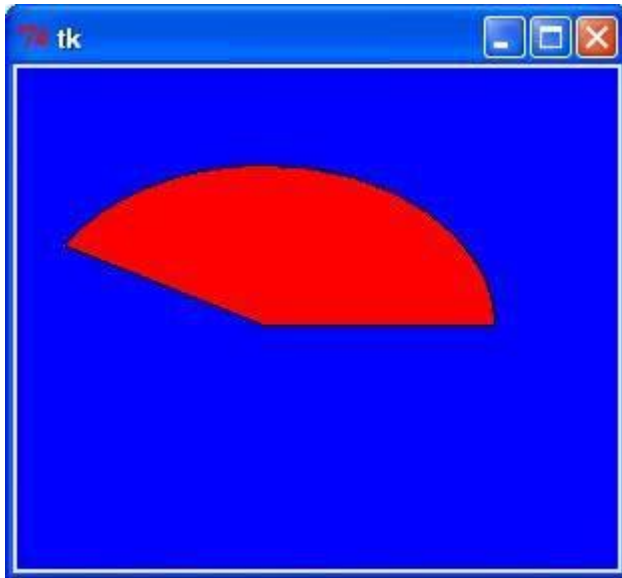
coord = 10, 50, 240, 210

arc = C.create_arc(coord, start=0, extent=150, fill="red")

C.pack()

top.mainloop()
```

When the above code is executed, it produces the following result –



## Creating Calculator

Every **GUI** apps include two steps.

- Creating User Interface
- Adding functionalities to the **GUI**

Let's start creating **Calculator**.

Let's start creating **Calculator**.

```
from tkinter import *
```

```
# creating basic window
```

```
window = Tk()
```

```
window.geometry("312x324") # size of the window width:-
```

```
500, height:- 375
```

```
window.resizable(0, 0) # this prevents from resizing the  
window
```

```
window.title("Calcualtor")
```

```
#####
```

functions

```
#####
```

```
# 'btn_click' function continuously updates the input field  
whenever you enters a number
```

```
def btn_click(item):
```

```
    global expression
```

```
    expression = expression + str(item)
```

```
    input_text.set(expression)
```

```
# 'btn_clear' function clears the input field
```

```
def btn_clear():
```

```
    global expression
```

```
    expression = ""
```

```
    input_text.set("")
```

```
# 'btn_equal' calculates the expression present in input field
```

```
def btn_equal():
```

```
    global expression
```

```
    result = str(eval(expression)) # 'eval' function evalutes the  
string expression directly
```

```
    # you can also implement your own function to evalute the  
expression instead of 'eval' function
```

```
    input_text.set(result)
```

```
    expression = ""
```

```
expression = ""
```

```
# 'StringVar()' is used to get the instance of input field
```

```
input_text = StringVar()
```

```
# creating a frame for the input field
```

```
input_frame = Frame(window, width = 312, height = 50, bd =  
0, highlightbackground = "black", highlightcolor = "black",  
highlightthickness = 1)
```

```
input_frame.pack(side = TOP)
```

```
# creating a input field inside the 'Frame'
```

```
input_field = Entry(input_frame, font = ('arial', 18, 'bold'),  
textvariable = input_text, width = 50, bg = "#eee", bd = 0,  
justify = RIGHT)
```

```
input_field.grid(row = 0, column = 0)
```

```
input_field.pack(ipady = 10) # 'ipady' is internal padding to  
increase the height of input field
```

```
# creating another 'Frame' for the button below the
```

```
'input_frame'
```

```
btns_frame = Frame(window, width = 312, height = 272.5, bg  
= "grey")
```

```
btns_frame.pack()
```

```
# first row
```

```
clear = Button(btns_frame, text = "C", fg = "black", width =  
32, height = 3, bd = 0, bg = "#eee", cursor = "hand2",
```

```
command = lambda: btn_clear()).grid(row = 0, column = 0,  
columnspan = 3, padx = 1, pady = 1)  
divide = Button(btns_frame, text = "/", fg = "black", width =  
10, height = 3, bd = 0, bg = "#eee", cursor = "hand2",  
command = lambda: btn_click("/")).grid(row = 0, column = 3,  
padx = 1, pady = 1)
```

```
# second row
```

```
seven = Button(btns_frame, text = "7", fg = "black", width =  
10, height = 3, bd = 0, bg = "#fff", cursor = "hand2",  
command = lambda: btn_click(7)).grid(row = 1, column = 0,  
padx = 1, pady = 1)  
eight = Button(btns_frame, text = "8", fg = "black", width =  
10, height = 3, bd = 0, bg = "#fff", cursor = "hand2",  
command = lambda: btn_click(8)).grid(row = 1, column = 1,  
padx = 1, pady = 1)  
nine = Button(btns_frame, text = "9", fg = "black", width =  
10, height = 3, bd = 0, bg = "#fff", cursor = "hand2",  
command = lambda: btn_click(9)).grid(row = 1, column = 2,  
padx = 1, pady = 1)  
multiply = Button(btns_frame, text = "*", fg = "black", width =  
10, height = 3, bd = 0, bg = "#eee", cursor = "hand2",  
command = lambda: btn_click("*")).grid(row = 1, column = 3,  
padx = 1, pady = 1)
```

```
# third row
```

```
four = Button(btns_frame, text = "4", fg = "black", width = 10,
height = 3, bd = 0, bg = "#fff", cursor = "hand2", command =
lambda: btn_click(4)).grid(row = 2, column = 0, padx = 1,
pady = 1)
five = Button(btns_frame, text = "5", fg = "black", width = 10,
height = 3, bd = 0, bg = "#fff", cursor = "hand2", command =
lambda: btn_click(5)).grid(row = 2, column = 1, padx = 1,
pady = 1)
six = Button(btns_frame, text = "6", fg = "black", width = 10,
height = 3, bd = 0, bg = "#fff", cursor = "hand2", command =
lambda: btn_click(6)).grid(row = 2, column = 2, padx = 1,
pady = 1)
minus = Button(btns_frame, text = "-", fg = "black", width =
10, height = 3, bd = 0, bg = "#eee", cursor = "hand2",
command = lambda: btn_click("-")).grid(row = 2, column = 3,
padx = 1, pady = 1)
```

# fourth row

```
one = Button(btns_frame, text = "1", fg = "black", width = 10,
height = 3, bd = 0, bg = "#fff", cursor = "hand2", command =
lambda: btn_click(1)).grid(row = 3, column = 0, padx = 1,
pady = 1)
two = Button(btns_frame, text = "2", fg = "black", width = 10,
height = 3, bd = 0, bg = "#fff", cursor = "hand2", command =
lambda: btn_click(2)).grid(row = 3, column = 1, padx = 1,
pady = 1)
three = Button(btns_frame, text = "3", fg = "black", width =
10, height = 3, bd = 0, bg = "#fff", cursor = "hand2",
```



```
command = lambda: btn_click(3)).grid(row = 3, column = 2,  
padx = 1, pady = 1)
```

```
plus = Button(btns_frame, text = "+", fg = "black", width =  
10, height = 3, bd = 0, bg = "#eee", cursor = "hand2",
```

```
command = lambda: btn_click("+")).grid(row = 3, column = 3,  
padx = 1, pady = 1)
```

```
# fourth row
```

```
zero = Button(btns_frame, text = "0", fg = "black", width =  
21, height = 3, bd = 0, bg = "#fff", cursor = "hand2",
```

```
command = lambda: btn_click(0)).grid(row = 4, column = 0,  
columnspan = 2, padx = 1, pady = 1)
```


```
point = Button(btns_frame, text = ".", fg = "black", width =  
10, height = 3, bd = 0, bg = "#eee", cursor = "hand2",
```

```
command = lambda: btn_click(".")).grid(row = 4, column = 2,  
padx = 1, pady = 1)
```

```
equals = Button(btns_frame, text = "=", fg = "black", width =  
10, height = 3, bd = 0, bg = "#eee", cursor = "hand2",
```

```
command = lambda: btn_equal()).grid(row = 4, column = 3,  
padx = 1, pady = 1)
```

```
window.mainloop()
```

<div>  <span>Calculador</span> <span>—</span> <span>□</span> <span>×</span> </div>			
C			/
7	8	9	*
4	5	6	-
1	2	3	+
0		.	=