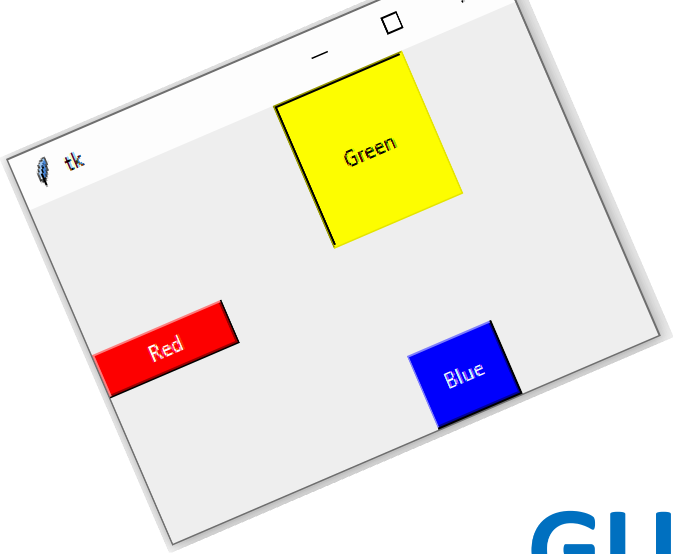
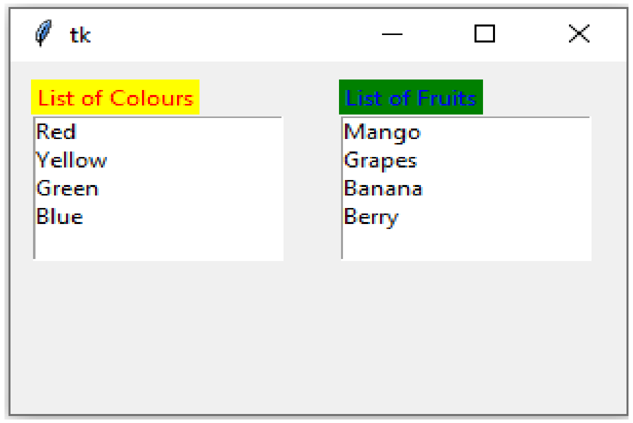
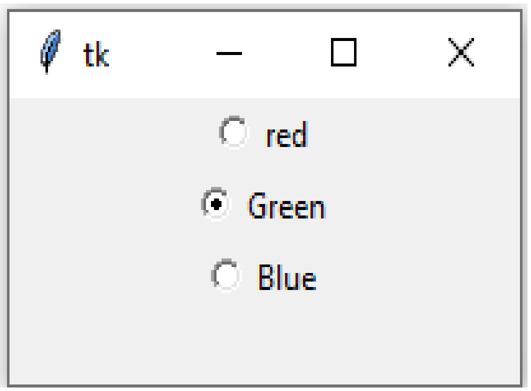


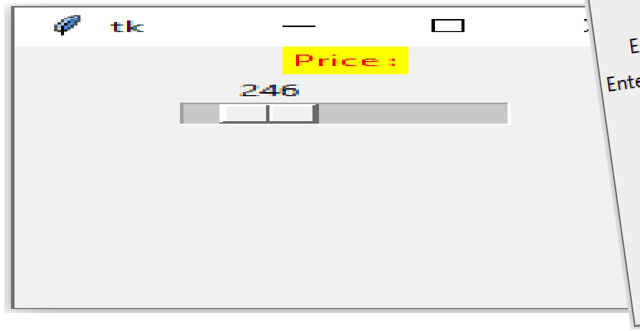


List of Colours	List of Fruits
Red	Mango
Yellow	Grapes
Green	Banana
Blue	Berry

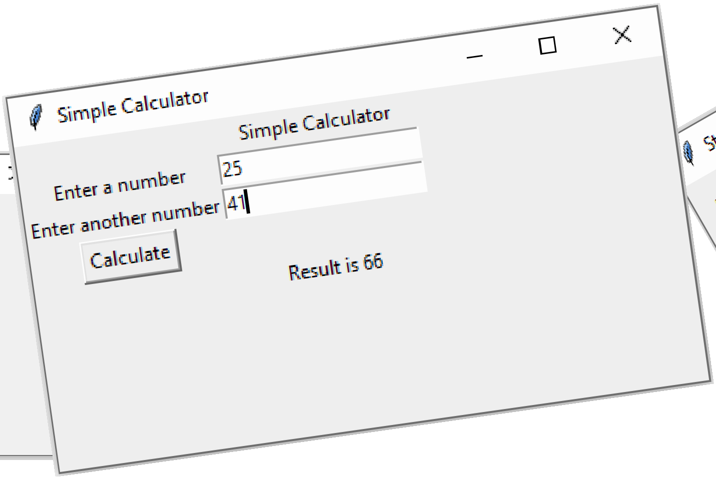
GUI Programming
in
Python



red [✓ Green] Blue



Price :
246

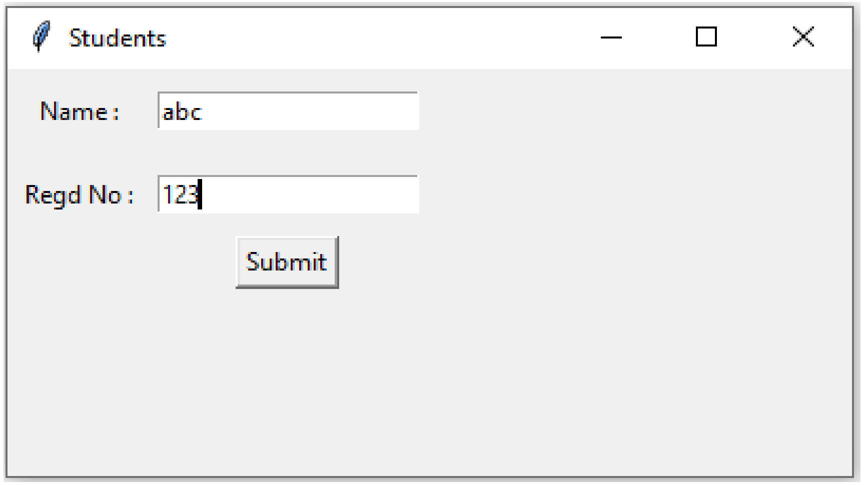


Simple Calculator
Enter a number 25
Enter another number 41
Result is 66

Name : abc
Regd No : 123

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 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 windowing objects can be text labels, buttons, list boxes, etc.These individual little GUI components are known as **widgets**.



Name : abc

Regd No : 123

- Python offers multiple options for developing GUI (Graphical User Interface). The most commonly used **GUI method** is **tkinter.**

- **Tkinter** 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

- since Tkinter is the Python interface to Tk (Tea Kay), it can be pronounced as **Tea-Kay-inter**. i.e tkinter = **t k inter**.

tkinter - GUI for Python:

- Python provides the standard library **tkinter** for creating the graphical user interface for **desktop based applications**.

- Developing desktop based applications with **tkinter** is not a complex task.

- A Tkinter window application 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.

1. 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**’.

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

2. 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()**

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

4. 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 ()

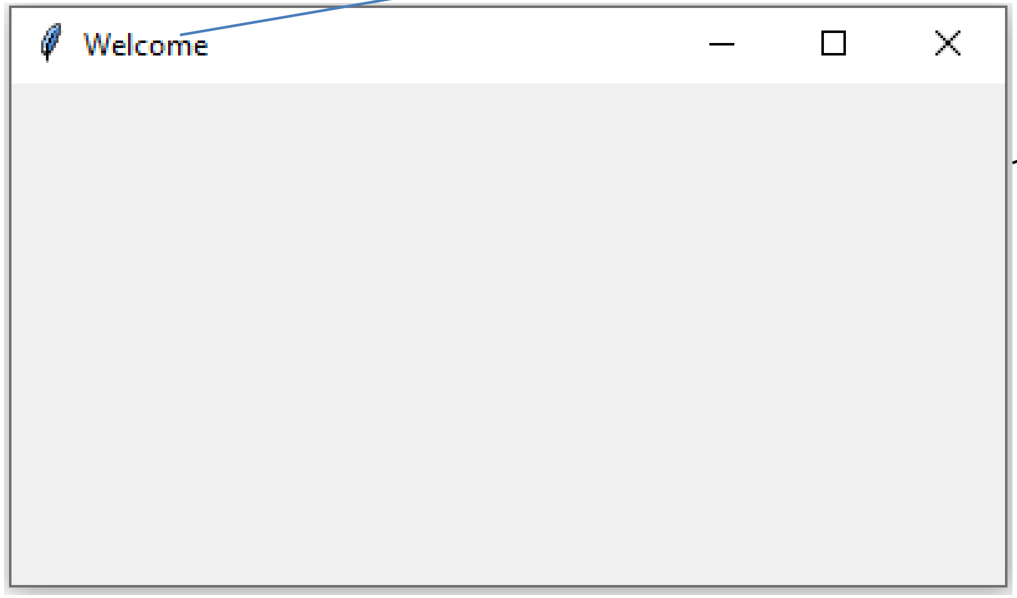
Example: **tkndemo.py**

```
import tkinter
top = tkinter.Tk()          #creating the application main window.
top.title("Welcome")        #title of main window
top.geometry("400x300")     #size of main window
top.mainloop()              #calling the event main loop
```

Output:

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

Title of window



Main Window (400x300)

- 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:

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

Syntax:

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

The possible options are

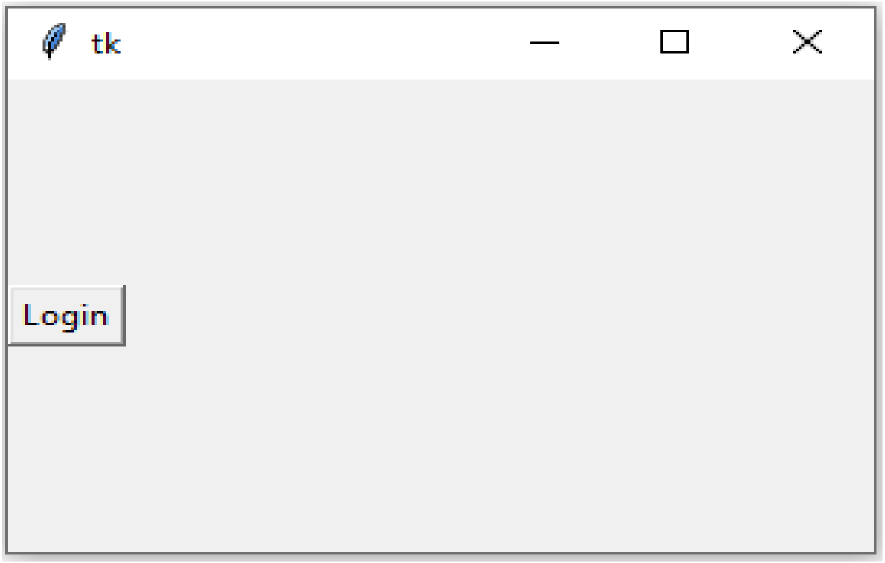
side: it represents the side to which the widget is to be placed on the window. Side may be **LEFT** or **RIGHT** or **TOP(default)** or **BOTTOM**.

Example: **tknpack.py**

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

This is a more organized way to place the widgets to the python application.

Syntax:

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

The possible options are

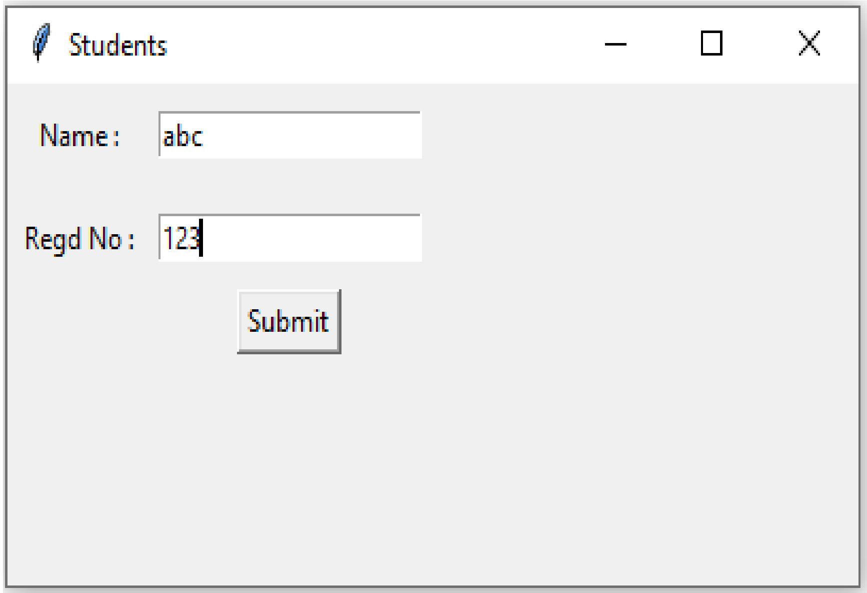
- **Column**
  The column number in which the widget is to be placed. The leftmost column is represented by **0**.
- **padx, pady**
  It represents the number of pixels to pad the widget outside the widget's border.
- **row**
  The row number in which the widget is to be placed. The topmost row is represented by **0**.

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)
```

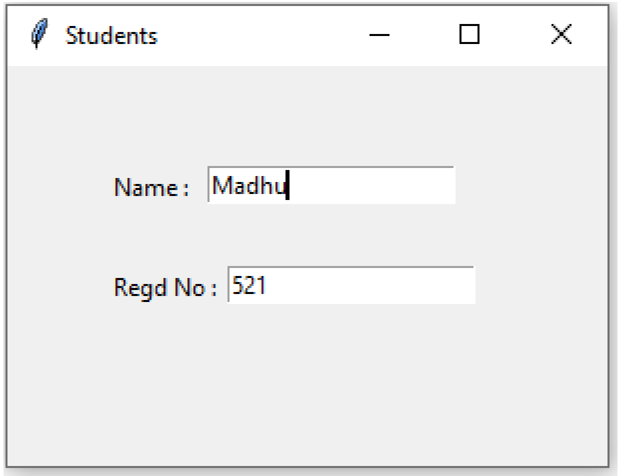
- **x, y:** It refers to the horizontal and vertical offset in the pixels.

Example: **tknplace.py**

```
from tkinter import *
parent = Tk()
parent.title("Students")
parent.geometry("300x200")
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)
parent.mainloop()
```

Output:

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



Students

Name : Madhu

Regd No : 521

- **Tkinter widgets or components:**

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

Widget	Description
Button	Creates various buttons in Python Application.
Checkbutton	Select one or more options from multiple options.(Checkbox)
Entry	Allows the user to enter single line of text(Textbox)
Frame	Acts like a container which can be used to hold the other widgets
Label	Used to display non editable text on window
Listbox	Display the list items, The user can choose one or more items.
Radiobutton	Select one option from multiple options.
Text	Allows the user to enter single or multiple line of text(Textarea)
Scale	Creates the graphical slider, the user can slide through the range of values
Toplevel	Used to create and display the top-level windows(Open a new window)

❖ 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)**

The options are

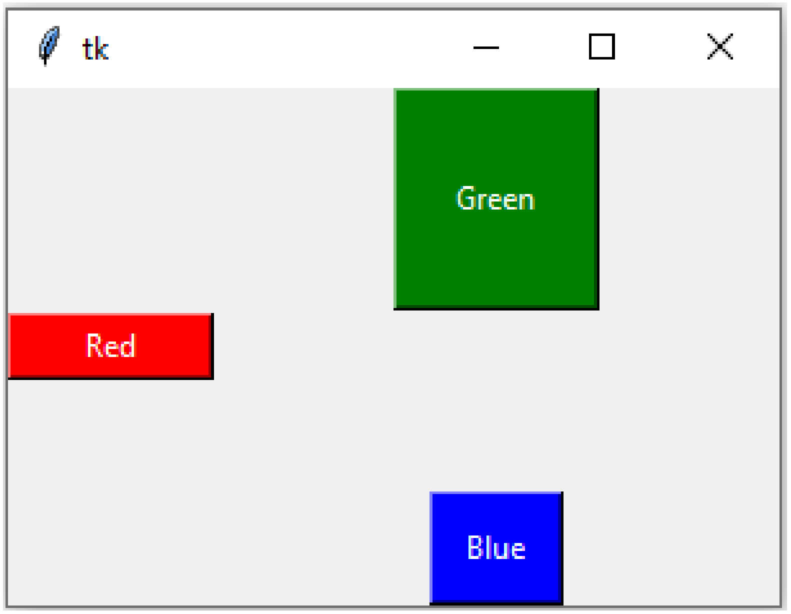
- **activebackground:**It represents the background of the button when it is active.
- **activeforeground:**It represents the font color of the button when it is active..
- **bd:** It represents the border width in pixels.
- **bg:** It represents the background color of the button.
- **command:**It is 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.
- **padx:**Additional padding to the button in the horizontal direction.
- **pady:**Additional padding to the button in the vertical direction.
- **width:**The width of the button.

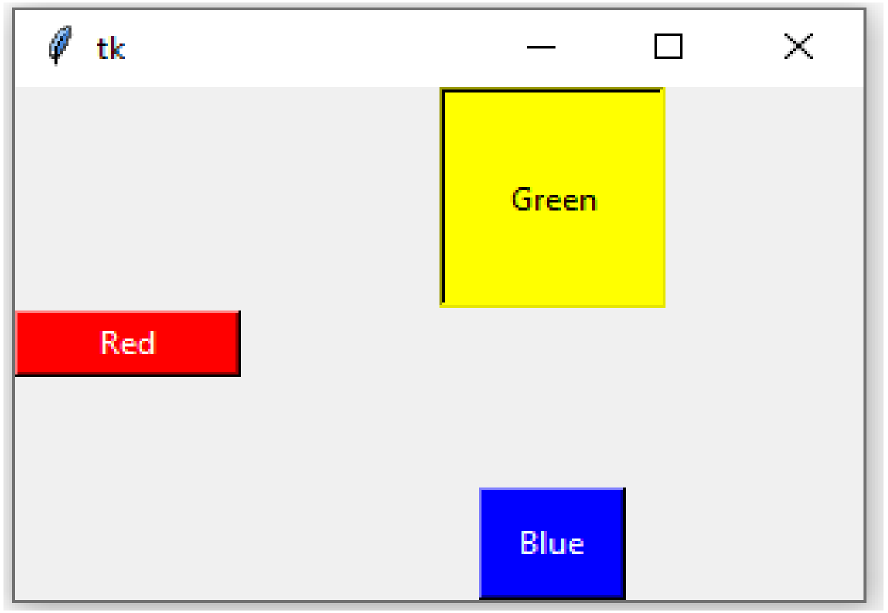
Example: **btndemo1.py**

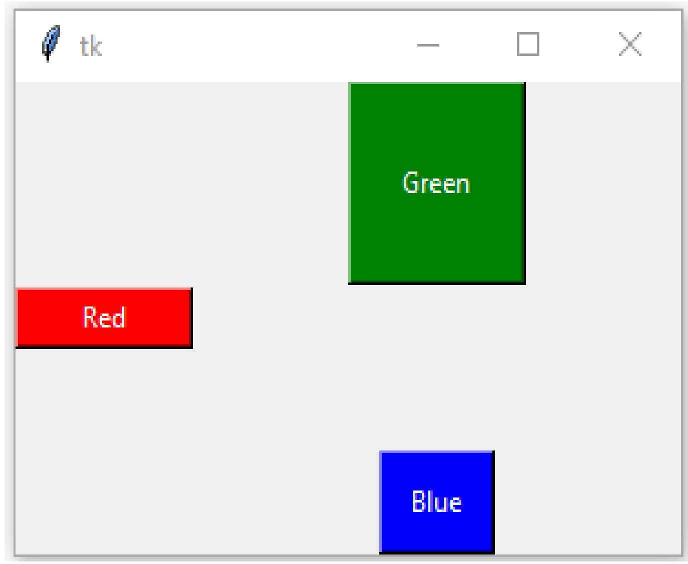
```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("300x200")

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()
```

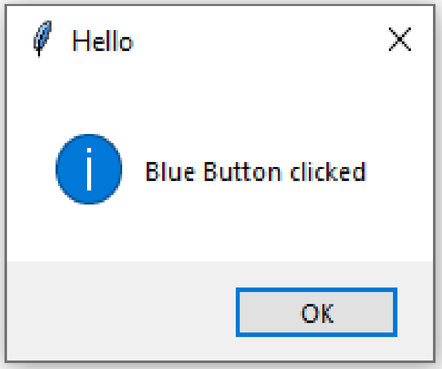
Output:

```
>>>python btndemo1.py
```









Blue Button clicked

❖ 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)**

The options are

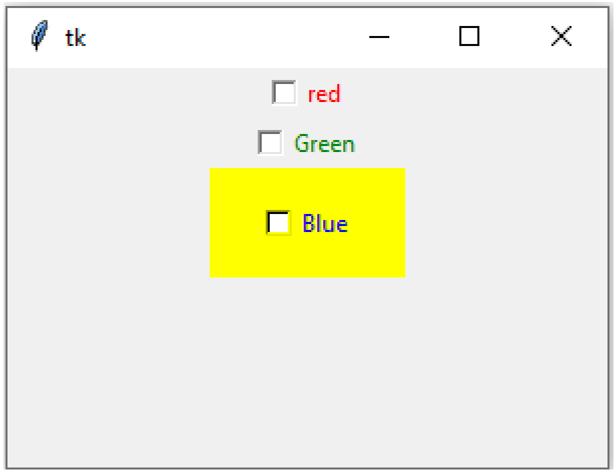
- **activebackground:**It represents the background of the Checkbutton when it is active.
- **activeforeground:**It represents the font color of the Checkbutton when when it is active.
- **bd:** It represents the border width in pixels.
- **bg:** It represents the background color of the Checkbutton.
- **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.
- **padx:** Additional padding to the Checkbutton in the horizontal direction.
- **pady:** Additional padding to the Checkbutton in the vertical direction.
- **width:** The width of the Checkbutton.

Example: **chbtndemo.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 chbtndemo.py



❖ 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)**

The options are

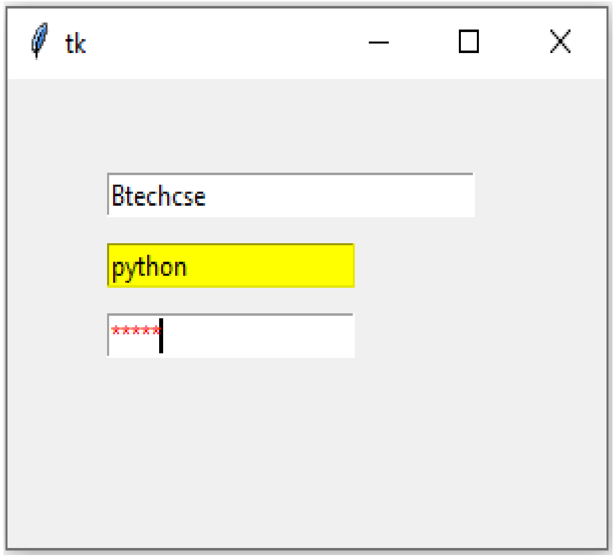
- **bd:** It represents the border width in pixels.
- **bg:** It represents the background color of the Entry.
- **show:** It is used to show the entry text of some other type instead of the string. For example, the password is typed using stars (*).
- **fg:** Foreground color of the Entry.
- **width:**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



```
Btechcse
python
*****
```

❖ 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)**

The options are

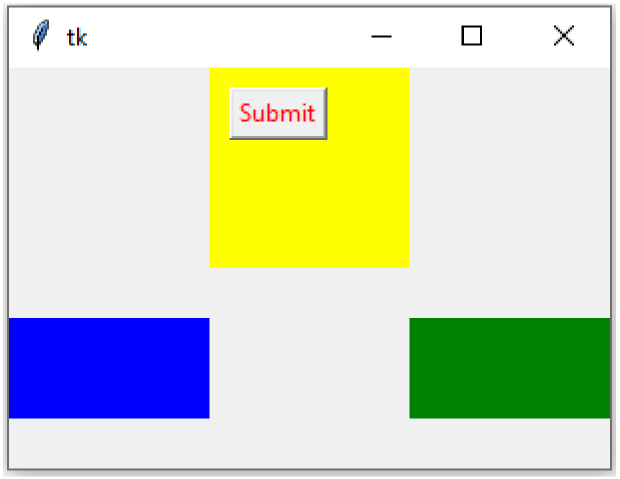
- **bd:** It represents the border width in pixels.
- **bg:** It represents the background color of the frame.
- **width:** The width of the frame.
- **height:** The height of the frame.

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



❖ Label Widget in Tkinter:

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

<u>**Syntax:**</u> **name = Label(parent, options)**

The options are

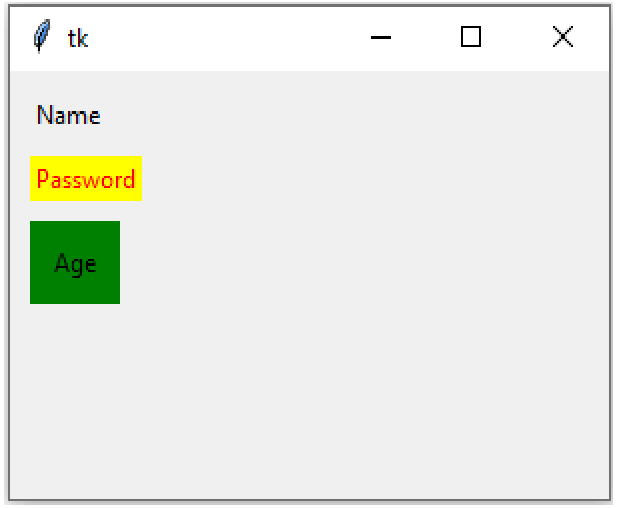
• **bd:**	It represents the border width in pixels.
• **bg:**	It represents the background color of the label.
• **text:**	It is set to the text displayed on the label.
• **fg:**	Foreground color of the label.
• **height:**	The height of the label.
• **image:**	It is set to the image displayed on the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

Example: **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()
```

Output:

>>>python labeldemo.py



❖ 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.

  Syntax: **name = Listbox(parent, options)**

The options are

- **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:

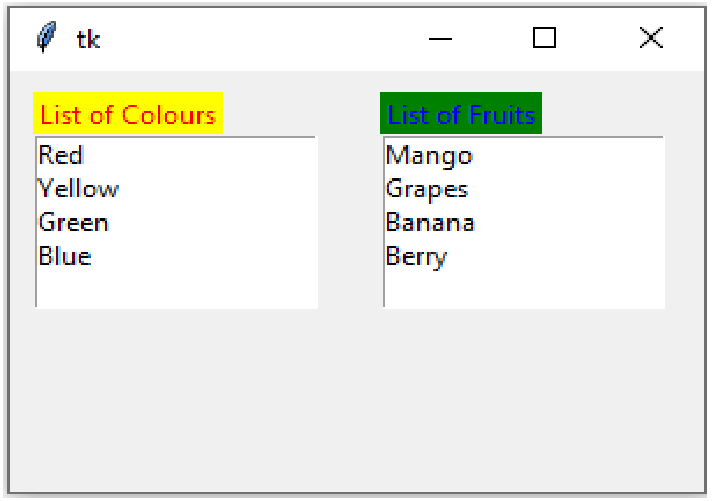
```
Listbox.insert (index, item)
```

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        List of Fruits
Red                    Mango
Yellow                 Grapes
Green                  Banana
Blue                   Berry
```

❖ 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)**

The options are

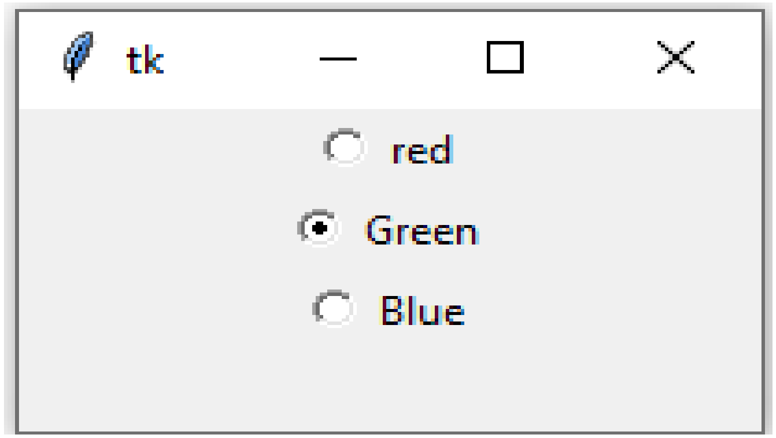
- **activebackground:**It represents the background of the Radiobutton when it is active.
- **activeforeground:**It represents the font color of the Radiobutton when when it is active.
- **bd:** It represents the border width in pixels.
- **bg:** It represents the background color of the Radiobutton.
- **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.

<u>Example:</u> 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()
```

<u>Output:</u>

>>>python rbtndemo.py



❖ 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)**

The options are

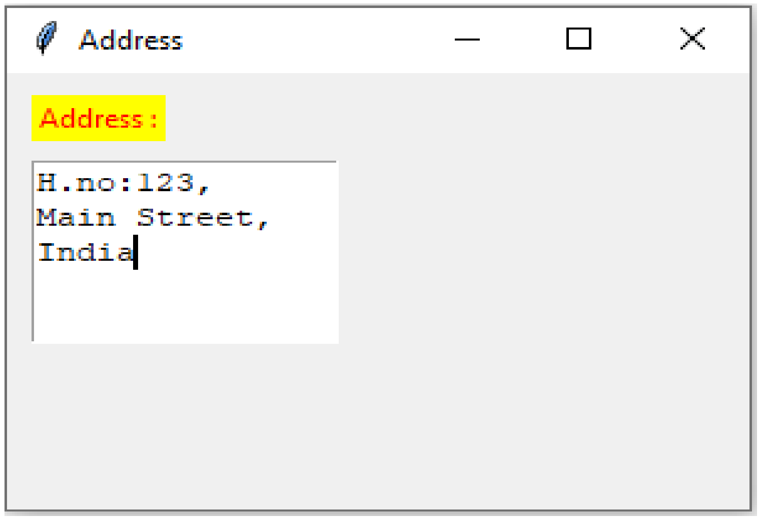
- **bd:** It represents the border width in pixels.
- **bg:** It represents the background color of the Text.
- **show:** It is used to show the entry text of some other type instead of the string. For example, the password is typed using stars (*).
- **fg:** Foreground color of the Text.
- **width:**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

❖ Scale 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 = Scale(parent, options)**

The options are

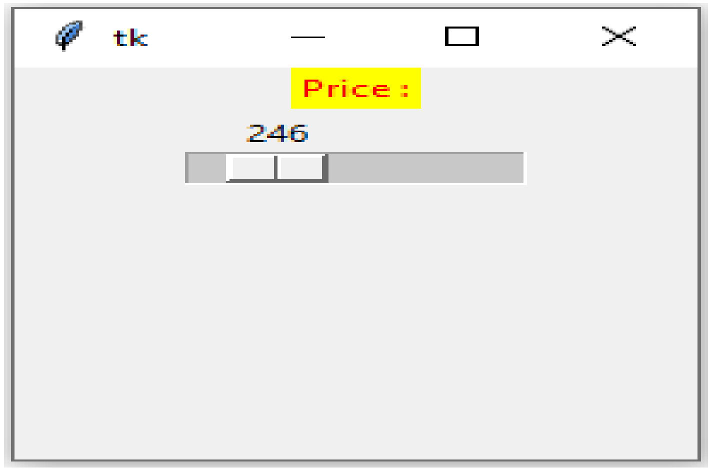
- **activebackground:**It represents the background of the Scale when it is active.
- **bd:** It represents the border width in pixels.
- **bg:** It represents the background color of the Scale.
- **command:** It is set to the function call which is scheduled when the function is called.
- **fg:** Foreground 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



❖ Toplevel Widget in Tkinter:

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

<u>**Syntax:**</u> **name = Toplevel(options)**

The options are

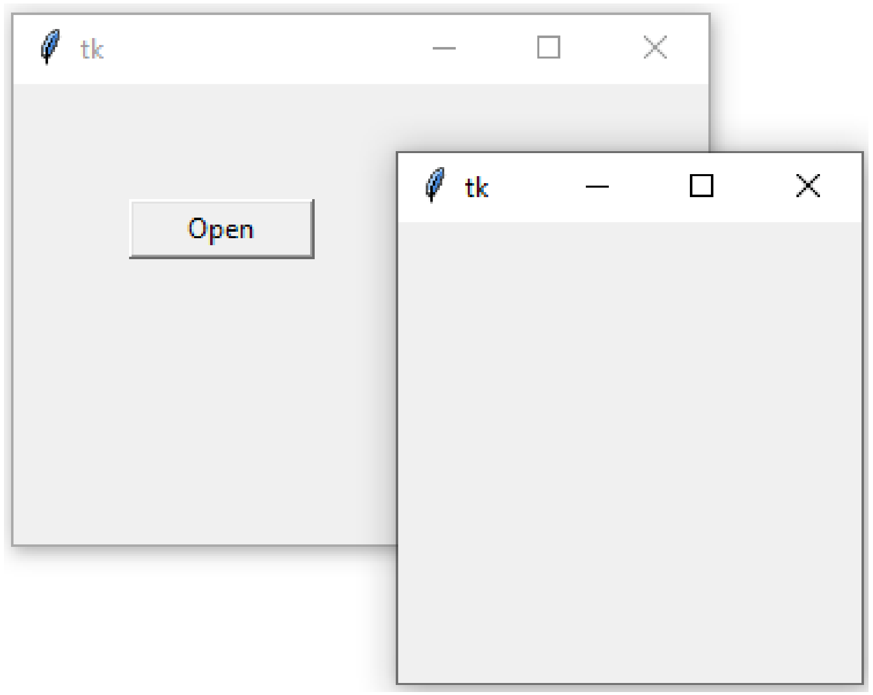
- **bd:** It represents the border width in pixels.
- **bg:** It represents the background color of the Toplevel.
- **fg:** Foreground color of the Toplevel.
- **width:** The width of the Toplevel.
- **height:** The vertical dimension of the widget in lines.

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:

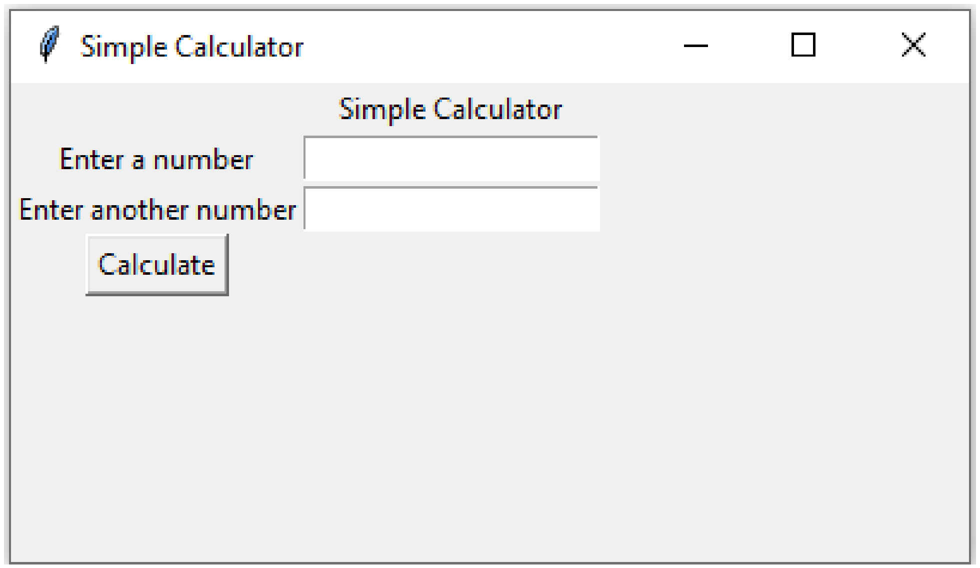
```
>>>python topleveldemo.py
```

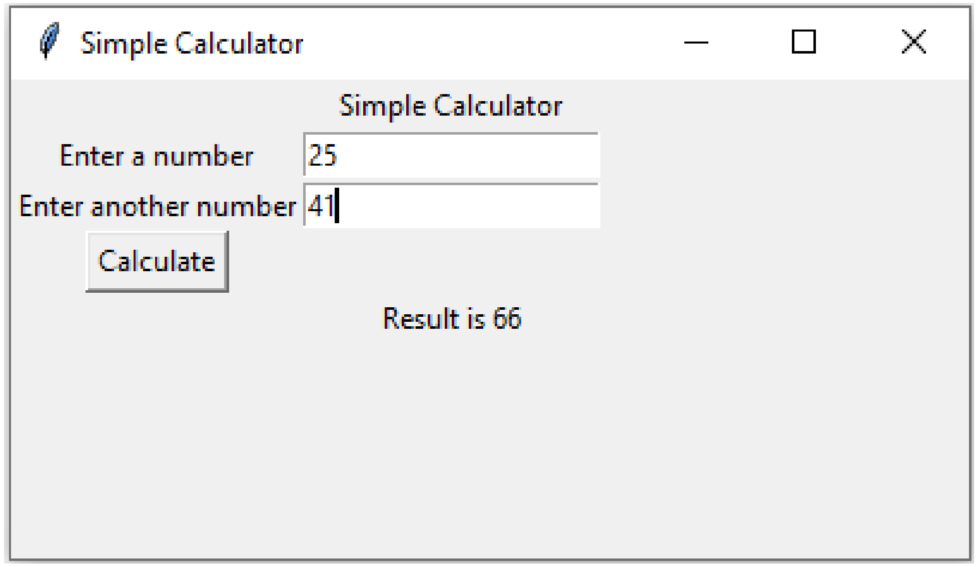


<u>Example:</u> simplecalc.py

```
import tkinter as tk
from functools import partial
def call_result(label_result, n1, n2):
    num1 = (n1.get())
    num2 = (n2.get())
    result = int(num1)+int(num2)
    label_result.config(text="Result is %d" % result)
    return
root = tk.Tk()
root.geometry('400x200+100+200')
root.title('Simple Calculator')
number1 = tk.StringVar()
number2 = tk.StringVar()
```

```
labelTitle = tk.Label(root, text="Simple Calculator").grid(row=0, column=2)
labelNum1 = tk.Label(root, text="Enter a number").grid(row=1, column=0)
labelNum2 = tk.Label(root, text="Enter another number").grid(row=2,
    column=0)
labelResult = tk.Label(root)
labelResult.grid(row=7, column=2)
entryNum1 = tk.Entry(root, textvariable=number1).grid(row=1, column=2)
entryNum2 = tk.Entry(root, textvariable=number2).grid(row=2, column=2)
call_result = partial(call_result, labelResult, number1, number2)
buttonCal = tk.Button(root, text="Calculate",
    command=call_result).grid(row=3, column=0)
root.mainloop()
```


Simple Calculator
Simple Calculator
Enter a number
Enter another number
Calculate


Simple Calculator
Simple Calculator
Enter a number
25
Enter another number
41
Calculate

Result is 66

- **Brief Tour of Other GUIs:**

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

1. **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.
- Tix includes the standard, widgets those are tixGrid,tixHList,tixInputOnly, tixTlist and etc.

2. **Pmw (Python MegaWidgets Tkinter extension)**:

- Pmw is a toolkit for building high-level compound widgets in Python using the Tkinter module.
- It consists of a set of base classes and a library of flexible and extensible megawidgets built on this foundation. These megawidgets include notebooks, comboboxes, selection widgets, paned widgets, scrolled widgets and dialog windows.

3. **wxPython (Python binding to wxWidgets)**:

- wxPython is a blending of the wxWidgets GUI classes and the Python programming language.
- wxPython is a Python package that can be imported at runtime that includes a collection of Python modules and an extension module (native code). It provides a series of Python classes that mirror (or shadow) many of the wxWidgets GUI classes.

Database Programming in python

Introduction:

- To build the real world applications, connecting with the databases is the necessity for the programming languages. However, python allows us to connect our application to the databases like MySQL, SQLite, MongoDB, and many others.
- Python also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements. For database programming, the Python **DB-API** is a widely used module that provides a database application programming interface.

The Python Programming language has powerful features for database programming, those are

- Python is famous for its portability.
- It is platform independent.
- In many programming languages, the application developer needs to take care of the open and closed connections of the database, to avoid further exceptions and errors. In Python, these connections are taken care of.
- Python supports relational database systems.
- Python database APIs are compatible with various databases, so it is very easy to migrate and port database application interfaces.

Environment Setup:

- In this topic we will discuss Python-MySQL database connectivity, and we will perform the database operations in python.
- The Python DB API implementation for MySQL is possible by **MySQLdb** or **mysql.connector.**

<u>Note:</u> The Python DB-API implementation for MySQL is possible by **MySQLdb** in **python2.x** but it deprecated in python3.x.In **Python3.x**,DB -API implementation for MySQL is possible by **mysql.connector.**

- ***You should have MySQL installed on your computer***

<u>Windows:</u>

You can download a free MySQL database at

https://www.mysql.com/downloads/.

<u>Linux(Ubuntu):</u>

sudo apt-get install mysql-server

- ***You need to install*** **MySQLdb: (in case of Python2.x)**
- MySQLdb is an interface for connecting to a MySQL database server from Python. The **MySQLdb** is not a built-in module, We need to install it to get it working.
- Execute the following command to install it.

➢ For (Linux)Ubuntu, use the following command -

```
sudo apt-get install python2.7-mysqldb
```

➢ For Windows command prompt, use the following command -

```
pip install MySQL-python
```

- To test if the installation was successful, or if you already have "**MySQLdb**" installed, execute following python statement at terminal or CMD.

```
import MySQLdb
```

- If the above statement was executed with no errors, "MySQLdb " is installed and ready to be used.

(OR)

- ***You need to install mysql.connector*: (in case of Python3.x)**
- To connect the python application with the MySQL database, we must import the **mysql.connector** module in the program.
- The **mysql.connector** is not a built-in module, We need to install it to get it working.
- Execute the following command to install it using pip installer.

➢ For (Linux)Ubuntu, use the following command -

```
pip install mysql-connector-python
```

➢ For Windows command prompt, use the following command -

```
pip install mysql-connector
```

- To test if the installation was successful, or if you already have " ***mysql.connector*** " installed, execute following python statement at terminal or CMD.

```
import  mysql.connector
```

- If the above statement was executed with no errors, "***mysql.connector*** " is installed and ready to be used.

Python Database Application Programmer's Interface (DB-API):

- Python DB-API is independent of any database engine, which enables you to write Python scripts to access any database engine.
- The Python DB API implementation for MySQL is possible by **MySQLdb** or **mysql.connector**.
- Using Python structure, **DB-API** provides standard and support for working with databases.

The API consists of:

1. Import module(**mysql.connector or MySQLdb)**
2. Create the connection object.
3. Create the cursor object
4. Execute the query
5. Close the connection

1. Import module(mysql.connector or MySQLdb):

- To interact with MySQL database using Python, you need first to import **mysql.connector or MySQLdb** module by using following statement.

- **MySQLdb(in python2.x)**

  import MySQLdb

- **mysql.connector(in python3.x)**

  import mysql.connector

2. Create the connection object:

- After importing **mysql.connector or MySQLdb** module, we need to create connection object, for that python DB-API supports one method i.e. **connect ()** method.
- It creates connection between MySQL database and Python Application.
- If you import **MySQLdb(**in python2.x**)** then we need to use following code to create connection.

Syntax:

*Conn-name=MySQLdb.**connect**(<hostname>,<username>,<password>,<database>)*

Example:

Myconn =*MySQLdb.**connect*** ("localhost","root","root",”emp”)

(Or)

- If you import **mysql.connector(**in python3.x**)** then we need to use following code to create connection.

Syntax:

*conn-name= mysql.connector.**connect** (**host**=<host-name>,*
*   **user**=<username>,**passwd**=<pwd>,**database**=<dbname>)*

Example:

```
myconn=mysql.connector.connect(host="localhost",user="root",
                                passwd="root",database=”emp”)
```

3. Create the cursor object:

- After creation of connection object we need to create cursor object to execute SQL queries in MySQL database.
- The cursor object facilitates us to have multiple separate working environments through the same connection to the database.
- The Cursor object can be created by using **cursor ()** method.

Syntax:

*cur_came = conn-name.**cursor()***

Example:

my_cur=myconn.**cursor()**

4. **Execute the query:**

- After creation of cursor object we need to execute required queries by using cursor object. To execute SQL queries, python DB-API supports following method i.e. **execute ().**

Syntax:

*cur-name.**execute(query)***

Example:

my_cur.**execute** (“select * from Employee”)

5. **Close the connection:**

- After completion of all required queries we need to close the connection.

Syntax:

*conn-name.**close()***

Example:

*conn-name.**close()***

MySQLdb(in python2.x):

- MySQLdb is an interface for connecting to a MySQL database server from Python. The following are example programs demonstrate interactions with MySQL database using **MySQLdb** module.

- **Note** − Make sure you have root privileges of MySQL database to interact with database.i.e. Userid and password of MySQL database.

- We are going to perform the following operations on MySQL database.

  - ➢ Show databases
  - ➢ Create database
  - ➢ Create table
  - ➢ To insert data into table
  - ➢ Read/Select data from table
  - ➢ Update data in table
  - ➢ Delete data from table

Example Programs:

To display databases :

We can get the list of all the databases by using the following MySQL query.

>show databases;

Example: showdb.py

```
import MySQLdb
 #Create the connection object
myconn = MySQLdb.connect("localhost","root","root")
#creating the cursor object
cur = myconn.cursor()
#executing the query
dbs = cur.execute("show databases")
#display the result
for x in cur:
   print(x)
#close the connection
myconn.close()
```

Output:

```
>>>python showdb.py
('information_schema',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('test',)
('Sampledb',)
```

To Create database :

The new database can be created by using the following SQL query.

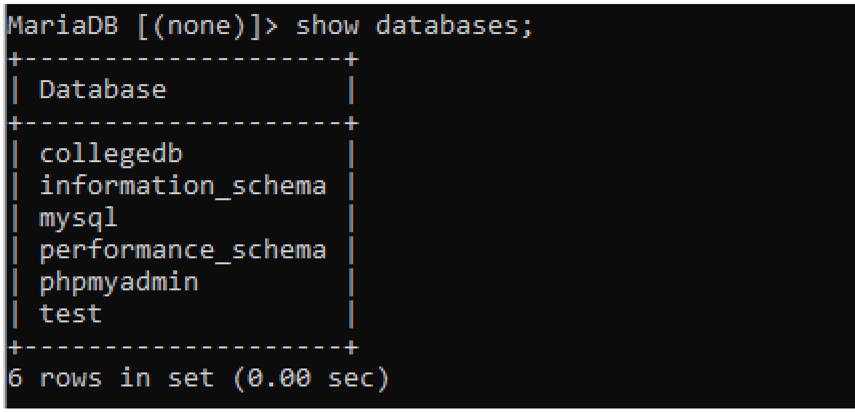
> create database <database-name>

Example: **createdb.py**

```
import MySQLdb
 #Create the connection object
myconn = MySQLdb.connect("localhost","root","root")
#creating the cursor object
cur = myconn.cursor()
#executing the query
cur.execute("create database Collegedb")
print("Database created successfully")
#close the connection
myconn.close()
```

Output:

```
>>>python createdb.py
Database created successfully
```



```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| collegedb          |
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
+--------------------+
6 rows in set (0.00 sec)
```

To Create table :

The new table can be created by using the following SQL query.

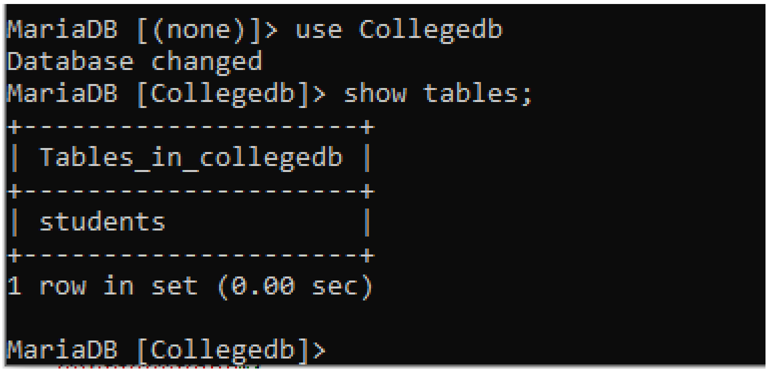
> create table <table-name> (column-name1 datatype, column-name2 datatype,…)

Example: **createtable.py**

```
import MySQLdb
 #Create the connection object
myconn = MySQLdb.connect("localhost","root","root",”Colleged”)
#creating the cursor object
cur = myconn.cursor()
#executing the query
cur.execute("create table students(sid varchar(20)primary key,sname varchar(25),age int(10))")
print("Table created successfully")
#close the connection
myconn.close()
```

Output:

```
>>>python createtable.py
Table created successfully
```



```
MariaDB [(none)]> use Collegedb
Database changed
MariaDB [Collegedb]> show tables;
+---------------------+
| Tables_in_collegedb |
+---------------------+
| students            |
+---------------------+
1 row in set (0.00 sec)

MariaDB [Collegedb]>
```

To Insert data into table :

The data can be inserted into table by using the following SQL query.

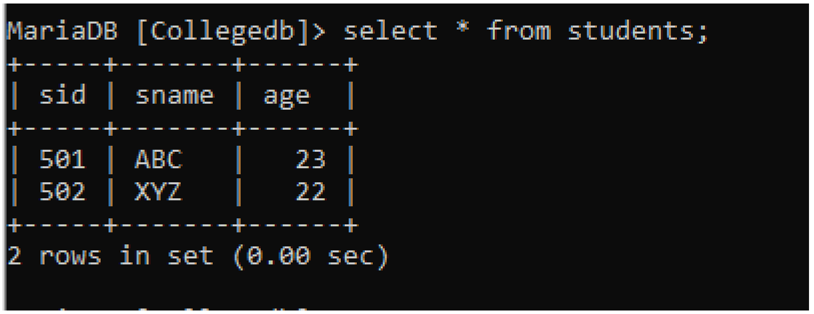
> insert into <table-name> values (value1, value2,…)

Example: **insertdata.py**

```
import MySQLdb
 #Create the connection object
myconn = MySQLdb.connect("localhost","root","root",”Colleged”)
#creating the cursor object
cur = myconn.cursor()
#executing the query
cur.execute("INSERT INTO students VALUES ('501', 'ABC', 23)")
cur.execute("INSERT INTO students VALUES ('502', 'XYZ', 22)")
#commit the transaction
myconn.commit()
print("Data inserted successfully")
#close the connection
myconn.close()
```

Output:

```
>>>python insertdata.py
Data inserted successfully
```



```
MariaDB [Collegedb]> select * from students;
+-----+-------+------+
| sid | sname | age  |
+-----+-------+------+
| 501 | ABC   |   23 |
| 502 | XYZ   |   22 |
+-----+-------+------+
2 rows in set (0.00 sec)
```

To Read/Select data from table ::

The data can be read/select data from table by using the following SQL query.

>select column-names from <table-name>

fetchall() method returns all rows in the table.
fetchone() method returns one row from table.

Example: **selectdata.py**

```
import MySQLdb
 #Create the connection object
myconn = MySQLdb.connect("localhost","root","root",”Colleged”)
#creating the cursor object
cur = myconn.cursor()
#executing the query
cur.execute("select * from students")
#fetching all the rows from the cursor object
result = cur.fetchall()
print("Student Details are :")
#printing the result
for x in result:
      print(x);
#close the connection
myconn.close()
```

Output:

```
>>>python selectdata.py

Student Details are:
('501', 'ABC', 23)
('502', 'XYZ', 22)
```

fetchall() method returns all rows in the table.
fetchone() method returns one row from table.

Example: **selectone.py**

```
import MySQLdb
 #Create the connection object
myconn = MySQLdb.connect("localhost","root","root",”Colleged”)
#creating the cursor object
cur = myconn.cursor()
#executing the query
cur.execute("select * from students")
#fetching all the rows from the cursor object
result = cur.fetchone()
print("One student Details are :")
 #printing the result
print(result)
#close the connection
myconn.close()
```

Output:

```
>>>python selectone.py

One student Details are:
('501', 'ABC', 23)
```

To Update data into table :

The data can be updated in table by using the following SQL query.

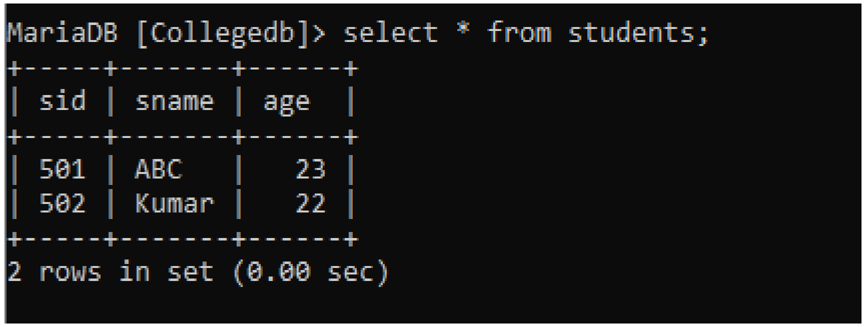
> update <table-name> set column-name=value where condition

Example: **updatedata.py**

```
import MySQLdb
 #Create the connection object
myconn = MySQLdb.connect("localhost","root","root",”Colleged”)
#creating the cursor object
cur = myconn.cursor()
#executing the query
cur.execute("update students set sname='Kumar' where sid='502'")
#commit the transaction
myconn.commit()
print("Data updated successfully")
#close the connection
myconn.close()
```

Output:

```
>>>python updatedata.py
Data updated successfully
```



```
MariaDB [Collegedb]> select * from students;
+-----+-------+------+
| sid | sname | age  |
+-----+-------+------+
| 501 | ABC   |   23 |
| 502 | Kumar |   22 |
+-----+-------+------+
2 rows in set (0.00 sec)
```

To Delete data from table :

The data can be deleted from table by using the following SQL query.

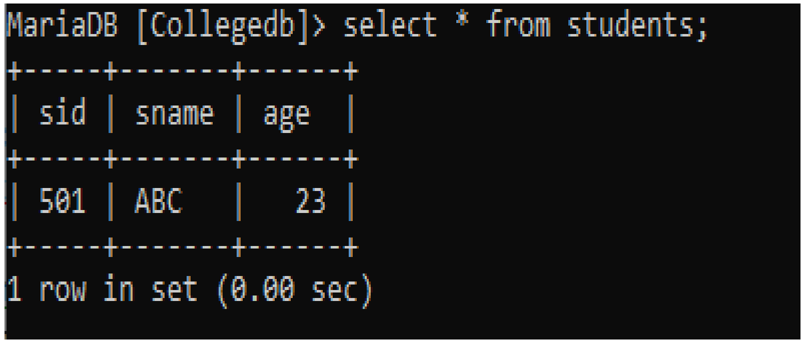
> delete from <table-name> where condition

Example: **deletedata.py**

```
import MySQLdb
 #Create the connection object
myconn = MySQLdb.connect("localhost","root","root",”Colleged”)
#creating the cursor object
cur = myconn.cursor()
#executing the query
cur.execute("delete from students where sid='502'")
#commit the transaction
myconn.commit()
print("Data deleted successfully")
#close the connection
myconn.close()
```

Output:

```
>>>python deletedata.py
Data deleted successfully
```



```
MariaDB [Collegedb]> select * from students;
+-----+-------+------+
| sid | sname | age  |
+-----+-------+------+
| 501 | ABC   |   23 |
+-----+-------+------+
1 row in set (0.00 sec)
```

DB-API for MySQL in Python

MySQLdb (python2.x)

```
#Import MySQLdb
import MySQLdb
 #Create the connection object
myconn =MySQLdb.connect
("localhost","root","root",”Colleged”)
```

Mysql.connector(python3.x)

```
#Import mysql.connector
import mysql.connector
 #Create the connection object
myconn=mysql.connector.connect
(host="localhost",user="root",
passwd="root",database="Colleged")
```

mysql.connector(in python3.x)::

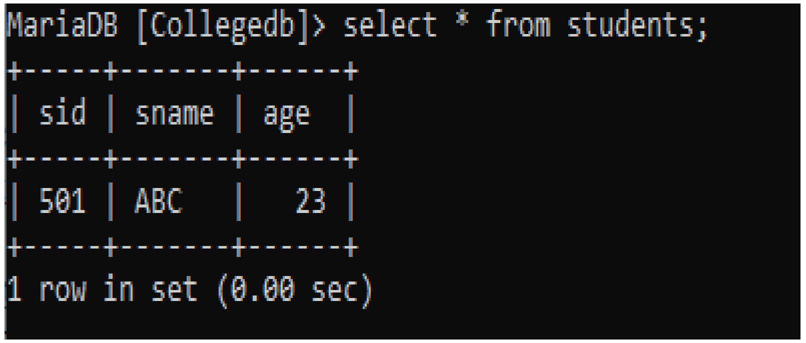
MySQL Connector enables Python programs to access MySQL databases.

Example: **deletedata.py**

```
import mysql.connector
 #Create the connection object
myconn=mysql.connector.connect(host="localhost",user="root",passwd="root",
database="Collegedb")
 #creating the cursor object
cur = myconn.cursor()
 #executing the querys
cur.execute("delete from students where sid='502'")
 #commit the transaction
myconn.commit()
print("Data deleted successfully")
 #close the connection
myconn.close()
```

Output:

```
>>>python deletedata.py
Data deleted successfully
```



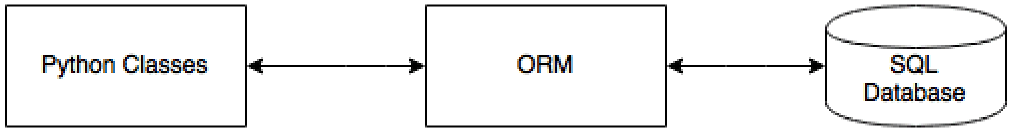
```
MariaDB [Collegedb]> select * from students;
+-----+-------+------+
| sid | sname | age  |
+-----+-------+------+
| 501 | ABC   |   23 |
+-----+-------+------+
1 row in set (0.00 sec)
```

Object Relational Mapping (ORM) in python

Introduction:

- **O**bject **R**elational **M**apping is a system of mapping objects to a database. That means it automates the transfer of data stored in relational databases tables into objects that are commonly used in application code.

- An object relational mapper maps a relational database system to objects. The ORM is independent of which relational database system is used. From within Python, you can talk to objects and the ORM will map it to the database.



- **ORM**s provide a high-level abstraction upon a relational database that allows a developer to write Python code instead of SQL to interact (create, read, update and delete data and schemas) with database.

The mapping like this…

- Python Class == SQL Table
- Instance of the Class == Row in the Table

- Developers can use the programming language they are comfortable with to work with a database instead of writing SQL statements or stored procedures.

- There are many ORM implementations written in Python, including
  - **SQLAlchemy**
  - Peewee
  - The Django ORM
  - PonyORM
  - SQLObject
  - Tortoise ORM

- We are going to discuss about **SQLAlchemy**,it pronounced as **SQL**-**All**-**Chemy**.

SQLAlchemy:

- **SQLAlchemy** is a library used to interact with a wide variety of databases. It enables you to create data models and queries in a manner that feels like normal Python classes and statements.

- It can be used to connect to most common databases such as Postgres, MySQL, SQLite, Oracle, and many others.

- **SQLAlchemy** is a popular SQL toolkit and Object Relational Mapper. It is written in Python and gives full power and flexibility of SQL to an application developer.

- It is necessary to install SQLAlchemy. To install we have to use following code at Terminal or CMD.

  pip install sqlalchemy

- To check if SQLAlchemy is properly installed or not, enter the following command in the Python prompt

  >>>import sqlalchemy

- If the above statement was executed with no errors, “sqlalchemy " is installed and ready to be used.

Connecting to Database:

- To connect with database using SQLAlchemy, we have to create engine for this purpose SQLAlchemy supports one function is **create_engine().**
- The **create_engine()** function is used to create engine; it takes overall responsibilities of database connectivity.

Syntax:

```
Database-server[+driver]://user:password@host/dbname
```

Example:

```
mysql+mysqldb://root:root@localhost/collegedb
```

- The main objective of the ORM-API of SQLAlchemy is to facilitate associating user-defined Python classes with database tables, and objects of those classes with rows in their corresponding tables.

Declare Mapping:

- First of all, create_engine() function is called to set up an engine object which is subsequently used to perform SQL operations.

To create engine in case of MySQL:

Example:

from sqlalchemy import create_engine

engine = create_engine('mysql+mysqldb://root:@localhost/Collegedb')

- When using ORM, we first configure database tables that we will be using. Then we define classes that will be mapped to them. Modern SQLAlchemy uses *Declarative* system to do these tasks.
- A *declarative base class* is created, which maintains a catalog of classes and tables. A declarative base class is created with the declarative_base() function.
- *The declarative_base() function is used to create base class. This function is defined in* ***sqlalchemy.ext.declarative*** *module.*

To create declarative base class:

Example:

from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

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

```
from sqlalchemy import Column, Integer, String
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
# create a engine
engine =create_engine('mysql+mysqldb://root:@localhost/Sampledb‘)
# create a declarative base class
Base = declarative_base()

class Students(Base):
   __tablename__ = 'students'
   id = Column(Integer, primary_key=True)
   name = Column(String(10))
   address = Column(String(10))
   email = Column(String(10))

Base.metadata.create_all(engine)
```