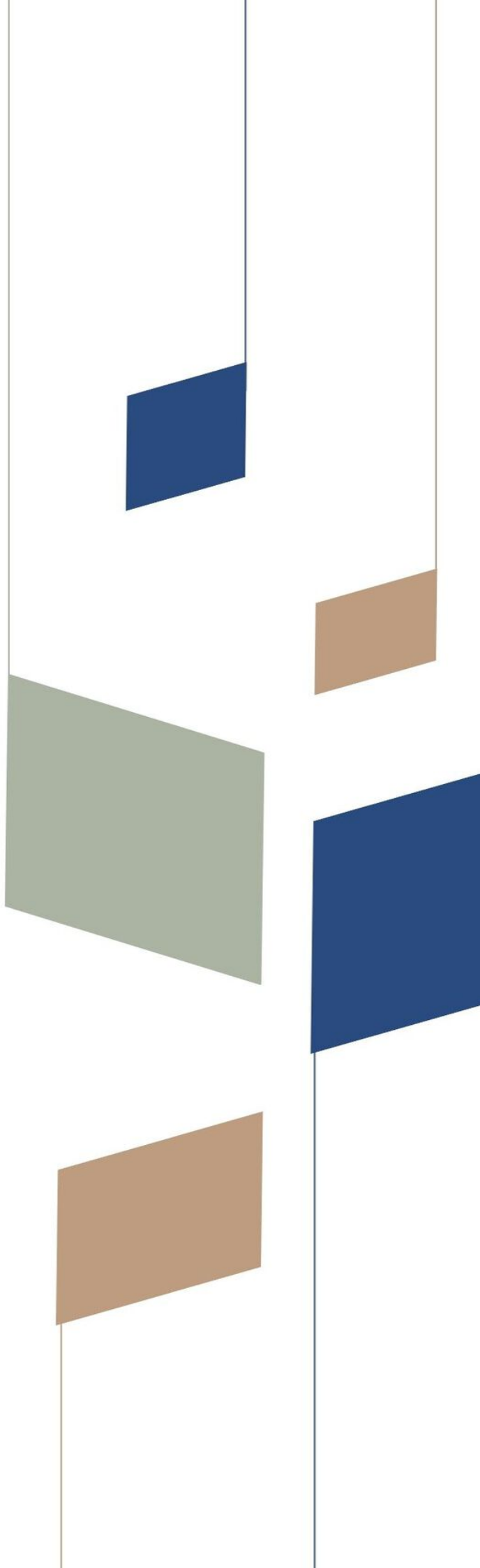
TKINTER GUI PROJECTS WITH PYTHON

JEFFREY LEON STROUP



About the Authors

Python is of the most popular and versatile programming languages in the tech industry. However, despite their popularity and versatility, mastering them can be challenging, especially for beginners. Technical challenges such as debugging and tight deadlines can cause stress and anxiety, and career advancement and staying up to date with the latest developments in the field can be daunting.

Table of Contents

Contents

Tkinter GUI Projects with Python

PART 1: Introduction to Python Tkinter Module

In this tutorial, we will cover an introduction to Tkinter, its prerequisites, different ways for GUI Programming, how to install Tkinter, and its working.

Tkinter is a **standard library** in python used for creating **Graphical User Interface (GUI)** for Desktop Applications. With the help of **Tkinter** developing **desktop applications** is not a tough task.

The **primary GUI toolkit** we will be using is Tk, which is Python's default GUI library. We'll access Tk from its Python interface called **Tkinter** (short for **Tk interface**).

Prerequisites for Tkinter

Before learning **Tkinter** you should have **basic knowledge of Python**. You can learn Python using our Complete Python Tutorial.

GUI Programming in Python

There are many ways to develop GUI based programs in Python. These different ways are given below:

1. **Tkinter:**

   In Python, **Tkinter** is a standard **GUI** (graphical user interface) package. Tkinter is **Python's default GUI module** and also the most common way that is used for **GUI programming** in Python. Note that **Tkinter** is a set of **wrappers** that implement the Tk widgets as Python classes.

2. **wxPython:**

   This is basically an open-source, cross-platform **GUI toolkit that is written in C++**. Also an **alternative to Tkinter**.

3. **JPython:**

JPython is a Python platform for Java that is providing Python scripts seamless access to **Java class Libraries** for the local machine.

We will cover GUI Programming with Tkinter.

What is Tkinter?

Tkinter in Python helps in **creating GUI** Applications with a minimum hassle. Among various GUI Frameworks, Tkinter is the only framework that is built-in into **Python's Standard Library**.

- An important feature in favor of Tkinter is that it is **cross-platform**, so the same code can easily work on **Windows**, **macOS**, and **Linux**.
- Tkinter is a **lightweight module**.
- It is **simple** to use.

What are Tcl, Tk, and Tkinter?

Let's try to understand more about the Tkinter module by discussing more about it origin.

- As mentioned, Tkinter is **Python's default GUI library**, which is nothing but a wrapper module on top of the **Tk toolkit**.
- Tkinter is based upon the Tk toolkit, and which was originally designed for the **Tool Command Language** (Tcl). As Tk is very popular thus it has been ported to a **variety of other scripting languages**, including **Perl (Perl/Tk)**, **Ruby (Ruby/Tk)**, and **Python (Tkinter)**.
- **GUI development portability** and **flexibility of Tk** makes it the right tool which can be used to design and **implement a wide variety of commercial-quality GUI applications.**

- Python **with Tkinter** provides us a **faster and efficient way** in order to build useful applications that would have taken much time if you had to program directly in C/C++ with the help of native OS system libraries.
- Once we have Tkinter up and running, we'll use basic building blocks known as **widgets** to create a variety of desktop applications.

Install Tkinter

Chances are, that Tkinter may be already installed on your system along with Python. But it is not true always. So let's first check if it is available.

If you do not have Python installed on your system - Install Python 3.8 first, and then check for Tkinter.

You can determine **whether Tkinter is available** for your Python interpreter by attempting to **import the Tkinter module** - If Tkinter is available, then there will be no errors, as demonstrated in the following code:

```
import tkinter
```

Nothing exploded, so we know we have **Tkinter available**. If you see any error like module not found, etc, then your **Python interpreter was not compiled with Tkinter enabled**, the module **import fails** and you might need to recompile your **Python interpreter to gain access to Tkinter**.

Adding Tk to your Applications

Basic steps of setting up a GUI application using Tkinter in Python are as follows:

1. First of all, **import the Tkinter module**.
2. The second step is to **create a top-level windowing object** that contains your entire GUI application.
3. Then in the third step, you need to **set up all your GUI components** and their functionality.

4. Then you need to **connect these GUI components** to the underlying application code.
5. Then just enter the main event loop using mainloop()

The above steps may sound gibberish right now. But just read them all, and we will explain everything as we move on with this tutorial.

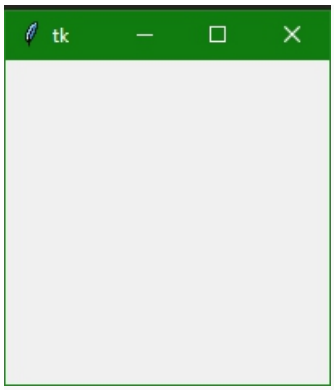
First Tkinter Example

As mentioned earlier that in GUI programming all main widgets are only built on the top-level window object.

The top-level window object is created by the Tk class in Tkinter.

Let us create a top-level window:

```
import tkinter as tk
win = tk.Tk()
###you can add widgets here
win.mainloop()
```



Tkinter Methods used above:

The two main methods are used while creating the **Python application** with **GUI**. You must need to remember them and these are given below:

1. **Tk(screenName=None, baseName=None, className='Tk', useTk=1)**

This method is mainly used **to create the main window**. You can also change the **name of the window** if you want, just by

changing the **className** to the desired one.

The code used to create the main window of the application is and we have also used it in our above example:

```
win = tkinter.Tk()    ## where win indicates name of the main window object
```

2. The mainloop() Function

This method is used to start the application.
The mainloop() function is an **infinite loop** which is used to run the application, it will wait for **an event to occur** and **process the event** as long as the window is not closed.

Summary:

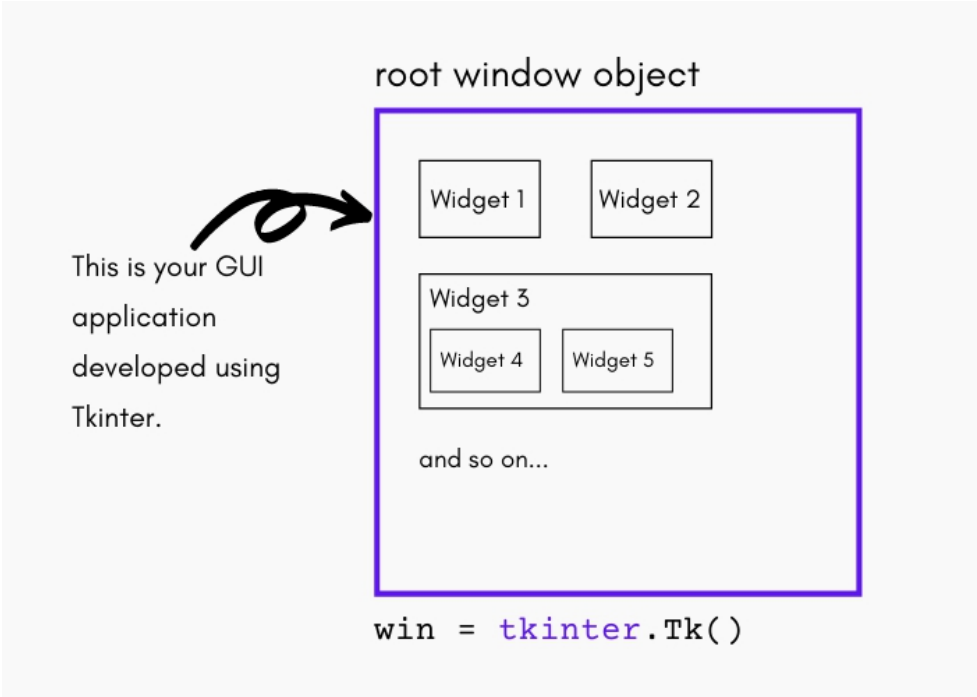
With this we have completed the introduction to Tkinter, we have installed the Tkinter module, and even know what are Windows and Widgets in Tkinter. We also create our first Tkinter GUI app and run it. In the next tutorial, we will learn more about Python Tkinter widgets.

PART 2: Tkinter Windows, Widgets and Frames

In this tutorial, we will cover the basics of the Tkinter module, to explain what is a Tkinter **Window, Widgets, and Frames** which are the **building blocks** of GUI application in Tkinter.

Introduction to Tkinter Windows and Widgets

Let's dive in a little deeper, to understand some basics about the Tkinter module and how it works.



- The **top-level window object** in GUI Programming contains all of the **little window objects** that will be Part of **your complete GUI**.
- The **little window objects** can be **text labels**, **buttons**, **list boxes**, etc., and these individual little GUI components are known as **Widgets**.

- So, having a top-level window object will act as a container where you will put all your widgets. In Python, you'd typically do so like this using the following code: `win = tkinter.Tk()`
- The object that is returned by making a call to `tkinter.Tk()` is usually referred to as **the Root Window**.
- **Top-level windows** are mainly **stand-alone as Part of your application**, also **you can have more than one top-level window** for your GUI, but only one of them should be your **root window.**
- First of all, you need to design all your widgets completely, and then **add the real functionality.**
- **The widgets can either be stand-alone** or can be **containers**. If one widget contains other widgets, it is considered the **parent of those widgets**.
- Similarly, if **a widget is contained within another widget**, it's known as **a child of the parent,** the **parent is the next immediate** enclosing container widget.
- The **widgets also have some associated behaviors**, such as **when a button is pressed**, or **text is filled into a text field**, so we have events attached to these actions.
- The behavior of widgets generates events, and the **GUI's response to events** is known as **Callbacks** - because **they 'call' a function just to handle the event** that occurred.

Tkinter Event-Driven Processing

In Tkinter, we have windows and widgets coming together to form a GUI application. But the GUI application is just the frontend of the application. You would want to execute some code logic when the end-user uses those widgets.

Whenever an action is performed on any widget, an event is generated, which we can handle to perform any operation.

- Events(behavior of widgets) can include **pressing a button**, **movement of the mouse**, **hitting the return or Enter key**, **gaining or losing 'focus'**, etc.
- The **entire system of events** that occurs from the **beginning until the end of any GUI application** is what drives it and hence it is also known as E**vent-Driven Processing**.
- Let us take a **simple mouse movement** example: Suppose that the **pointer of the mouse is just sitting somewhere** on top of your GUI application. If you will move the mouse to another Part of your application, something has to cause the movement of the mouse to be replicated by the cursor on your screen(on top of your GUI application). These are **'cursor move'** events that the system **must process to portray** your cursor moving across the window. At the time you will **stop moving the mouse**, no more events need to be processed, so everything **just stays still on the screen again**.

Here are some basic definitions through which you will be able to understand the concept of Windows, widgets, and frames in Tkinter.

Tkinter Windows

The term "Window" has different meanings in the different contexts, But generally **"Window"** refers to a rectangular area somewhere on the user's display screen through which you can interact.

Then there comes the concept of Top Level Window in Tkinter.

Tkinter Top-Level Window

The Top-Level Window is a window that **exists independently on the screen**. You can decorate the top-level window with the standard frame and controls for the desktop manager. It can usually be moved **around the desktop, and also you can resize it if you want to do so**.

Then there comes the concept of Widgets. Let us try to understand it.

Tkinter Widgets

The term “Widgets” is a generic term that refers to the **building blocks that make up an application in a graphical user interface**.

Let us list out the core widgets with their categories:

- **Container**

  Under this category, the widgets that lies are frame, labelframe, toplevel, and paned window.

- **Buttons**

  Under the category of Buttons, there are buttons, radiobuttons, checkbuttons (checkbox), and menubuttons.

- **Text Widgets**

  Under the category of text widgets, there are labels, messages, text.

- **Entry Widgets**

  Under this category, the widgets are scale, scrollbar, Listbox, slider, spinbox, entry (single-line), optionmenu, text (multiline), and canvas (vector and pixel graphics).

Now let us move on to Frames in Tkinter.

Tkinter Frames

A frame is basically a rectangular area that can contain other widgets. In Tkinter, there is a Frame widget that is the basic unit of organization for complex layouts. It is a widget which has no special styling or GUI component of its own. It is just used to hold other Tkinter widgets in case of complex GUI layouts.

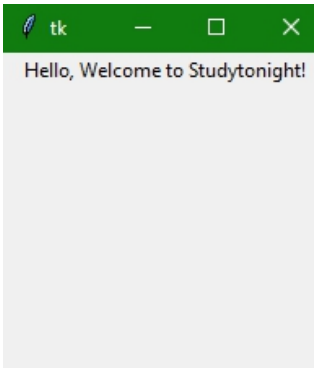
Note: It is important to note here that whenever any widget is created, there is also a parent-child relationship that is created.

Just take an example, if you place a button inside a frame, the frame widget is called the parent of the Button widget.

Tkinter Basic Example

Let us take an example where we will create a Tkinter application with a simple Text widget:

```
from tkinter import *
win = Tk() # Create the root (base) window
win.geometry("200x200")
w = Label(win, text="Hello, Welcome to Studytonight!") # Create a label with words
w.pack() # Put the label into the window
win.mainloop()# Start the event loop
```



The above code will create a window with the label widget and output will look like as shown above. We have created a Tkinter Window and then added a basic Label widget to it.

Summary:

In this tutorial, we learned about the basic building blocks of GUI application using Tkinter which are Windows, Widgets, and Frames, which are used to develop different GUI applications. In the next tutorial, we will learn how to create a Tkinter Window which is the starting point for any application, because in a Tkinter Window we add all the widgets.

PART 3: Tkinter Windows

In this tutorial, we will learn about Tkinter Windows in Python which is the main Window of the GUI application inside which every other component runs. We have covered the basic Tkinter GUI application Components in which we have explained how Tkinter Windows, Widgets, and Frames are building blocks of a Tkinter App.

Tkinter Windows

The Tkinter window is the **foundational element** of the Tkinter GUI. Tkinter window is a **container in which all other GUI elements(widgets) live**.

Here is the **syntax** for creating a basic Tkinter Window:

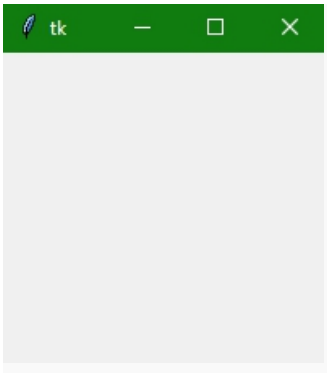
```
win = Tk()
```

Yes, we use the Tk() function to create our Tkinter app window in which all the other components are added.

Tkinter Windows Example:

Here is a simple example,

```
from tkinter import *
win = Tk()
# run the app window
win.mainloop()
```



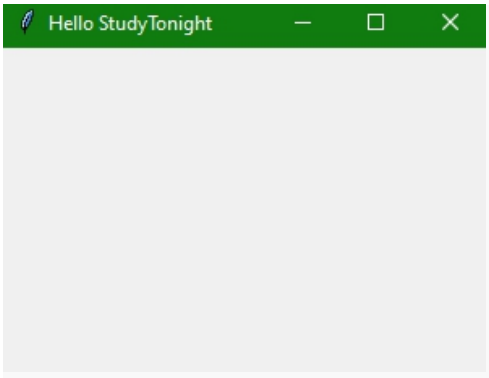
In the above example, the mainloop() function is used to run the GUI application.

Tkinter Customized Window

Let us now cover a basic example where we will create a Basic GUI Application using properties like **title** and **geometry**.

Here we have the code to demonstrate the steps used in the creation of a customized Tkinter Window:

```
from tkinter import *
window = Tk()
# You can add your widgets here
window.title('Hello StudyTonight')
window.geometry("300x200+10+20")
window.mainloop()
```



Here is what we have done in the code:

- The first step is to import the **Tkinter module in the code.**
- After importing, the second step is to set up the application object by calling the Tk() function. This will create a top-level window (root) having a frame with a **title bar** and **control box** with the minimize and close buttons, and a **client area to hold other widgets**.
- After that, you can add the widgets you might want to add in your code like buttons, textbox, scrollbar,

labels, and many more.

- The window.title() function is used to provide the title to the user interface as you can see in the output.
- In the line window.geometry("300x200+10+20); the geometry() method defines the **width**, **height,** and **coordinates** of the **top left corner** of the frame as follows (all values are generally in pixels) in the same way. Here is the syntax:

```
window.geometry("widthxheight+XPOS+YPOS")
```

- After that, the application object enters an **event listening loop** by calling the mainloop() method. In this way, the application is now **constantly waiting for any event generated** on the elements in it. There could be an event like **text entered in a text field**, a **selection made from the dropdown** or radio button, single/double click actions of the mouse, etc.

The application's functionality involves **executing appropriate callback functions** in response to a Particular type of event.

The event loop will terminate as soon as there is a click made on the **close button on the title bar**.

Summary:

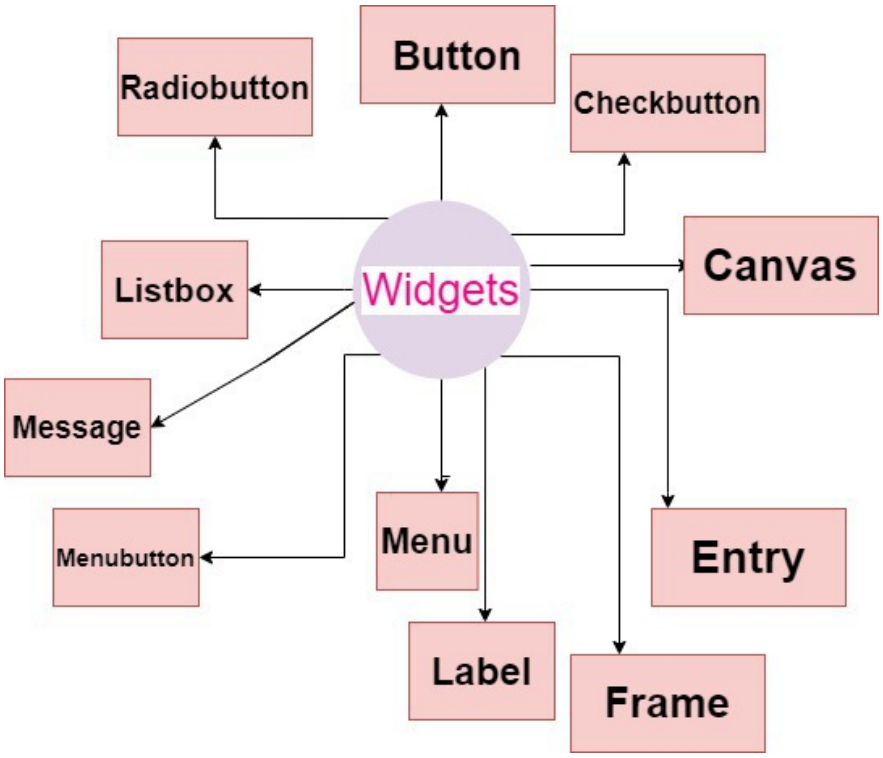
In this tutorial, we learned how to create a Tkinter window to create a GUI application. The Tkinter window contains all the widgets that make the application.

PART 4: Python Tkinter Widgets

In this tutorial, we will cover an overview of Tkinter widgets in Python. These widgets are the functional units on any Tkinter GUI application.

Tkinter Widgets

There are various controls, such as **buttons**, **labels**, **scrollbars**, **radio buttons**, and **text boxes** used in a GUI application. These **little components** or controls of **Graphical User Interface (GUI)** are known as **widgets** in Tkinter.



Summary:

So in this tutorial, we got a basic introduction to Trinket Widgets. In our upcoming tutorial pages, we will cover each widget in detail with their respective code examples.

PART 5: Python Tkinter Geometry Manager

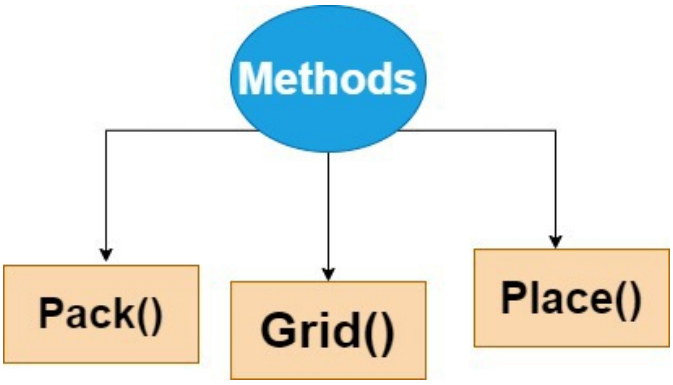
In this tutorial, we will learn how to control the **layout of the Application** with the help of the Tkinter **Geometry Managers**.

Controlling Tkinter Application Layout

In order to **organize or arrange or place all the widgets** in the parent window, Tkinter provides us the **geometric configuration** of the widgets. The GUI Application Layout is mainly controlled by Geometric Managers of Tkinter.

It is important to note here that each window and Frame in your application is allowed to use **only one geometry manager**. Also, **different frames** can use different geometry managers, even if they're already assigned to a frame or window using another geometry manager.

There are mainly three methods in Geometry Managers:



Let us discuss each method in detail one by one.

1. Tkinter pack() Geometry Manager

The pack() method mainly uses a **packing algorithm** in order to place widgets in a Frame or window in a specified order.

This method is mainly used to **organize the widgets in a block**.

Packing Algorithm:

The steps of Packing algorithm are as follows:

- Firstly this algorithm will **compute a rectangular area** known as a **Parcel** which is tall (or wide) enough to hold the widget and then it will fill the remaining width (or height) in the window with **blank space**.
- It will **center the widget** until any different location is specified.

This method is powerful but it is difficult to visualize.

Here is the **syntax** for using pack() function:

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

The possible options as a parameter to this method are given below:

- **fill**

The default value of this option is set to **NONE**. Also, we can set it to **X or Y** in order to determine whether the **widget contains any extra space**.

- **side**

This option specifies which **side to pack the widget against**. If you want to pack **widgets vertically**, use **TOP** which is the default value. If you want to pack **widgets horizontally**, use **LEFT**.

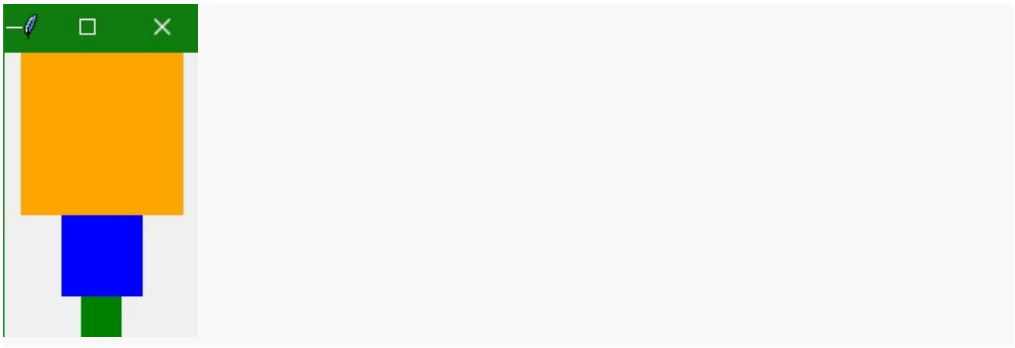
- **expand**

This option is used to **specify whether the widgets should be expanded to fill any extra space** in the geometry master or not. Its default value is false. If it is false then the **widget is not expanded** otherwise widget expands to fill extra space.

Tkinter pack() Geometry Manager Example:

Let us discuss an example where we will see what happens when you pack() three colored Frame widgets(here is the Tkinter Frame Widget) into a window:

```
import tkinter as tk
win = tk.Tk()
# add an orange frame
frame1 = tk.Frame(master=win, width=100, height=100, bg="orange")
frame1.pack()
# add blue frame
frame2 = tk.Frame(master=win, width=50, height=50, bg="blue")
frame2.pack()
# add green frame
frame3 = tk.Frame(master=win, width=25, height=25, bg="green")
frame3.pack()
window.mainloop()
```



According to the output of the above code, the pack() method just places each Frame below the **previous one by default**, in the same **order in which they're assigned to the window**.

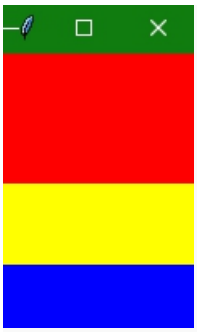
Tkinter pack() with Parameters

Let's take a few more code examples using the parameters of this function like fill, side and, expand.

You can set the **fill** argument in order to specify in which **direction** you want the frames should fill. If you want to fill in the **horizontal** direction then the option is tk.X, whereas, tk.Y is used to **fill vertically**, and to fill in both directions tk.BOTH is used.

Let's take another example where we will stack the three frames so that each one fills the whole window horizontally:

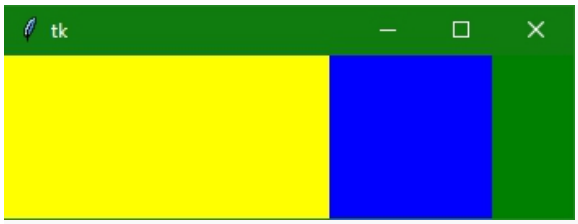
```
import tkinter as tk
win= tk.Tk()
frame1 = tk.Frame(master=win, height=80, bg="red")
# adding the fill argument with
# horizontal fill value
frame1.pack(fill=tk.X)
frame2 = tk.Frame(master=win, height=50, bg="yellow")
frame2.pack(fill=tk.X)
frame3 = tk.Frame(master=win, height=40, bg="blue")
frame3.pack(fill=tk.X)
win.mainloop()
```



In the above output, you can see that the frames fill the entire width of the application window because we used the tk.X value for the fill parameter.

Now let's take another code example, where we will be using all options, namely, fill, side, and expand options of the pack() method:

```
import tkinter as tk
win = tk.Tk()
frame1 = tk.Frame(master=win, width=200, height=100, bg="Yellow")
# setting fill, side and expand
frame1.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)
frame2 = tk.Frame(master=win, width=100, bg="blue")
frame2.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)
frame3 = tk.Frame(master=win, width=50, bg="green")
frame3.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)
win.mainloop()
```



If you will run this above code in your system then you can see this output is able to expand in both directions.

2. Tkinter grid() Geometry Manager

The most used geometry manager is grid() because it provides all the power of pack() function but in an easier and maintainable way.

The grid() geometry manager is mainly used to split either a window or frame into rows and columns.

- You can **easily specify the location of a widget** just by calling grid() function and passing the **row** and **column indices** to the row and column keyword arguments, respectively.
- Index of both the row and column starts from 0, so a row index of **2** and a column index of **2** tells the grid() function to place a widget in the **third**

column of the third row(0 is first, 1 is second and 2 means third).

Here is the **syntax** of the grid() function:

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

The possible options as a parameter to this method are given below:

- **Column**

This option specifies the column number in which the widget is to be placed. The index of **leftmost** column is **0**.

- **Row**

This option specifies the row number in which the widget is to be placed. The **topmost** row is represented by **0**.

- **Columnspan**

This option specifies the width of the widget. It mainly represents the number of columns up to which, the column is expanded.

- **Rowspan**

This option specifies the height of the widget. It mainly represents the number of rows up to which, the row is expanded.

- **padx, pady**

This option mainly represents the number of pixels of padding to be added to **the widget just outside the widget's border**.

- **ipadx, ipady**

This option is mainly used to represents the number of pixels of padding to be added to the widget **inside the widget's border**.

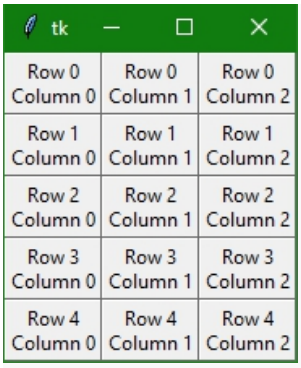
- **Sticky**

If any cell is larger than a widget, then sticky is mainly **used to specify the position of the widget inside the cell.** It is basically concatenation of the sticky letters which represents the position of the widget. It may be N, E, W, S, NE, NW, NS, EW, ES.

Tkinter grid() Geometry Manager Example:

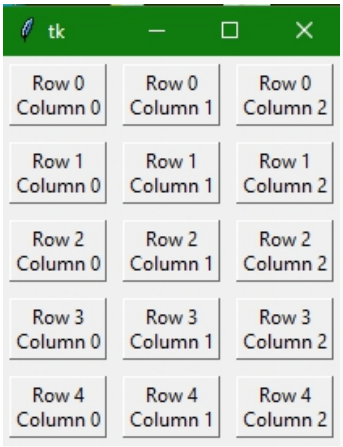
The following code script will help you to create a **5 × 3 grid** of frames with Label widgets packed into them:

```
import tkinter as tk
win = tk.Tk()
for i in range(5):
    for j in range(3):
        frame = tk.Frame(
            master = win,
            relief = tk.RAISED,
            borderwidth = 1
        )
        frame.grid(row=i, column=j)
        label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")
        label.pack()
win.mainloop()
```



If you want to **add some padding** then you can do it by using the following code snippet:

```
import tkinter as tk
win = tk.Tk()
for i in range(5):
    for j in range(3):
        frame = tk.Frame(
            master=win,
            relief=tk.RAISED,
            borderwidth=1
        )
        frame.grid(row=i, column=j, padx=5, pady=5)
        label = tk.Label(master=frame, text=f"Row {i}\nColumn {j}")
        label.pack()
win.mainloop()
```



As you can see in the code example above, we have used the padx and pady parameters because of which padding is applied outside the widget. To add padding inside the Frame widget, use the parameters ipadx and ipady in your code.

Similarly, do try using other parameters too for the grid() geometry manager.

3. Trinket place() Geometry Manager

The place() Geometry Manager organizes the widgets to **place them in a specific position** as directed by the programmer.

- This method basically **organizes the widget** in accordance with its **x and y coordinates**. Both x and y coordinates are in **pixels**.
- Thus the origin (where x and y are both 0) is the **top-left corner** of the Frame or the window.
- Thus, the y argument specifies the **number of pixels of space from the top of the window**, to place the widget, and the x argument specifies the **number of pixels from the left of the window**.

Here is the **syntax** of the place() method:

```
widget.place(options)
```

The possible options as a parameter to this method are given below:

- **x, y**

This option indicates the **horizontal and vertical** offset in the pixels.

- **height, width**

This option indicates the **height and weight** of the widget in the pixels.

- **Anchor**

This option mainly represents the **exact position** of the widget within the container. The default value (direction) is **NW** that is (the upper left corner).

- **bordermode**

This option indicates the **default value of the border type** which is INSIDE and it also refers to ignore the parent's inside the border. The other option is OUTSIDE.

- **relx, rely**

This option is used to represent the float between 0.0 and 1.0 and it is the **offset in the horizontal and vertical direction.**

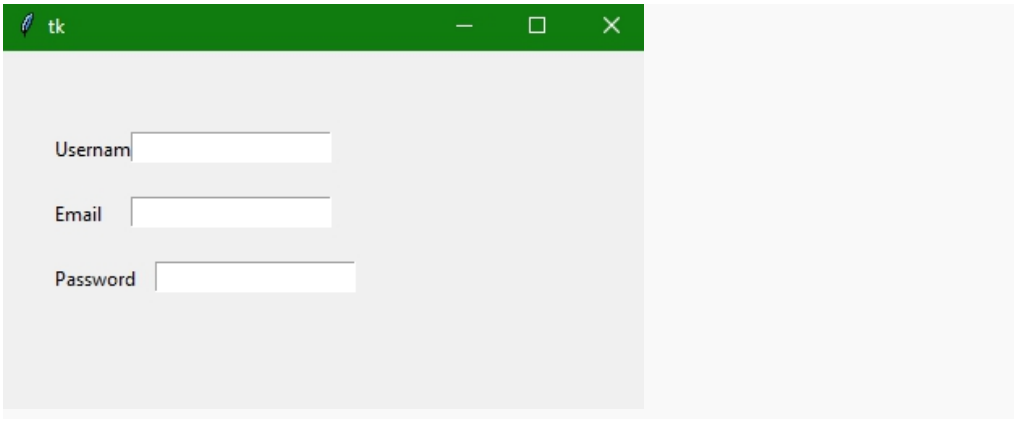
- **relheight, relwidth**

This option is used to represent the float value between 0.0 and 1.0 indicating the fraction of the **parent's height and width**.

Tkinter place() Geometry Manager Example:

The code snippet for this is given below:

```
from tkinter import *
top = Tk()
top.geometry("400x250")
Username = Label(top, text = "Username").place(x = 30,y = 50)
email = Label(top, text = "Email").place(x = 30, y = 90)
password = Label(top, text = "Password").place(x = 30, y = 130)
e1 = Entry(top).place(x = 80, y = 50)
e2 = Entry(top).place(x = 80, y = 90)
e3 = Entry(top).place(x = 95, y = 130)
top.mainloop()
```



In the above code example, we have used Tkinter Label and Tkinter Entry widget, we will cover them in detail

in the upcoming tutorials.

Summary:

In this tutorial, we learned how we can position our widgets inside the frame or window of our GUI application. We learned about the three Tkinter geometry managers, namely, pack(), grid() and place().

From the next tutorial, we will start covering different Tkinter widgets.

PART 6: Python Tkinter Label Widget

In this tutorial, we will cover the Tkinter **Label widget** in Python, which is used to create a Label in the GUI application in which we can show any text or image.

The **label widget** in Tkinter is used to **display boxes** where **you can place your images and text**.

- The **label widget** is mainly used to **provide a message** about the **other widgets** used in the Python Application **to the user**.
- You can **change or update the text** inside the label widget anytime you want.
- This widget uses **only one font at the time** of displaying some text.
- You can perform other tasks like **underline some Part of the text** and you can also **span text to multiple lines**.
- There are various options available **to configure the text** or the **Part of the text** shown in the **Label**.

Tkinter Label Widget

The **syntax** of the label widget is given below,

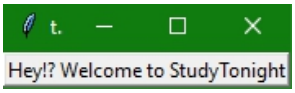
```
W = Label(master,options)
```

In the above syntax, the master parameter denotes **the parent window.** You can use many options to configure the text and these options are written as **comma-separated key-value pairs**.

Tkinter Label Widget Example

Now let us see a basic example of the label widget and the code snippet is given below:

```
import tkinter
from tkinter import *
win = Tk()
var = StringVar()
label = Label( win, textvariable=var, relief=RAISED )
# set label value
var.set("Hey!? Welcome to StudyTonight")
label.pack()
win.mainloop()
```



In the above code, we created a simple variable StringVar() and then assigned a value to it, and this variable is assigned as value to the textvariable option of the Label widget.

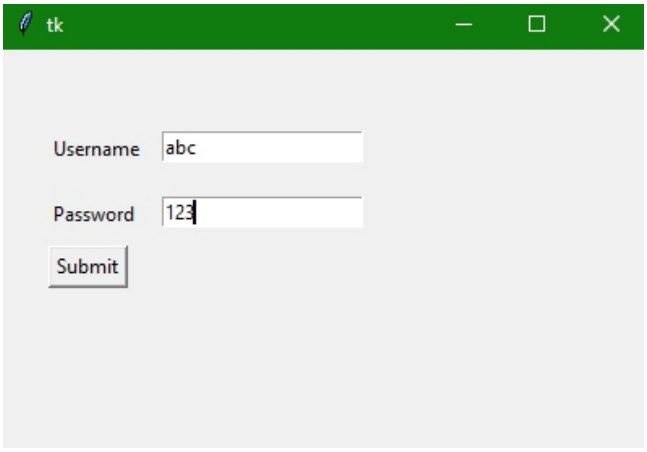
Tkinter Label Widget - Another Example

Below we have another code snippet for more clear understanding. Let us see the code snippet given below:

```
from tkinter import *
win = Tk()
win.geometry("400x250")
#creating a label
username = Label(win, text = "Username").place(x = 30,y = 50)
#creating second label
```

```
password = Label(win, text = “Password”).place(x = 30, y = 90)

submitbutton = Button(win, text = “Submit”,activebackground = “red”, activeforeground = “blue”).place(x = 30, y = 120)

e1 = Entry(win,width = 20).place(x = 100, y = 50)

e2 = Entry(win, width = 20).place(x = 100, y = 90)

win.mainloop()
```

Whenever you will run the above code, after putting **values into username and password label**, when you will click on the submit button then its **color gets changed to red**.



Don’t worry about the Button Widget and Entry Widget used in the above code, we will cover them shortly in the upcoming tutorials. This example is to give you an idea about how Tkinter widgets are used to create user interfaces for your Tkinter application.

Summary:

In this tutorial, we covered the Tkinter Label Widget which is used to show text and images in Tkinter GUI application or to add texts with form input fields like we have done in the example above.

PART 7: Python Tkinter Button Widget

In this tutorial, we will cover the **Button widget** of Tkinter module in Python.

The **Button widget** in Tkinter is mainly used to **add a button in any GUI Application**. In Python, while using the Tkinter button widget, we can easily modify the style of the button like adding a background colors to it, adjusting height and width of button, or the placement of the button, etc. very easily.

- You can add various types of buttons(as per your applications UI) to your application with the help of the Button Widget.
- You can also associate any **method or function** with a button if you want and then that method will get called **whenever you press the button**.
- There are **many options** of **button widget** which you can **reset or set according to your requirements**.

Tkinter Button Widget

The **syntax** of the button widget is given below,

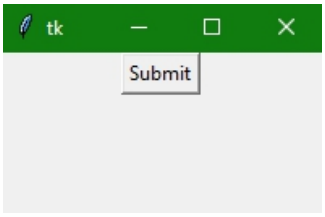
```
W = Button(master, options)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many options to change the **look of the buttons and these options are written as comma-separated**.

Tkinter Button Widget Example

Now let us create a simple submit button with the help of code snippet given below:

```
from tkinter import *
win = Tk()  ## win is a top or parent window
win.geometry("200x100")
b = Button(win, text = "Submit")
b.pack()  #using pack() geometry
win.mainloop()
```



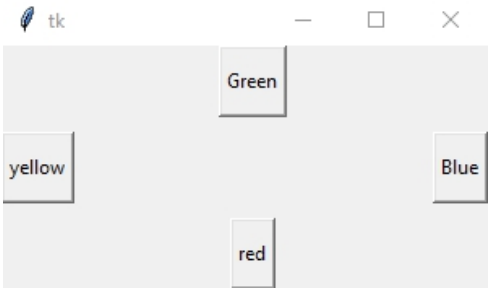
In the above code example, we created a simple window of given width and height. Then we added a button widget to it with providing the **window created as the master of that button** and adding a **text** for the button.

Tkinter Button Widget - Add style and Event handler

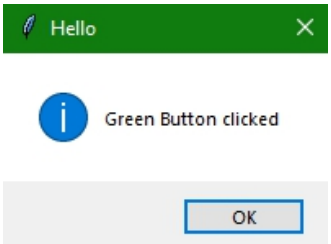
Below we have another code snippet where we will change the look of buttons by adding morestyle to it. Let us see how we do it:

```
import tkinter
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("300x150")
def click():
    messagebox.showinfo("Hello", "Green Button clicked")
```

```
a = Button(top, text="yellow", activeforeground="yellow", activebackground="orange", pady=10)
b = Button(top, text="Blue", activeforeground="blue", activebackground="orange", pady=10)
# adding click function to the below button
c = Button(top, text="Green", command=click, activeforeground = "green", activebackground="orange", pady=10)
d = Button(top, text="red", activeforeground="yellow", activebackground="orange", pady=10)
a.pack(side = LEFT)
b.pack(side = RIGHT)
c.pack(side = TOP)
d.pack(side = BOTTOM)
top.mainloop()
```



In the above code, we have added some styling using different options and we have added an event handler to handle the click event of the 3rd button. So whenever you will make a click on the **button with text Green**, you will see a Tkinter messagebox with a message.



Summary:

In this tutorial, we learned how to make a Tkinter Button widget with various options like chaning the style of the button, adding text to the button or positioning the button. We also saw a code example for adding an event handler function to any button to perform some action when the button is clicked.

PART 8: Python Tkinter Checkbutton Widget

In this tutorial, we will cover Tkinter Checkbutton widget in Python, which is used to create checkboxes on GUI while developing desktop applications using Tkinter.

If you want to display a number of options in a form, where users can check to select any option, we can use the Tkinter **Checkbutton widget**. It allows you to **select multiple options** or a **single** option at a time just by clicking the button corresponding to each option.

For example, in a form, you see option to fill in your Gender, it has options, Male, Female, Others, etc., and you can tick on any of the options, that is a checkbox. We use the **<input> tag** in HTML, to create checkbox

It can either contain **text or image**. There are a number of options available to configure the **Checkbutton widget** as per your requirement.

Tkinter Checkbutton Widget

The syntax of the **checkbutton widget** is given below:

```
w = CheckButton(master, option=value)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many **options** to configure your checkbutton widget and these **options** are written as **comma-separated key-value pair**.

Tkinter Checkbutton Widget Methods:

Following are the methods used with checkbutton widgets:

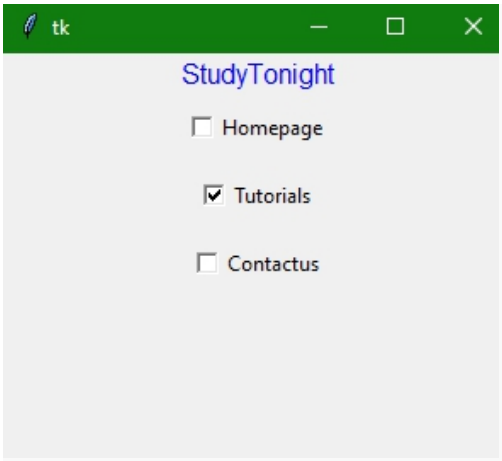
Method Name	Description
invoke()	This method in checkbutton widget is used to **invoke the method associated** with the checkbutton.
select()	This method in the checkbutton widget is called **to turn on** the checkbutton.
deselect()	This method in the checkbutton widget is called **to turn off** the checkbutton.
toggle()	This method in the checkbutton widget is **used to toggle between** the different Checkbuttons.
flash()	This method in the checkbutton widget is used to **flashed between active and normal colors**.

Tkinter Checkbutton Widget Example

Below we have a basic example of this widget to gain a basic understanding of this method:

```
from tkinter import *
```

```
root = Tk()
root.geometry(“300x300”)
w = Label(root, text =‘StudyTonight’, fg=“Blue”,font = “100”)
w.pack()
Checkbutton1 = IntVar()
Checkbutton2 = IntVar()
Checkbutton3 = IntVar()
Button1 = Checkbutton(root, text = “Homepage”,
        variable = Checkbutton1,
        onvalue = 1,
        offvalue = 0,
        height = 2,
        width = 10)
Button2 = Checkbutton(root, text = “Tutorials”,
        variable = Checkbutton2,
        onvalue = 1,
        offvalue = 0,
        height = 2,
        width = 10)
Button3 = Checkbutton(root, text = “Contactus”,
        variable = Checkbutton3,
        onvalue = 1,
        offvalue = 0,
        height = 2,
        width = 10)
Button1.pack()
Button2.pack()
Button3.pack()
mainloop()
```



As you can see in the code above, three IntVar() variables are created and then we have created three **checkbutton** with different text like **Homepage**, **Tutorials**, and **Contactus**.

So we have created here checkbuttons using some options like text, variable, onvalue, offvalue, height and width. You can try using more options.

Summary

So in this tutorial we learned how to create check boxes in a GUI application using Tkinter checkbutton widget.

PART 9: Python Tkinter Radiobutton Widget

In this tutorial, we will cover the Tkinter **Radiobutton widget** in Python, which is used when we want to add a radio button to the GUI of our application.

Tkinter **radiobutton widget** is used to implement multiple-choice **options** that are mainly created in user input forms.

- This widget offers **multiple selections to the user** and **allows the user to select only one option from the given ones**. Thus it is also known as implementing **one-of-many selection** in a Python Application.
- Also, **different methods** can also be associated with radiobutton.
- You can also **display multiple line text and images** on the radiobutton.
- Each radiobutton displays a **single value** for a Particular **variable**.
- You can also **keep a track of the user's selection of the radiobutton** because **it is associated with a single variable**

Tkinter Radiobutton Widget

The syntax of the Radiobutton widget is given below:

```
W = Radiobutton(master, options)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many options to change the **look of the radiobutton** and these options are written as **comma-separated key-value** pairs.

Tkinter Radiobutton Widget Methods:

Following are the various methods used with the Tkinter Radiobutton widgets:

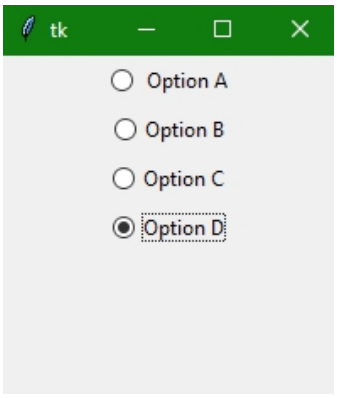
Method Name	Description
deselect()	This method is used to deselect or turns off the radio button
select()	This method is used to select the radio button
invoke()	This method is generally used to call a function when the state of radio button gets changed.
flash()	This method is generally used to flash the radio button between its normal and active colors many times.

Tkinter Radiobutton Widget Example

Below we have a basic example for the radio button widget. let us see the code snippet for the Radiobutton widget:

```
#firstly ImportTkinter module
from tkinter import *
from tkinter.ttk import *
# Creating parent Tkinter window
win = Tk()
win.geometry("200x200")
# let us create a Tkinter string variable
# that is able to store any string value
v = StringVar(win, "1")
# here is a Dictionary to create multiple buttons
options = {" Option A" : "1",
        "Option B" : "2",
        "Option C" : "3",
        "Option D" : "4"
        }
# We will use a Loop just to create multiple
# Radiobuttons instaed of creating each button separately
for (txt, val) in options.items():
        Radiobutton(win, text=txt, variable=v, value=val).pack(side = TOP, ipady =
4)
mainloop()
```

The above code will give the following output:



Note: If you will try above code snippet by yourself then you will see that in the output you can select only one button at a time.

Tkinter Radiobutton Widget Another Example

Below we have another example of this widget where we will add styling to radiobutton using style class:

```
from tkinter import *
```

```
from tkinter.ttk import *
win= Tk()
win.geometry('200x200')
v = StringVar(win, "1")
# we will add Style class to add style to Radiobutton
style = Style(win)
style.configure("TRadiobutton", background = "light blue",
        foreground = "orange", font = ("arial", 14, "bold"))
# Dictionary to create multiple buttons
values = {"RadioButton 1" : "1",
        "RadioButton 2" : "2",
        "RadioButton 3" : "3"
        }
for (text, value) in values.items():
        Radiobutton(win, text = text, variable = v,
        value = value).pack(side = TOP, ipady = 3)
mainloop()
```

The above code will change the font style as well as background and foreground colors. In the above TRadiobutton is used in style class, which automatically applies styling to all the available Radiobuttons.

Summary:

In this tutorial, we learned about the Radiobutton widget which is used to create multiple options out of which a single can be selected. This is mainly used when we create a user form, like registration form.

PART 10: Python Tkinter Entry Widget

In this tutorial, we will cover the **entry widget** of Tkinter in Python with its various options, and with the help of **Few Examples,** we will understand the concept in detail.

If you need to get a little bit of text from a user, like a name or an email address or a contact number then use an **Entry** widget.

- The Entry widget is mainly used to display a **small text box** that the user can type some text into.
- There are the number of options available to change the styling of the **Entry Widget.**
- It is important to note that the **Entry widget** is only used to get a **single-line text** from the user because in the case of **multiline text** the **text widget** will be used.
- This widget is mainly used **to accept text strings** from the user.

Tkinter Entry Widget

The syntax of the **entry widget** is given below:

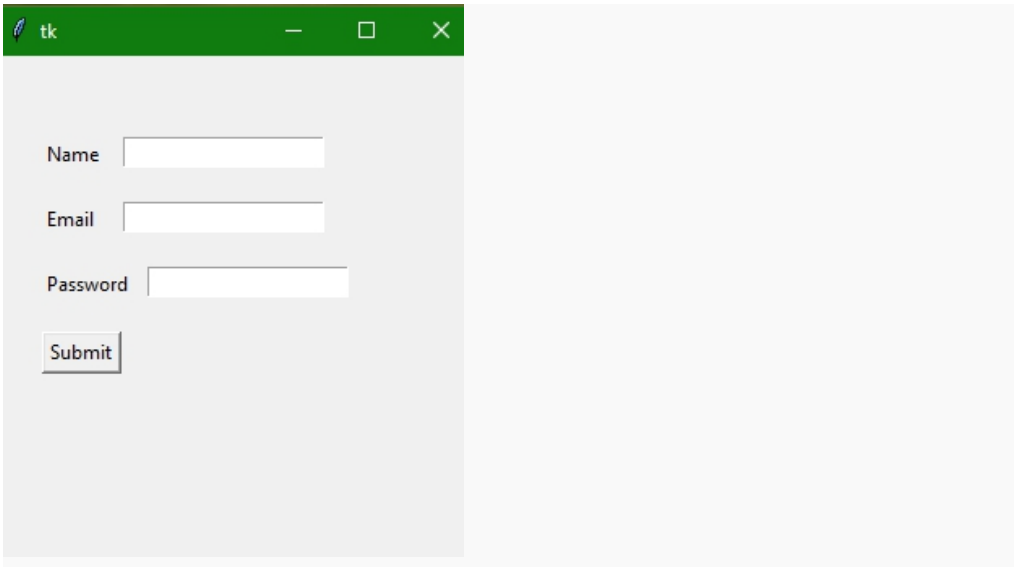
```
w = Entry(master, option=value)
```

In the above syntax, the master parameter denotes **the parent window.** You can use many options to change the styling of the **entry widget and these options are written as comma-separated.**

Tkinter Entry Widget Example

Below we have a basic example of the Tkinter Entry widget. Let us see the code snippet:

```
from tkinter import *
win = Tk()
win.geometry("400x250")
name = Label(win, text = "Name").place(x = 30,y = 50)
email = Label(win, text = "Email").place(x = 30, y = 90)
password = Label(win, text = "Password").place(x = 30, y = 130)
submitbtn = Button(win, text = "Submit",activebackground = "red", activeforeground = "blue")
            .place(x = 30, y = 170)
entry1 = Entry(win).place(x = 80, y = 50)
entry2 = Entry(win).place(x = 80, y = 90)
entry3 = Entry(win).place(x = 95, y = 130)
win.mainloop()
```



In the above code example, we have done the following:

1. Create Tkintor Labels to name the text input fields. For all the 3 text input fields(Entry widget), we have created three labels.

2. We have used the place() geometry manager to place the labels on the application window.
3. Then we have created a Button which is the submit button. And used the place() geometry manager to position it on the application GUI.
4. And finally, we have the three **Entry** widgets which will create the three text input fields. And used the place() geometry manager to position it on the application GUI.

Summary

In this tutorial, we learned how to use the Tkinter Entry widget to create a text input field while creating a GUI application.

PART 11: Python Tkinter Message Widget

In this tutorial, we will learn about the Tkinter **Message widget** in Python which is used to show some text message on the GUI application that you create using Tkinter.

The Tkinter **Message Widget** in Python is mainly used to **show some message** to the user who is using the application.

- The message displayed by the Tkinter **Message Widget** is of **non-editable type** and it can be **in multiline**.
- The **message** displayed by the Tkinter **Message widget** contains **single font text**.
- The **functionality of this widget** is very **similar to** the Tkinter Label widget, **but there is a difference** and that is the message widget can **automatically wrap the text**.

Tkinter Message Widget

The **syntax** of the Tkinter Message widget is given below:

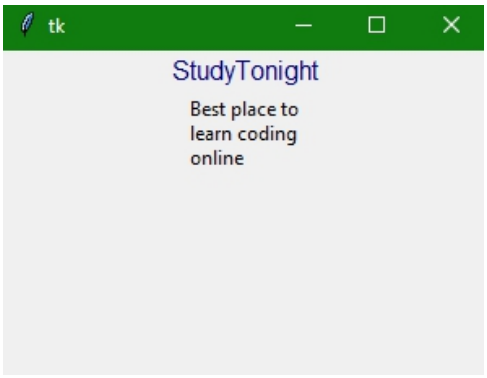
```
W = Message(master,options)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many options to change the **look of the message** and these options are written as **comma-separated key-value pair**.

Tkinter Message Widget Example

Below we have a basic example for the Tkinter Message widget:

```
from tkinter import *
win = Tk()
win.geometry("300x200")
w = Label(win, text ='StudyTonight', font = "90",fg="Navyblue")
w.pack()
msg = Message(win, text = "Best place to learn coding online")
msg.pack()
win.mainloop()
```



In the code example above, we have created a simple label widget and a message widget with some text message in it.

Summary:

So with this, we have covered the Tkineter Message Widget which is used to show message in Tkinter GUI application. We can also create message widget dynamically to show error or success message in Tkinter application.

PART 12: Python Tkinter Menu Widget

In this tutorial, we will cover the Tkinter **Menu widget** in Python which is used to create Menus with options for users to choose from.

The Tkinter **Menu widget** is used to **create different types of menus** in Python Application.

- The following types of menus can be created using the Tkinter Menu widget: **pop-up**, **pull-down**, **and top level**.
- Top-level menus are those **menus that are displayed just under the title bar** of the root or any other top-level windows. For example, all the websites have a top navigation menu just below the URL bar in the browser.
- Menus are commonly used to provide convenient access to options like **opening any file**, **quitting any task**, **and manipulating data in an application**.

Tkinter Menu Widget

The syntax of the Menu widget is given below:

```
W = Menu(master, options)
```

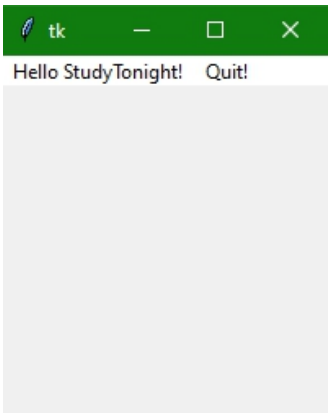
In the above syntax, the master parameter denotes **the parent window**. You can use many options to change the **look of the menu** and these options are written as **comma-separated key-value pairs**.

Tkinter Menu Widget Example

Below we have a basic example using this widget:

```
from tkinter import *
```

```
root = Tk()
def hello():
    print("hello!")
menubar = Menu(root)
menubar.add_command(label="Hello StudyTonight!",
command=hello)
menubar.add_command(label="Quit!", command=root.quit)
root.config(menu=menubar)
root.mainloop()
```



After running the above code, you will see the above output. Whenever you will click on **Hello StudyTonight!** menu item then it will print a **hello!** on your console. While on clicking the **Quit!** menu item, the application will close.

Summary:

In this tutorial, we learned how to create a menu for our Tkinter application and how to add menu items to it and perform some operation when the user selects any menu items.

PART 13: Python Tkinter Menubutton Widget

In this tutorial, we will cover the Tkinter **Menubutton widget** in Python which is used to create a **dropdown menu** which can be clicked by the user to see the

- This widget is used to provide **various types of menus** in the Python Application.
- It is important to note that **every Menubutton in an application** is associated with a **Menu widget** and that in return **can display the choices for that menubutton** whenever the user clicks on it.
- The Tkinter Menubutton widget provides **the user with an option** to select the **appropriate choice that exists** within the application.

Tkinter Menubutton Widget

The syntax of the Tkinter Menubutton widget is given below:

```
W = Menubutton(master, options)
```

In the above syntax, the master parameter denotes **the parent window.** You can use many options to change the **look of the menubuttons** and these options are written as **comma-separated key-value pairs**.

Tkinter Menubutton Widget Options:

Following are the various options used with Tkinter Menubutton widgets:

Option name	Description
activebackground	This option indicates the background color of the menubutton at the time **when the mouse hovers the menubutton**.
bd	This option is used to **represent the width of the border** in pixels. The default value is 2 pixels.
bitmap	This option will be set to the graphical content **which is to be displayed to the widget**.
bg	This option is used to **represent the background color of the widget**.
cursor	This option indicates the cursor **when the mouse hovers the menubutton**.
activeforeground	This option mainly **represents the font color** of the widget at the time **when the widget is under the focus**
fg	This option represents the **foreground color of the widget**.
direction	With the help of this option, you can specify the direction so that menu can be **displayed to the specified direction of the button**. You can Use **LEFT, RIGHT, or ABOVE** to place the widget accordingly.
disabledforeground	This option indicates the **text color of the widget when the widget is disabled**
height	This option indicates the height of the menubutton. This height indicates the **number of text lines** in the case of **text lines** and it indicates **the number of pixels** in the case **of images**.

Option name	Description
image	This option indicates the image displayed on the menubutton.
higlightcolor	This option indicates the highlight color when there is a focus on the button
justify	This option is used to indicate the way **by which the multiple text lines are represented.** For left justification, it is set to LEFT and it is set to RIGHT for the right justification, and CENTER for the center justification.
padx	This option indicates the additional padding of the widget in the **horizontal direction**.
pady	This option indicates the additional padding of the widget in the **vertical direction**.
menu	This option is used to **indicate the menu associated with the menubutton**
width	This option specifies the width of the widget. For textual buttons, It exists as a number of letters or for image buttons it indicates the pixels
Wraplength	In this case, if this option's value is **set to a positive number**, the text lines will be wrapped **in order to fit within this length**.
state	As the normal state of **menubutton is enabled**.It can be **set to disable to make the menubutton unresponsive**.
text	This option is used to indicate **the text on the widget**.

Option name	Description
textvariable	A control variable of class **StringVar** can be associated with this menubutton. If you will **set that control variable** then it will change the displayed text.
underline	This **option is mainly used to represent the index** of the character in the **text of the widget which is to be underlined**. The **indexing generally starts with zero in the text**.
relief	This option is used to **specify the border type**. Its default value is RAISED

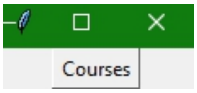
Tkinter Menubutton Widget Example

Now let us see a code snippet for the Tkinter Menubutton widget:

```
from tkinter import *
import tkinter
win = Tk()
mbtn = Menubutton(win, text="Courses", relief=RAISED)
mbtn.grid()
mbtn.menu = Menu(mbtn, tearoff = 0)
mbtn["menu"] = mbtn.menu
pythonVar = IntVar()
javaVar = IntVar()
phpVar = IntVar()
mbtn.menu.add_checkbutton(label="Python",
variable=pythonVar)
mbtn.menu.add_checkbutton(label="Java", variable=javaVar)
mbtn.menu.add_checkbutton(label="PHP", variable=phpVar)
```

```
mbtn.pack()
win.mainloop()
```

The output of the above code is as follows:



You can try the above example by yourself. Whenever you will execute the above code, you will see there is a Tkinter **menubutton** named **Courses** on a window whenever you will click on it then it will show you a **drop-down menu** like this:

Summary:

In this tutorial, we covered the Tkinter Menubutton widget which is used to create drop-down menus in a Tkinter application.

PART 14: Python Tkinter Frame Widget

The Tkinter **Frame widget** is used to **group and organize** the widgets in a better and friendly way. The **Frame widget** is basically a **container** (an invisible container) whose task is to hold **other widgets** and **arrange** them with respect to each other. The Tkinter frame widget makes up a **rectangular region** on the screen.

It basically acts as a **foundation class** which then **implements complex widgets**. It is like the HTML div tag, which is just to define divisions on a webpage in which we have other HTML elements.

Tkinter Frame Widget

The **syntax** of the frame widget is given below:

```
W = Frame(master, options)
```

In the above syntax, the master parameter denotes **the parent window.** You can use many options to change the **look of the frame** and these **options** are written as **comma-separated key-value pairs**.

Tkinter Frame Widget Example

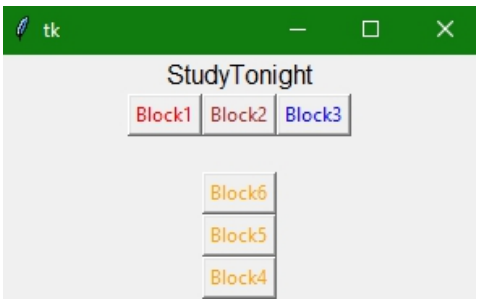
Below we have a basic example where we will organize different button widgets in a Frame widget. Let us see the code snippet given below:

```
from tkinter import *
```

```
root = Tk()
```

```
root.geometry("300x150")
```

```
w = Label(root, text ='StudyTonight', font = "80")
w.pack()
frame = Frame(root)
frame.pack()
bottomframe = Frame(root)
bottomframe.pack(side = BOTTOM)
button1 = Button(frame, text ="Block1", fg ="red")
button1.pack(side = LEFT)
button2 = Button(frame, text ="Block2", fg ="brown")
button2.pack(side = LEFT)
button3 = Button(frame, text ="Block3", fg ="blue")
button3.pack(side = LEFT)
button4 = Button(bottomframe, text ="Block4", fg ="orange")
button4.pack(side = BOTTOM)
button5 = Button(bottomframe, text ="Block5", fg ="orange")
button5.pack(side = BOTTOM)
button6 = Button(bottomframe, text ="Block6", fg ="orange")
button6.pack(side = BOTTOM)
root.mainloop()
```



In the code example above, we have created two frame widgets and then we have added 3 each button in those frames and have use Tkinter geometry manager to arrange the buttons inside the application window,

Summary

In this tutorial, we learned about Tkinter Frame widget and how we can use it to manage other Tkinter widgets and position them properly.

PART 15: Python Tkinter Canvas Widget

Tkinter Canvas widget is mainly used as a **general-purpose** widget which is **used to draw** anything on the application window in Tkinter.

- This widget is mainly used to draw graphics and **plots, drawings, charts, and showing images**.
- You can draw **several complex layouts** with the help of canvas, for example, **polygon**, **rectangle**, **oval**, **text**, **arc bitmap**, **graphics**, etc.
- Canvas widget is also used to **create graphical editors**.
- There are a number of options available to configure and control **the Canvas Widget**.

Tkinter Canvas Widget

The **syntax** of the **canvas widget** is given below:

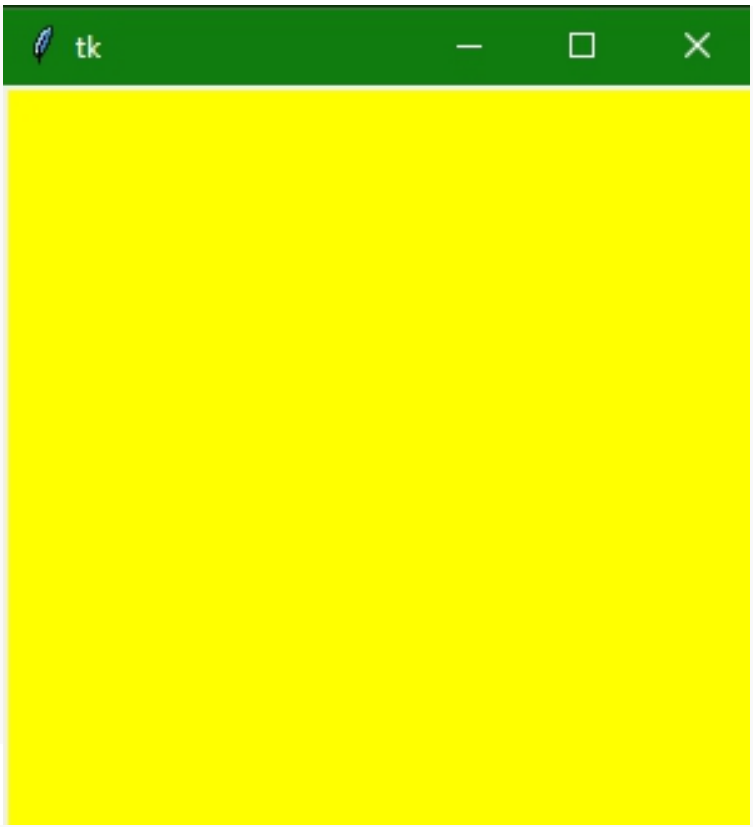
```
w = Canvas(master, option=value)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many options to change the layout of the canvas and these **options** are written **as comma-separated key-values**.

Tkinter Canvas Widget Basic Example

Let us create a simple canvas with the help of the canvas widget:

```
from tkinter import *
# window named top
top = Tk()
# set height and width of window
top.geometry("300x300")
#creating a simple canvas with canvas widget
cv = Canvas(top, bg = "yellow", height = "300")
cv.pack()
top.mainloop()
```

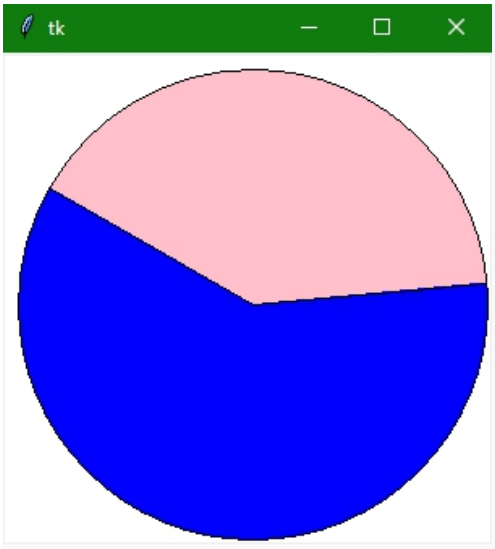


The above code will create a **simple canvas** with background color **yellow** and you can draw anything above it.

Tkinter Canvas Widget - Pie Chart using Arcs

Let us create a canvas and then an **arc** on it with the help of code snippet given below:

```
import tkinter
# init tk
root = tkinter.Tk()
# creating canvas
mCanvas = tkinter.Canvas(root, bg="white", height=300, width=300)
# drawing two arcs
coord = 10, 10, 300, 300
arc1 = mCanvas.create_arc(coord, start=0, extent=150, fill="pink")
arc2 = mCanvas.create_arc(coord, start=150, extent=215, fill="blue")
# adding canvas to window and display it
mCanvas.pack()
root.mainloop()
```



The above code will open a window, then **add a canvas, and then draws two arcs on it**. The two arcs drawn of **pink** and **blue** color together make up a circle as shown in the output above.

Summary:

In this tutorial, we learned about the Tkinter canvas widget which can be used to draw anything on the canvas, maybe a chart, and image, or some dynamic shape.

PART 16: Python Tkinter Listbox Widget

In this tutorial, we will cover the Tkinter **Listbox widget** in Python which is used to display **different types of items** to the user in form of a List inside a box and the user can select the items.

- The items contain the **same type of font** and the **same font color**.
- It is important to note here that **only text items** can be placed inside a **Listbox widget**.
- From this list of items, the **user can select one or more items** according to the requirements.

Tkinter Listbox Widget

The **syntax** of the Tkinter Listbox widget is given below:

```
W = Listbox(master, options)
```

In the above syntax, the master parameter denotes **the parent window.** You can use many options to change the **look of the ListBox** and these **options** are written as **comma-separated key-value pairs**.

Tkinter Listbox Widget Options:

Following are the various options used with Listbox widgets:

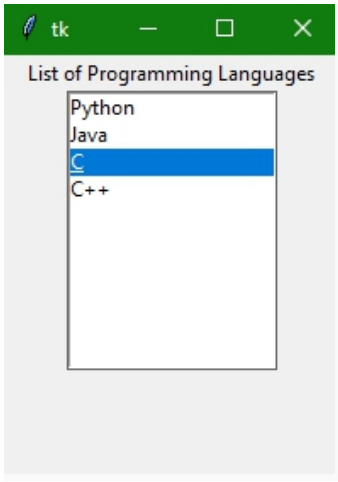
Name of Option	Description
bg	This option indicates the **background color** of the widget.
bd	This option is used to represent the **size of the border**. The default value is 2 pixels.
cursor	With the help of this option, the **mouse pointer will look** like the cursor type like dot, arrow, etc.
font	This option indicates the **font type of the Listbox items**.
fg	This option indicates the **color of the text**.
height	This option is used to represents the **count of the lines** shown in the Listbox. The default value of this option is 10.
highlightcolor	This option is used to **indicate the color of the Listbox items** when the widget is **under focus**.
highlightthickness	This option is used to indicate the **thickness of the highlight**.
relief	This option indicates the type of border. The default value is **SUNKEN**.

Name of Option	Description
selectbackground	This option is used to indicate the **background color** that is used to display the **selected text**.
selectmode	This option is used to determine the number of items that can be selected from the list. It can set to **BROWSE**, **SINGLE**, **MULTIPLE**, **EXTENDED**.
width	This option is used to represent the **width of the widget** in characters.
xscrollcommand	This option is used to let the **user scroll the Listbox horizontally**.
yscrollcommand	This option is used to let the **user scroll the Listbox vertically**.

Tkinter ListBox Widget Example

Below we have a basic example using this widget:

```
from tkinter import *
top = Tk()
top.geometry("200x250")
lbl = Label(top, text="List of Programming Languages")
listbox = Listbox(top)
listbox.insert(1,"Python")
listbox.insert(2, "Java")
listbox.insert(3, "C")
listbox.insert(4, "C++")
lbl.pack()
listbox.pack()
top.mainloop()
```



In the code example above, we have created a simple Listbox widget with some items along with specifying the top list item which is the heading on the Listbox widget.

Summary:

In this tutorial, we learned about the Tkinter Listbox Widget, its basic syntax, the commonly used methods for the ListBox Widget, and a code example.

PART 17: Python Tkinter Scrollbar Widget

In this tutorial, we will cover the Tkinter **Scrollbar widget** in Python, using which we can add a scrollbar to the user interface of our Tkinter application.

To scroll **up or down or right or left the content** in a Python desktop application, the Tkinter **Scrollbar widget** is used.

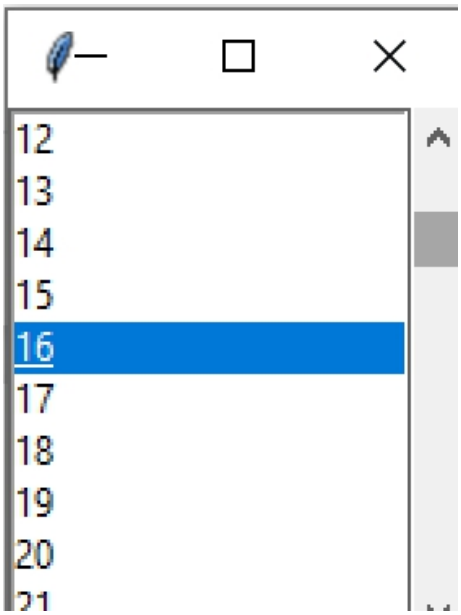
- To **scroll** the content of other widgets like **Listbox**, **canvas**, etc we use this widget.
- **Both Horizontal and Vertical scrollbars** can be created in the Trinket Entry widget.

Below we have an image showing scrollbar widget used with a Listbox:

Name of the Option	Description
activebackground	This option represents **the background color of the widget when it is under focus**.
bg	This option represents the **background color of the widget**
bd	This option **represents the border size of the widget**. The default value is 2 pixels.
cursor	With the help of this option, **the mouse pointer will be changed to a specific cursor type and it can be an arrow, dot, etc**.
command	This option will be **set to the procedure associated which is called** every time the scrollbar is moved.

Name of the Option	Description
elementborderwidth	This option mainly represents the border width around the arrowheads and the slider. The default value of this option is -1.
highlightthickness	This option **represents the thickness** of the focus highlights
highlightbackground	This **option indicates the highlight color** when the **widget is not under the focus**
highlightcolor	This **option indicates the highlight color** when the **widget is under the focus**
jump	This option is **used to control the behavior of the scroll jump**. If this option is set to 1, then the callback is called at the time **when the user releases the mouse button**.
orient	This option can be set to either **horizontal or vertical depending** upon the orientation of the scrollbar.
width	This option **represents the width of the scrollbar**.
troughcolor	This option is used **to set the color for the trough**
takefocus	By default, you **can tab the focus through this widget**. If you don't want this behavior you can set this option to 0.
repeatdelay	This option is mainly used to **tell the duration up to which the button is to be pressed** before the slider starts moving **in that direction repeatedly**. its default value is **300 ms**

Name of the Option	Description
repeatinterval	The default value of **this option is 100**



Tkinter Scrollbar Widget

The **syntax** of the **Scrollbar widget** is given below:

```
W = Scrollbar(master, options)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many **options** to configure your scrollbar widget and these **options** are written as **comma-separated key-value pairs**.

Tkinter Scrollbar Widget Options:

Following are the various options used with Tkinter Scrollbar widgets:

Tkinter Scrollbar Widget Methods:

Few methods used with Tkinter Scrollbar widgets are:

- get():

This method returns the two numbers
suppose a and b which **represents the current position of**

the scrollbar.

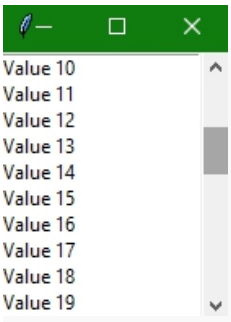
- set(first, last):

This method is used to connect the scrollbar to any other widget. That is yscrollcommand or xscrollcommand of the other widget to this method.

Tkinter Scrollbar Widget Example

Below we have a basic example of a scrollbar widget.

```
from tkinter import *
win= Tk()
sbb = Scrollbar(win)
sbb.pack(side = RIGHT, fill = Y)
mylist = Listbox(win, yscrollcommand = sbb.set)
for line in range(45):
    mylist.insert(END, "Value " + str(line))
mylist.pack(side = LEFT)
sbb.config(command = mylist.yview)
mainloop()
```



As you can see in the above code, we have created a Listbox widget with numbers as list items in it. Then we have created a Scrollbar widget and have used the yscrollcommand option of the Listbox widget to set the Scrollbar widget with it. We have used the Scrollbar widget's set function here.

Summary:

So we have covered the Tkinter Scrollbar widget. It is very useful where we have a Tkinter application with long lists or some widget which is too long to fit in the application window. Then we can use the Scrollbar widget for such applications.

PART 18: Python Tkinter Scale Widget

In this tutorial, we will cover the Tkinter **Scale widget** in Python which is used to add a **graphical slider** object which the user can slide and choose a number, as a numeric value is attached to this slider scale and as you move the slider up/down or right/left the numeric value attached to it increases or decreases and you can set the slider to the value you wish to select.

- The **sliding bar** provided by **the scale widget** is helpful in **selecting the values just by sliding** from **left to right** or **top to bottom** depending upon the **orientation of the sliding bar** in our application.
- The **scale widget** is used as an **alternative to the Entry widget** if the purpose of the Entry widget is to take numeric input from user within a given range of values.
- You can also **control minimum and maximum values** along with the resolution of the scale.

Tkinter Scale Widget

The **syntax** of the Tkinter **Scale widget** is given below:

```
W = Scale(master, options)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many options to change the layout of the scale widget and these **options** are written **as comma-separated key-values**.

Tkinter Scale Widget Methods

Following are the few methods used with Scale widgets:

- get():

This method is used to get the current value of the scale.

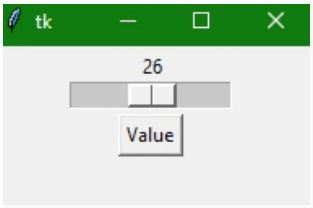
- set(value):

This method is used to set the value of the scale.

Tkinter Scale Widget - Horizontal Example

Below we have a basic example where we will **create a horizontal slide bar**.

```
from tkinter import *
win = Tk()
win.geometry("200x100")
v = DoubleVar()
scale = Scale( win, variable=v, from_=1, to=50, orient=HORIZONTAL)
scale.pack(anchor=CENTER)
btn = Button(win, text="Value")
btn.pack(anchor=CENTER)
label = Label(win)
label.pack()
win.mainloop()
```

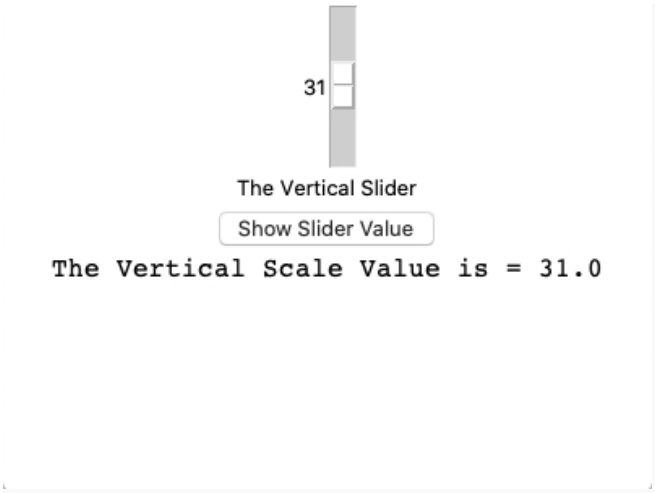


In this tutorial, we have created a horizontal Scale widget. If you see in the code, we have specified the orient as **HORIZONTAL** for this. We have also specified the **range of numeric values** for the slider scale.

Tkinter Scale Widget - Vertical Example

Below we have another example where we will create a **vertical slider**:

```
from tkinter import *
win = Tk()
win.geometry("400x300")
v = DoubleVar()
def show():
        sel = "The Vertical Scale Value is = " + str(v.get())
    # adding scale value to label to show
        scale_val.config(text=sel, font=("Courier", 16))
scl = Scale(win, variable=v, from_=60, to=1,
orient=VERTICAL)
mainlabel = Label(win, text="The Vertical Slider")
btn = Button(win, text ="Show Slider Value",
        command = show,
        bg = "darkblue",
        fg = "white")
# creating another label to show the scale value
scale_val = Label(win)
scl.pack(anchor = CENTER)
mainlabel.pack()
btn.pack()
scale_val.pack()
win.mainloop()
```



You can move the slider from bottom to top as it is a vertical slider. In the given example, we have also **added a button to our application**, and we have defined a function show() that is attached to the button widget as an **event handler**. So after the user uses the slider to select any value, and then clicks on the button, the value will be displayed in a Label widget below the button.

Summary:

In this tutorial, we learned about the Tkinter Scale widget which is a good UI component for taking numerical input value within a specific range from the end user.

PART 19: Python Tkinter Toplevel Widget

In this tutorial, we will cover the Tkinter **Toplevel widget** in Python which is used to **create and display** top-level windows other than the application window.

- With the help of the Tkinter **Toplevel widget**, you can provide **extra information** to the user in a separate window on top of the parent window.
- This **top-level window** created using the **Toplevel widget** is directly organized and managed **by the window manager**.
- It is not necessary for the **top-level windows** to have parents on their top.
- You can create **multiple top-level windows** one over the other.
- Top-level windows created using **Top-level widgets** contain **title bars**, **borders**, **and some window decorations too**.
- With the help of this widget, you can provide **pop-ups**, **some extra information**, **or some widgets** on the new window if you want.

Python Tkinter Toplevel Widget

The syntax of the Tkinter Toplevel widget is given below:

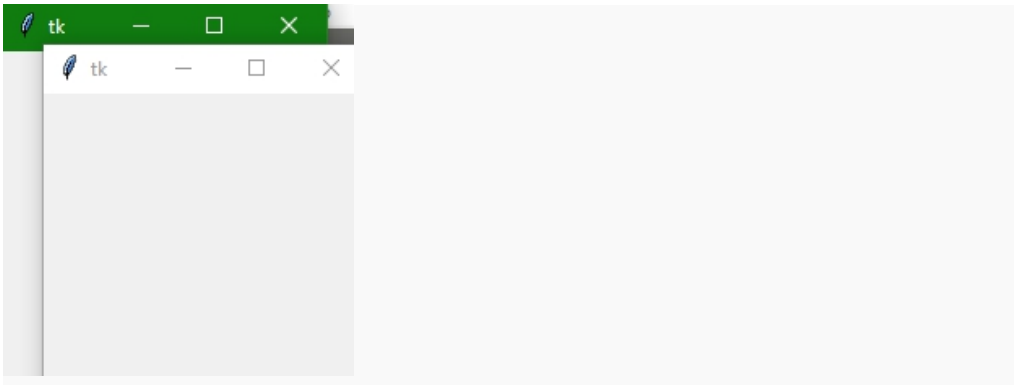
```
W = Toplevel(master,options)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many **options** to configure your Toplevel widget and these **options** are written as **comma-separated key-value pairs**.

Tkinter Toplevel Widget Example

Below we have a basic example where we will create a simple top-level window.

```
from tkinter import *
win = Tk()
win.geometry("200x200")
def open():
    top = Toplevel(win)
    top.mainloop()
btn = Button(win, text="open", command=open)
btn.place(x=75, y=50)
win.mainloop()
```



In the code above, we have created a Toplevel widget that is created and started when the Button is clicked.

Summary:

So now we know what a Tkinter Toplevel widget is and how to create it. The Tkinter Toplevel widget is good to show some section of your application in a different window which is displayed on the top of the main application window.

PART 20: Python Tkinter Spinbox Widget

In this tutorial, we will cover Tkinter **Spinbox widget** in Python with its syntax and few examples. The Spinbox widget in Tkinter in Python is used to **select a value from the specified given range of values**.

It is different from the **Tkinter Scale widget** (Scale widget being more stylish) in terms of style, but more or less, fulfils the same purpose.

For example, when you want to have a dropdown of numerical values like year of birth (from 1950 to 2020) or a dropdown for user to choose their age, we can use the Tkinter Spinbox widget.

- This widget is **an alternative to Entry widget**, when we want user to enter a numeric value within a specific range.
- This widget is used only in the case **where users need to chose from a given range of choices**.

Tkinter Spinbox Widget

The syntax of the **Spinbox widget** is given below:

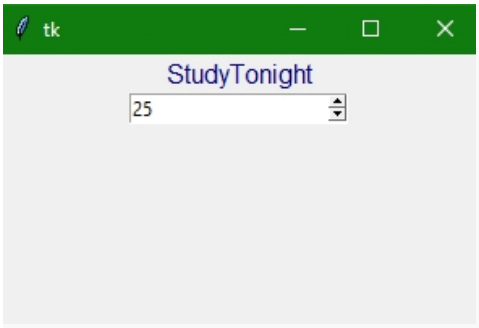
```
w = Spinbox(master, option=value)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many **options** to configure your spinbox widget and these **options** are written as **comma-separated key-value pairs**.

Tkinter Spinbox Widget Example

Below we have a basic example of the Spinbox widget. Let us see the code snippet given below:

```
from tkinter import *
win = Tk()
win.geometry("300x200")
w = Label(win, text ='StudyTonight', fg="navyblue",font = "50")
w.pack()
sp = Spinbox(win, from_ = 0, to = 50)
sp.pack()
win.mainloop()
```



In the above code, we created a simple application window, with a Label widget and a Spinbox widget with range from **0** to **50**.

Summary:

So this how a TKinter Spinbox widget works. It is used for creating input for a specified range of numeric values, which user can select. If we have a defined range of numbers then its better to use the Spinbox widget rather than using a Tkinter Entry widget.

PART 21: Python Tkinter LabelFrame Widget

In this tutorial, we will cover the Tkinter **LabelFrame widget** in Python with its syntax and few examples. The LabelFrame widget is mainly used **to draw borders around the child widgets**.

- This widget is a **bordered container widget** and is **used to group the related widgets** in a Tkinter application to provide a better user experience to the user.
- For example, we can group the radiobutton widgets used in an application using the labelframe widget.
- One can also **add a title** for the LabelFrame widget(we will see this in the code example).
- The **LabelFrame widget** is **simply a variant** of the **Frame widget** and it has all the features of a frame.

Note: If you have used HTML for web development, then the labelframe is just like HTML fieldset tag.

Tkinter LabelFrame Widget

The syntax of the **LabelFrame widget** is given below. Let us see:

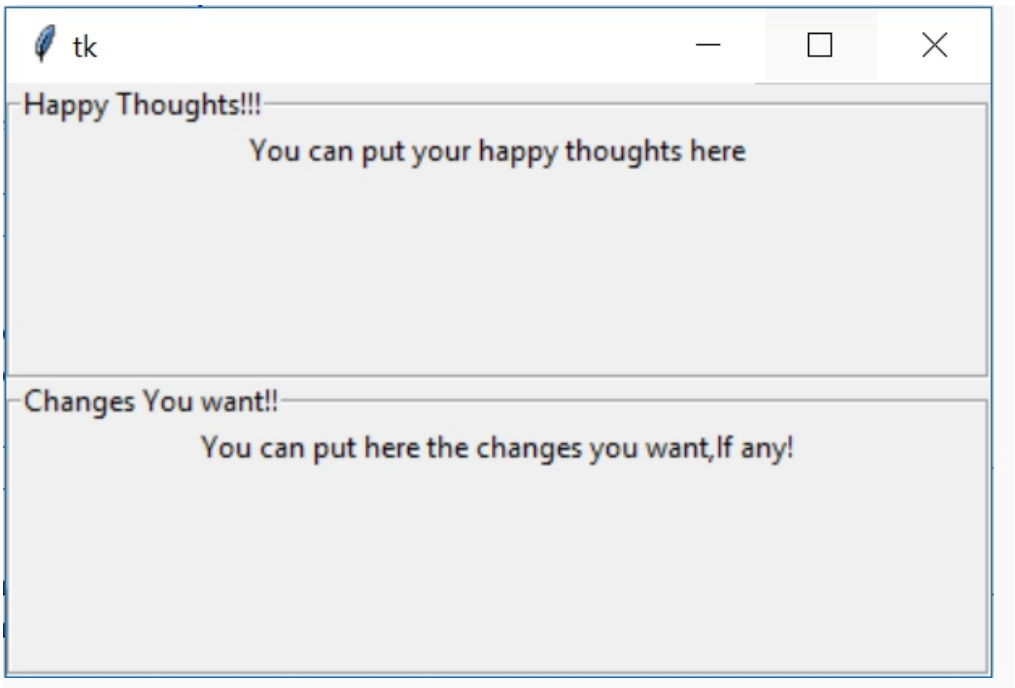
```
w = LabelFrame(master, option=value)
```

In the above syntax, the master parameter denotes **the parent window.** You can use many options to configure the labelframe and these options are written as **comma-separated key-value pairs**.

Tkinter LabelFrame Widget Example

Below we have a basic example of the LabelFrame widget. Let us see the code snippet given below:

```
from tkinter import *
win = Tk()
win.geometry("300x200")
labelframe1 = LabelFrame(win, text="Happy Thoughts!!!")
labelframe1.pack(fill="both", expand="yes")
toplabel = Label(labelframe1, text="You can put your happy thoughts here")
toplabel.pack()
labelframe2 = LabelFrame(win, text = "Changes You want!!")
labelframe2.pack(fill="both", expand = "yes")
bottomlabel = Label(labelframe2, text = "You can put here the changes you want,If any!")
bottomlabel.pack()
win.mainloop()
```



As you can see in the output above, we have created two labelframe widgets, in which we have added text for the labelframe widgets, and inside the labelframe widget we have one label widget. We can have as many widgets inside the labelframe widget, as we want.

Summary:

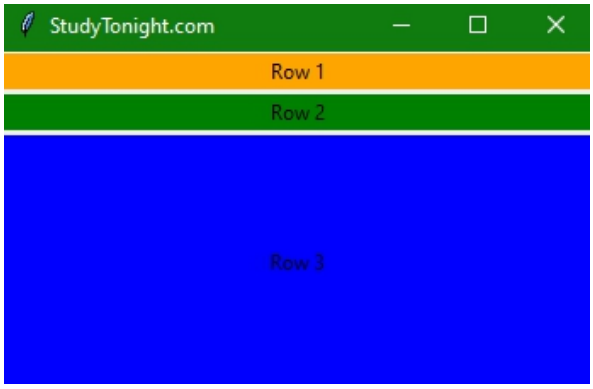
So with this we have completed the Tkinter labelframe widget which is just like HTML fieldset tag, if you know HTML. The labelframe widget is used to create a border around other widgets to group other widgets present in your application.

PART 22: Python Tkinter PanedWindow Widget

In this tutorial, we will cover the Tkinter **PanedWindow widget** which is mainly a **container widget containing one or more than one child widgets** which are also known as **Panes**.

- This widget arranges child widgets either in a **vertical** or in a **horizontal manner**.
- It is also known as the **Geometry Manager** widget.
- This widget is used to **implement different layouts** in a Python desktop application created using the Tkinter module.
- The **child widgets** inside the **PanedWindow widget** can be **resized by the user** by moving separator lines **sashes** using the mouse.
- You can implement **multiple panes** using the **PanedWindow widget**.

Here is a simple Tkinter application window with three widgets stacked vertically inside a PanedWindow widget.



Tkinter PanedWindow Widget

The **syntax** of the PanedWindow widget is given below:

```
W = PanedWindow(master, options)
```

In the above syntax, the master parameter denotes **the parent window**. You can use many options to change the **look of the PanedWindow and these options are written as comma-separated**.

Tkinter PanedWindow Widget Methods:

Following are some methods used with PanedWindow widget:

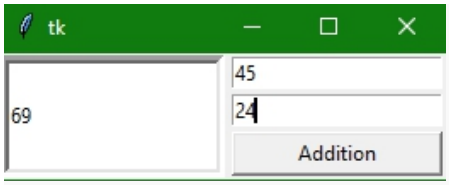
Method Name	Description
config(options)	This method is mainly used to configure any widget with some specified options.

Method Name	Description
get(startindex,endindex)	This method is used to get the text at the specified given range.
add(child,options)	This method is used to add a window to a parent window.

Tkinter PanedWindow Widget Example

Below we have a basic example for the understanding of the PanedWindow widget. Let us see the code snippet given below:

```
from tkinter import *
# event handler for button
def addition():
    x = int(e1.get())
    y = int(e2.get())
    leftdata = str(x+y)
    leftinput.insert(1, leftdata)
# first paned window
w1 = PanedWindow()
w1.pack(fill=BOTH, expand=1)
leftinput = Entry(w1, bd=5)
w1.add(leftinput)
# second paned window
w2 = PanedWindow(w1, orient=VERTICAL)
w1.add(w2)
e1 = Entry(w2)
e2 = Entry(w2)
w2.add(e1)
w2.add(e2)
bottomBtn = Button(w2, text="Addition", command=addition)
w2.add(bottomBtn)
mainloop()
```



As you can see above, in the output, we have an application window, in which we have 3 tkinter Entry widgets and 1 tkinter button widget, stacked using 2 PanedWindow widgets **packed vertically** besides each other.

If you will provide, two numbers in the right side entry widgets and then click on the Addition button, the result of addition of the numbers in the right, will be shown in the

entry widget on the left hand side.

Tkinter PanedWindow Widget - Multiple Panes Example

Let us see another code snippet of this widget given below:

```
from tkinter import *
from tkinter import tk
win = Tk()
pw = PanedWindow(orient =‘vertical’)
#creating Button widget
top = tk.Button(pw, text =“Just Click Me!!!\nI am a Button”)
top.pack(side=TOP)
#Adding button widget to the panedwindow
pw.add(top)
#Creating Checkbutton Widget
bot = Checkbutton(pw, text=“I am Checkbutton Choose Me!”)
bot.pack(side=TOP)
pw.add(bot)
label = Label(pw, text=“I am a Label”)
label.pack(side=TOP)
pw.add(label)
string = StringVar()
entry = Entry(pw, textvariable=string, font=(‘arial’, 15, ‘bold’))
entry.pack()
# This is used to force focus on Particular widget
# that means widget is already selected for some operations
entry.focus_force()
pw.add(entry)
pw.pack(fill = BOTH, expand = True)
# To show sash
pw.configure(sashrelief = RAISED)
mainloop()
```

In the above code example, we have create multiple widgets inside a panedwindow widget. We have also used StringVar() variable and we have used the focus_force() function to have entry widget in focus when the application is loaded.

Summary:

In this tutorial, we learned about PanedWindow widget which is a great widget if you want to create multi-column grid like arrangement of widgets in your application.

PART 23: Python Tkinter Text Widget

In this tutorial, we will cover the Tkinter **Text widget** in Python. If you want a **text-editor** in your desktop application then you can use the Tkinter **Text widget**.

- The **text widget** is used to provide a **multiline textbox** (input box) because in Tkinter **single-line textbox** is provided using **Entry widget**.
- You can use various **styles and attributes** with the Text widget.
- You can also use **marks and tabs** in the Text widget **to locate the specific sections of the text**.
- Media files like **images and links** can also be inserted in the Text Widget.
- There are some variety of applications where you need multiline text like **sending messages** or taking **long inputs** from users, or to **show editable long format text** content in application, etc. use cases are fulfilled by this widget.
- Thus in order **to show textual information**, we will use the Text widget.

Tkinter Text Widget

The syntax of the Text widget is given below:

```
W = Text(master, options)
```

In the above syntax, the master parameter denotes **the parent window.** You can use many options to configure the text editor and these options are written as **comma-separated key-value pairs**.

Tkinter Text Widget Methods:

Some methods used with the text widget are given below:

Method	Description
index(index)	This method is used to get the specified index.
see(index)	This method returns true or false on the basis that if the string is visible or not at the specified index.
insert(index,string)	This method is used to insert a string at the specified index.
get(startindex,endindex)	This method returns the characters in the specified range

Method	Description
delete(startindex,endindex)	This method deletes the characters in the specified range

Methods for Tag Handling

Mainly tags are used to **configure different areas of the text widget** separately. Tag is basically the name given to separate areas of the text. Some Methods for handling tags are given below:

- tag_config()

To **configure the properties** of the tag this method will be used.

- tag_add(tagname, startindex, endindex)

This method is mainly used **to tag the string that is present at the specified index**.

- tag_delete(tagname)

This method is mainly **used to delete the specified tag**.

- tag_remove(tagname, startindex, endindex)

To **remove the tag from the specified range this method is used**.

Methods for Mark Handling

In a given text widget in order to **bookmark specified positions** between the characters **Mark** is used. Some methods for the same are given below:

- index(mark)

This method is mainly used to get the index of the mark specified.

- mark_names()

This method is used to get all the names of the mark in the range in the text widget.

- mark_gravity(mark, gravity)

To get the gravity of the given mark this method will be used.

- mark_set(mark, index)

This method is used to inform the new position of the given mark.

- mark_unset(mark)

In order to remove the given mark from the text this method will be used.

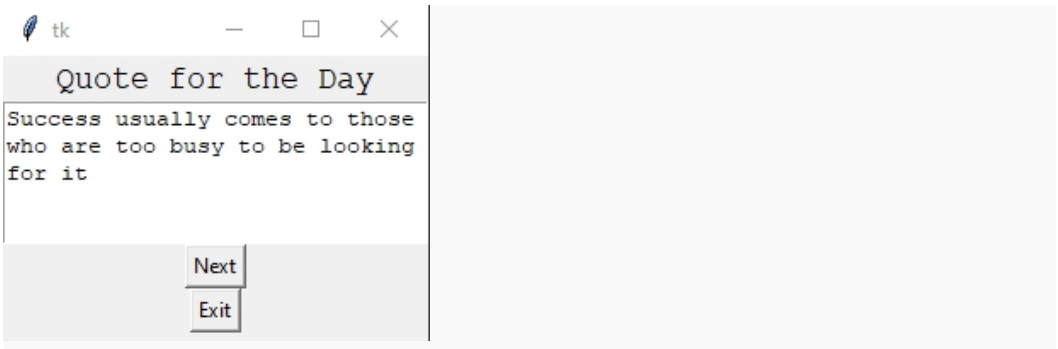
Tkinter Text Widget Example

Let us discuss a basic example for the text widget. The code snippet for the example of the text widget is given below:

```
import tkinter as tk
```

```
from tkinter import *
win = Tk()
#to specify size of window.
win.geometry("250x170")
# To Create a text widget and specify size.
T = Text(win, height = 6, width = 53)
# TO Create label
l = Label(win, text = "Quote for the Day")
l.config(font =("Courier", 14))
Quote = """Success usually comes to those who are too busy to be looking for it"""
# Create a button for the next text.
b1 = Button(win, text = "Next", )
# Create an Exit button.
b2 = Button(win, text = "Exit",
            command = win.destroy)
l.pack()
T.pack()
b1.pack()
b2.pack()
# Insert the Quote
T.insert(tk.END, Quote)
tk.mainloop()
```

The output for the code snippet given above is as follows:



If you want to destroy this window just click on the **Exit** button.

Summary:

In this tutorial, we learned Tkinter Text widget which is used when we want to add a textarea in our application for taking large user inputs.

PART 24: Python Tkinter MessageBox

In this tutorial, we will cover how to create and use **Tkinter MessageBox** while developing desktop applications.

In order **to display message boxes** in a desktop application, we use the **MessageBox** module in **Tkinter**.

- There are **various functions** present in this module which helps to **provide an appropriate type of message** according to **the requirement**.
- With the help of this module, we can create **pop-up message boxes to take user input**.
- The functions of the MessageBox module are as follows: showError(), askretrycancel(), showwarning(), etc., all of which are used to create a messagebox.

Tkinter MessageBox

To use the messagebox module, we first need to import it in our python script:

```
from tkinter import messagebox
```

Then following is the **basic syntax** to use the messagebox:

```
messagebox.function_name(title, message [, options])
```

In the above syntax, we have used the following:

- **function_name**

This is used to indicate the name of the appropriate MessageBox Function.

- **title**

This is used to indicate the text to be displayed in the title bar of the appropriate message box.

- **message**

This is used to indicate the text to be displayed as a message in the message box.

- **options**

It is used to indicate various options in order to configure the MessageBox. There are two values of it and these are **default** and **parent**.

  - **default:** It is used to specify a default button such as ABORT, RETRY, IGNORE.
  - **parent:** It is used to specify a window on the top of which we will display the MessageBox.

The functions present in the **MessageBox module** uses the same syntax but the functionalities of each function are different.

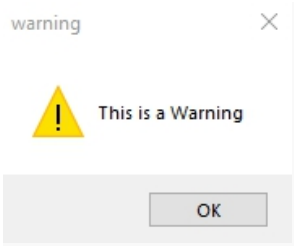
Let us see a few functions of the Tkinter MessageBox module.

Tkinter MessageBox - showwarning()

This method is used to **display any warning** to the user in a Python application.

Here is a code for the same:

```
from tkinter import *
from tkinter import messagebox
win = Tk()
win.geometry("200x200")
messagebox.showwarning("warning","This is a Warning")
win.mainloop()
```

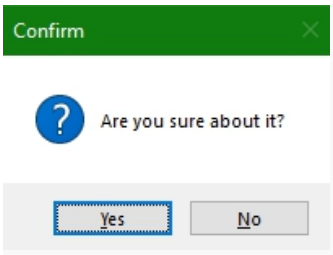


Tkinter MessageBox - askquestion()

This method in MessageBox is mainly used to **ask some question to the user** which can be answered either in **Yes** or **No**.

The code for this method is as follows:

```
from tkinter import *
from tkinter import messagebox
win = Tk()
win.geometry("100x100")
messagebox.askquestion("Confirm","Are you sure about it?")
win.mainloop()
```



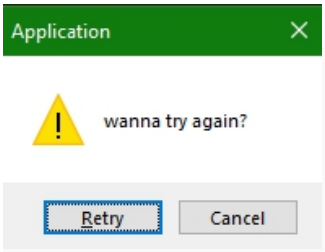
Tkinter MessageBox - askretrycancel()

If you want to **ask a user to do any Particular task** or not then this method will be used.

Let us see the code for the same:

```
from tkinter import *
from tkinter import messagebox
win= Tk()
win.geometry("100x100")
```

```
messagebox.askretrycancel("Application"," wanna try again?")
win.mainloop()
```

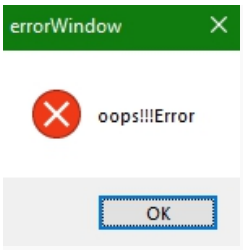


Tkinter MessageBox - showerror()

To display an **error message** this method will be used.

Let us see the code snippet given below:

```
from tkinter import *
from tkinter import messagebox
top = Tk()
top.geometry("100x100")
messagebox.showerror("errorWindow","oops!!!Error")
top.mainloop()
```



Summary:

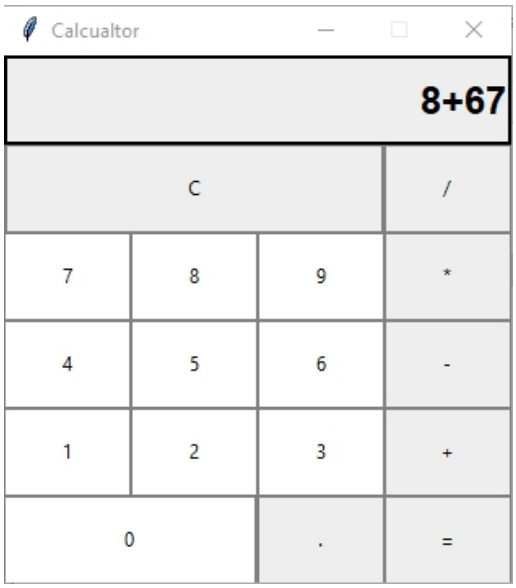
In this tutorial, we learned about the different types of message boxes that we can create to show information to user or to take input from users, like to confirm any action, etc. The message boxes are like pop-ups in which we can show errors, warnings, etc. to the user.

PART 25: Calculator Application using Tkinter (Python Project)

In this tutorial, we will cover **how to create a simple calculator app using Python** Tkinter.

As in our previous tutorials, we have covered how to create tkinter buttons, tkinter labels, tkinter entry, tkinter frames and tkinter checkbuttons, and many more. Now with the help of all the widgets discussed in previous sections, we are going to create a Calculator App using Tkinter.

Here is how our calculator will look, which is made by using the input field, buttons and for the calculation purpose we will use logic in our code defined in functions, like if you want to add two numbers then behind this there must be a logic for addition purpose, similarly for substraction, multiplication, etc, we have created functions whose task is to perform these operations.



We have an **Input Field** in which the user input will be shown and the final result of the calculation will be displayed.

And the buttons are like 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, =, ., and C(clear button)

What is a Calculator?

For those who do not know, a **calculator** is basically a program on a **computer that simulates** the behavior of any **hand-held calculator** useful for **performing Mathematical Calculations**. It is a very basic device used in our everyday lives. Now all the smartphones also have a Calculator application in them.

While creating any GUI Application there are mainly two steps:

- The first step is to **create a User Interface**.
- The second step is the most important one and in this, to **add functionalities to the GUI**

Now let's begin with **creating a simple calculator app** using Tkinter in Python which is used for **basic arithmetic calculations**.

Calculator App Code

Now it's time to take a look at the code to create a Calculator App using Tkinter:

```
from tkinter import *

win = Tk() # This is to create a basic window

win.geometry("312x324")  # this is for the size of the window

win.resizable(0, 0)  # this is to prevent from resizing the window

win.title("Calculator")

######################Starting with functions ######################

# 'btn_click' function :
```

```
# This Function continuously updates the
# input field whenever you enter a number
def btn_click(item):
    global expression
    expression = expression + str(item)
    input_text.set(expression)
# ‘bt_clear’ function :This is used to clear
# the input field
def bt_clear():
    global expression
    expression = ””
    input_text.set(””)
# ‘bt_equal’:This method calculates the expression
# present in input field
def bt_equal():
    global expression
    result = str(eval(expression)) # ‘eval’:This function is used to evaluates the string expression directly
    input_text.set(result)
    expression = ””
expression = ””
# ‘StringVar()’ :It is used to get the instance of input field
input_text = StringVar()
# Let us creating a frame for the input field
input_frame = Frame(win, width=312, height=50, bd=0, highlightbackground=“black”, highlightcolor=“black”, highlightthickness=2)
input_frame.pack(side=TOP)
#Let us create 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

#Let us creating another 'Frame' for the button below the 'input_frame'

btns_frame = Frame(win, 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: bt_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: bt_equal()).grid(row = 4, column = 3, padx = 1, pady = 1)
```

```
win.mainloop()
```

There are a **variety of functions in Tkinter** with the help of them **it becomes easy** and **convenient to make a simple calculator** just with **this little code**.

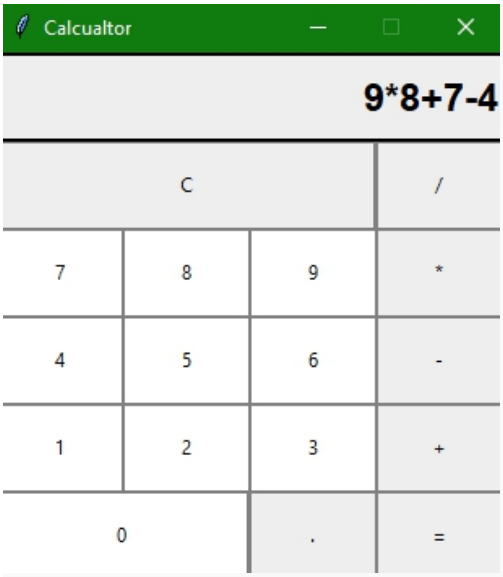
APart from the Tkinter widgets, we have defined the following functions in our code:

btn_click() **Function:** This function handles the button click on various numeric buttons to add them to the operation.

bt_clear() **Function:** This function is used to handle the clear operation to clean the previous input in the Calculator application.

bt_equal() **Function:** This function is used to handle the equal button to execute the operation and show the result.

Now we will show you a snapshot as the output of the above code. And yes you can implement it on your system for more clear understanding of Calculator App Using Tkinter:



Summary:

In this tutorial, we developed a basic Calculator application using Tkinter and various widgets of Tkinter about which we have covered in our Tkinter Tutorial. Click on Next to see more Apps developed using Tkinter as this will help you Part what you have learned.

PART 26: Text Editor Application Using Tkinter (Python Project)

In this tutorial, we will help you to build a simple Text Editor Application using Tkinter which is a very good beginner project for Tkinter.

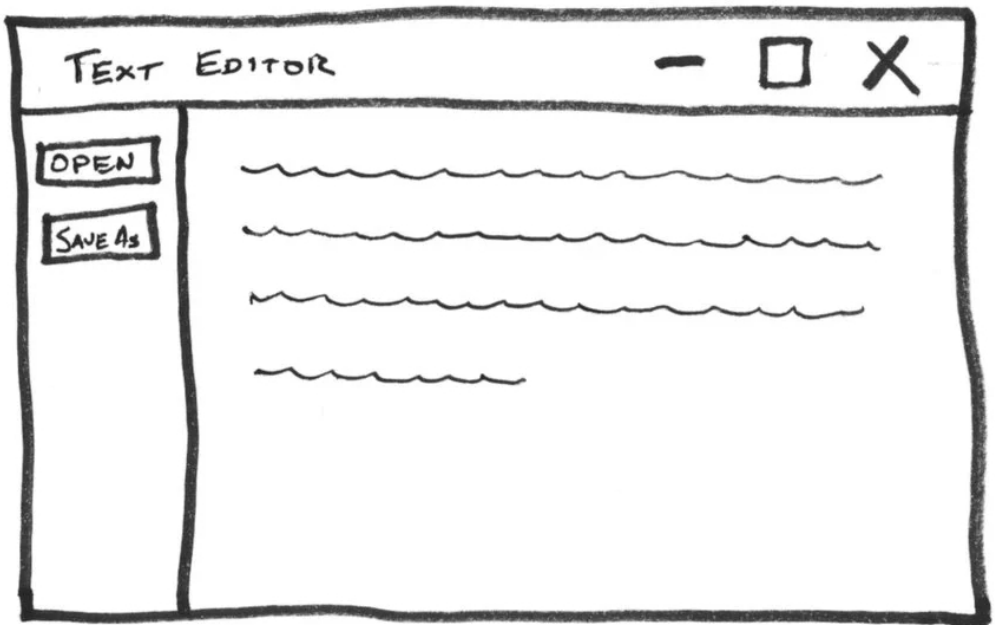
Text Editor Application is an application where you can write your text, open any text file, you can edit any text file and you can also save a file if you want. In this tutorial, we will build a Text Editor Application from scratch.

Essential Elements for the Text editor application are as follows:

- There is a **Button widget** called **btn_open** that is used for opening a file for editing
- Second one is a Button widget called **btn_save** for saving a file
- Third, there is a **Text widget** called **txt_edit** for creating and editing any text file.

The arrangement of three widgets is done in a way such that the two buttons are on the left-hand side of the window, and the text box is on the right-hand side. The minimum height of the whole window should be 900 pixels and txt_edit should have a minimum width of 900 pixels. And The whole layout should be responsive if the window is resized, then txt_edit is resized as well. The width of the Frame that holds the buttons should not change.

Let us show you with a rough sketch of how this text editor will look like:



The desired layout of the Text Editor Application can be achieved using the .grid() geometry manager. And this layout contains a single row and two columns:

1. **On the left side, there is A narrow column** for the buttons
2. **On the right side, there is A wider column** for the text box

In order to set the minimum sizes for the window and txt_edit, you just need to set the minsize parameters of the window methods .rowconfigure() and .columnconfigure() to 900. In order to handle the resizing, the weight parameters of these methods will be set to 1.

If you want both the buttons in the same column then you'll need to create a Frame widget called fr_buttons. According to the above-shown sketch, the two buttons should be stacked vertically inside of this frame, having btn_open on top. This can be done either by .grid() or .pack() geometry manager. For now, you'll just need to stick with .grid() as it is easier to work with it.

Let us start with the code for building the Application:

1. Creating all the needed widgets

The code snippet that is used is as follows:

```
import tkinter as tk
window = tk.Tk()
window.title("Text Editor Application")
window.rowconfigure(0, minsize=900, weight=1)
window.columnconfigure(1, minsize=900, weight=1)
txt_edit = tk.Text(window)
fr_buttons = tk.Frame(window)
btn_open = tk.Button(fr_buttons, text="Open")
btn_save = tk.Button(fr_buttons, text="Save As…")
```

Explanation of the above code:

- **The first command is used to** import the tkinter.
- **Then the next two lines are used to** create a new window with the title "Text Editor Application".
- **The next two lines of code are used to** set the row and column configurations.
- **Then the lines from 9 to 12** will create the four widgets you'll need for the text box, the frame, and the open and save buttons.

```
window.rowconfigure(0, minsize=900, weight=1)
```

The above-given line in the code indicates
The minsize parameter of .rowconfigure() is set to 900 and weight is set to 1.The first argument is 0, which is used to set the height of the first row to 900 pixels and makes sure that the height of the row grows proportionally to the height of the window. There's only one row in the application layout, so these settings are applied to the entire window.

Then take a look at this line in the code :

```
window.columnconfigure(1, minsize=900, weight=1)
```

In the above code the .columnconfigure() to set the width and weight attributes of the column with index 1 to 900 and 1, respectively. Keep it in mind that, row and column indices are zero-based, so these settings apply only to the second column. By configuring the second column, the text box will expand and contract naturally when the window is resized, while the column containing the buttons will always remain at a fixed width.

2.Creation of Application Layout

```
btn_open.grid(row=0, column=0, sticky="ew", padx=5, pady=5)
```

```
btn_save.grid(row=1, column=0, sticky="ew", padx=5)
```

The above two lines of code will **create a grid** with two rows and one column in the fr_buttons frame since both btn_open and btn_save have their master attribute set to fr_buttons. btn_open is put in the first row and btn_save will be in the second row so that btn_open appears above btn_save in the layout, as planned in the above sketch.

- The btn_open and btn_save both have their sticky attributes set to "ew", which forces the buttons to **expand horizontally** in both directions and in order to fill the entire frame. It makes sure both buttons are of the same size.
- You place 5 pixels of **padding** around each button just by setting the padx and pady parameters to 5. The btn_open has vertical padding. Because it's on top, the vertical padding offsets the button down from the top of the window a bit and makes sure that there's a small gap between this and btn_save.

Now the fr_buttons is laid out and ready to go, you can just set up the grid layout for the rest of the window now:

```
fr_buttons.grid(row=0, column=0, sticky="ns")
```

```
txt_edit.grid(row=0, column=1, sticky="nsew")
```

These above two lines of code are used to **create a grid** with one row and two columns for window. You place fr_buttons in the

first column and txt_edit in the second column so that fr_buttons appears to the left of txt_edit in the window layout.

The sticky parameter for fr_buttons will be set to "ns", which forces the whole frame to **expand vertically** and fill the entire height of its column. txt_edit is used to fill its entire grid cell because you set its sticky parameter to "nsew", which forces it to **expand in every direction**.

Now we have just created the buttons but these do not work until we add functioning to them So let's start adding the Functioning of Buttons:

1. Function to Open the File

The code snippet for open_file is as follows:

```
def open_file():
    """Open a file for editing."""
    filepath = askopenfilename(
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
    )
    if not filepath:
        return
    txt_edit.delete(1.0, tk.END)
    with open(filepath, "r") as input_file:
        text = input_file.read()
        txt_edit.insert(tk.END, text)
    window.title(f"Text Editor Application - {filepath}")
```

Explanation:

- **The Lines from 3 to 5** use the askopenfilename dialog from the tkinter.filedialog module to display a file open dialog and store the selected file path to filepath.
- **Lines 6 and 7** checks to see if the user closes the dialog box or clicks the **Cancel** button. If so,

then filepath will be None, and the function will return without executing any of the code to read the file and set the text of txt_edit.

- **Line 8** clears the current contents of txt_edit using .delete().
- **Lines 9 and 10** are used to open the selected file and .read() its contents before storing the text as a string.
- **Line number 11** assigns the string text to txt_edit using .insert().
- **Line 12** sets the title of the window so that it contains the path of the open file.

2. Function to Save the File

The code snippet for save_file is as follows:

```
def save_file():
    """Save the current file as a new file."""
    filepath = asksaveasfilename(
        defaultextension="txt",
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")],
    )
    if not filepath:
        return
    with open(filepath, "w") as output_file:
        text = txt_edit.get(1.0, tk.END)
        output_file.write(text)
    window.title(f"Text Editor Application - {filepath}")
```

Explanation:

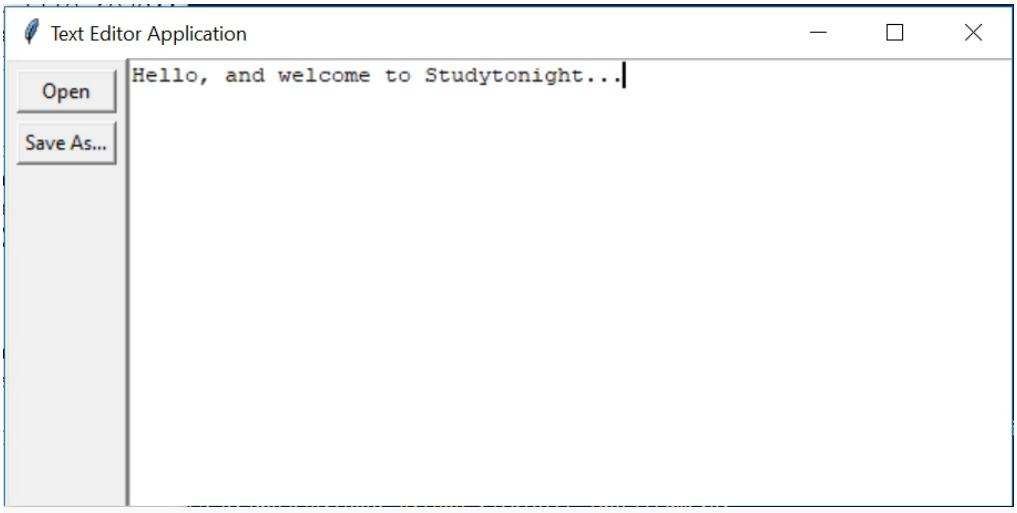
- **Lines 3 to 6** use the asksaveasfilename dialog box to get the desired save location from the user. The selected file path is stored in the filepath variable.

- **Lines 7 and 8** checks to see if the user closes the dialog box or clicks the **Cancel** button. If so, then filepath will be None, and the function will return without executing any of the code to save the text to a file.
- **Line 9** creates a new file at the selected file path.
- **Line 10** extracts the text from txt_edit with .get() method and assigns it to the variable text.
- **Line 11** writes text to the output file.
- **Line 12** updates the title of the window so that the new file path is displayed in the window title.

Complete Source code is:

```
import tkinter as tk
from tkinter.filedialog import askopenfilename,
asksaveasfilename
def open_file():
    """Open a file for editing."""
    filepath = askopenfilename(
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
    )
    if not filepath:
        return
    txt_edit.delete(1.0, tk.END)
    with open(filepath, "r") as input_file:
        text = input_file.read()
        txt_edit.insert(tk.END, text)
    window.title(f"Text Editor Application - {filepath}")
def save_file():
    """Save the current file as a new file."""
    filepath = asksaveasfilename(
```

```
        defaultextension="txt",
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")],
    )
    if not filepath:
        return
    with open(filepath, "w") as output_file:
        text = txt_edit.get(1.0, tk.END)
        output_file.write(text)
    window.title(f"Text Editor Application - {filepath}")
window = tk.Tk()
window.title("Text Editor Application")
window.rowconfigure(0, minsize=800, weight=1)
window.columnconfigure(1, minsize=800, weight=1)
txt_edit = tk.Text(window)
fr_buttons = tk.Frame(window, relief=tk.RAISED, bd=2)
btn_open = tk.Button(fr_buttons, text="Open",
command=open_file)
btn_save = tk.Button(fr_buttons, text="Save As…",
command=save_file)
btn_open.grid(row=0, column=0, sticky="ew", padx=5,
pady=5)
btn_save.grid(row=1, column=0, sticky="ew", padx=5)
fr_buttons.grid(row=0, column=0, sticky="ns")
txt_edit.grid(row=0, column=1, sticky="nsew")
window.mainloop()
```



As you can see in the output, we have a basic text editor application in which **we can write** something and then **save the text in a new file** or use the **Open button to open a file** in the editor and then edit it.

PART 27: Music Player Application using Tkinter (Python Project)

In this tutorial, we will create a Music Player Application in Python using **Tkinter** and **Pygame** module.

In our daily life, we see every person has a hobby and that is listening to music. So in order to listen to music, they all need a Music player(hardware or software) where they can play their favorite songs. And we have to install this music player on our computer, based on the Operating system i.e Windows, Macintosh, Android, Linux, etc. Then we can listen to our favorite songs.

Now we will help you to code and create a Music Player from scratch.

Libraries used for Music Player Application:

Now we will tell you about the Libraries we will use in our code :

1. Tkinter

We had already told you in the title of this page that we are going to use the Tkinter library, which is a standard library for GUI creation. The Tkinter library is most popular and very

easy to use and it comes with many widgets (these widgets help in the creation of nice-looking GUI Applications).

Also, Tkinter is a very light-weight module and it is helpful in creating cross-platform applications(so the same code can easily work on **Windows**, **macOS**, and **Linux)**

To use all the functions of Tkinter you need to import it in your code and the command for the same is:

```
from tkinter import *
```

2. Pygame module

Pygame is a Python module that works with computer graphics and sound libraries and is designed with the power of playing with different multimedia formats like audio, video, etc. While creating our Music Player application, we will be using Pygame's mixer.music module for providing different functionality to our music player application that is usually related to the manipulation of the song tracks.

Command used to install the **pygame** module is:

```
pip install pygame
```

To use this module in your code you need to write this:

```
import pygame
```

3. OS module

There is no need to install this module explicitly, as it comes with the standard library of Python. This module provides different functions for interaction with the Operating System. In this tutorial, we are going to use the OS module for **fetching the playlist of songs from the specified directory** and make it available to the music player application.

To use this module in your code you need to import its and command for the same is as follows:

```
import OS
```

After importing Libraries and modules, now it's time to create a basic window where we will add our UI elements or Tkinter widgets. You can add this code either after importing libraries or also at the end just before the looping of the root window and the code is as follows:

```
root = Tk() # In order to create an empty window
# Passing Root to MusicPlayer Class
MusicPlayer(root)
```

MusicPlayer Class

Here we have the constructor and the other functions defined in the MusicPlayer class.

1. _init_ Constructor

With the help of this constructor, we will set the **title for the window** and **geometry** for the window. We will **initiate pygame and pygame mixer** and then declare the track variable and status variable.

- We will then **Create the Track Frames** for the **Song label** & **status label** and then after Insert the Song Track Label and Status Label.
- After that, we will create the **Button Frame** and **insert play, pause, unpause, and stop buttons** into it.
- Then we will create the **playlist frame** and **add the scrollbar to it** and we will add songs into the playlist.

The code snippet is as follows:

```
def __init__(self,root):
    self.root = root
```

```
    # Title of the window
    self.root.title("MusicPlayer")
    # Window Geometry
    self.root.geometry("1000x200+200+200")
    # Initiating Pygame
    pygame.init()
    # Initiating Pygame Mixer
    pygame.mixer.init()
    # Declaring track Variable
    self.track = StringVar()
    # Declaring Status Variable
    self.status = StringVar()
    # Creating the Track Frames for Song label & status label
    trackframe = LabelFrame(self.root,text="Song Track",font=("times new roman",15,"bold"),bg="Navyblue",fg="white",bd=5,relief=GROOVE)
    trackframe.place(x=0,y=0,width=600,height=100)
    # Inserting Song Track Label
    songtrack = Label(trackframe,textvariable=self.track,width=20,font=("times new roman",24,"bold"),bg="Orange",fg="gold").grid(row=0,column=0,padx=10,pady=5)
    # Inserting Status Label
    trackstatus = Label(trackframe,textvariable=self.status,font=("times new roman",24,"bold"),bg="orange",fg="gold").grid(row=0,column=1,padx=10,pady=5)
    # Creating Button Frame
```

```
    buttonframe = LabelFrame(self.root,text="Control Panel",font=("times new roman",15,"bold"),bg="grey",fg="white",bd=5,relief=GROOVE)
    buttonframe.place(x=0,y=100,width=600,height=100)
    # Inserting Play Button
    playbtn = Button(buttonframe,text="PLAYSONG",command=self.playsong,width=10,height=1,font=("times new roman",16,"bold"),fg="navyblue",bg="pink").grid(row=0,column=0,padx=10,pady=5)
    # Inserting Pause Button
    playbtn = Button(buttonframe,text="PAUSE",command=self.pausesong,width=8,height=1,font=("times new roman",16,"bold"),fg="navyblue",bg="pink").grid(row=0,column=1,padx=10,pady=5)
    # Inserting Unpause Button
    playbtn = Button(buttonframe,text="UNPAUSE",command=self.unpausesong,width=10,height=1,font=("times new roman",16,"bold"),fg="navyblue",bg="pink").grid(row=0,column=2,padx=10,pady=5)
    # Inserting Stop Button
    playbtn = Button(buttonframe,text="STOPSONG",command=self.stopsong,width=10,height=1,font=("times new roman",16,"bold"),fg="navyblue",bg="pink").grid(row=0,column=3,padx=10,pady=5)
    # Creating Playlist Frame
    songsframe = LabelFrame(self.root,text="Song Playlist",font=("times new roman",15,"bold"),bg="grey",fg="white",bd=5,relief=GROOVE)
    songsframe.place(x=600,y=0,width=400,height=200)
```

```
    # Inserting scrollbar
    scrol_y = Scrollbar(songsframe,orient=VERTICAL)
    # Inserting Playlist listbox
    self.playlist = Listbox(songsframe,yscrollcommand=scrol_y.set,selectbackground="gold",selectmode=SINGLE,font=("times new roman",12,"bold"),bg="silver",fg="navyblue",bd=5,relief=GROOVE)
    # Applying Scrollbar to listbox
    scrol_y.pack(side=RIGHT,fill=Y)
    scrol_y.config(command=self.playlist.yview)
    self.playlist.pack(fill=BOTH)
    # Changing Directory for fetching Songs
    os.chdir("PATH/OF/DIRECTORY")
    # Fetching Songs
    songtracks = os.listdir()
    # Inserting Songs into Playlist
    for track in songtracks:
      self.playlist.insert(END,track)
```

In the code above, change the **PATH/OF/DIRECTORY** with an appropriate path where the song files are stored.

2. The playsong() Function

Now we will define the Play Song Function and the code is:

```
def playsong(self):
    # Displaying Selected Song title
    self.track.set(self.playlist.get(ACTIVE))
    # Displaying Status
    self.status.set("-Playing")
    # Loading Selected Song
```

```
    pygame.mixer.music.load(self.playlist.get(ACTIVE))
    # Playing Selected Song
    pygame.mixer.music.play()
```

3. The stopsong() Function

The code snippet to stop the song:

```
def stopsong(self):
    # Displaying Status
    self.status.set("-Stopped")
    # Stopped Song
    pygame.mixer.music.stop()
```

4. The pausesong() Function

The code snippet to pause the song:

```
def pausesong(self):
    # Displaying Status
    self.status.set("-Paused")
    # Paused Song
    pygame.mixer.music.pause()
```

5. The unpausesong() Function

The code snippet to unpause the song:

```
 def unpausesong(self):
    # It will Display the  Status
    self.status.set("-Playing")
    # Playing back Song
    pygame.mixer.music.unpause()
```

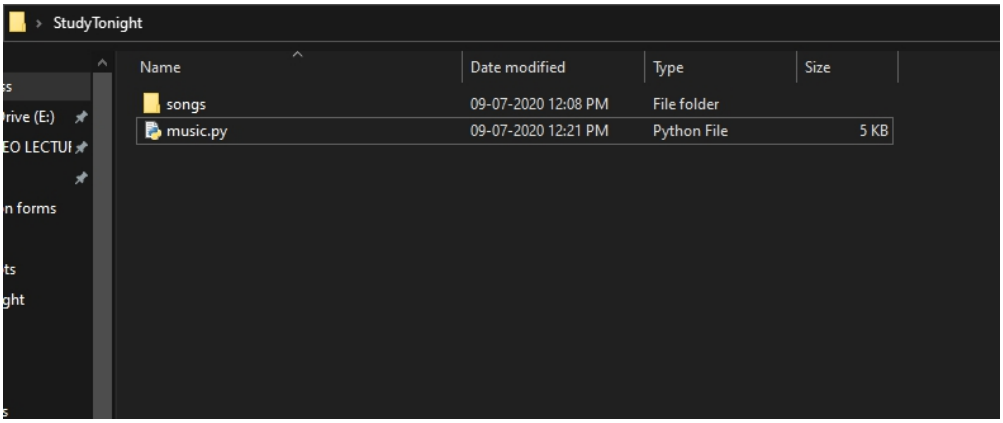
6. Root Window Looping

The command will be:

```
root.mainloop()
```

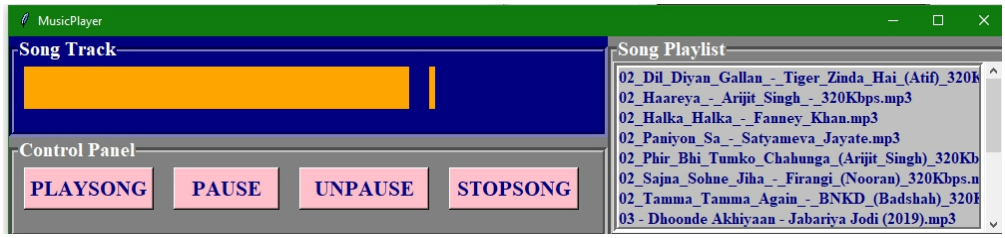
Now here are some of the screenshots of our application:

The Folder named studytonight where the code file and folder for songs is placed looks like this:



We will provide the path of the songs folder in our code where all songs are placed in order to access them.

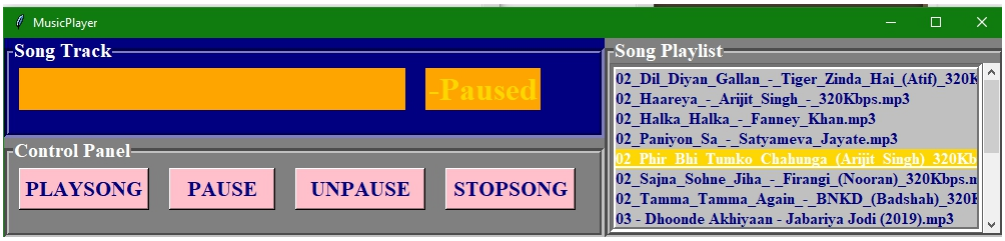
Now the following screenshot is to show you how the application will look like:



whenever you made a **click on the song** it will look like this:

On clicking the **PLAYSONG Button**:

On clicking the **PAUSE Button**:



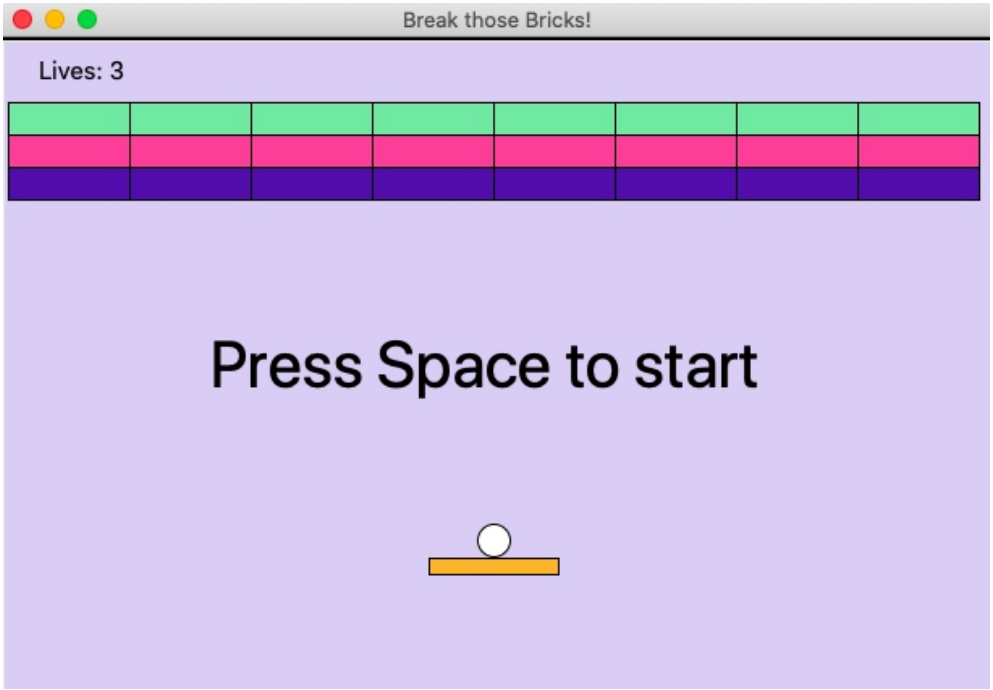
On clicking the **STOP Button**:

So this was all about building the Music Player Application using Tkinter. Hope you all like this application. Do share more ideas with us for Tkinter based desktop applications and we will definitely add more Tkinter projects to our Tkinter tutorial.

PART 28: Brick Breaker Game using Tkinter (Python Project)

It's time to code a game in Python. We will start with a simple Brick breaker game in which there is a ball that bounces of a platform to break a brick wall and the player has to keep the ball going by making sure the paddle is always there to bounce off the ball back.

The game will have three layers of bricks, and each layer of brick will have a different hit capacity, which means some bricks will break in a **single hit**, some will require a **double hit** and some will require **three hits**.



To simplify the logic and user interface of the game, the following are the type of bricks:

Green brick: Requires three hits to break

Pink bricks: Requires two hits to break

Purple bricks: Requires three hits to break

Prerequisites:

To understand the code to build the brick breaker game in Python, you should know the following concepts:

- Create Class in Python
- Constructor for Class in Python
- Inheritance in Python
- Access modifiers in Python
- __name__ as the main method in Python

APart from that the overall understanding of Python is required, which you can also get from our FREE tutorial series to **Learn Python**.

Code for Brick Breaker Game

We will be creating 5 classes in the code, the name of the classes are:

1. GameObject
2. Ball
3. Paddle
4. Brick
5. Game

All these classes will have methods to perform various handling and operations on the respective game element.

Let's see the code for each class.

GameObject class

This class is the parent class of all the other classes, which means all the other classes will inherit this class.

The code for this class is as follows:

```
class GameObject(object):
    def __init__(self, canvas, item):
        self.canvas = canvas
        self.item = item
    def get_position(self):
        return self.canvas.coords(self.item)
    def move(self, x, y):
        self.canvas.move(self.item, x, y)
    def delete(self):
        self.canvas.delete(self.item)
```

This class has functions to handle the basic lifecycle of different objects that form the game.

Ball Class

Here is the code for the Ball class:

```
class Ball(GameObject):
    def __init__(self, canvas, x, y):
        self.radius = 10
        self.direction = [1, -1]
        # increase the below value to increase the speed of ball
        self.speed = 5
        item = canvas.create_oval(x-self.radius, y-self.radius,
                                  x+self.radius, y+self.radius,
                                  fill='white')
        super(Ball, self).__init__(canvas, item)
    def update(self):
        coords = self.get_position()
        width = self.canvas.winfo_width()
        if coords[0] <= 0 or coords[2] >= width:
```

```
            self.direction[0] *= -1
        if coords[1] <= 0:
            self.direction[1] *= -1
        x = self.direction[0] * self.speed
        y = self.direction[1] * self.speed
        self.move(x, y)
    def collide(self, game_objects):
        coords = self.get_position()
        x = (coords[0] + coords[2]) * 0.5
        if len(game_objects) > 1:
            self.direction[1] *= -1
        elif len(game_objects) == 1:
            game_object = game_objects[0]
            coords = game_object.get_position()
            if x > coords[2]:
                self.direction[0] = 1
            elif x < coords[0]:
                self.direction[0] = -1
            else:
                self.direction[1] *= -1
        for game_object in game_objects:
            if isinstance(game_object, Brick):
                game_object.hit()
```

In the Ball class the `collide()` method is used to handle the collision and is used in the check_collisions() function of the Game class. Whereas the `update()` function is to reset the Ball object after a life is lost and the game starts again.

Paddle Class

Here is the code for the Paddle class:

```
class Paddle(GameObject):
    def __init__(self, canvas, x, y):
        # set the shape and position of paddle
        self.width = 80
        self.height = 10
        self.ball = None
        item = canvas.create_rectangle(x - self.width / 2,
                                       y - self.height / 2,
                                       x + self.width / 2,
                                       y + self.height / 2,
                                       fill='#FFB643')
        super(Paddle, self).__init__(canvas, item)
    def set_ball(self, ball):
        self.ball = ball
    def move(self, offset):
        coords = self.get_position()
        width = self.canvas.winfo_width()
        if coords[0] + offset >= 0 and coords[2] + offset <=
width:
            super(Paddle, self).move(offset, 0)
            if self.ball is not None:
                self.ball.move(offset, 0)
```

APart from the constructor, the `move()` method in this class is important as that **enables the movement for the paddle**.

Brick Class

Here is the code for the Brick class:

```
class Brick(GameObject):
    COLORS = {1: '#4535AA', 2: '#ED639E', 3: '#8FE1A2'}
```

```
    def __init__(self, canvas, x, y, hits):
        self.width = 75
        self.height = 20
        self.hits = hits
        color = Brick.COLORS[hits]
        item = canvas.create_rectangle(x - self.width / 2,
                                       y - self.height / 2,
                                       x + self.width / 2,
                                       y + self.height / 2,
                                       fill=color, tags='brick')
        super(Brick, self).__init__(canvas, item)
    def hit(self):
        self.hits -= 1
        if self.hits == 0:
            self.delete()
        else:
            self.canvas.itemconfig(self.item,
                                   fill=Brick.COLORS[self.hits])
```

We start the class definition by setting the colors for the different brick types. Then in the constructor, the size and position of the bricks are defined. And then there is the `hit()` function that handles whether the brick will disappear or it will take the hit and change color to take the next hit.

Game Class

This is the main class that controls everything. Here is the code:

```
class Game(tk.Frame):
    def __init__(self, master):
        super(Game, self).__init__(master)
```

```
        self.lives = 3
        self.width = 610
        self.height = 400
        self.canvas = tk.Canvas(self, bg='#D6D1F5',
                                width=self.width,
                                height=self.height,)
        self.canvas.pack()
        self.pack()
        self.items = {}
        self.ball = None
        self.paddle = Paddle(self.canvas, self.width/2, 326)
        self.items[self.paddle.item] = self.paddle
        # adding brick with different hit capacities - 3,2 and 1
        for x in range(5, self.width - 5, 75):
            self.add_brick(x + 37.5, 50, 3)
            self.add_brick(x + 37.5, 70, 2)
            self.add_brick(x + 37.5, 90, 1)
        self.hud = None
        self.setup_game()
        self.canvas.focus_set()
        self.canvas.bind('<Left>',
                         lambda _: self.paddle.move(-10))
        self.canvas.bind('<Right>',
                         lambda _: self.paddle.move(10))
    def setup_game(self):
        self.add_ball()
        self.update_lives_text()
        self.text = self.draw_text(300, 200,
```

```
                                'Press Space to start')
        self.canvas.bind('<space>', lambda _:
self.start_game())
    def add_ball(self):
        if self.ball is not None:
            self.ball.delete()
        paddle_coords = self.paddle.get_position()
        x = (paddle_coords[0] + paddle_coords[2]) * 0.5
        self.ball = Ball(self.canvas, x, 310)
        self.paddle.set_ball(self.ball)
    def add_brick(self, x, y, hits):
        brick = Brick(self.canvas, x, y, hits)
        self.items[brick.item] = brick
    def draw_text(self, x, y, text, size='40'):
        font = ('Forte', size)
        return self.canvas.create_text(x, y, text=text,
                                       font=font)
    def update_lives_text(self):
        text = 'Lives: %s' % self.lives
        if self.hud is None:
            self.hud = self.draw_text(50, 20, text, 15)
        else:
            self.canvas.itemconfig(self.hud, text=text)
    def start_game(self):
        self.canvas.unbind('<space>')
        self.canvas.delete(self.text)
        self.paddle.ball = None
        self.game_loop()
```

```
    def game_loop(self):
        self.check_collisions()
        num_bricks = len(self.canvas.find_withtag('brick'))
        if num_bricks == 0:
            self.ball.speed = None
            self.draw_text(300, 200, 'You win! You the Breaker of
Bricks.')
        elif self.ball.get_position()[3] >= self.height:
            self.ball.speed = None
            self.lives -= 1
            if self.lives < 0:
                self.draw_text(300, 200, 'You Lose! Game Over!')
            else:
                self.after(1000, self.setup_game)
        else:
            self.ball.update()
            self.after(50, self.game_loop)
    def check_collisions(self):
        ball_coords = self.ball.get_position()
        items = self.canvas.find_overlapping(*ball_coords)
        objects = [self.items[x] for x in items if x in self.items]
        self.ball.collide(objects)
```

In this class, we have the following:

1. We define the number of lives, the background colors of the game, and other styling of the game window.
2. The game setup is done by this class.
3. Adding the bricks, adding the paddle, ball, and text is done by this class.
4. Handling of lives is taken care of by this class.

5. Starting the game and running the game loop, is also done in this class.

Complete Code for the Brick Breaker Game:

Here is the complete code for the game.

```
import tkinter as tk
class GameObject(object):
    def __init__(self, canvas, item):
        self.canvas = canvas
        self.item = item
    def get_position(self):
        return self.canvas.coords(self.item)
    def move(self, x, y):
        self.canvas.move(self.item, x, y)
    def delete(self):
        self.canvas.delete(self.item)
class Ball(GameObject):
    def __init__(self, canvas, x, y):
        self.radius = 10
        self.direction = [1, -1]
        # increase the below value to increase the speed of ball
        self.speed = 5
        item = canvas.create_oval(x-self.radius, y-self.radius,
                                  x+self.radius, y+self.radius,
                                  fill='white')
        super(Ball, self).__init__(canvas, item)
    def update(self):
```

```
        coords = self.get_position()
        width = self.canvas.winfo_width()
        if coords[0] <= 0 or coords[2] >= width:
            self.direction[0] *= -1
        if coords[1] <= 0:
            self.direction[1] *= -1
        x = self.direction[0] * self.speed
        y = self.direction[1] * self.speed
        self.move(x, y)
    def collide(self, game_objects):
        coords = self.get_position()
        x = (coords[0] + coords[2]) * 0.5
        if len(game_objects) > 1:
            self.direction[1] *= -1
        elif len(game_objects) == 1:
            game_object = game_objects[0]
            coords = game_object.get_position()
            if x > coords[2]:
                self.direction[0] = 1
            elif x < coords[0]:
                self.direction[0] = -1
            else:
                self.direction[1] *= -1
        for game_object in game_objects:
            if isinstance(game_object, Brick):
                game_object.hit()
class Paddle(GameObject):
    def __init__(self, canvas, x, y):
```

```
        self.width = 80
        self.height = 10
        self.ball = None
        item = canvas.create_rectangle(x - self.width / 2,
                                       y - self.height / 2,
                                       x + self.width / 2,
                                       y + self.height / 2,
                                       fill='#FFB643')
        super(Paddle, self).__init__(canvas, item)
    def set_ball(self, ball):
        self.ball = ball
    def move(self, offset):
        coords = self.get_position()
        width = self.canvas.winfo_width()
        if coords[0] + offset >= 0 and coords[2] + offset <=
width:
            super(Paddle, self).move(offset, 0)
            if self.ball is not None:
                self.ball.move(offset, 0)
class Brick(GameObject):
    COLORS = {1: '#4535AA', 2: '#ED639E', 3: '#8FE1A2'}
    def __init__(self, canvas, x, y, hits):
        self.width = 75
        self.height = 20
        self.hits = hits
        color = Brick.COLORS[hits]
        item = canvas.create_rectangle(x - self.width / 2,
                                       y - self.height / 2,
```

```
                                x + self.width / 2,
                                y + self.height / 2,
                                fill=color, tags='brick')
        super(Brick, self).__init__(canvas, item)
    def hit(self):
        self.hits -= 1
        if self.hits == 0:
            self.delete()
        else:
            self.canvas.itemconfig(self.item,
                                   fill=Brick.COLORS[self.hits])
class Game(tk.Frame):
    def __init__(self, master):
        super(Game, self).__init__(master)
        self.lives = 3
        self.width = 610
        self.height = 400
        self.canvas = tk.Canvas(self, bg='#D6D1F5',
                                width=self.width,
                                height=self.height,)
        self.canvas.pack()
        self.pack()
        self.items = {}
        self.ball = None
        self.paddle = Paddle(self.canvas, self.width/2, 326)
        self.items[self.paddle.item] = self.paddle
        # adding brick with different hit capacities - 3,2 and 1
        for x in range(5, self.width - 5, 75):
```

```
            self.add_brick(x + 37.5, 50, 3)
            self.add_brick(x + 37.5, 70, 2)
            self.add_brick(x + 37.5, 90, 1)
        self.hud = None
        self.setup_game()
        self.canvas.focus_set()
        self.canvas.bind('<Left>',
                         lambda _: self.paddle.move(-10))
        self.canvas.bind('<Right>',
                         lambda _: self.paddle.move(10))
    def setup_game(self):
            self.add_ball()
            self.update_lives_text()
            self.text = self.draw_text(300, 200,
                                       'Press Space to start')
            self.canvas.bind('<space>', lambda _:
self.start_game())
    def add_ball(self):
        if self.ball is not None:
            self.ball.delete()
        paddle_coords = self.paddle.get_position()
        x = (paddle_coords[0] + paddle_coords[2]) * 0.5
        self.ball = Ball(self.canvas, x, 310)
        self.paddle.set_ball(self.ball)
    def add_brick(self, x, y, hits):
        brick = Brick(self.canvas, x, y, hits)
        self.items[brick.item] = brick
    def draw_text(self, x, y, text, size='40'):
```

```
        font = ('Forte', size)
        return self.canvas.create_text(x, y, text=text,
                                       font=font)
    def update_lives_text(self):
        text = 'Lives: %s' % self.lives
        if self.hud is None:
            self.hud = self.draw_text(50, 20, text, 15)
        else:
            self.canvas.itemconfig(self.hud, text=text)
    def start_game(self):
        self.canvas.unbind('<space>')
        self.canvas.delete(self.text)
        self.paddle.ball = None
        self.game_loop()
    def game_loop(self):
        self.check_collisions()
        num_bricks = len(self.canvas.find_withtag('brick'))
        if num_bricks == 0:
            self.ball.speed = None
            self.draw_text(300, 200, 'You win! You the Breaker of
Bricks.')
        elif self.ball.get_position()[3] >= self.height:
            self.ball.speed = None
            self.lives -= 1
            if self.lives < 0:
                self.draw_text(300, 200, 'You Lose! Game Over!')
            else:
                self.after(1000, self.setup_game)
```

```
        else:
            self.ball.update()
            self.after(50, self.game_loop)
    def check_collisions(self):
        ball_coords = self.ball.get_position()
        items = self.canvas.find_overlapping(*ball_coords)
        objects = [self.items[x] for x in items if x in self.items]
        self.ball.collide(objects)
if __name__ == '__main__':
    root = tk.Tk()
    root.title('Break those Bricks!')
    game = Game(root)
    game.mainloop()
```

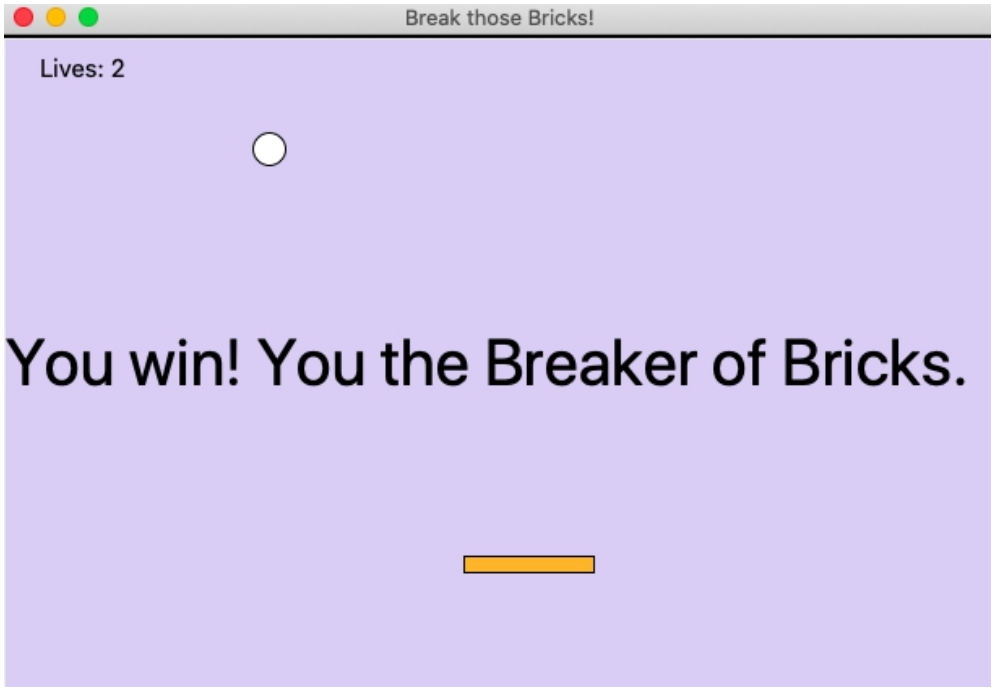
To run the game, you must have Tkinter module installed. Save the code given above in a file with some name, let's say **brick-breaker.py** file. Then to run the code execute the following command:

```
python brick-breaker.py
```

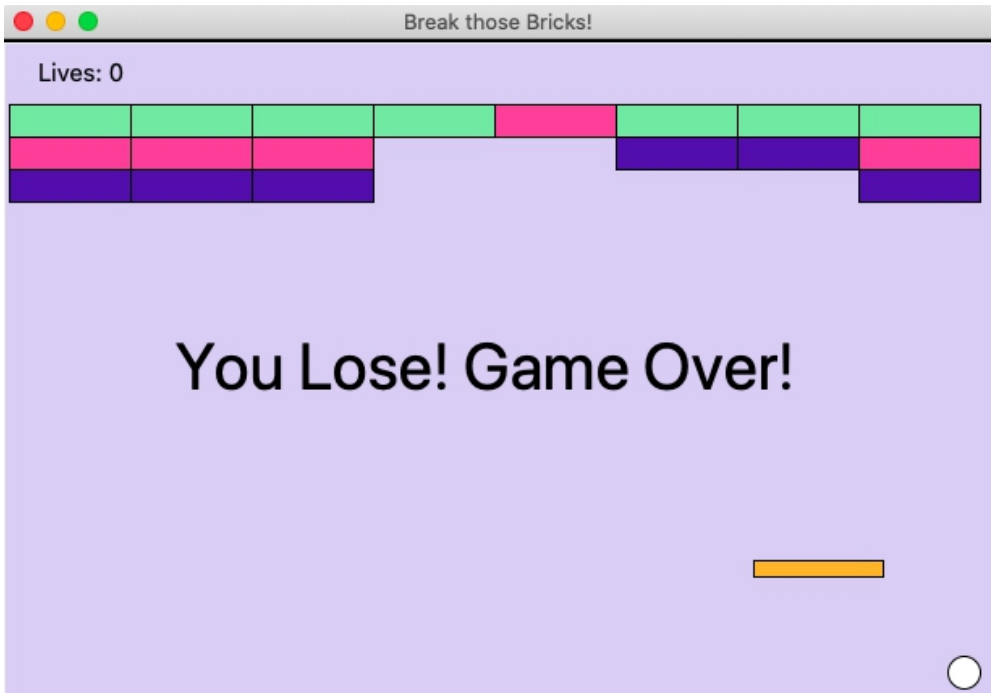
Brick Breaker Game UI:

Here are a few snapshots of how the game looks.

When the **player wins** the game:



When the **player loses** the game:

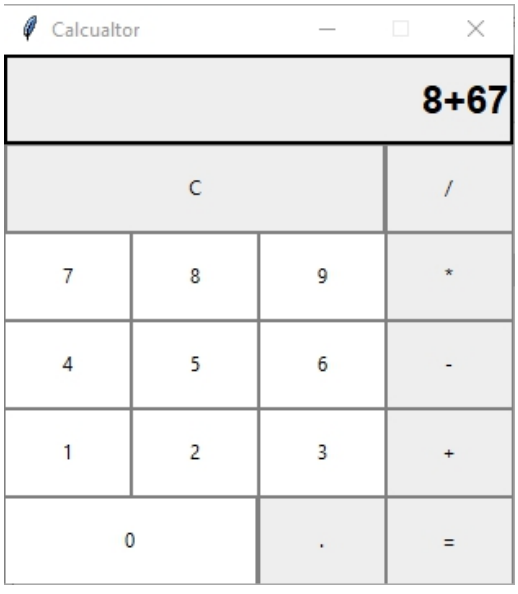


And with that, this tutorial ends. Hope the game works fine for you. We recommend you to edit and customize the code to make the game better. Try to add more features to the game.

PART 29: Calculator Application Using Python Language

The Calculator is one application that we all utilize in our **day-to-day** lives. If you are attempting to get your hands dirty with programming in python, Calculator is a straightforward and helpful project at the same time. Today, we will develop a **Python Calculator** using Tkinter with simple to follow instructions.

Here is how our calculator will appear, which is created using the input field, buttons, and logic defined in functions for calculation purposes. For example, if you want to add two numbers, there must be logic for addition, similarly for subtraction, multiplication, and so on, we have created functions whose task is to perform these operations.



We have an Input Field where the user input will be shown along with the calculation's ultimate result.

The buttons are numbered 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, =, and C. (clear button)

What is a Calculator?

For those unfamiliar, a calculator is a computer application that mimics the behaviour of any hand-held calculator used for completing mathematical calculations. It is a really simple item that we use on a daily basis. A Calculator application is now available on all cellphones.

There are primarily two processes involved in developing any GUI application:

- Creating a User Interface is the initial stage.
- The second and most crucial stage is to provide the functionality to the graphical user interface.

Let's get started by making a simple calculator app in Python utilizing Tkinter for basic arithmetic calculations.

Source Code for Calculator Application

Now it's time to look at the code for making a Tkinter Calculator App:

```
from tkinter import *

win = Tk() # This is to create a basic window

win.geometry("312x324")  # this is for the size of the window

win.resizable(0, 0)  # this is to prevent from resizing the window

win.title("Calculator")

######################Starting with functions ######################

# 'btn_click' function :
```

```
# This Function continuously updates the
# input field whenever you enter a number
def btn_click(item):
    global expression
    expression = expression + str(item)
    input_text.set(expression)
# ‘bt_clear’ function :This is used to clear
# the input field
def bt_clear():
    global expression
    expression = ””
    input_text.set(””)
# ‘bt_equal’:This method calculates the expression
# present in input field
def bt_equal():
    global expression
    result = str(eval(expression)) # ‘eval’:This function is used to evaluates the string expression directly
    input_text.set(result)
    expression = ””
expression = ””
# ‘StringVar()’ :It is used to get the instance of input field
input_text = StringVar()
# Let us creating a frame for the input field
input_frame = Frame(win, width=312, height=50, bd=0, highlightbackground=“black”, highlightcolor=“black”, highlightthickness=2)
input_frame.pack(side=TOP)
#Let us create 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

#Let us creating another 'Frame' for the button below the 'input_frame'

btns_frame = Frame(win, 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: bt_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: bt_equal()).grid(row = 4, column = 3, padx = 1, pady = 1)
```

```
win.mainloop()
```

Tkinter has a number of methods that make it simple and quick to create a basic calculator with only a few lines of code.

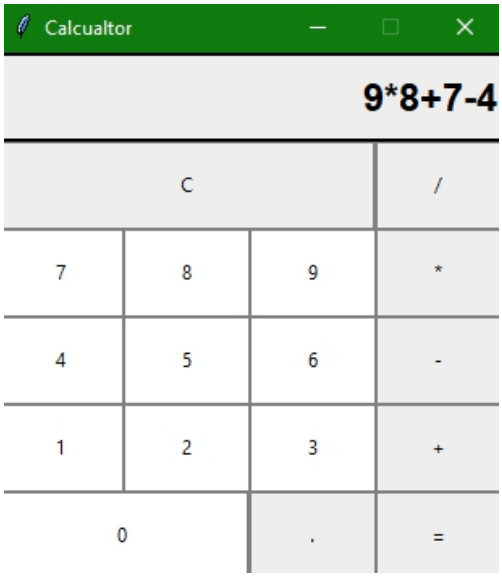
We've defined the following functions in our code, in addition to the Tkinter widgets:

- **btn click() Function:** This function adds various numeric buttons to the operation by handling button clicks.
- **The bt clear()** method is used to handle the clear action in the Calculator application, which cleans the previous input.

The equal button is handled by the bt equal() function, which does the action and displays the result.

Now we'll show you a screenshot of the aforementioned code's result. Yes, you may use it on your machine to have a better knowledge of Calculator App Using Tkinter:

Output:



Summary:

In this article, we used Tkinter and several **Tkinter** widgets to create a simple **Calculator** application, which we covered in our Tkinter Tutorial.

-

PART 30: Alarm Clock Using Python Language

The Alarm Clock Using **Python** is a project built in a python programming language, The purpose of this Python **Alarm Clock** Tutorial is to create a Python Alarm Clock GUI.

A Python Alarm Clock Script includes essential libraries such as **DateTime and Tkinter,** which assist us in constructing projects utilizing the current date and time. They also give a user interface to set the alarm according to the demand in a 24-hour format. To start developing this primary project, Alarm Clock Using Python, make sure that you have **Pycharm IDE** installed on your computer.

Steps on How to Make An Alarm Clock Using Python.

Alarm Clock Using Python With Source Code

Step 1: Create a project name.

First, open Pycharm IDE and then create a “project name” after generating a project name, click the “create” button.

Step 2: Create a python file.

Second, after establishing a project name, “right-click” your project name and then choose “new” after that, click the “python file“.

Step 3: Name your python file.

Third, after generating a python file, Name your python file; click “enter“.

Step 4: The actual code.

You can the code supplied below and download the entire source code below.

Code For Importing Modules

```
from tkinter import *
import datetime
import time
import winsound
```

While the following code imports every module that is being called in running the application.

Code For The Module Actual Time

```
def actual_time():
    set_alarm_timer = f"{hour.get()}:{min.get()}:{sec.get()}"
    alarm(set_alarm_timer)
```

This module is the real or the current time that is being called while setting the alarm.

Code For The Module Of Setting The Alarm

```
def alarm():
    # Infinite Loop
    while True:
        # Set Alarm
        set_alarm = f"{hour.get()}:{minute.get()}:
{second.get()}"
        # Wait for one seconds
        time.sleep(1)
        # Get current time
        current_time =
datetime.datetime.now().strftime("%H:%M:%S")
        # Check whether set alarm is equal to current time or not
        if current_time == set_alarm:
```

```
        print("Time to Wake up")
        # Playing sound
        winsound.PlaySound("sound.wav",winsound.SND_ASYNC)
```

In this module which is the setup of alarm that is performed.

Code For The GUI

```
clock = Tk()
clock.title("DataFlair Alarm Clock")
clock.geometry("400x200")
time_format=Label(clock, text= "Enter time in 24 hour format!",
fg="red",bg="black",font="Arial").place(x=60,y=120)
addTime = Label(clock,text = "Hour  Min Sec",font=60).place(x = 110)
setYourAlarm = Label(clock,text = "When to wake you up",fg="blue",relief = "solid",font=
("Helevetica",7,"bold")).place(x=0, y=29)
# The Variables we require to set the alarm(initialization):
hour = StringVar()
min = StringVar()
sec = StringVar()
#Time required to set the alarm clock:
hourTime= Entry(clock,textvariable = hour,bg = "pink",width = 15).place(x=110,y=30)
minTime= Entry(clock,textvariable = min,bg = "pink",width = 15).place(x=150,y=30)
secTime = Entry(clock,textvariable = sec,bg = "pink",width = 15).place(x=200,y=30)
#To take the time input by user:
submit = Button(clock,text = "Set Alarm",fg="red",width = 10,command = actual_time).place(x =110,y=70)
```

```
clock.mainloop()
#Execution of the window.
```

This module is the design or the graphical user interface or (GUI) of this project.

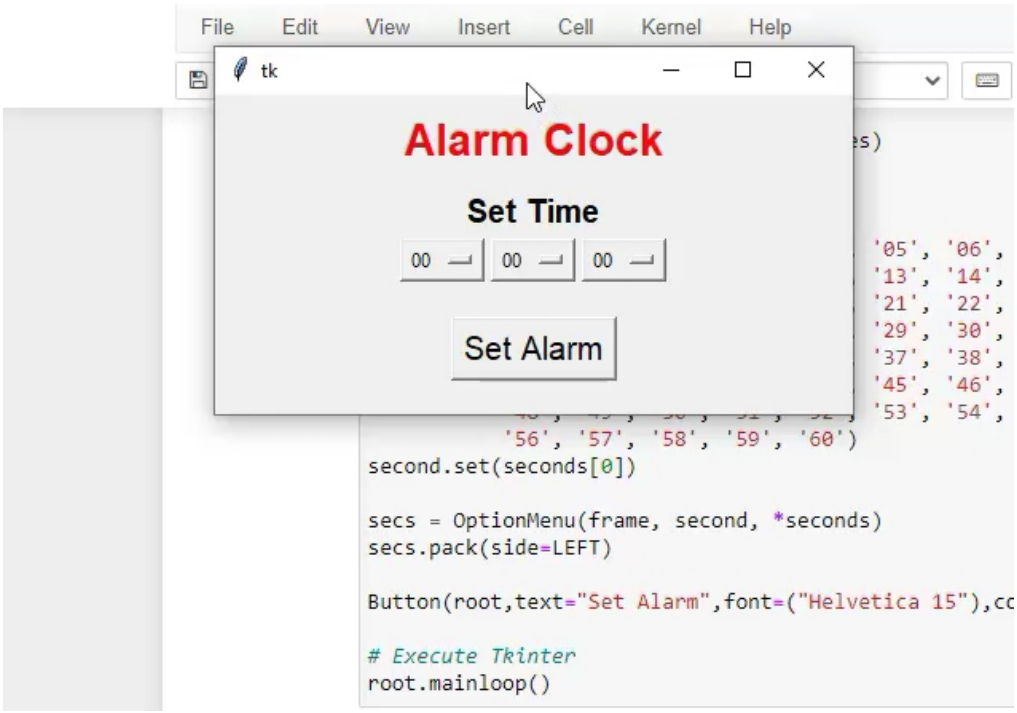
Complete Source Code of Alarm Clock Using Python

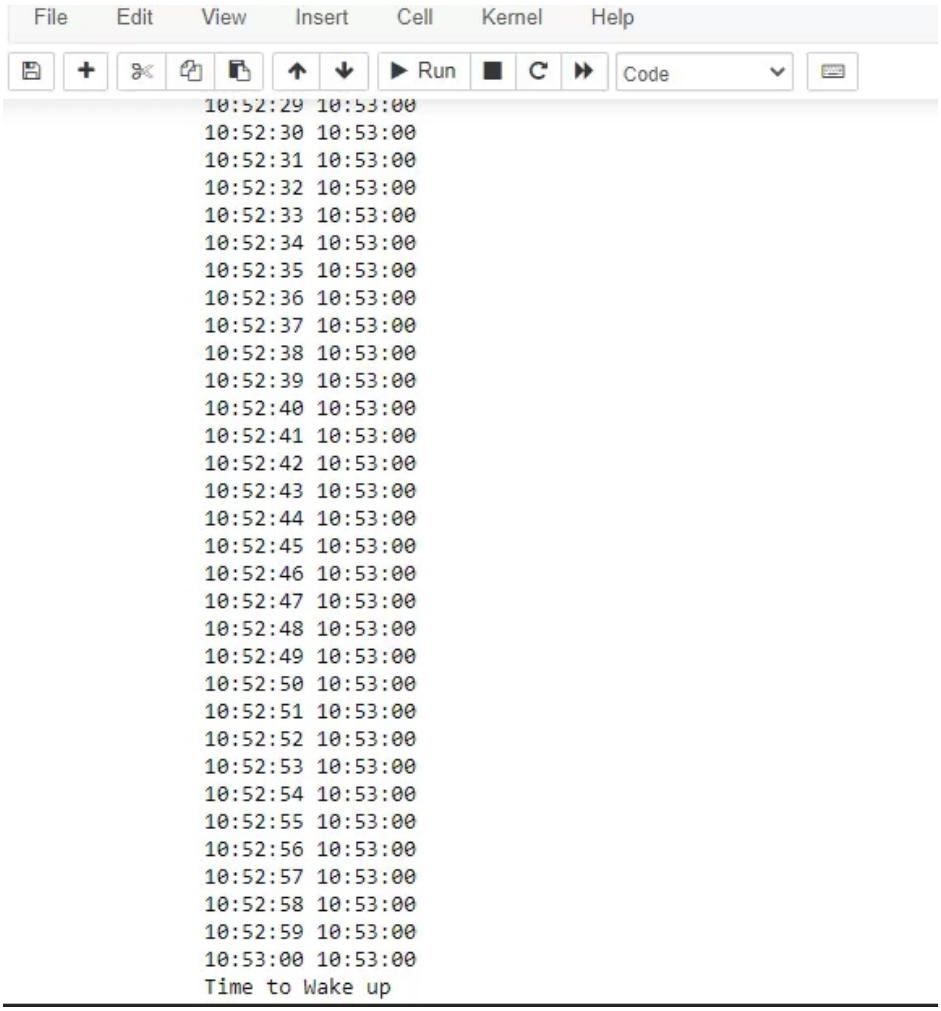
```
# Import Required Library
from tkinter import *
import datetime
import time
import winsound
from threading import *
# Create Object
root = Tk()
# Set geometry
root.geometry("400x200")
# Use Threading
def Threading():
    t1=Thread(target=alarm)
    t1.start()
def alarm():
    # Infinite Loop
    while True:
        # Set Alarm
        set_alarm_time = f"{hour.get()}:{minute.get()}:
{second.get()}"
        # Wait for one seconds
        time.sleep(1)
```

```
        # Get current time
        current_time =
datetime.datetime.now().strftime("%H:%M:%S")
        print(current_time,set_alarm_time)
        # Check whether set alarm is equal to current time or not
        if current_time == set_alarm_time:
            print("Time to Wake up")
            # Playing sound
            winsound.PlaySound("sound.wav",winsound.SND_A
SYNC)
# Add Labels, Frame, Button, Optionmenus
Label(root,text="Alarm Clock",font=("Helvetica 20
bold"),fg="red").pack(pady=10)
Label(root,text="Set Time",font=("Helvetica 15 bold")).pack()
frame = Frame(root)
frame.pack()
hour = StringVar(root)
hours = ('00', '01', '02', '03', '04', '05', '06', '07',
          '08', '09', '10', '11', '12', '13', '14', '15',
          '16', '17', '18', '19', '20', '21', '22', '23', '24'
        )
hour.set(hours[0])
hrs = OptionMenu(frame, hour, *hours)
hrs.pack(side=LEFT)
minute = StringVar(root)
minutes = ('00', '01', '02', '03', '04', '05', '06', '07',
            '08', '09', '10', '11', '12', '13', '14', '15',
            '16', '17', '18', '19', '20', '21', '22', '23',
            '24', '25', '26', '27', '28', '29', '30', '31',
```

```
        '32', '33', '34', '35', '36', '37', '38', '39',
        '40', '41', '42', '43', '44', '45', '46', '47',
        '48', '49', '50', '51', '52', '53', '54', '55',
        '56', '57', '58', '59', '60')
minute.set(minutes[0])
mins = OptionMenu(frame, minute, *minutes)
mins.pack(side=LEFT)
second = StringVar(root)
seconds = ('00', '01', '02', '03', '04', '05', '06', '07',
        '08', '09', '10', '11', '12', '13', '14', '15',
        '16', '17', '18', '19', '20', '21', '22', '23',
        '24', '25', '26', '27', '28', '29', '30', '31',
        '32', '33', '34', '35', '36', '37', '38', '39',
        '40', '41', '42', '43', '44', '45', '46', '47',
        '48', '49', '50', '51', '52', '53', '54', '55',
        '56', '57', '58', '59', '60')
second.set(seconds[0])
secs = OptionMenu(frame, second, *seconds)
secs.pack(side=LEFT)
Button(root,text="Set Alarm",font=("Helvetica
15"),command=Threading).pack(pady=20)
# Execute Tkinter
root.mainloop()
```

Output:


File Edit View Insert Cell Kernel Help
tk
Alarm Clock
Set Time
00 00 00
Set Alarm
'05', '06',
'13', '14',
'21', '22',
'29', '30',
'37', '38',
'45', '46',
'53', '54',
'56', '57', '58', '59', '60')
second.set(seconds[0])
secs = OptionMenu(frame, second, *seconds)
secs.pack(side=LEFT)
Button(root,text="Set Alarm",font=("Helvetica 15"),cc
# Execute Tkinter
root.mainloop()


File Edit View Insert Cell Kernel Help
Run
Code
10:52:30 10:53:00
10:52:31 10:53:00
10:52:32 10:53:00
10:52:33 10:53:00
10:52:34 10:53:00
10:52:35 10:53:00
10:52:36 10:53:00
10:52:37 10:53:00
10:52:38 10:53:00
10:52:39 10:53:00
10:52:40 10:53:00
10:52:41 10:53:00
10:52:42 10:53:00
10:52:43 10:53:00
10:52:44 10:53:00
10:52:45 10:53:00
10:52:46 10:53:00
10:52:47 10:53:00
10:52:48 10:53:00
10:52:49 10:53:00
10:52:50 10:53:00
10:52:51 10:53:00
10:52:52 10:53:00
10:52:53 10:53:00
10:52:54 10:53:00
10:52:55 10:53:00
10:52:56 10:53:00
10:52:57 10:53:00
10:52:58 10:53:00
10:52:59 10:53:00
10:53:00 10:53:00
Time to Wake up

Conclusion

The goal of this project, **Alarm Clock** Using Python, is to create an alarm clock using Python. **Python** contains some essential libraries such as DateTime and Tkinter, which assist us in constructing a project utilizing the current date and time. They also give a user interface to set the alarm according to the demand in a **24-hour format.**

PART 31: Number Guessing Game In Python

A number guessing game is a frequent mini-project for beginner **programmers** who grip random number generation and conditional statements with iteration.

The number guessing game is built on the player's notion to estimate a number between the **provided** range. If the player predicts the desired number, the player wins; otherwise, the player loses the game. Since this game has limited efforts thus, the player needs to indicate the number with the **limited** attempts. Otherwise, the player will **lose** the game.

In this tutorial, we will construct a number guessing game in Python.

Number Guessing Game Rules

1. You must input only valid integers within the provided range.
2. You will be granted limited tries to estimate the number.
3. You cannot exit the game once begun.

If the input number is less than or more significant than the **needed** number, the player receives the message (hint) to move further in up or down range.

We first produce a random number between a **defined** range in such a game. We ask the user to estimate this number. If the guess is accurate, we report that the guess is correct and break out of the loop. Else we specify whether the number is less or more significant than the actual number. We also ask the user for the total guesses they are authorized to take. When the number of guesses surpasses this, we **break** off the loop.

The user may make use of this to know the **actual** number. For example, if the user guesses that the number is 45 and the result is that the **exact** number is smaller than 45, the user might infer that the number won't lie between 45 and 100 (provided that the range is up to 100). (given that the content is till 100). This way, the user may keep guessing and interpreting the outcome. We publish the number of **guesses** it takes the user to get the answer correctly.

Number Guessing Game Implementation in Python Language

```
import random
t = 0
g = int(input("Total Guesses: "))
low = int(input("Enter the lower range: "))
high = int(input("Enter the upper range: "))
x = random.randint(low, high)
n = int(input("Enter an integer between the given range: "))
while (x != 'n'):
    if(t<(g-1)):
        if n < x:
            print("The number guessed is low")
            t = t+1
            n = int(input("Enter an integer between the given range: "))
        elif (n > x):
            print("The number guessed is high")
            t = t+1
            n = int(input("Enter an integer between the given range: "))
        else:
```

```
            print("The number guessed is right")
            print("Total guesses taken: ", t+1)
            break
    else:
        print("Ran out of tries!")
        break
```

Output:

```
Total Guesses: 5
Enter the lower range: 0
Enter the upper range: 7
Enter an integer between the given range: 5
The number guessed is low
Enter an integer between the given range: 6
The number guessed is right
Total guesses taken: 2
```

We generated the program in **Python 3.** Here are the instructions for making a number guessing game in **Python**:

1. We initially asked the user to choose the range for the number to be created. A random integer is created using the randint() function from the unexpected package.
2. We started a variable with 0 to keep track of the total predictions made.
3. We executed the while loop until the number estimated was not equal to the actual number.
4. We used an if-else ladder to check whether the estimated number is lower or greater than the actual number and increase the total guesses in each iteration.
5. We broke out of the loop when the estimate matched the number.

6. We displayed the total guesses made when the guess was accurate.

Similarly **replicating** the logic, we may make this game in Python 2 or any other computer language.

PART 32: Python Game : Rock, Paper, Scissors

Game programming is a fun way to learn any programing language. In this tutorial, we will learn how to code a simple Rock, Paper, and Scissors game.

Winning Rules in Rock, Paper, and Scissors as follows :

- **Rock vs paper-> paper wins**
- **Rock vs scissor-> Rock wins**
- **paper vs scissor-> scissor wins.**

Rock, Paper, and Scissors Source Code

Here is the complete code for the project, that you can run and see how it works.

To make the above code run again and again in a loop, add the following code at the end of the while loop code in the above code. The code below will ask the user, after 1 game, if they want to play again, and if the user enters **y**, then the game will restart.

You can try this code in your laptop locally and play the game again and again.

```
ch = input("do you wish to continue(y/n)") #choice for continuing loop
if ch=="y":
    player_choice=1
else:
```

```
    break
```

To understand the flow, the code has been divided into the following contents-

1. Assign a choice to computer

```
t = [“Rock”, “Paper”, “Scissors”]
#random choice for computer
comp_choice = t[randint(0,2)]
```

Import random module to use randomint() function. We assign a play option for the computer. t is a list of possible play options. randomint(0,2) will randomly generate a number from the given range each time it is called. It can give 0,1 or 2 according to our code.

comp_choice will store the value as

t[0] = “Rock”

t[1]=“Paper”

t[2]=“Scissor”

2. Take input from the player

```
player_choice = input(“Chose Rock, Paper, Scissors?”)
```

Players will give an input of their choice

3. Using while loop to play multiple rounds

Take a look at the while body

```
?
while player_choice == 1:
#player choses its option
    player_choice = input(“Chose Rock, Paper, Scissors?”)
    comp_choice = t[randint(0,2)]
```

```
    if player_choice == comp_choice:
        print("Tie!")
        print("Score")
        print("computer win:",computer_win)
        print("player win:",player_win)
    elif player_choice == "Rock":
        if comp_choice == "Paper":
            print("You lose! computer chose", comp_choice,
"player chose", player_choice)
            computer_win+=1
            print("Score")
            print("computer win:",computer_win)
            print("player win:",player_win)
        else:
            print("You win! player chose", player_choice,
"computer chose", comp_choice)
            player_win+=1
            print("Score")
            print("computer win:",computer_win)
            print("player win:",player_win)
    elif player_choice == "Paper":
        if comp_choice == "Scissors":
            print("You lose! computer chose", comp_choice,
"player chose", player_choice)
            computer_win+=1
            print("Score")
            print("computer win:",computer_win)
            print("player win:",player_win)
        else:
```

```
            print("You win! player chose", player_choice, "computer chose", comp_choice)
            player_win+=1
            print("Score")
            print("computer win:",computer_win)
            print("player win:",player_win)
    elif player_choice == "Scissors":
        if comp_choice == "Rock":
            print("You lose computer chose", comp_choice, "player chose", player_choice)
            computer_win+=1
            print("Score")
            print("computer win:",computer_win)
            print("player win:",player_win)
        else:
            print("You win! player chose", player_choice, "computer chose", comp_choice)
            player_win+=1
            print("Score")
            print("computer win:",computer_win)
            print("player win:",player_win)
    else:
        print("That's not a valid play. Check your spelling!")
    print("")
    ch=input("do you wish to continue(y/n)") #choice for continuing loop
    if ch=="y":
        player_choice=1
    else:
```

```
        break
```

When the loop starts, the player's choice and computer's choice are compared and whoever wins is given a point.

4. Display Score

```
computer_win=0
player_win=0
```

player_win and computer_win are initialised at 0. These variables will store the scores.

5. Option to play again or quit

```
    ch=input("do you wish to continue(y/n)") #choice for continuing loop
    if ch=="y":
        player_choice=1
        comp_choice = t[randint(0,2)]
    else:
        break
```

Player is given the choice to continue or leave. Player's choice is stored in a variable ch. If it is 'y' then player_choice is set to 1 to continue the loop, otherwise the control executes break and comes out of the loop.

Output-

Here we have played four rounds with the computer. After each round scores are printed for the player and computer.

```
Chose Rock, Paper, Scissors?Rock

Tie!

Score

computer win: 0

player win: 0
```

do you wish to continue(y/n)y

Chose Rock, Paper, Scissors?Paper

You lose! computer chose Scissors player chose Paper

Score

computer win: 1

player win: 0

do you wish to continue(y/n)y

Chose Rock, Paper, Scissors?Paper

Tie!

Score

computer win: 1

player win: 0

do you wish to continue(y/n)y

Chose Rock, Paper, Scissors?Scissors

You win! player chose Scissors computer chose Paper

Score

computer win: 1

player win: 1

do you wish to continue(y/n)n

PART 33: Desktop Notifier Application Python Project

Have you ever attempted to design a desktop notification program that meets your specific requirements? Achieve you know that you can do this using Python in just a few steps?

Don’t worry, we’ll start from the beginning; in this post, we’ll create a desktop notification application for tracking the coronavirus’s daily statistics.

What You’ll Discover In This Article

- Installing the Python packages that are necessary.
- Obtaining coronavirus info via the internet.
- Making a desktop notification program.
- Making your software run in the background is a great way to save time.

Let’s get going.

We need to obtain two vital python packages for this application while installing needed python packages.

Note: If you’re using Windows, execute these two instructions at the command prompt; if you’re using Linux, type them in the terminal (for fetching data from the web)

```
pip install requests
```

```
pip install plyer
```

We may get coronavirus data from the web by using the URL supplied below; you are free to substitute the nation name with your own, but we will be utilizing India's coronavirus data for this application.

```
https://corona-rest-api.herokuapp.com/Api/india
```

Now that we have all of the tools we need to construct this app, let's get to work coding it.

Note: It will be easier if you code this in offline compiler rather than online compiler because we will be making this application run as a background process in your PC in the later stages of this article; if you code in online compiler, you will need to download the file, which is not required in offline compiler. I recommend that you use Visual Studio.

Step 1: Importing Libraries

```
import datetime #for reading present date
import time
import requests #for retreiving coronavirus data from web
from plyer import notification #for getting notification on your PC
```

Step 2: Retrieving the Data From The Web

```
#let there is no data initially
covidData = None
try:
    covidData = requests.get("https://corona-rest-api.herokuapp.com/Api/india")
except:
    #if the data is not fetched due to lack of internet
    print("Please! Check your internet connection")
```

Step 3: Creating Custom Notification

```
#if we fetched data
if (covidData != None):
    #converting data into JSON format
    data = covidData.json()[‘Success’]
    #repeating the loop for multiple times
    while(True):
        notification.notify(
            #title of the notification,
            title = “COVID19 Stats on
{}”.format(datetime.date.today()),
            #the body of the notification
            message = “Total cases : {totalcases}\nToday cases :
{todaycases}\nToday deaths :{todaydeaths}\nTotal active :
{active}”.format(
                    totalcases = data[‘cases’],
                    todaycases = data[‘todayCases’],
                    todaydeaths = data[‘todayDeaths’],
                     active = data[“active”]),
            #creating icon for the notification
            #we need to download a icon of ico file format
            app_icon = “Paomedia-Small-N-Flat-Bell.ico”,
            # the notification stays for 50sec
            timeout  = 50
        )
        #sleep for 4 hrs => 60*60*4 sec
        #notification repeats after every 4hrs
        time.sleep(60*60*4)
```

That’s it; we’re ready to launch our program. Before we do, however, you should be aware of several adjustments you may make to tailor your application to your specific requirements.

timeout — specifies how long a notice should appear on the desktop. **sleep ()**— specifies the time period after which the notice should appear.

After you've ran your app, this is how you'll receive your notice.

Having your application operate in the background is a great way to save time.

Source Code for Desktop Notifier Application

```
import datetime #for reading present date
import time
import requests #for retreiving coronavirus data from web
from plyer import notification #for getting notification on your PC
#let there is no data initially
covidData = None
try:
    covidData = requests.get("https://corona-rest-api.herokuapp.com/Api/india")
except:
    #if the data is not fetched due to lack of internet
    print("Please! Check your internet connection")
#if we fetched data
if (covidData != None):
    #converting data into JSON format
    data = covidData.json()['Success']
    #repeating the loop for multiple times
    while(True):
        notification.notify(
```

```
            #title of the notification,
            title = "COVID19 Stats on
{}".format(datetime.date.today()),
            #the body of the notification
            message = "Total cases : {totalcases}\nToday cases :
{todaycases}\nToday deaths :{todaydeaths}\nTotal active :
{active}".format(
                        totalcases = data['cases'],
                        todaycases = data['todayCases'],
                        todaydeaths = data['todayDeaths'],
                        active = data["active"]),
            #creating icon for the notification
            #we need to download a icon of ico file format
            app_icon = "Paomedia-Small-N-Flat-Bell.ico",
            # the notification stays for 50sec
            timeout  = 50
        )
        #sleep for 4 hrs => 60*60*4 sec
        #notification repeats after every 4hrs
        time.sleep(60*60*4)
```

Output:

What is the best way to have a Python program run in the background?

Simply execute this command in command prompt in Windows or terminal in Linux to have your program operate in the background. Note that you must type this command in command prompt in Windows and terminal in Linux.

Note: your-file-name-here> should be replaced with pythonw.exe.your-file-name-here>.

```
pythonw.exe .\<your-file-name-here>
```

example

```
pythonw.exe .\desktopNotifier.py
```

That's all there is to it; your program will now operate in the background.

How Can I Turn Off Notifications?

It's as easy as killing the python process in Task Manager. If you have any trouble stopping the notice, please share your experience in the comments area of this page. There are a variety of other situations in which you may use this strategy.

- **Every day, you will be reminded to take your medication.**
- **The need to drink water is reminded every hour.**

and many more; how you utilize this program is entirely up to you.

Conclusion

This program is compatible with any operating system, including Windows, Linux, and Mac. Please feel free to ask in the comments area of this post if you desire a simpler desktop notification program.