

2020

Tkinter Notes

Creating the GUI using Python

In this Book, you are going to learn how to create GUI apps in Python using Tkinter. You'll also learn about all the elements needed to develop GUI apps in Python.



Learn Tkinter

In this course, you are going to learn how to create GUI apps in Python using Tkinter. You'll also learn about all the elements needed to develop GUI apps in Python.

After completing this course, you will be...

- Able to design own GUI using Tkinter module
- Bag strong fundamentals required to create great GUI

Course Contents

1. Notepad & Calculator in Tkinter + Why Tkinter
2. Our first Tkinter GUI
3. Tkinter Widgets and Attributes
4. Label, Geometry, Maxsize & Minsize
5. Displaying images Using Label
6. Attributes of Label and Pack
7. Creating Newspaper GUI
8. Frame in Tkinter
9. Packing Buttons in Tkinter
10. Entry Widget & Grid Layout in Tkinter
11. Travel from Using Checkbuttons & Entry Widget
12. Accepting User Input in Tkinter Form
13. Canvas Widget in Python Tkinter
14. Handling Events In Tkinter GUI
15. Python GUI Exercise 1: Solution
16. Python GUI Exercise 2: Window Resizer GUI()
17. Menus and Submenus in Tkinter Python
18. Message Box in Tkinter Python
19. Sliders in Tkinter Using Scale()
20. Creating RadioButtons in Tkinter
21. ListBox in Tkinter
22. ScrollBar in Tkinter GUI
23. Tkinter GUI Exercise 2 Solution
24. Status Bar in Tkinter
25. Using Classes and Objects to Create GUIs
26. More Tkinter Tips, Tricks and Functions
27. Creating a Calculator Using Tkinter
28. Tkinter GUI Text Editor Announcement
29. Creating a GUI Notepad in Tkinter
30. Tkinter Tutorials Conclusions + Resources

Requirements

- Basic knowledge of Python programming

Description

This course is free of cost. Take advantage of this course and enhance your skills in GUI Python programming!!!

Most of us write code and run it in a command-line terminal or an IDE (Integrated Development Environment). The code produces an output based on what you expect out of it, either on the terminal or on the IDE itself. However, if we want our system to have a fancy looking user-interface, our application (use-case) requires having a GUI. GUI is nothing but a desktop app that provides us with an interface which helps us to interact with the computers and enriches our experience of giving a command (command-line input) to our code. Some of the applications like Chrome, Firefox, Microsoft Edge, Text-Editors, Sudoku, etc. utilize the power of GUIs

Why Tkinter?

Tkinter is a built-in Python module for developing a GUI application. It's easy to use and shipped as a package with Python. We can visualize our data with GUI applications.

Why Tkinter?

Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the *Tk* GUI toolkit. Creating a GUI application using Tkinter is an easy task.

- Import the module *Tkinter*
- Create the GUI application's main window.
- Add one or more widgets like Button, Radiobutton, Checkbutton, Canvas, Menubutton, etc.
- Apply the event Trigger on the widgets.
- All Tkinter widgets have access to specific geometry management methods that have the purpose of organizing widgets throughout the parent widget area.
- We can use standard attributes like color, size, etc.

GUI Calculator:

GUI based simple calculator can be created using Python Tkinter module ([Creating a Calculator Using Tkinter](#)), which can perform basic arithmetic operations i.e. addition, subtraction, multiplication, division etc.

GUI Notepad:

GUI based notepad can be created using Python Tkinter module([Creating a GUI Notepad in Tkinter](#)) which has three main menu items: File, Edit & Help. The file menu item will have four sub-items- New, Open, Save & Exit. In this GUI Notepad the user can make and save new file and open any file for further use. Many more operations can be created in Notepad using GUI.

Our First Tkinter GUI

Tkinter is the standard GUI library for Python. Here, we'll start to develop our first Tkinter GUI. Tkinter provides a powerful object-oriented interface to the *Tk* GUI toolkit.

Let's start with easy code to know the basics of Tkinter. All you need to do is perform the following steps –

- Import the *Tkinter*
- Create the GUI application's main window.
- Add one or more widgets to the GUI application (we'll discuss these later more specifically)
- Enter the main event loop to take action against each event triggered by the user.

Code is described below:

```
from tkinter import *  
rahulrathva_root = Tk()  
# gui logic here  
rahulrathva_root.mainloop()
```

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

```
from tkinter import *
```

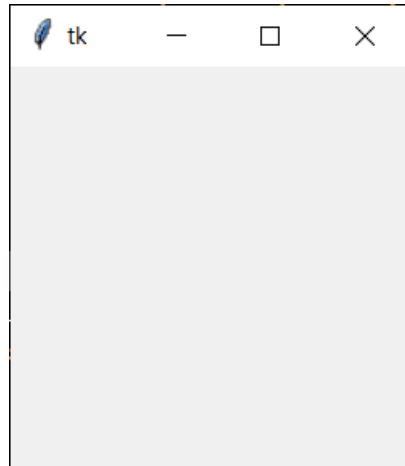
- To create the main window, tkinter offers a method '*Tk*'. To change the name of the window, you can change the *className* to the desired one.

```
rahulrathva_root = Tk()
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

Note: There are currently 15 types of widgets in Tkinter. Button, Canvas, RadioButton, CheckButton, Menu, Frame, Label, etc. are different types of widgets used in Tkinter.

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *  
rahulrathva_root = Tk()  
# gui logic here  
rahulrathva_root.mainloop()
```

Question - Other than Tkinter what are the other ways to create a GUI in python

Tkinter Widgets & Attributes

- The **Graphical user interface(GUI)** is used by the most commercially popular computer operating systems and software programs today. It's the kind of interface that allows users to manipulate elements on the screen using a mouse, a stylus, or even a finger.
- **Widgets** are the basic building blocks for graphical user interface (GUI) applications. Each GUI component (e.g. buttons, labels) is a *widget* that is placed somewhere within a user interface window or is displayed as an independent window.
- *Tkinter* also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes.
 1. **pack() method:** It organizes the widgets in blocks before placing in the parent widget.
 2. **grid() method:** It organizes the widgets in the grid (table-like structure) before placing in the parent widget.
 3. **place() method:** It organizes the widgets by placing them on specific positions directed by the programmer.
- In Python, almost each widget object has several *attributes*. Here, we'll talk about standard widget attributes, including cursors, reliefs, colors, and fonts.
 1. **Tkinter Widget state:** The state of the widget is defined by state attributes. **NORMAL, ACTIVE, and DISABLED** are the values of the attributes.
 2. **Tkinter Widget padding:** *padx* and *pady* –these two attributes come under this which, respectively, add extra horizontal and vertical space to the widget. The *padx* and *pady* attributes add space between the buttons.
 3. **Tkinter Background colors:** The background colors of widgets can be set with background attribute. It can be abbreviated to **bg**.
 4. **Width & Height:** The *width* and *height* attributes set the width and height of the widget.
 5. **Tkinter Fonts:** For working with fonts it has a *font* module. It has some built-in fonts such as TkToolTipFont, TkDefaultFont.
 6. **Tkinter Cursors:** The *cursor* in Tkinter is set with the cursor
 7. **Tkinter Reliefs:** A *relief* is a border decoration. The possible values are **SUNKEN, RAISED, GROOVE, RIDGE, and FLAT**.

Label, Geometry, Maxsize & Minsize

Label(): A Label is a Tkinter Widget class, which is used to display text or an image. The label is a widget that the user just views but does not interact with.

Geometry(): This method is used to set the dimensions of the Tkinter window and is used to set the position of the main window on the user's desktop.

Minsize(): This method is used to set the minimum size of the Tkinter window. Using this method user can set the window's initialized size to its minimum size, and still be able to maximize and scale the window larger.

Maxsize(): This method is used to set the **maximum size** of the Tkinter window i.e. maximum size a window can be expanded. Users will still be able to shrink the size of the window to the minimum possible.

Code is described below:

```
from tkinter import *
imaginary_tech_root = Tk()
# Width x Height
imaginary_tech_root.geometry("644x434")
# width, height
imaginary_tech_root.minsize(300,100)
# width, height
imaginary_tech_root.maxsize(1200, 988)
shakaib = Label(text="Shakaib is a good boy and this is his GUI")
shakaib.pack()
imaginary_tech_root.mainloop()
```

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

```
from tkinter import *
```

- To create the main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
imaginary_tech_root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the width is 644 pixels and the height is 434 pixels so we can write the function as *geometry(644x434)*.

```
imaginary_tech_root.geometry("644x434")
```

- To set the minimum size of the Tkinter window we use minsized() function and to set the maximum size of the Tkinter window we use maxsize() As in the example: minsized(300,100) depicts the minimum width and height of the Tkinter window must be 300 pixels and 100 pixels respectively. In the same way, maxsize(1200,988) depicts the maximum width, and

the height of the Tkinter window must be 1200 pixels and 988 pixels respectively. **Note: We have to put a comma between width and height, unlike geometry() function.**

```
imaginary_tech_root.minsize(300,100)
imaginary_tech_root.maxsize(1200, 988)
```

- To call the Label widget which is a child of the root widget. The keyword parameter "text" specifies the text "Shakaib is a good boy and this is his GUI" to be shown:

```
shakaib = Label(text="Shakaib is a good boy and this is his GUI")
```

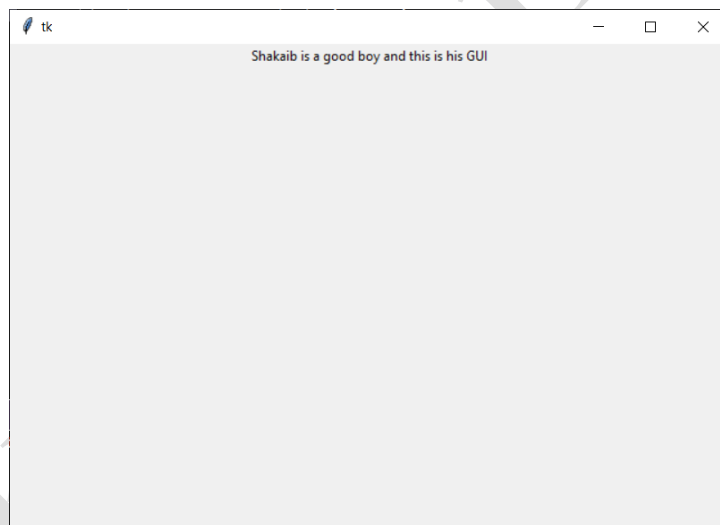
- The pack method tells Tk to fit the size of the window to the given text.

```
shakaib.pack()
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur, and process the event as long as the window is not closed.

```
imaginary_tech_root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *

imaginary_tech_root = Tk()

# Width x Height
imaginary_tech_root.geometry("644x434")

# width, height
imaginary_tech_root.minsize(300,100)

# width, height
```

```
imaginary_tech_root.maxsize(1200, 988)

shakaib = Label(text="Shakaib is a good boy and this is his GUI")
shakaib.pack()

imaginary_tech_root.mainloop()
```

TKinter notes

Displaying Images Using Label

We perhaps add one or more images to make our GUI more creative and presentable. The **PhotoImage()** class is used to display images in labels, buttons, canvases, and text widgets. You can use the PhotoImage() class whenever you need to display an icon or an image in a Tkinter application. This being said, let's see how to see an image using with Python in the shortest way.

Code is described below:

```
from tkinter import *
mahmudul_root = Tk()
mahmudul_root.geometry("1255x944")
photo = PhotoImage(file="1.png")
varun_label = Label(image=photo)
varun_label.pack()
mahmudul_root.mainloop()
```

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

```
from tkinter import *
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
mahmudul_root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, **geometry()** function is used. As in example: the width is 1255 pixels and height is 944 pixels so we can write the function as **geometry(1255x944)**.

```
mahmudul_root.geometry("1255x944")
```

- To display an image i.e. "1.png" in a Tkinter application, the PhotoImage() class is used.

```
photo = PhotoImage(file="1.png")
```

- To call the Label widget which is a child of the root widget. The keyword parameter "**image**" specifies the *photo* "**1.png**" to be shown:

```
varun_label = Label(image=photo)
```

- The pack method tells Tk to fit the size of the window to the given image.

```
varun_label.pack()
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.\

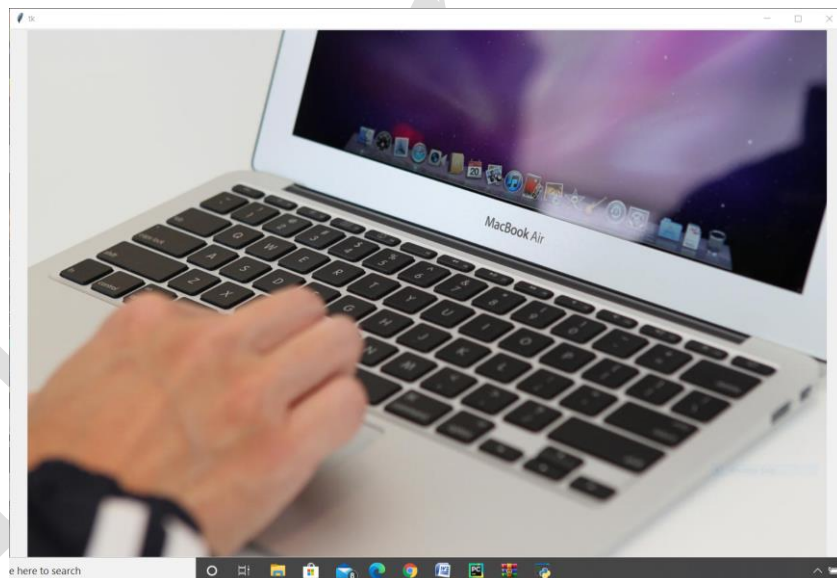
```
mahmudul_root.mainloop()
```

Note: If we need to work with other file formats (example: .jpg) the **Python Imaging Library (PIL)** contains classes that lets you load images in over 30 formats, and convert them to Tkinter-compatible image objects. So in this case, we have to install *pillow* library (if it is not installed in the system) by writing `pip install pillow` in the terminal and then write the code as follows:

```
from tkinter import *
from PIL import Image, ImageTk
mahmudul_root = Tk()
mahmudul_root.geometry("1255x944")
image = Image.open("1.jpg")
photo = ImageTk.PhotoImage(image)
varun_label = Label(image=photo)
varun_label.pack()
mahmudul_root.mainloop()
```

- Import **Image** and **ImageTk** from PIL library.
- Take *image* as a variable when the image "**1.jpg**" will be opened.
- Take *photo* as another variable to store the image using *PhotoImage()* class.
- Call the `Label()` widget and pack it.
- Write the `mainloop()` method to run the application.

Output: *The output of the code (or the GUI window) is given below:*



Code as described/written in the video

```
from tkinter import *
from PIL import Image, ImageTk

mahmudul_root = Tk()

mahmudul_root.geometry("1255x944")
# photo = PhotoImage(file="1.png")

# For Jpg Images
```

```
image = Image.open("photo.jpg")
photo = ImageTk.PhotoImage(image)

varun_label = Label(image=photo)
varun_label.pack()

mahmudul_root.mainloop()
```

TKinter notes

Attributes Of Label & Pack

- **Attributes:** A set of properties of a widget that defines its visual appearance on the computer screen and how it responds to user events. Here we'll discuss about the **attributes** of Label and Pack.
- **Attributes of Label:** The **Label** widget is a standard Tkinter widget used to display a text or image on the screen. There are a lot of attributes of Label widget. Some important attributes are discussed below:
 1. **bg:** The normal background color displayed behind the label and indicator.
 2. **fg:** This option specifies the color of the text (foreground color). If you are displaying a bitmap, this is the color that will appear at the position of the 1-bits in the bitmap.
 3. **padx:** Extra space added to the left and right of the text within the widget. **Default is 1.**
 4. **pady:** Extra space added above and below the text within the widget. **Default is 1.**
 5. **relief:** Specifies the appearance of a decorative border around the label. There are five types of reliefs, such that FLAT, RAISED, SUNKEN, GROOVE, RIDGE. **The default is FLAT.**
 6. **font:** If you are displaying text in this label (with the text or textvariable option), the font option specifies the style, size and other characteristics (i.e. bold, italic etc.) of the font and in this style the text will be displayed.
 7. **text:** To display one or more lines of text in a label widget, set this option to a string containing the text. Internal newlines ("\\n") will force a line break.
 8. **justify:** Specifies how multiple lines of text will be aligned with respect to each other: **LEFT** for flush left, **CENTER** for centered (**the default**), or **RIGHT** for right-justified.
 9. **height:** The vertical dimension of the new frame.
 10. **width:** The horizontal dimension of the new frame. If this option is not set, the label will be sized to fit its contents.
- **Attributes of Pack:** The Pack geometry manager packs widgets in rows or columns. We can use options like **fill**, **expand**, and **side** to control this geometry manager.
 1. **fill:** Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically).
 2. **side:** Determines which side of the parent widget packs against: TOP, BOTTOM, LEFT, or RIGHT. **The default is TOP.**
 3. **anchor:** Anchors are used to define where text is positioned relative to a reference point. The anchor attributes are: n, s, e, w, center, nw, sw, ne and se. **The default is center.**

Code is described below (using attributes of Label & Pack):

```
# padx - x padding
# pady - y padding
# relief - border styling - SUNKEN, RAISED, GROOVE, RIDGE

title_label = Label(text = '''
Abdul Rashid Salim Salman Khan is an Indian
\\nfilm actor, producer, occasional playback singer and television
personality. In a film career spanning
```

```

\almost thirty years, Khan has received numerous awards, including two
National Film Awards as a film
\nproducer, and two Filmfare Awards for acting. He has a significant
following in Asia and the Indian
\ndiaspora worldwide, and is cited in the media as one of the most
commercially successful actors of Indian
\ncinema. According to the Forbes 2018 list of Top-Paid 100 Celebrity
Entertainers in world, Khan was
\nthe highest ranked Indian with 82nd rank with earnings of $37.7
million.'''', bg="red", fg="white", padx=13, pady=94, font="comicsansms 9
bold", borderwidth=3, relief=SUNKEN)

```

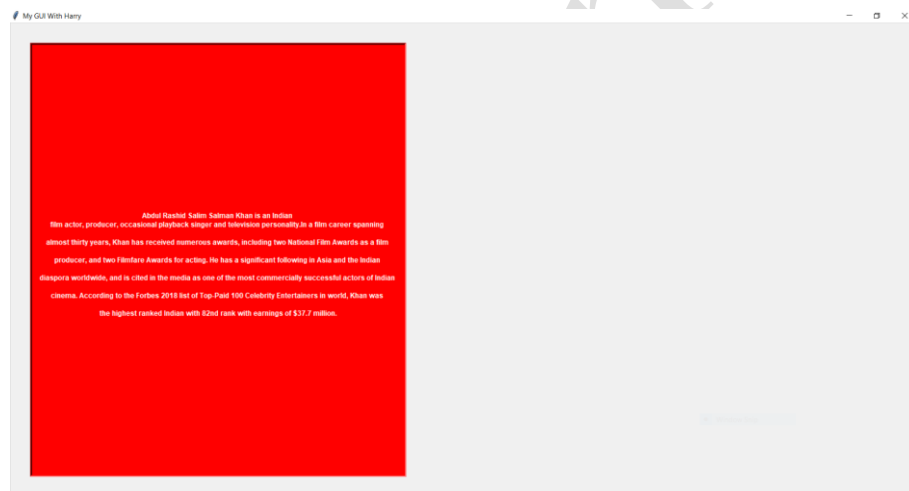
```

# Important Pack options
# anchor = nw
# side = top, bottom, left, right
# fill
# padx
# pady

# title_label.pack(side=BOTTOM, anchor="sw", fill=X)
title_label.pack(side=LEFT, fill=Y, padx=34, pady=34)
root.mainloop()

```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```

from tkinter import *
root = Tk()
root.geometry("744x133")
root.title("My GUI With Harry")

# Important Label Options
# text - adds the text
# bd - background
# fg - foreground
# font - sets the font
# 1. font=("comicsansms", 19, "bold")
# 2. font="comicsansms 19 bold"

# padx - x padding
# pady - y padding
# relief - border styling - SUNKEN, RAISED, GROOVE, RIDGE

```

```

title_label = Label(text ='''
Abdul Rashid Salim Salman Khan is an Indian \nfilm actor, producer,
occasional playback singer and television personality. In a film career
spanning \nalmost thirty years, Khan has received numerous awards,
including two National Film Awards as a film \nproducer, and two Filmfare
Awards for acting. He has a significant following in Asia and the Indian
\ndiaspora worldwide, and is cited in the media as one of the most
commercially successful actors of Indian \ncinema. According to the Forbes
2018 list of Top-Paid 100 Celebrity Entertainers in world, Khan was \nthe
highest ranked Indian with 82nd rank with earnings of $37.7 million.''' , bg
="red", fg="white", padx=13, pady=94, font="comicsansms 9 bold",
borderwidth=3, relief=SUNKEN)

# Important Pack options
# anchor = nw
# side = top, bottom, left, right
# fill
# padx
# pady

# title_label.pack(side=BOTTOM, anchor ="sw", fill=X)
title_label.pack(side=LEFT, fill=Y, padx=34, pady=34)

root.mainloop()

```


Exercise 1: Creating Newspaper GUI

Problem Statement

Create a Newspaper GUI using Tkinter in Python

-

Hints

Use the Tkinter module for GUI purposes.

Show text and images to tell any news to the reader by using Label and Pack attributes or options.

Tkinter notes

Frame In Tkinter

- A **Frame** widget is a rectangular region on the screen which can be used as a foundation class to implement complex widgets. This widget is very important for the process of grouping and organizing other widgets in a friendly way. It works like a container, which is responsible for arranging the position of other widgets. We can use *bg*, *relief*, *borderwidth*, *bd* as the attributes of Frame widget. Some important attributes are discussed below:
- **bg**: The normal background color displayed behind the label and indicator.
- **relief**: The type of the border of the frame. **Its default value is set to FLAT**. We can set it to any other styles i.e. *FLAT*, *RAISED*, *SUNKEN*, *GROOVE*, *RIDGE*.
- **borderwidth**: TkinterLabel **doesn't have any border by default**. We need to assign the borderwidth option to add a border around Label widget along with the relief option to be any option rather than flat to make visible.
- **bd**: The size of the border around the indicator. **Default is 2 pixels**.

Code is described below:

```
from tkinter import *
root = Tk()
root.geometry("655x333")
f1 = Frame(root, bg="grey", borderwidth=6, relief=SUNKEN)
f1.pack(side=LEFT, fill="y")

f2 = Frame(root, borderwidth=8, bg="grey", relief=SUNKEN)
f2.pack(side=TOP, fill="x")

l = Label(f1, text="Project Tkinter - Pycharm")
l.pack(pady=142)

l = Label(f2, text="Welcome to sublime text", font="Helvetica 16 bold",
fg="red", pady=22)
l.pack()

root.mainloop()
```

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

```
from tkinter import *
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the width is 655 pixels and height is 333 pixels so we can write the function as *geometry(655x333)*.

```
root.geometry("655x333")
```

- To take Frame variable as *f1* and set the attributes- *bg* (background color) =“**grey**”, *borderwidth*(thickness of the Frame’s border)=**6** and *relief*=**SUNKEN**. Then the Frame *f1* must be packed.

```
f1 = Frame(root, bg="grey", borderwidth=6, relief=SUNKEN)
f1.pack(side=LEFT, fill="y")
```

- In the same way, another Frame *f2* is taken and the attributes are set to the Frame.

```
f2 = Frame(root, borderwidth=8, bg="grey", relief=SUNKEN)
f2.pack(side=TOP, fill="x")
```

- To call the Label widget which is a child of the root widget. The keyword parameter "text" specifies the *text* “Project Tkinter - Pycharm” to be shown and the pack method tells Tk to fit the size of the window to the given text where *pady* adds extra space (i.e. 142 in this example) from the upper and lower portion of the Frame widget.

```
l = Label(f1, text="Project Tkinter - Pycharm")
l.pack( pady=142)
```

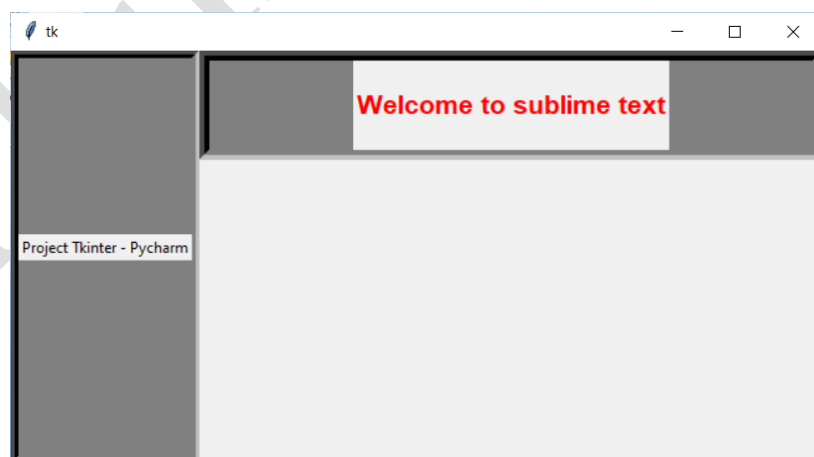
- In the same way, another label is taken and the attributes are set to the Label.

```
l = Label(f2, text="Welcome to sublime text", font="Helvetica 16 bold",
fg="red", pady=22)
l.pack()
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *
root = Tk()
root.geometry("655x333")
f1 = Frame(root, bg="grey", borderwidth=6, relief=SUNKEN)
f1.pack(side=LEFT, fill="y")

f2 = Frame(root, borderwidth=8, bg="grey", relief=SUNKEN)
f2.pack(side=TOP, fill="x")

l = Label(f1, text="Project Tkinter - Pycharm")
l.pack(pady=142)

l = Label(f2, text="Welcome to sublime text", font="Helvetica 16 bold",
fg="red", pady=22)
l.pack()

root.mainloop()
```

Packing Buttons In Tkinter

The **Button** widget is a standard Tkinter widget, which is used for various kinds of buttons. A button is a widget which is designed for the user to interact with (i.e. if the button is pressed by mouse click some action might be started). They can also contain text and images like labels. While labels can display text in various fonts, a button can only display text in a single font. The text of a button can span more than one line.

A Python function or method can be associated with a button. This function or method will be executed, if the button is pressed in some way. We can use *bd*, *bg*, *command*, *fg*, *text*, *image*, *relief* etc. as attributes of Button widget. Some important attributes are discussed below:

- **bg**: The normal background color displayed behind the label and indicator.
- **relief**: The type of the border of the frame. **Its default value is set to FLAT**. We can set it to any other styles i.e. *FLAT*, *RAISED*, *SUNKEN*, *GROOVE*, *RIDGE*.
- **bd**: The size of the border in pixels. **Default is 2 pixels**.
- **font**: Text font to be used for the button's label.
- **image**: Instead of text Image to be displayed on the button.
- **command**: Function or method to be called when the button is clicked. **Note: we only have to write the function or method name in the command attribute. We should not call the function in the attribute (i.e. If the function is *name()*, we just have to write *command=name*).**

Code is described below:

```
from tkinter import *
```

```
root =Tk()  
root.geometry("655x333")
```

```
def hello():  
    print("Hello tkinter Buttons")
```

```
def name():  
    print("Name is harry")
```

```
frame = Frame(root, borderwidth=6, bg="grey", relief=SUNKEN)  
frame.pack(side=LEFT, anchor="nw")
```

```
b1 = Button(frame, fg="red", text="Print now", command=hello)  
b1.pack(side=LEFT, padx=23)
```

```
b2 = Button(frame, fg="red", text="Tell me name now", command=name)  
b2.pack(side=LEFT, padx=23)
```

```
b3 = Button(frame, fg="red", text="Print now")  
b3.pack(side=LEFT, padx=23)
```

```
b4 = Button(frame, fg="red", text="Print now")  
b4.pack(side=LEFT, padx=23)  
root.mainloop()
```

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

```
from tkinter import *
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
root =Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the width is 655 pixels and height is 333 pixels so we can write the function as *geometry(655x333)*.

```
root.geometry("655x333")
```

- To define a function using 'def' (i.e. here two functions hello() and name() are defined) and use it in button attributes.

```
def hello():
    print("Hello tkinter Buttons")
```

```
def name():
    print("Name is harry")
```

- To take Frame variable as *frame* and set the attributes- *bg* (background color) = "**grey**", *borderwidth* (thickness of the Frame's border) = **6** and *relief* = **SUNKEN**. Then the Frame *frame* must be packed (here it is packed in left side and north-west corner).

```
frame = Frame(root, borderwidth=6, bg="grey", relief=SUNKEN)
frame.pack(side=LEFT, anchor="nw")
```

- To take the Button variable (here we take four Button variables b1, b2, b3 & b4) and use attributes in the Button widget (i.e. for Button b1, the attributes are fg (foreground color) = "**red**", text= "Print now" and command= hello, such that when the Button name showing as "Print now" is clicked it will call the hello() function through the command "hello" attribute and it will print "Hello tkinter Buttons".

```
b1 = Button(frame, fg="red", text="Print now", command=hello)
b1.pack(side=LEFT, padx=23)
```

```
b2 = Button(frame, fg="red", text="Tell me name now", command=name)
b2.pack(side=LEFT, padx=23)
```

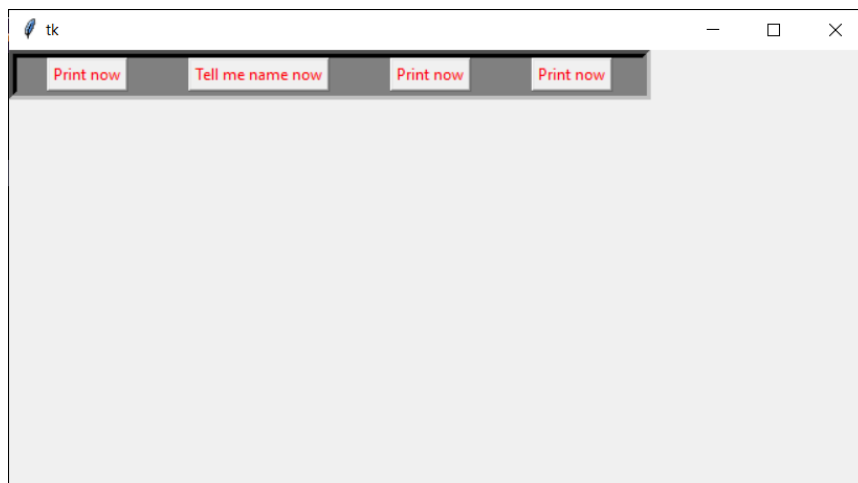
```
b3 = Button(frame, fg="red", text="Print now")
b3.pack(side=LEFT, padx=23)
```

```
b4 = Button(frame, fg="red", text="Print now")
b4.pack(side=LEFT, padx=23)
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *

root = Tk()
root.geometry("655x333")

def hello():
    print("Hello tkinter Buttons")

def name():
    print("Name is harry")

frame = Frame(root, borderwidth=6, bg="grey", relief=SUNKEN)
frame.pack(side=LEFT, anchor="nw")

b1 = Button(frame, fg="red", text="Print now", command=hello)
b1.pack(side=LEFT, padx=23)

b2 = Button(frame, fg="red", text="Tell me name now", command=name)
b2.pack(side=LEFT, padx=23)

b3 = Button(frame, fg="red", text="Print now")
b3.pack(side=LEFT, padx=23)

b4 = Button(frame, fg="red", text="Print now")
b4.pack(side=LEFT, padx=23)
root.mainloop()
```

Tkinter notes

Entry Widget & Grid Layout In Tkinter

- **Entry() widgets** are the basic widgets of Tkinter which is used to get input, i.e. text strings, from the user of an application. This widget allows the user to enter a single line of text.

Note: **If the user enters a string, which is longer than the available display space of the widget, the content will be scrolled.** This means that the string cannot be seen in its entirety. The arrow keys can be used to move to the invisible parts of the string. **If you want to enter multiple lines of text, you have to use the text widget. An entry widget is also limited to single font.**

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

Note: **We can also create layouts using Pack() manager** but it takes a number of extra frame widgets, and a lot of work to make things look good. If you use the grid manager instead, you only need one call per widget to get everything laid out properly.

Code is described below:

```
from tkinter import *

def getvals():
    print(f"The value of username is {uservalue.get()}")
    print(f"The value of password is {passvalue.get()}")

root = Tk()
root.geometry("655x333")

user = Label(root, text="Username")
password = Label(root, text="Password")
user.grid()
password.grid(row=1)

# Variable classes in tkinter
# BooleanVar, DoubleVar, IntVar, StringVar

uservalue = StringVar()
passvalue = StringVar()

userentry = Entry(root, textvariable = uservalue)
passentry = Entry(root, textvariable = passvalue)

userentry.grid(row=0, column=1)
passentry.grid(row=1, column=1)

Button(text="Submit", command=getvals).grid()

root.mainloop()
```

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

```
from tkinter import *
```

- To define a function 'def' (i.e. here the function *getvals()* is defined) is used and use the *get()* method on the Entry objects (i.e. *uservalue* and *passvalue*).

```
def getvals():
    print(f"The value of username is {uservalue.get()}")
    print(f"The value of password is {passvalue.get()}")
```

- To create a main window, *tkinter* offers a method '*Tk*'. To change the name of the window, you can change the *className* to the desired one.

```
root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, *geometry()* function is used. As in example: the width is 655 pixels and height is 333 pixels so we can write the function as *geometry(655x333)*.

```
root.geometry("655x333")
```

- To call the Label widget which is a child of the root widget. The keyword parameter "text" specifies the text "Username" for the variable *user* and "Password" for the variable *password* to be shown:

```
user = Label(root, text="Username")
password = Label(root, text="Password")
```

- To use the *grid()* method for placing the labels in proper place by passing attributes row and column. If we don't pass anything in the *grid()* method the default is 0 for both row and column (Here for *user* we pass nothing in *grid()* so row=0 and column=0 and for *password* we pass row=1 so row=1 and column=0).

```
user.grid()
password.grid(row=1)
```

- To take the variables and set them to the variable classes (i.e. *StringVar()*). The attribute *textvariable* of *Entry()* widget is used to allow the user to enter some text.

```
uservalue = StringVar()
passvalue = StringVar()
```

```
userentry = Entry(root, textvariable = uservalue)
passentry = Entry(root, textvariable = passvalue)
```

- To use the *grid()* method for placing the *Entry()* widgets in proper place by passing attributes row and column. If we don't pass anything in the *grid()* method the default is 0 for both row and column (Here for *userentry* we pass row=0 and column=1 and for *passentry* we pass row=1 and column=1 in *grid()*).

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

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

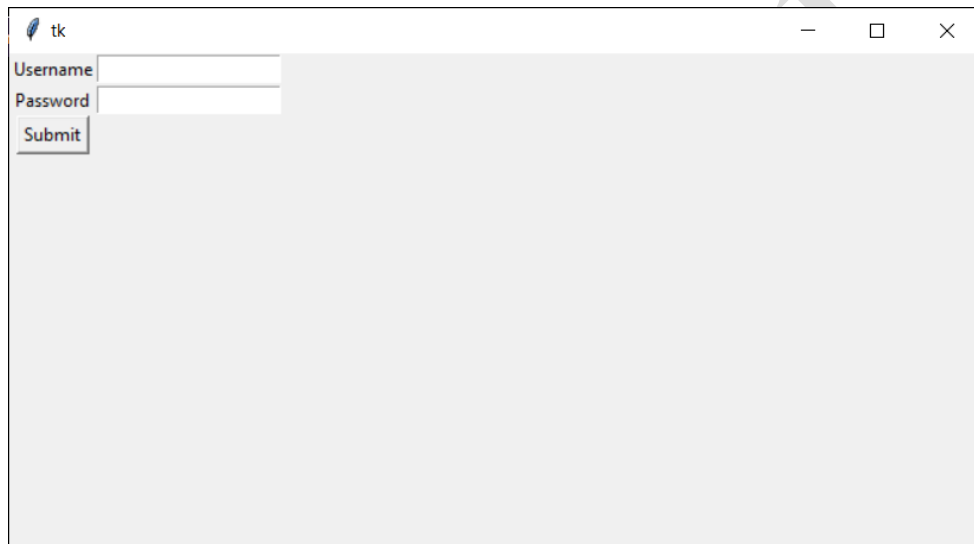
- We extend our little script by two buttons "Submit". We bind the function `getvals()`, which is using the `get()` method on the Entry objects, to the *Submit* button. So, every time this button is clicked, the content of the Entry fields will be printed on the terminal from which we had called the script.

```
Button(text="Submit", command=getvals).grid()
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *

def getvals():
    print(f"The value of username is {username.get()}")
    print(f"The value of password is {password.get()}")

root = Tk()
root.geometry("655x333")

user = Label(root, text="Username")
password = Label(root, text="Password")
user.grid()
password.grid(row=1)

# Variable classes in tkinter
# BooleanVar, DoubleVar, IntVar, StringVar
```

```
uservalue = StringVar()
passvalue = StringVar()

userentry = Entry(root, textvariable = uservalue)
passentry = Entry(root, textvariable = passvalue)

userentry.grid(row=0, column=1)
passentry.grid(row=1, column=1)

Button(text="Submit", command=getvals).grid()

root.mainloop()
```

TKinter notes

Travel Form Using Checkbuttons & Entry Widget

In this tutorial we'll make a Travel form using Label, Button, CheckButton and Entry widgets and grid() method. The title of the form is created using Label() widget. Different text sections are made with the Label() and are placed them properly using grid() method. The inputs for the entry sections are defined and entry sections are created using Entry() widget and placed using grid() method.

Code is described below:

```
from tkinter import *

root = Tk()

def getvals():
    print("It works!")
    root.geometry("644x344")
    #Heading
    Label(root, text="Welcome to Harry Travels", font="comicsansms 13 bold",
    pady=15).grid(row=0, column=3)

    #Text for our form
    name = Label(root, text="Name")
    phone = Label(root, text="Phone")
    gender = Label(root, text="Gender")
    emergency = Label(root, text="Emergency Contact")
    paymentmode = Label(root, text="Payment Mode")

    #Pack text for our form
    name.grid(row=1, column=2)
    phone.grid(row=2, column=2)
    gender.grid(row=3, column=2)
    emergency.grid(row=4, column=2)
    paymentmode.grid(row=5, column=2)

    # Tkinter variable for storing entries
    namevalue = StringVar()
    phonevalue = StringVar()
    gendervalue = StringVar()
    emergencyvalue = StringVar()
    paymentmodevalue = StringVar()
    foodservicevalue = IntVar()

    #Entries for our form
    nameentry = Entry(root, textvariable=namevalue)
    phoneentry = Entry(root, textvariable=phonevalue)
    genderentry = Entry(root, textvariable=gendervalue)
    emergencyentry = Entry(root, textvariable=emergencyvalue)
    paymentmodeentry = Entry(root, textvariable=paymentmodevalue)

    # Packing the Entries
    nameentry.grid(row=1, column=3)
```

```

phoneentry.grid(row=2, column=3)
genderentry.grid(row=3, column=3)
emergencyentry.grid(row=4, column=3)
paymentmodeentry.grid(row=5, column=3)

#Checkbox & Packing it
foodservice = Checkbutton(text="Want to prebook your meals?", variable =
foodservicevalue)
foodservice.grid(row=6, column=3)

#Button & packing it and assigning it a command
Button(text="Submit to Harry Travels", command=getvals).grid(row=7,
column=3)

root.mainloop()

```

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

```
from tkinter import *
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
root = Tk()
```

- To define a function 'def' (i.e. here the function getvals() is defined) is used.

```
def getvals():
    print("It works!")
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the width is 644 pixels and height is 344 pixels so we can write the function as *geometry(644x344)*.

```
root.geometry("644x344")
```

- The heading is made using Label() widget and font and pady are fixed as attributes. The the Label is put at row=0 and column=3 using grid() method so that the heading places at the right position.

```
Label(root, text="Welcome to Harry Travels", font="comicsansms 13 bold",
pady=15).grid(row=0, column=3)
```

- The text sections (i.e. name, phone, gender etc.) are made using Label() widget and "text" attribute is passed through this widget so that the given name in the "text" attribute is shown in GUI. For example, if text="Name", in GUI on the label "Name" will be written.

```

name = Label(root, text="Name")
phone = Label(root, text="Phone")
gender = Label(root, text="Gender")
emergency = Label(root, text="Emergency Contact")
paymentmode = Label(root, text="Payment Mode")

```

- The labels are set to proper place using `grid()` method passing row and column attributes.

```
name.grid(row=1, column=2)
phone.grid(row=2, column=2)
gender.grid(row=3, column=2)
emergency.grid(row=4, column=2)
paymentmode.grid(row=5, column=2)
```

- Tkinter variables (i.e `StringVar()`, `IntVar()` etc.) are made to store the entries.

```
namevalue = StringVar()
phonevalue = StringVar()
gendervalue = StringVar()
emergencyvalue = StringVar()
paymentmodevalue = StringVar()
foodservicevalue = IntVar()
```

- For taking the entries of the sections mentioned above, `Entry()` widget is used and "textvariable" attribute is passed through the widget to get the information of the entries.

```
nameentry = Entry(root, textvariable=namevalue)
phoneentry = Entry(root, textvariable=phonevalue)
genderentry = Entry(root, textvariable=gendervalue)
emergencyentry = Entry(root, textvariable=emergencyvalue)
paymentmodeentry = Entry(root, textvariable=paymentmodevalue)
```

- Entries are packed using `grid()` method and their positions are set using row and column attributes.

```
nameentry.grid(row=1, column=3)
phoneentry.grid(row=2, column=3)
genderentry.grid(row=3, column=3)
emergencyentry.grid(row=4, column=3)
paymentmodeentry.grid(row=5, column=3)
```

- The "foodservice" variable is defined as `IntVar()` as the `CheckBox` is made using "Checkbutton" widget and "text" is passed through it as attribute where "Want to prebook your meals?" text is shown. Then we need to use `grid()` method to set it proper place.

```
foodservice = Checkbutton(text="Want to prebook your meals?", variable =
foodservicevalue)
foodservice.grid(row=6, column=3)
```

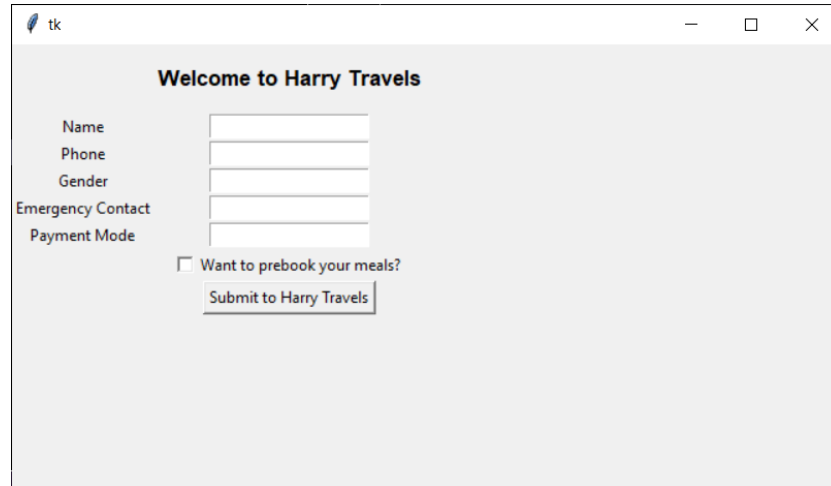
- We extend our little script by button "Submit". We bind the function `getvals()` to the *Submit* button. So, every time this button is clicked, the text "It works!" will be printed on the terminal from which we had called the script. Then we need to use `grid()` method to set it proper place.

```
Button(text="Submit to Harry Travels", command=getvals).grid(row=7,
column=3)
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *

root = Tk()

def getvals():
    print("It works!")
    root.geometry("644x344")
    #Heading
    Label(root, text="Welcome to Harry Travels", font="comicsansms 13 bold",
    pady=15).grid(row=0, column=3)

    #Text for our form
    name = Label(root, text="Name")
    phone = Label(root, text="Phone")
    gender = Label(root, text="Gender")
    emergency = Label(root, text="Emergency Contact")
    paymentmode = Label(root, text="Payment Mode")

    #Pack text for our form
    name.grid(row=1, column=2)
    phone.grid(row=2, column=2)
    gender.grid(row=3, column=2)
    emergency.grid(row=4, column=2)
    paymentmode.grid(row=5, column=2)

    # Tkinter variable for storing entries
    namevalue = StringVar()
    phonevalue = StringVar()
    gendervalue = StringVar()
    emergencyvalue = StringVar()
    paymentmodevalue = StringVar()
    foodservicevalue = IntVar()
```



```
#Entries for our form
nameentry = Entry(root, textvariable=namevalue)
phoneentry = Entry(root, textvariable=phonevalue)
genderentry = Entry(root, textvariable=gendervalue)
emergencyentry = Entry(root, textvariable=emergencyvalue)
paymentmodeentry = Entry(root, textvariable=paymentmodevalue)

# Packing the Entries
nameentry.grid(row=1, column=3)
phoneentry.grid(row=2, column=3)
genderentry.grid(row=3, column=3)
emergencyentry.grid(row=4, column=3)
paymentmodeentry.grid(row=5, column=3)

#Checkbox & Packing it
foodservice = Checkbutton(text="Want to prebook your meals?", variable =
foodservicevalue)
foodservice.grid(row=6, column=3)

#Button & packing it and assigning it a command
Button(text="Submit to Harry Travels", command=getvals).grid(row=7,
column=3)

root.mainloop()
```

Accepting User Input In Tkinter Form

In this tutorial, we'll make a form and learn how to accept user input in Tkinter form. We'll also learn how to open a text (.txt) file and write or append those input values in that file.

Code is described below:

```
from tkinter import *

root = Tk()

def getvals():
    print("Submitting form")

    print(f"{namevalue.get(), phonevalue.get(), gendervalue.get(),
emergencyvalue.get(), paymentmodevalue.get(), foodservicevalue.get()} ")

    with open("records.txt", "a") as f:
        f.write(f"{namevalue.get(), phonevalue.get(), gendervalue.get(),
emergencyvalue.get(), paymentmodevalue.get(), foodservicevalue.get()}\n ")

root.geometry("644x344")
#Heading
Label(root, text="Welcome to Harry Travels", font="comicsansms 13 bold",
pady=15).grid(row=0, column=3)

#Text for our form
name = Label(root, text="Name")
phone = Label(root, text="Phone")
gender = Label(root, text="Gender")
emergency = Label(root, text="Emergency Contact")
paymentmode = Label(root, text="Payment Mode")

#Pack text for our form
name.grid(row=1, column=2)
phone.grid(row=2, column=2)
gender.grid(row=3, column=2)
emergency.grid(row=4, column=2)
paymentmode.grid(row=5, column=2)

# Tkinter variable for storing entries
namevalue = StringVar()
phonevalue = StringVar()
gendervalue = StringVar()
emergencyvalue = StringVar()
paymentmodevalue = StringVar()
foodservicevalue = IntVar()

#Entries for our form
nameentry = Entry(root, textvariable=namevalue)
phoneentry = Entry(root, textvariable=phonevalue)
```

```

genderentry = Entry(root, textvariable=gendervalue)
emergencyentry = Entry(root, textvariable=emergencyvalue)
paymentmodeentry = Entry(root, textvariable=paymentmodevalue)

# Packing the Entries
nameentry.grid(row=1, column=3)
phoneentry.grid(row=2, column=3)
genderentry.grid(row=3, column=3)
emergencyentry.grid(row=4, column=3)
paymentmodeentry.grid(row=5, column=3)

#Checkbox & Packing it
foodservice = Checkbutton(text="Want to prebook your meals?", variable =
foodservicevalue)
foodservice.grid(row=6, column=3)

#Button & packing it and assigning it a command
Button(text="Submit to Harry Travels", command=getvals).grid(row=7,
column=3)

```

```

root.mainloop()

```

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

```

from tkinter import *

```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```

root = Tk()

```

- To define a function 'def' (i.e. here the function getvals() is defined) is used.
- This function "getvals()" is called from a Button (which is described below). After clicking this button it will print "Submitting form" as well as the values gotten from the entry widgets (for that the get() method is used) in the terminal.
- As within this function the text file ("records.txt") is opened in "append" mode the values of the Entry() widgets will be written in the text file itself. Note: We can open a file as read, write and append mode.

```

def getvals():
    print("Submitting form")

    print(f"{namevalue.get(), phonevalue.get(), gendervalue.get(),
emergencyvalue.get(), paymentmodevalue.get(), foodservicevalue.get()} ")

    with open("records.txt", "a") as f:
        f.write(f"{namevalue.get(), phonevalue.get(), gendervalue.get(),
emergencyvalue.get(), paymentmodevalue.get(), foodservicevalue.get()}\n ")

```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, `geometry()` function is used. As in example: the width is 644 pixels and height is 344 pixels so we can write the function as `geometry(644x344)`.

```
root.geometry("644x344")
```

- The heading is made using `Label()` widget and `font` and `pady` are fixed as attributes. The the `Label` is put at `row=0` and `column=3` using `grid()` method so that the heading places at the right position.

```
Label(root, text="Welcome to Harry Travels", font="comicsansms 13 bold",  
pady=15).grid(row=0, column=3)
```

- The text sections (i.e. name, phone, gender etc.) are made using `Label()` widget and `"text"` attribute is passed through this widget so that the given name in the `"text"` attribute is shown in GUI. For example, if `text="Name"`, in GUI on the label `"Name"` will be written.

```
name = Label(root, text="Name")  
phone = Label(root, text="Phone")  
gender = Label(root, text="Gender")  
emergency = Label(root, text="Emergency Contact")  
paymentmode = Label(root, text="Payment Mode")
```

- The label are set to proper place using `grid()` method passing row and column attributes.

```
name.grid(row=1, column=2)  
phone.grid(row=2, column=2)  
gender.grid(row=3, column=2)  
emergency.grid(row=4, column=2)  
paymentmode.grid(row=5, column=2)
```

- Tkinter variables (i.e `StringVar()`, `IntVar()` etc.) are made to store the entries.

```
namevalue = StringVar()  
phonevalue = StringVar()  
gendervalue = StringVar()  
emergencyvalue = StringVar()  
paymentmodevalue = StringVar()  
foodservicevalue = IntVar()
```

- For taking the entries of the sections mentioned above, `Entry()` widget is used and `"textvariable"` attribute is passed through the widget to get the information of the entries.

```
nameentry = Entry(root, textvariable=namevalue)  
phoneentry = Entry(root, textvariable=phonevalue)  
genderentry = Entry(root, textvariable=gendervalue)  
emergencyentry = Entry(root, textvariable=emergencyvalue)  
paymentmodeentry = Entry(root, textvariable=paymentmodevalue)
```

- Entries are packed using `grid()` method and their positions are set using row and column attributes.

```
nameentry.grid(row=1, column=3)  
phoneentry.grid(row=2, column=3)  
genderentry.grid(row=3, column=3)
```

```
emergencyentry.grid(row=4, column=3)
paymentmodeentry.grid(row=5, column=3)
```

- The "foodservice" variable is defined as IntVar() as the CheckBox is made using "CheckButton" widget and "text" is passed through it as attribute where "Want to prebook your meals?" text is shown. Then we need to use grid() method to set it proper place.

```
foodservice = Checkbutton(text="Want to prebook your meals?", variable =
foodservicevalue)
foodservice.grid(row=6, column=3)
```

- We extend our little script by button "Submit". We bind the function getvals() to the *Submit* button. So, every time this button is clicked, the text "It works!" will be printed on the terminal from which we had called the script. Then we need to use grid() method to set it proper place.

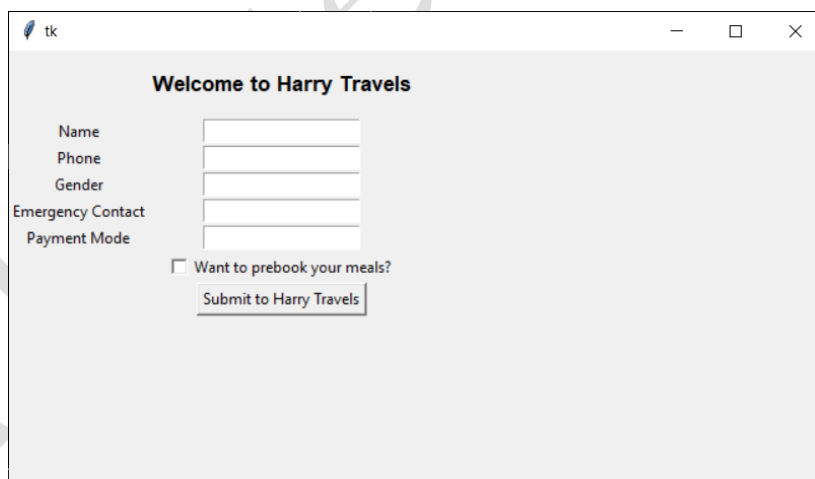
```
Button(text="Submit to Harry Travels", command=getvals).grid(row=7,
column=3)
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:

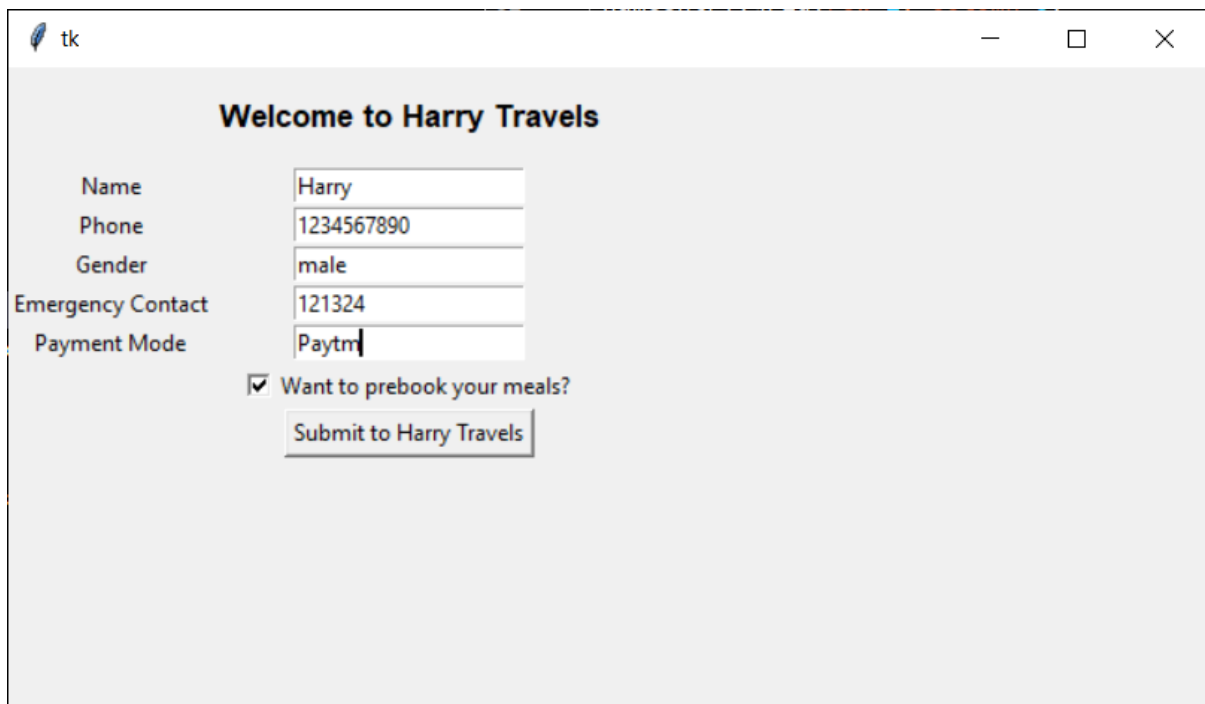
1. Initial GUI window:



The screenshot shows a Tkinter window titled "Welcome to Harry Travels". Inside the window, there is a form with the following elements:

- Labels: Name, Phone, Gender, Emergency Contact, Payment Mode.
- Input fields: Four text entry boxes corresponding to the labels above them.
- Checkbox: A checkbox labeled "Want to prebook your meals?".
- Button: A button labeled "Submit to Harry Travels".

2. The user is giving input and click on submit button:



tk

Welcome to Harry Travels

Name: Harry

Phone: 1234567890

Gender: male

Emergency Contact: 121324

Payment Mode: Paytm

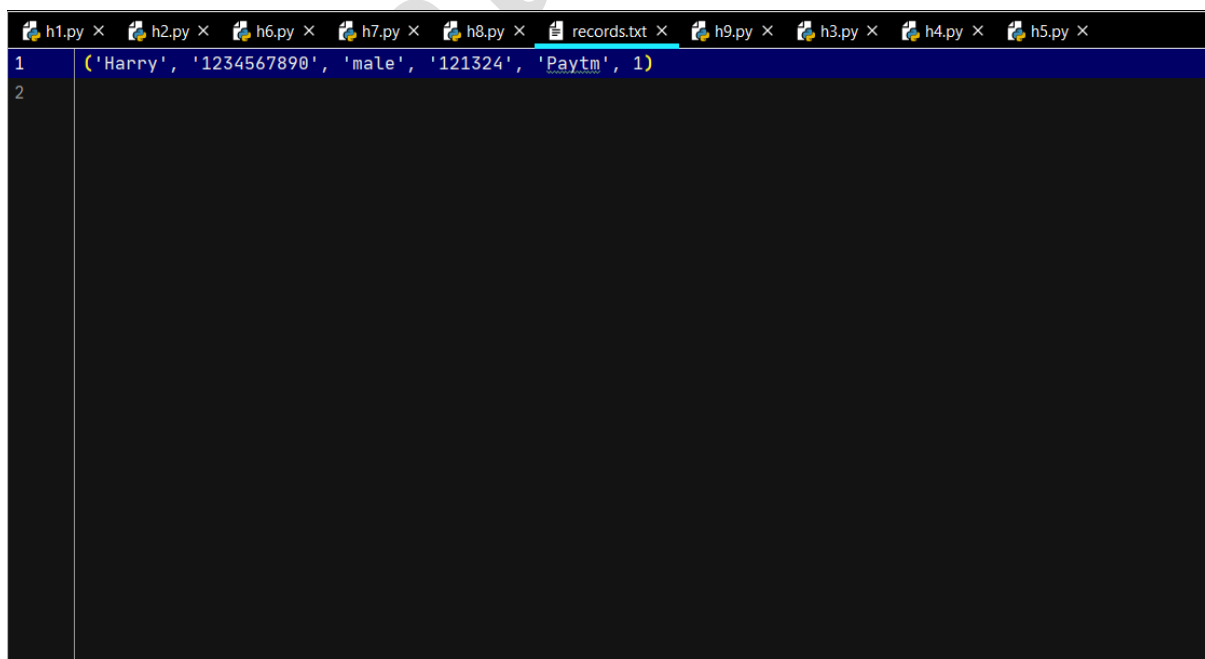
☒ Want to prebook your meals?

Submit to Harry Travels

4 The output in terminal:

```
Submitting form  
( 'Harry', '1234567890', 'male', '121324', 'Paytm', 1)
```

4. The output in the text file (records.txt):



```
h1.py × h2.py × h6.py × h7.py × h8.py × records.txt × h9.py × h3.py × h4.py × h5.py ×  
1 ('Harry', '1234567890', 'male', '121324', 'Paytm', 1)  
2
```

Code as described/written in the video

```

from tkinter import *

root = Tk()

def getvals():
    print("Submitting form")

    print(f"{namevalue.get(), phonevalue.get(), gendervalue.get(),
emergencyvalue.get(), paymentmodevalue.get(), foodservicevalue.get()} ")

    with open("records.txt", "a") as f:
        f.write(f"{namevalue.get(), phonevalue.get(), gendervalue.get(),
emergencyvalue.get(), paymentmodevalue.get(), foodservicevalue.get()}\n ")

root.geometry("644x344")
#Heading
Label(root, text="Welcome to Harry Travels", font="comicsansms 13 bold",
pady=15).grid(row=0, column=3)

#Text for our form
name = Label(root, text="Name")
phone = Label(root, text="Phone")
gender = Label(root, text="Gender")
emergency = Label(root, text="Emergency Contact")
paymentmode = Label(root, text="Payment Mode")

#Pack text for our form
name.grid(row=1, column=2)
phone.grid(row=2, column=2)
gender.grid(row=3, column=2)
emergency.grid(row=4, column=2)
paymentmode.grid(row=5, column=2)

# Tkinter variable for storing entries
namevalue = StringVar()
phonevalue = StringVar()
gendervalue = StringVar()
emergencyvalue = StringVar()
paymentmodevalue = StringVar()
foodservicevalue = IntVar()

#Entries for our form
nameentry = Entry(root, textvariable=namevalue)
phoneentry = Entry(root, textvariable=phonevalue)
genderentry = Entry(root, textvariable=gendervalue)
emergencyentry = Entry(root, textvariable=emergencyvalue)
paymentmodeentry = Entry(root, textvariable=paymentmodevalue)

# Packing the Entries
nameentry.grid(row=1, column=3)
phoneentry.grid(row=2, column=3)
genderentry.grid(row=3, column=3)
emergencyentry.grid(row=4, column=3)
paymentmodeentry.grid(row=5, column=3)

#Checkbox & Packing it

```

```
foodservice = Checkbutton(text="Want to prebook your meals?", variable =  
foodservicevalue)  
foodservice.grid(row=6, column=3)
```

```
#Button & packing it and assigning it a command  
Button(text="Submit to Harry Travels", command=getvals).grid(row=7,  
column=3)
```

```
root.mainloop()
```

TKinter notes

Canvas Widget In Python Tkinter

The Canvas() widget is a rectangular area for drawing pictures or other complex layouts. This is a highly versatile widget which can be used to draw graphs and plots, create graphics editors, and implement various kinds of custom widgets. The canvas is a general purpose widget, which is typically used to display and edit graphs and other drawings. Another common use for this widget is to implement various kinds of custom widgets. For example, you can use a canvas as a completion bar, by drawing and updating a rectangle on the canvas.

Here is the list of most commonly used options for this widget (these options can be used as key-value pairs separated by commas):

- **bd:** It defines the border width in pixels. The default value is 2.
- **bg:** Normal background color.
- **height:** It defines the size of the canvas in the Y dimension.
- **width:** It defines the size of the canvas in the X direction.
- **relief:** It specifies the type of the border. Some of the values are SUNKEN, RAISED, GROOVE, and RIDGE.

Here is the list of some standard items which the Canvas widget can support:

- **arc** – It creates an arc item which can be a chord, a pieslice or a simple arc.

```
coord = 10, 20, 300, 250
arc = canvas.create_arc(coord, start=0, extent=150, fill="red")
```

- **line** – It creates a line item.

```
#line = canvas.create_line(x0, y0, x1, y1, ..., xn, yn, options)
line = canvas.create_line(0, 0, 800, 400, 20, 100, fill="red")
```

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

```
#oval = canvas.create_oval(x0, y0, x1, y1, options)
oval = canvas.create_oval(344,233,244,355, fill="blue")
```

- **rectangle:** It creates a rectangle at the given coordinates. It takes two pairs of coordinates; the top left corner and the bottom right corner of the rectangle.

```
#rectangle = canvas.create_rectangle(x0, y0, x1, y1, options)
oval = canvas.create_rectangle(0, 100, 100, 0, fill="blue")
```

Code is described below:

```
from tkinter import *

root = Tk()

canvas_width = 800
canvas_height = 400

root.geometry(f"{canvas_width}x{canvas_height}")
root.title("Harry Ka GUI")
can_widget = Canvas(root, width=canvas_width, height=canvas_height)
can_widget.pack()

# The line goes from the point x1, y1 to x2, y2
can_widget.create_line(0, 0, 800, 400, fill="red")
can_widget.create_line(0, 400, 800, 0, fill="red")

# To create a rectangle specify parameters in this order - coors of top
left and coors of bottom right
can_widget.create_rectangle(3, 5, 700, 300, fill="blue")

can_widget.create_text(200, 200, text="python")

can_widget.create_oval(344, 233, 244, 355)

root.mainloop()
```

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

```
from tkinter import *
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the canvas_width is defined as 800 pixels and canvas_height is defined as 400 pixels so we can write the function as *geometry(f"{canvas_width}x{canvas_height}")*.

```
canvas_width = 800
canvas_height = 400
```

```
root.geometry(f"{canvas_width}x{canvas_height}")
```

- To set the title of the GUI window we use title() function. Here the title is taken as "Harry Ka GUI".

```
root.title("Harry Ka GUI")
```

- For drawing or writing something in a Canvas the Canvas() widget is used and as attributes the width and height are passed which are similar with the size of the GUI window.

```
can_widget = Canvas(root, width=canvas_width, height=canvas_height)
can_widget.pack()
```

- To draw a line create_line() is used where we should pass the X and Y coordinates of the line (i.e. if x1=0, y1=0, x2=800, y2=400 we should write `create_line(0, 0, 800, 400)`)

```
can_widget.create_line(0, 0, 800, 400, fill="red")
can_widget.create_line(0, 400, 800, 0, fill="red")
```

- To draw a rectangle create_rectangle is used where we should pass the coordinates of the top left corner and the bottom right corner of the rectangle (i.e. if (x1, y1) are the coordinates of top-left corner and (x2,y2) are the coordinates of the bottom-right corner and if x1=3, y1=5, x2=700, y2=300 we should write `create_line(3, 5, 700, 300)`).

```
can_widget.create_rectangle(3, 5, 700, 300, fill="blue")
```

- To add any text create_text is used and "text" attribute is passed to write on the canvas (i.e. text="python")

```
can_widget.create_text(200, 200, text="python")
```

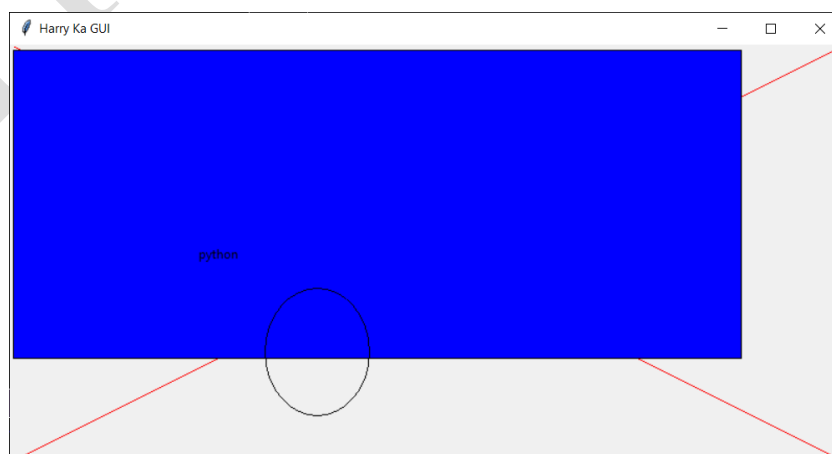
- To draw an oval create_oval is used where we should pass the two pairs of coordinates; the top left and bottom right corners of the bounding rectangle for the oval (i.e. x1=344, y1=233, x2= 244, y2=355 so we can write `create_oval(344, 233, 244, 355)`).

```
can_widget.create_oval(344, 233, 244, 355)
```

- There is a method known by the name `mainloop()` which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *

root = Tk()

canvas_width = 800
canvas_height = 400

root.geometry(f"{canvas_width}x{canvas_height}")
root.title("Harry Ka GUI")
can_widget = Canvas(root, width=canvas_width, height=canvas_height)
can_widget.pack()

# The line goes from the point x1, y1 to x2, y2
can_widget.create_line(0, 0, 800, 400, fill="red")
can_widget.create_line(0, 400, 800, 0, fill="red")

# To create a rectangle specify parameters in this order - coors of top
left and coors of bottom right
can_widget.create_rectangle(3, 5, 700, 300, fill="blue")

can_widget.create_text(200, 200, text="python")

can_widget.create_oval(344, 233, 244, 355)

root.mainloop()
```

Handling Events In Tkinter GUI

Tkinter is a GUI (Graphical User Interface) module which is widely used in the desktop applications. It provides variety of Widget classes and functions with the help of which one can make our GUI more attractive and user friendly in terms of both looks and functionality. Binding function is used to deal with the events. We can bind **Python's Functions** and methods to an event as well as we can bind these functions to any particular widget. Events can come from various sources, including key presses and mouse operations by the user, and redraw events from the window manager (indirectly caused by the user, in many cases).

The most important part of an event specifier is the *type* field. It specifies the kind of event that we wish to bind, and can be user actions like **Button**, and **Key**, or window manager events like **Enter**, **Configure**, and others.

Event formats:

- **<Button-1>**: A mouse button is pressed over the widget. Button 1 is the leftmost button, Button 2 is the middle button (where available), and Button 3 the rightmost button.
- **<B1-Motion>**: The mouse is moved, with mouse button 1 being held down (use B2 for the middle button, B3 for the right button).
- **<Double-Button-1>**: Button 1 was double clicked. You can use **Double** or **Triple** as prefixes. **Note: If you bind to both a single click (<Button-1>) and a double click, both bindings will be called.**
- **<Enter>**: The mouse pointer entered the widget. **Note: this event doesn't mean that the user pressed the Enter key.**

Event attributes:

As Event attributes, we can use **width, height, char, num, widget** etc.

Code is described below:

```
from tkinter import *

def harry(event):
    print(f"You clicked on the button at {event.x}, {event.y}")

root = Tk()
root.title("Events in Tkinter")
root.geometry("644x334")

widget = Button(root, text='Click me please')
widget.pack()

widget.bind('<Button-1>', harry)
widget.bind('<Double-1>', quit)
root.mainloop()
```

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

```
from tkinter import *
```

- To define a function 'def' (i.e. here the function `harry()` is defined) is used and within `def` the event is called.
- It prints the X and Y coordinates of the event when the event is occurred (i.e. `<Button-1>`, `<Double-1>`).

```
def harry(event):  
    print(f"You clicked on the button at {event.x}, {event.y}")
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the `className` to the desired one.

```
root = Tk()
```

- To set the title of the GUI window `title()` function is used.

```
root.title("Events in Tkinter")
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, `geometry()` function is used. As in example: the width is 644 pixels and height is 334 pixels so we can write the function as `geometry(644x334)`.

```
root.geometry("644x334")
```

- We extend our little script by button "Click me please" passing the attribute "text" within Button widget and then the button is packed using the `pack()` method.

```
widget = Button(root, text='Click me please')  
widget.pack()
```

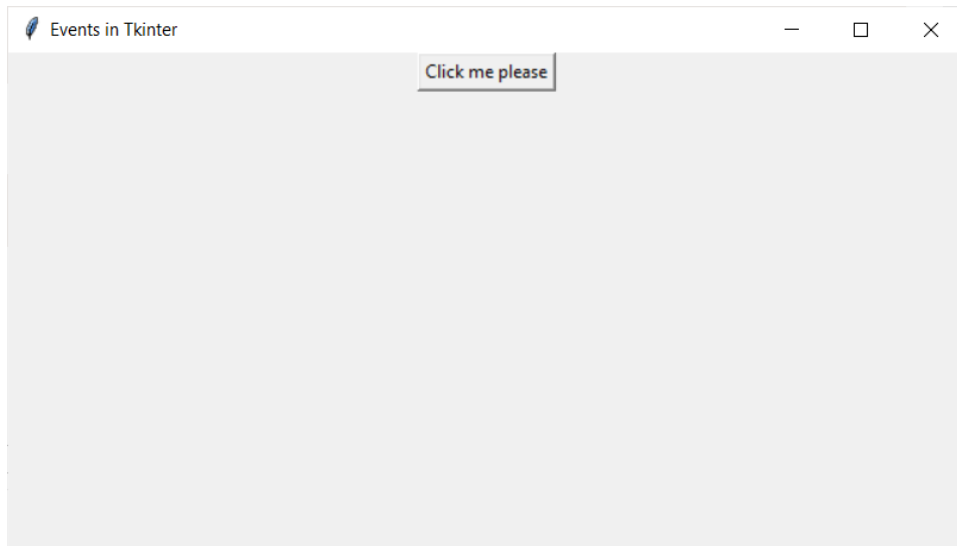
- The button "widget" is binded using the `bind` method and we can pass any attribute (i.e. `<Button-1>`, `<Double-1>` etc.). Whenever the button will be clicked the functions (i.e. `harry`) will be called and the event will be occurred.

```
widget.bind('<Button-1>', harry)  
widget.bind('<Double-1>', quit)
```

- There is a method known by the name `mainloop()` which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed. In this example, when the button will be clicked for once it will call `harry` and it will print the coordinates of x and y of the place where the button was clicked and if the button is clicked for twice the program will quit.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



```
You clicked on the button at 50, 10
You clicked on the button at 50, 10
You clicked on the button at 56, 17
```

Code as described/written in the video

```
from tkinter import *

def harry(event):
    print(f"You clicked on the button at {event.x}, {event.y}")

root = Tk()
root.title("Events in Tkinter")
root.geometry("644x334")

widget = Button(root, text='Click me please')
widget.pack()

widget.bind('1', harry)
widget.bind('1', quit)

root.mainloop()
```

Python GUI Exercise 1: Solution

```
from tkinter import *
from PIL import ImageTk, Image

def every_100(text):
    final_text = ""
    for i in range(0, len(text)):
        final_text += text[i]
        if i%100==0 and i!=0:
            final_text += "\n"
    return final_text

root = Tk()
root.title("CodeWithHarry News - Aapka Apna Akhabaar")
root.geometry("1000x1000")

texts = []
photos = []
for i in range(0, 3):
    with open(f"{i+1}.txt") as f:
        text = f.read()
        texts.append(every_100(text))

    image = Image.open(f"{i+1}.png")
    #TODO: Resize these images
    image = image.resize((225, 265), Image.ANTIALIAS)
    photos.append(ImageTk.PhotoImage(image))

f0 = Frame(root, width=800, height=70)
Label(f0, text="Code With Harry News", font="lucida 33 bold").pack()
Label(f0, text="January 19, 2050", font="lucida 13 bold").pack()
f0.pack()

f1 = Frame(root, width=900, height=200, pady=14)
Label(f1, text=texts[0], padx=22, pady=22).pack(side="left")
Label(f1, image=photos[0], anchor="e").pack()
f1.pack(anchor="w")

f2 = Frame(root, width=900, height=200, pady=14, padx=22)
Label(f2, text=texts[1], padx=22, pady=22).pack(side="right")
Label(f2, image=photos[1], anchor="e", padx=22).pack()
f2.pack(anchor="w")

f3 = Frame(root, width=900, height=200, pady=34)
Label(f3, text=texts[2], padx=22, pady=22).pack(side="left")
Label(f3, image=photos[2], anchor="e").pack()
f3.pack(anchor="w")

root.mainloop()
```


Python GUI Exercise 2: Window Resizer GUI

Problem Statement

Create a Window Resizer GUI using Tkinter in Python

-

Hints

Create a GUI window which takes the input of width and height

Create a Button "Apply" and after clicking on this button it will change the size of the GUI window accordingly

Menus & Submenus In Tkinter Python

The **Menu** widget is used to implement different types of menus (toplevel, pulldown, and popup menus). The goal of this widget is to allow us to create all kinds of menus that can be used by our applications. It is also possible to use other extended widgets to implement new types of menus, such as the *OptionMenu* widget, which implements a special type that generates a pop-up list of items within a selection.

Attributes:

- **bg:** The background color for choices not under the mouse.
- **bd:** The width of the border around all the choices. **Default is 1.**
- **fg:** The foreground color used for choices not under the mouse.
- **tearoff:** Normally, a menu can be torn off. The first position (position 0) in the list of choices is occupied by the tear-off element and the additional choices are added starting at position 1. If you set **tearoff=0**, the menu will not have a tear-off feature and choices will be added starting at position 0.
- **relief:** The default 3-D effect for menus is **relief=RAISED**.
- **title:** Normally, the title of a tear-off menu window will be the same as the text of the menubutton or cascade that lead to this menu. If you want to change the title of that window, set the title option to that string.

Code is described below:

```
from tkinter import *
root = Tk()
root.geometry("733x566")
root.title("Pycharm")

def myfunc():
    print("Mai ek bahut hi natkhat aur shaitaan function hoon")

# #Use these to create a non dropdown menu
# mymenu = Menu(root)
# mymenu.add_command(label="File", command=myfunc)
# mymenu.add_command(label="Exit", command=quit)
# root.config(menu=mymenu)

mainmenu = Menu(root)

m1 = Menu(mainmenu, tearoff=0)
m1.add_command(label="New project", command=myfunc)
m1.add_command(label="Save", command=myfunc)
m1.add_separator()
m1.add_command(label="Save As", command=myfunc)
m1.add_command(label="Print", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="File", menu=m1)

m2 = Menu(mainmenu, tearoff=0)
m2.add_command(label="Cut", command=myfunc)
m2.add_command(label="Copy", command=myfunc)
```

```
m2.add_separator()
m2.add_command(label="Paste", command=myfunc)
m2.add_command(label="Find", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="Edit", menu=m2)
```

```
root.mainloop()
```

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

```
from tkinter import *
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the width is 733 pixels and height is 566 pixels so we can write the function as *geometry(733x566)*.

```
root.geometry("733x566")
```

- To set a title of the GUI window title() method is used. Here it is taken as "Pycharm".

```
root.title("Pycharm")
```

- To define a function 'def' (i.e. here the function myfunc() is defined) is used. Whenever the function will be called the printing statement within this function will be executed.

```
def myfunc():
    print("Mai ek bahut hi natkhat aur shaitaan function hoon")
```

- **For creating non-dropdown menu widget:**

1. Menu() widget is created in the root or parent window and this widget is taken as variable named "mymenu".
2. For adding a menu item to the Menu widget we use add_command() method. We pass the attribute "label" which shows the name of the given menu item (i.e. "File", "Exit") and command which call the function and executes the statement within the function (i.e. myfunc, quit).
3. config() is used to access an object's attributes after its initialisation and then the menu is set with the variable "mymenu".

```
mymenu = Menu(root)
mymenu.add_command(label="File", command=myfunc)
mymenu.add_command(label="Exit", command=quit)
root.config(menu=mymenu)
```

- **For creating a dropdown menu widget:**

1. `Menu()` widget is created in the root or parent window and this widget is taken as variable named "mainmenu".
2. Three dropdown menus are created (m1, m2 & m3). With each menu four menu items are added and their names are set using Label attribute (i.e. New Project, Save, Print etc.). `Tearoff=0` is used so that the menu will not have a tear-off feature, and choices will be added starting at position 0 (the menu section will be attached with GUI window always).
3. **`config()`** is used to access an object's attributes after its initialisation and then the menu is set with the variable "mainmenu".
4. For adding the new hierarchical menu by associating a given menu m1 to a parent menu mainmenu `add_cascade()` is used and label is passed to set the name of the label (i.e. "File").
5. For adding a separator line to the menu we use `add_separator()` method.

```
mainmenu = Menu(root)

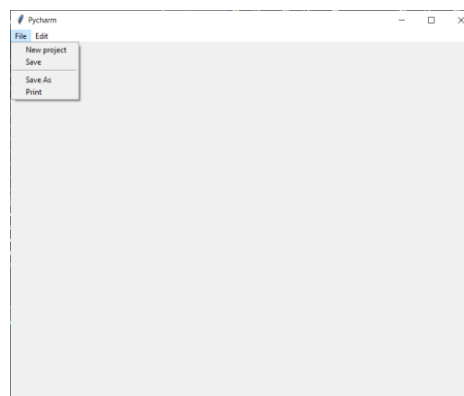
m1 = Menu(mainmenu, tearoff=0)
m1.add_command(label="New project", command=myfunc)
m1.add_command(label="Save", command=myfunc)
m1.add_separator()
m1.add_command(label="Save As", command=myfunc)
m1.add_command(label="Print", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="File", menu=m1)

m2 = Menu(mainmenu, tearoff=0)
m2.add_command(label="Cut", command=myfunc)
m2.add_command(label="Copy", command=myfunc)
m2.add_separator()
m2.add_command(label="Paste", command=myfunc)
m2.add_command(label="Find", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="Edit", menu=m2)
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *
root = Tk()
root.geometry("733x566")
root.title("Pycharm")

def myfunc():
    print("Mai ek bahut hi natkhat aur shaitaan function hoon")

# #Use these to create a non dropdown menu
# mymenu = Menu(root)
# mymenu.add_command(label="File", command=myfunc)
# mymenu.add_command(label="Exit", command=quit)
# root.config(menu=mymenu)

mainmenu = Menu(root)

m1 = Menu(mainmenu, tearoff=0)
m1.add_command(label="New project", command=myfunc)
m1.add_command(label="Save", command=myfunc)
m1.add_separator()
m1.add_command(label="Save As", command=myfunc)
m1.add_command(label="Print", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="File", menu=m1)

m2 = Menu(mainmenu, tearoff=0)
m2.add_command(label="Cut", command=myfunc)
m2.add_command(label="Copy", command=myfunc)
m2.add_separator()
m2.add_command(label="Paste", command=myfunc)
m2.add_command(label="Find", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="Edit", menu=m2)

root.mainloop()
```

Message Box In Tkinter Python

Python **Tkinter-MessageBox** module is used to display the message boxes in the python applications. This module provides a number of functions that an user can use to display an appropriate message. Some of these functions are *showinfo*, *askquestion*, *showwarning*, *showerror*, *askokcancel*, *askyesno* and *askretrycancel*.

Parameters:

`tmsg.Function_Name(title, message [, options])`

- **tmsg** is used to defining TKMessageBox. For this we have to import `tkinter.messagebox` as `tmsg`.
 - **Function_Name**: This is the name of the appropriate message box function.
 - **title**: This is the text to be displayed in the title bar of a message box.
 - **message**: This is the text to be displayed as a message in the message box.
 - **options**: Options are alternative choices that may be used to tailor a standard message box.
1. **default**: The default option is used to specify the default button, such as ABORT, RETRY, or IGNORE in the message box.
 2. **parent**: The parent option is used to specify the window on top of which the message box is to be displayed.

Code is described below:

```
from tkinter import *
import tkinter.messagebox as tmsg
root = Tk()
root.geometry("733x566")
root.title("Pycharm")

def myfunc():
    print("Mai ek bahut hi natkhat aur shaitaan function hoon")

def help():
    print("I will help you")
    tmsg.showinfo("Help", "Harry will help you with this gui")

def rate():
    print("Rate us")
    value = tmsg.askquestion("Was your experience Good?", "You used this
gui.. Was your experience Good?")
    if value == "yes":
        msg = "Great. Rate us on appstore please"
    else:
        msg = "Tell us what went wrong. We will call you soon"
    tmsg.showinfo("Experience", msg)

def divya():
    ans = tmsg.askretrycancel("Divya se dosti kar lo", "Sorry divya nahi
banegi aapki dost")
    if ans:
        print("Retry karne pe bhi kuch nahi hoga")
```

```

else:
    print("Bahut badiya bhai cancel kar diya warna pitte")

mainmenu = Menu(root)

m1 = Menu(mainmenu, tearoff=0)
m1.add_command(label="New project", command=myfunc)
m1.add_command(label="Save", command=myfunc)
m1.add_separator()
m1.add_command(label="Save As", command=myfunc)
m1.add_command(label="Print", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="File", menu=m1)

m2 = Menu(mainmenu, tearoff=0)
m2.add_command(label="Cut", command=myfunc)
m2.add_command(label="Copy", command=myfunc)
m2.add_separator()
m2.add_command(label="Paste", command=myfunc)
m2.add_command(label="Find", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="Edit", menu=m2)

m3 = Menu(mainmenu, tearoff=0)
m3.add_command(label = "Help", command=help)
m3.add_command(label = "Rate us", command=rate)
m3.add_command(label = "Befriend Divya", command=divya)
mainmenu.add_cascade(label="Help", menu=m3)
root.config(menu=mainmenu)
root.mainloop()

```

- In this above code, within the functions help(), rate() and divya() TkMeassgeBox module is used to show message or dialogue box. These three functions are called from menu m3 labelled as "Help".
1. Within help() function, showinfo() function is used where we can pass title's name as the first parameter and show some relevant information to the user (i.e. "Harry" is the title of the message box and we can see "Harry will help you with this gui" as information in the message box.
 2. Within rate() function, askquestion() function is used where we can pass title's name as the first parameter and ask question to user which has to be answered in yes or no. In this example, "Was your experience Good?" is the title of the message box and "You used this gui.. Was your experience Good?" is passed as question. Also depending on the user's answer, it can give output . Here if the answer is "yes" it will show message "Great. Rate us on appstore please", else (if the anser is "no") it will show the message "Tell us what went wrong. We will call you soon". This both condition's statements are taken in "msg" variable which is passed in the showinfo() function so that it can show the information creating another message box whose title is taken as "Experience".
 3. Within "divya" function, asktrycancel() function is used where we can pass title's name as the first parameter and ask the user about doing a particular task again or not (Retry or Cancel). Here in this example based on the two conditions "Retry" & "Cancel" on the message box it will print different statements in terminal.

```
def help():
```

```

print("I will help you")
tmsg.showinfo("Help", "Harry will help you with this gui")

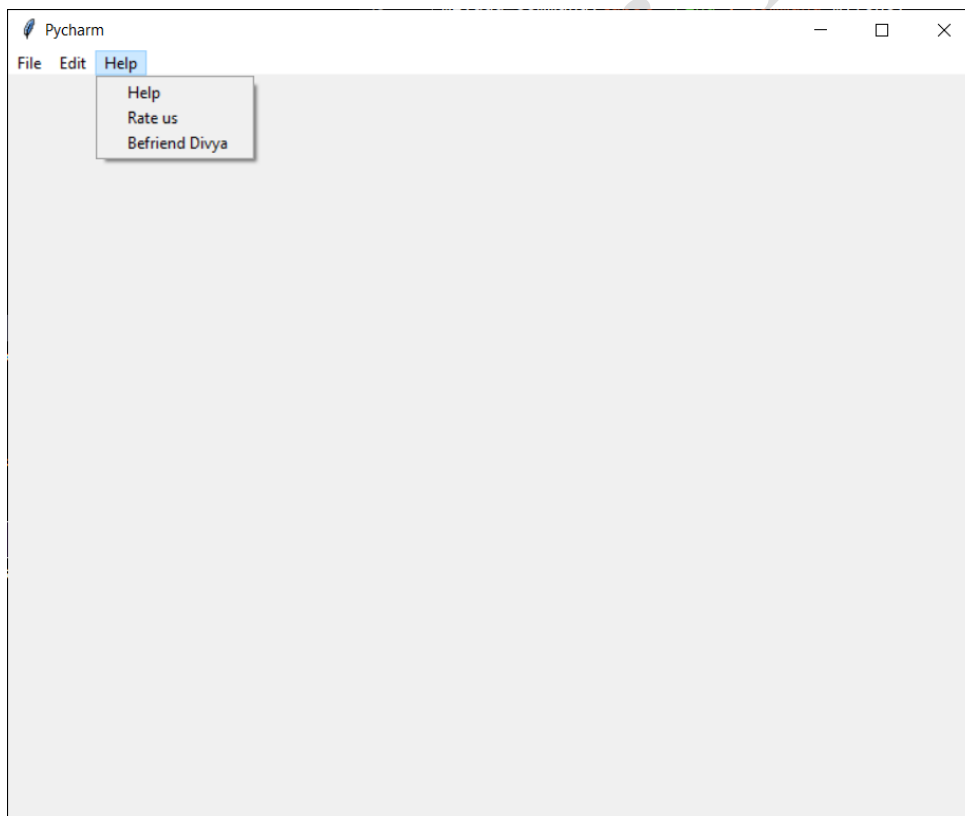
def rate():
    print("Rate us")
    value = tmsg.askquestion("Was your experience Good?", "You used this
gui.. Was your experience Good?")
    if value == "yes":
        msg = "Great. Rate us on appstore please"
    else:
        msg = "Tell us what went wrong. We will call you soon"
    tmsg.showinfo("Experience", msg)

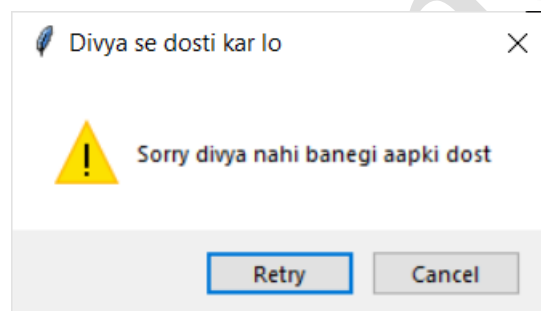
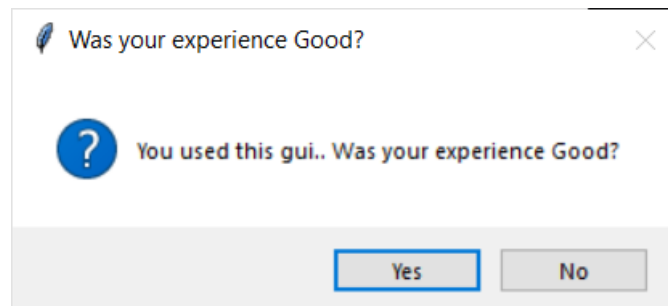
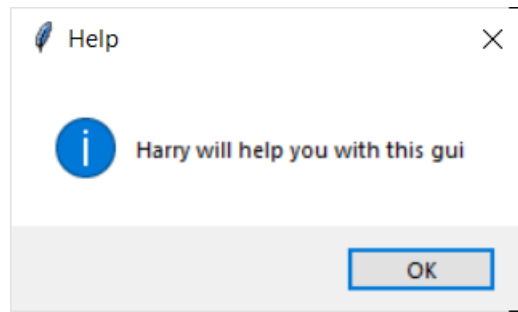
def divya():
    ans = tmsg.askretrycancel("Divya se dosti kar lo", "Sorry divya nahi
banegi aapki dost")
    if ans:
        print("Retry karne pe bhi kuch nahi hoga")

    else:
        print("Bahut badiya bhai cancel kar diya warna pitte")

```

- The rest part of the code (Menus & Submenus) is described in [Menus and Submenus in Tkinter Python](#)
-
- **Output:** The output of the code (or the GUI window) is given below:





Code as described/written in the video

```
from tkinter import *
import tkinter.messagebox as tmsg
root = Tk()
root.geometry("733x566")
root.title("Pycharm")

def myfunc():
    print("Mai ek bahut hi natkhat aur shaitaan function hoon")

def help():
    print("I will help you")
    tmsg.showinfo("Help", "Harry will help you with this gui")

def rate():
    print("Rate us")
```

```

        value = tmsg.askquestion("Was your experience Good?", "You used this
gui.. Was your experience Good?")
        if value == "yes":
            msg = "Great. Rate us on appstore please"
        else:
            msg = "Tell us what went wrong. We will call you soon"
        tmsg.showinfo("Experience", msg)

def divya():
    ans = tmsg.askretrycancel("Divya se dosti kar lo", "Sorry divya nahi
banegi aapki dost")
    if ans:
        print("Retry karne pe bhi kuch nahi hoga")

    else:
        print("Bahut badiya bhai cancel kar diya warna pitte")

mainmenu = Menu(root)

m1 = Menu(mainmenu, tearoff=0)
m1.add_command(label="New project", command=myfunc)
m1.add_command(label="Save", command=myfunc)
m1.add_separator()
m1.add_command(label="Save As", command=myfunc)
m1.add_command(label="Print", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="File", menu=m1)

m2 = Menu(mainmenu, tearoff=0)
m2.add_command(label="Cut", command=myfunc)
m2.add_command(label="Copy", command=myfunc)
m2.add_separator()
m2.add_command(label="Paste", command=myfunc)
m2.add_command(label="Find", command=myfunc)
root.config(menu=mainmenu)
mainmenu.add_cascade(label="Edit", menu=m2)

m3 = Menu(mainmenu, tearoff=0)
m3.add_command(label = "Help", command=help)
m3.add_command(label = "Rate us", command=rate)
m3.add_command(label = "Befriend Divya", command=divya)
mainmenu.add_cascade(label="Help", menu=m3)
root.config(menu=mainmenu)
root.mainloop()

```

Sliders In Tkinter Using Scale()

The **Scale()** widget provides a graphical slider object that allows the user to select numerical values by moving the “slider” knob along the specific scale. User can control the minimum and maximum values as well as the resolution. In Slider() widget we use some new attributes i.e. *to, from, orient* etc.

Attributes:

- **from_** : This should be a float or integer value that defines one end or the starting value of the scale's range.
- **to**: This should be also a float or integer value that defines other end or the finishing value of the scale's range. **Note: The to value can be either greater than or less than the from_ value. For vertical scales, the to value defines the bottom of the scale; for horizontal scales, the right end.**
- **orient**: It sets the orientation of the scale. If we set orient=HORIZONTAL the scale runs along the x dimension or we set orient=VERTICAL it runs parallel to the y-axis. **Default is horizontal.**
- **length**: It sets the length of the Scale() widget. It defines the x dimension if the scale is horizontal or the y dimension if vertical. **The default value is 100 pixels.**
- **resolution**: Normally, the user is only able to change the scale in whole units. But this attribute sets this option to some other value to change the smallest increment of the scale's value. For example, if from_=-10 and to=10, and you set resolution=5, the scale will have 5 possible values: -10, -5, 0, +5, and +10.
- **tickinterval**: It displays periodic scale values. If we set this option to a number ticks will be displayed on multiples of that value. For example, if from_=0, to=10, and tickinterval=2, labels will be displayed along the scale at values 0, 2, 4, 6, 8, 10. These labels appear below the scale if horizontal, to its left if vertical. **Default is 0, which suppresses display of ticks.**

Methods:

- **get()**: This method returns the current value of the scale.
- **set(value)**: It sets the scale's value. For example, if we give set(30) the initial scale value will show 30 (the scale will starts from 30).

Code is described below:

```
from tkinter import *
import tkinter.messagebox as tmsg

root = Tk()
root.geometry("455x233")
root.title("Slider tutorial")

def getdollar():
    print(f"We have credited {myslider2.get()} dollars to your bank account")
    tmsg.showinfo("Amount Credited!", f"We have credited {myslider2.get()} dollars to your bank account")
```

```
# myslider = Scale(root, from_=0, to=100)
# myslider.pack()
Label(root, text="How many dollars do you want?").pack()
myslider2 = Scale(root, from_=0, to=100, orient=HORIZONTAL,
tickinterval=50)
# myslider2.set(34)
myslider2.pack()

Button(root, text="Get dollars!", pady=10, command=getdollar).pack()

root.mainloop()
```

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

```
from tkinter import *
```

- Tkinter-MessageBox (tkinter.messagebox) is imported as '*tmsg*'.

```
import tkinter.messagebox as tmsg
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the width is 455 pixels and height is 233 pixels so we can write the function as *geometry(455x233)*.

```
root.geometry("455x233")
```

- Title of the GUI window is set using title() function. Here we have taken the title as "Slider tutorial".

```
root.title("Slider tutorial")
```

- A function "getdollar" is defined. Within this function, print statement is written from where the user will get the myslider2 value (myslider2 is discussed below). A message box named "Amount Credited!" is also created and within that also get() method is written so that user can see the myslider2 value in the message box.

```
def getdollar():
    print(f"We have credited {myslider2.get()} dollars to your bank account")
    tmsg.showinfo("Amount Credited!", f"We have credited {myslider2.get()} dollars to your bank account")
```

- A label is taken and text is passed as attribute and then the label is packed using pack() method.

```
Label(root, text="How many dollars do you want?").pack()
```

- Slider is created using Scale() widget. Its beginning range is set to 0 using **from_** attribute and its ending range is set to 100 using **to** attribute. The orientation is taken as **HORIZONTAL** using **orient** attribute. Also **tickinterval** attribute is taken as 50 so that the scale only shows the multiple values of 50. Atlast the slider is packed using pack() method.

```
myslider2 = Scale(root, from_=0, to=100, orient=HORIZONTAL,
tickinterval=50)
myslider2.pack()
```

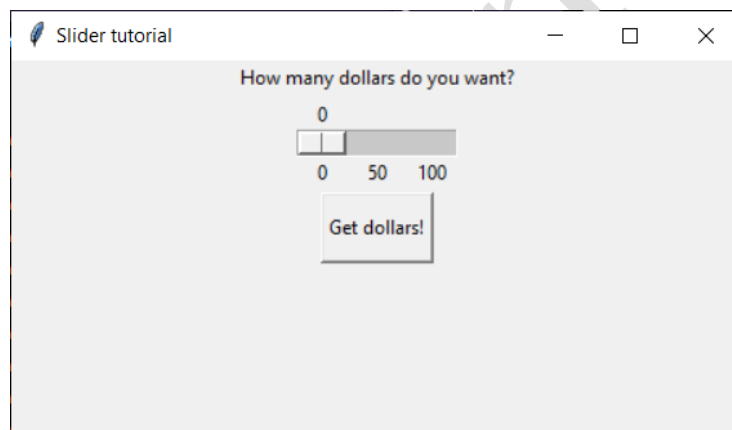
- A button is created using Button() widget and getdollars() function is called from this button using **command** attribute. The button is packed using pack() method.

```
Button(root, text="Get dollars!", pady=10, command=getdollar).pack()
```

- There is a method known by the name **mainloop()** which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *
import tkinter.messagebox as tmsg

root = Tk()
root.geometry("455x233")
root.title("Slider tutorial")

def getdollar():
    print(f"We have credited {myslider2.get()} dollars to your bank account")
    tmsg.showinfo("Amount Credited!", f"We have credited {myslider2.get()} dollars to your bank account")

# myslider = Scale(root, from_=0, to=100)
# myslider.pack()
```

```
Label(root, text="How many dollars do you want?").pack()
myslider2 = Scale(root, from_=0, to=100, orient=HORIZONTAL,
tickinterval=50)
# myslider2.set(34)
myslider2.pack()

Button(root, text="Get dollars!", pady=10, command=getdollar).pack()

root.mainloop()
```

TKinter notes

Creating RadioButtons In Tkinter

Radiobutton() widget is a graphical user interface element of Tkinter, which allows the user to choose (exactly) one of a predefined set of options. Sometimes it is called option button. Radiobuttons can contain text or images. The button can only display text in a single font. A Python function or method can be associated with a radio button. This function or method will be called, if you press this radio button.

In order to implement this functionality, each group of radiobuttons must be associated to the same *variable* (we have to initialize the variable firstly, if it is *StringVar()*) and each one of the buttons must symbolize a single *value*. You can use the Tab key to switch from one radiobutton to another.

Attributes:

- **anchor:** If the widget inhabits a space larger than it needs, this option specifies where the radiobutton will sit in that space (i.e. e, w). The default is anchor=CENTER.
- **justify:** If the text contains multiple lines, this option controls how the text is justified: CENTER (the default), LEFT, or RIGHT.
- **value:** When a radiobutton is turned on by the user, its control variable is set to its current value option. If the control variable is an *IntVar*, give each radiobutton in the group a different integer value option. If the control variable is a *StringVar*, give each radiobutton a different string value option.
- **variable:** The control variable that this radiobutton shares with the other radiobuttons in the group. This can be either an *IntVar()* or a *StringVar()*. If it is *StringVar()* it should be initialized beforehand with some other string variable name using *set()* function.
- **bg:** It shows the normal background color behind the indicator and label.
- **font:** It is used to set the font used for the text.
- **text:** It is the label displayed next to the radiobutton. Use newlines ("**\n**") to display multiple lines of text.

Code is described below:

```
from tkinter import *
import tkinter.messagebox as tmsg

root = Tk()
root.geometry("455x233")
root.title("Radiobutton tutorial")

def order():

    tmsg.showinfo("Order Received!", f"We have received your order for {var.get()}. Thanks for ordering")

# var = IntVar()
var = StringVar()
var.set("Radio")
# var.set(1)
Label(root, text = "What would you like to have sir?",font="lucida 19 bold",
      justify=LEFT, padx=14).pack()
```

```

radio = Radiobutton(root, text="Dosa", padx=14, variable=var,
value="Dosa").pack(anchor="w")
radio = Radiobutton(root, text="Idly", padx=14, variable=var,
value="Idly").pack(anchor="w")
radio = Radiobutton(root, text="Paratha", padx=14, variable=var,
value="Paratha").pack(anchor="w")
radio = Radiobutton(root, text="Samosa", padx=14, variable=var,
value="Samosa").pack(anchor="w")

Button(text="Order Now", command=order).pack()
root.mainloop()

```

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

```
from tkinter import *
```

- Tkinter-MessageBox (tkinter.messagebox) is imported as '*tmsg*'.

```
import tkinter.messagebox as tmsg
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the width is 455 pixels and height is 233 pixels so we can write the function as *geometry(455x233)*.

```
root.geometry("455x233")
```

- Title of the GUI window is set using title() function. Here we have taken the title as "Radiobutton tutorial".

```
root.title("Radiobutton tutorial")
```

- A function "order()" is defined. Within this function, a message box named "Order Received!" is created where showinfo() function is used to show the given information and within that also get() method is written so that user can see the selected Radiobutton value in the message box.

```
def order():
```

```

    tmsg.showinfo("Order Received!", f"We have received your order for
{var.get()}. Thanks for ordering")

```

- A variable is taken as StringVar() and is set to a string value "Radio" (so that the Radiobuttons become unchecked initially).

```

var = StringVar()
var.set("Radio")

```


- A label is created using `label()` widget where the text is passed as attribute and font is set as "lucida 19 bold". We justified the label in the left side and set the `padx` as 14. Then the label is packed using `pack()` function.

```
Label(root, text = "What would you like to have sir?", font="lucida 19 bold",
      justify=LEFT, padx=14).pack()
```

- Four Radiobuttons are created using `Radiobutton()` widget. For setting the names of the buttons "text" is passed as attribute and different names (i.e. "Idly", "Dosa" etc.) are set. For taking the variable (`StringVar()`) "var" we pass the attribute "variable" and give all the radiobuttons different values. Then all the radiobuttons are packed (here we packed it at the west or the left most side so we used `anchor="w"`).

```
radio = Radiobutton(root, text="Dosa", padx=14, variable=var,
                    value="Dosa").pack(anchor="w")
radio = Radiobutton(root, text="Idly", padx=14, variable=var,
                    value="Idly").pack(anchor="w")
radio = Radiobutton(root, text="Paratha", padx=14, variable=var,
                    value="Paratha").pack(anchor="w")
radio = Radiobutton(root, text="Samosa", padx=14, variable=var,
                    value="Samosa").pack(anchor="w")
```

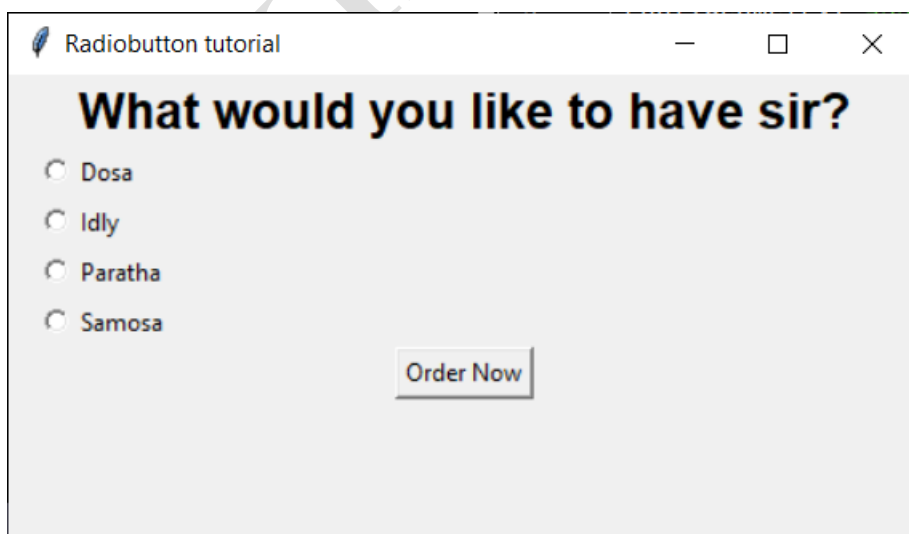
- A button "Order Now" is created using `Button()` widget and `order()` function is called from this button using `command` attribute. The button is packed using `pack()` method.

```
Button(text="Order Now", command=order).pack()
```

- There is a method known by the name `mainloop()` which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *
import tkinter.messagebox as tmsg

root = Tk()
root.geometry("455x233")
root.title("Radiobutton tutorial")

def order():

    tmsg.showinfo("Order Received!", f"We have received your order for {var.get()}. Thanks for ordering")

# var = IntVar()
var = StringVar()
var.set("Radio")
# var.set(1)
Label(root, text = "What would you like to have sir?", font="lucida 19 bold",
      justify=LEFT, padx=14).pack()
radio = Radiobutton(root, text="Dosa", padx=14, variable=var,
value="Dosa").pack(anchor="w")
radio = Radiobutton(root, text="Idly", padx=14, variable=var,
value="Idly").pack(anchor="w")
radio = Radiobutton(root, text="Paratha", padx=14, variable=var,
value="Paratha").pack(anchor="w")
radio = Radiobutton(root, text="Samosa", padx=14, variable=var,
value="Samosa").pack(anchor="w")

Button(text="Order Now", command=order).pack()
root.mainloop()
```

ListBox In Tkinter

Listbox() widget is a standard Tkinter widget which is used to display a list of items from which a user can select a number of items. The listbox can only contain text items, and all items must have the same font and color. Depending on the widget configuration, the user can choose one or more alternatives from the list.

Listboxes are used to select from a group of textual items. Depending on how the listbox is configured, the user can select one or many items from that list.

Attributes:

- **bg:** It is used to set the background color of the widget.
- **fg:** It is used to set the color of the text.
- **bd:** It represents the size of the border. Default value is 2 pixel.
- **font:** It sets the font type of the Listbox items (the font type of the all listbox items will be same).

Methods:

- **activate(index):** It is used to select the lines at the specified index.
- **delete(first, last = None):** It is used to delete the lines which exist in the given range.
- **insert(index, *elements):** It is used to insert the new lines with the specified number of elements before the specified index.
- **get(first, last = None):** It is used to get the list items that exist in the given range.
- **index(i):** It is used to place the line with the specified index at the top of the widget.

Code is described below:

```
from tkinter import *

def add():
    global i
    lbx.insert(ACTIVE, f"{i}")
    i+=1

i = 0
root = Tk()
root.geometry("455x233")
root.title("Listbox tutorial")

lbx = Listbox(root)
lbx.pack()
lbx.insert(END, "First item of our listbox")

Button(root, text="Add Item", command=add).pack()

root.mainloop()
```

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

```
from tkinter import *
```

- To define a function 'def' (i.e. here the function `add()` is defined) is used. Within this function, `insert()` method is used to insert the new lines with the specified number of elements before the specified index. Here the index is `i` and it will be incremented. Here a special index **ACTIVE** is used, which refers to the "active" item (set when you click on an item, or by the arrow keys) and start insert further items before that index only.

```
def add():
    global i
    lbx.insert(ACTIVE, f"{i}")
    i+=1
```

- `i` is initialized to 0. So the list will start adding item from 0.

```
i = 0
```

- To create a main window, *tkinter* offers a method '*Tk*'. To change the name of the window, you can change the `className` to the desired one.

```
root = Tk()
```

- To set the dimensions of the *Tkinter* window and to set the position of the main window on the user's desktop, `geometry()` function is used. As in example: the width is 455 pixels and height is 233 pixels so we can write the function as *geometry(455x233)*.

```
root.geometry("455x233")
```

- For setting the title of the GUI window, `title()` function is used. Here the title is "Listbox tutorial".

```
root.title("Listbox tutorial")
```

- A Listbox "`lbx`" is taken as variable and the list variable is packed using `pack()` method. Then an item is inserted into the listbox ("First item of our listbox") using `insert()` method and a special index **END** is used to append items to the list.

```
lbx = Listbox(root)
lbx.pack()
lbx.insert(END, "First item of our listbox")
```

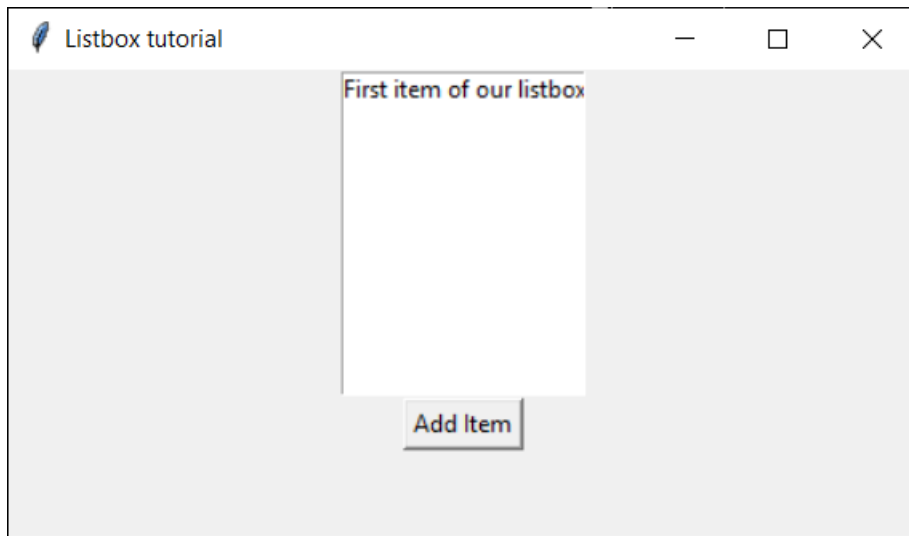
- We extend our little script by button "Add Item". We bind the function `add()` to the "Add Item" button. So, every time this button is clicked, it will call the `add()` function and add list item in the `Listbox()`. The button is packed using `pack()` method.

```
Button(root, text="Add Item", command=add).pack()
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *

def add():
    global i
    lbx.insert(ACTIVE, f"{i}")
    i+=1

i = 0
root = Tk()
root.geometry("455x233")
root.title("Listbox tutorial")

lbx = Listbox(root)
lbx.pack()
lbx.insert(END, "First item of our listbox")

Button(root, text="Add Item", command=add).pack()

root.mainloop()
```

ScrollBar In Tkinter GUI

Scrollbar() widget provides a slide controller that is used to implement vertical scrolled widgets, such as Listbox, Text and Canvas. Horizontal scrollbars can also be used with the **Entry** widget.

To connect a vertical scrollbar to such a widget, you have to do two things:

- Set the widget's **yscrollcommand** callbacks to the **set** method of the scrollbar.

```
widget(yscrollcommand = scrollbar.set)
```

- Set the scrollbar's **command** to the **yview** method of the widget using **config()**.

```
scrollbar.config(command=widget.yview)
```

Attributes:

- **bg:** It is used to set the color of the slider and arrowheads when the mouse is not over them.
- **bd:** The width of the 3-d borders around the entire perimeter of the trough, and also the width of the 3-d effects on the arrowheads and slider. Default is no border around the trough, and a 2-pixel border around the arrowheads and slider.
- **orient:** It is used to set orient=HORIZONTAL for a horizontal scrollbar, orient=VERTICAL for a vertical one.
- **width:** It sets the width of the scrollbar (its y dimension if horizontal, and its x dimension if vertical). Default is 16.

Code is described below:

```
from tkinter import *
root = Tk()
root.geometry("455x233")
root.title("Scrollbar tutorial")
# For connecting scrollbar to a widget
# 1. widget(yscrollcommand = scrollbar.set)
# 2 scrollbar.config(command=widget.yview)
scrollbar = Scrollbar(root)
scrollbar.pack(side=RIGHT, fill=Y)

listbox = Listbox(root, yscrollcommand = scrollbar.set)
for i in range(344):
    listbox.insert(END, f"Item {i}")

listbox.pack(fill="both",padx=22)
#text = Text(root, yscrollcommand = scrollbar.set)
#text.pack(fill=BOTH)
scrollbar.config(command=listbox.yview)
#scrollbar.config(command=text.yview)

root.mainloop()
```

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

```
from tkinter import *
```

- To create a main window, tkinter offers a method 'Tk'. To change the name of the window, you can change the className to the desired one.

```
root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, geometry() function is used. As in example: the width is 455 pixels and height is 233 pixels so we can write the function as *geometry(455x233)*.

```
root.geometry("455x233")
```

- To set the title of the GUI window, title() function is used. Here the title is "Scrollbar Tutorial".

```
root.title("Scrollbar tutorial")
```

- A variable of the Scrollbar() widget is taken as "scrollbar" and is packed at the right hand side of the window using fill=Y (so that whenever the window will be expanded the scrollbar will also be expanded at Y direction).

```
scrollbar = Scrollbar(root)
scrollbar.pack(side=RIGHT, fill=Y)
```

- A ListBox() variable is created and the scrollbar variable is set with the label vertically using "yscrollcommand=scrollbar.set".

```
listbox = Listbox(root, yscrollcommand = scrollbar.set)
```

- In the listbox 0-343 list items are inserted (as range is 344, it will take items 0 to (range-1)). END index is used to append the list.

```
for i in range(344):
    listbox.insert(END, f"Item {i}")
```

- The listbox is packed using pack() method along with the fill and padx attributes.

```
listbox.pack(fill="both", padx=22 )
```

- The scrollbar's **command** is set to the **yview** method of the listbox using config().

```
scrollbar.config(command=listbox.yview)
```

- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

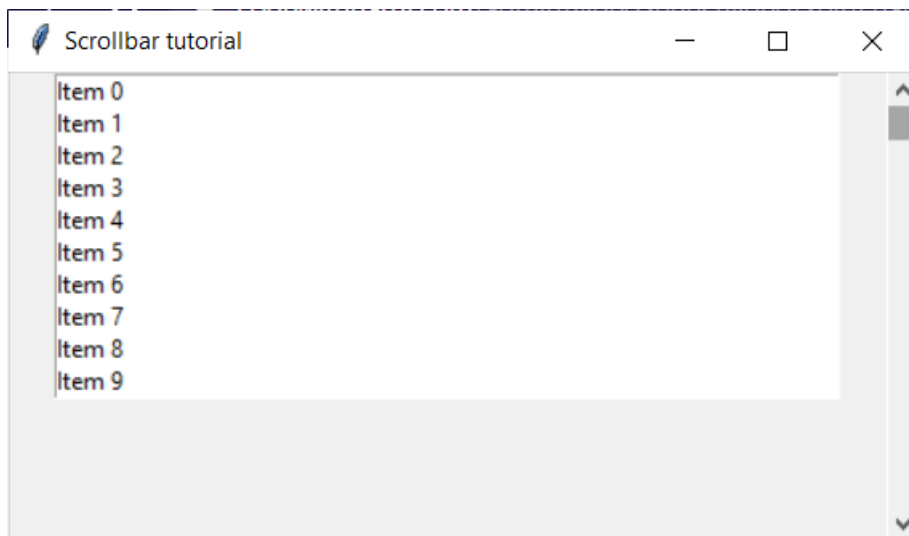
Note: We can use scrollbar with Text() and Canvas() widgets also. One example is shown below with the Text() widget.

```
scrollbar = Scrollbar(root)
scrollbar.pack(side=RIGHT, fill=Y)

text = Text(root, yscrollcommand = scrollbar.set)
text.pack(fill=BOTH)

scrollbar.config(command=text.yview)
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *
root = Tk()
root.geometry("455x233")
root.title("Scrollbar tutorial")
# For connecting scrollbar to a widget
# 1. widget(yscrollcommand = scrollbar.set)
# 2 scrollbar.config(command=widget.yview)
scrollbar = Scrollbar(root)
scrollbar.pack(side=RIGHT, fill=Y)

# listbox = Listbox(root, yscrollcommand = scrollbar.set)
# for i in range(344):
#     listbox.insert(END, f"Item {i}")

# listbox.pack(fill="both",padx=22 )
text = Text(root, yscrollcommand = scrollbar.set)
text.pack(fill=BOTH)
# scrollbar.config(command=listbox.yview)
scrollbar.config(command=text.yview)

root.mainloop()
```


Status Bar In Tkinter

status bar is normally a narrow bar at the bottom of the GUI to indicate some extra information like word counts of the file or anything that could add extra value to the user interface. Tkinter doesn't have a dedicated status bar widget but uses **Label()** widget with appropriate configuration to work as the status bar in the GUI.

Code is described below:

```
from tkinter import *

def upload():
    statusvar.set("Busy..")
    sbar.update()
    import time
    time.sleep(2)
    statusvar.set("Ready")

root = Tk()
root.geometry("455x233")
root.title("Status bar tutorial")

statusvar = StringVar()
statusvar.set("Ready")
sbar = Label(root, textvariable=statusvar, relief=SUNKEN, anchor="w")
sbar.pack(side=BOTTOM, fill=X)
Button(root, text="Upload", command=upload).pack()
root.mainloop()
```

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

```
from tkinter import *
```

- To define a function 'def' (i.e. here the function `upload()` is defined) is used. Within this function "statusvar" variable (`StringVar()`) is set to "Busy.." and `update()` function is used so that the it updates the string "Busy.." on the `Label()` widget. Time module is imported and `time.sleep(2)` is used so that it waits 2 sec. in the previous state ("Busy..") and then it sets to "Ready" state.

```
def upload():
    statusvar.set("Busy..")
    sbar.update()
    import time
    time.sleep(2)
    statusvar.set("Ready")
```

- To create a main window, tkinter offers a method '`Tk`'. To change the name of the window, you can change the className to the desired one.

```
root = Tk()
```

- To set the dimensions of the Tkinter window and to set the position of the main window on the user's desktop, `geometry()` function is used. As in example: the width is 455 pixels and height is 233 pixels so we can write the function as `geometry(455x233)`.

```
root.geometry("455x233")
```

- The title of the GUI window is created using the `"title()"` function. Here the title is "Status bar tutorial".

```
root.title("Status bar tutorial")
```

- A `StringVar()` is taken as "statusvar" (a string variable) and it is set to "Ready" initially.

```
statusvar = StringVar()
statusvar.set("Ready")
```

- A `Label()` is created (this'll actually show as a statusbar) and "statusvar" (`StringVar()`) is taken as "textvariable" where the text is shown after each updation. The attribute "relief" determines how the label appears. We prefer the label to appear "SUNKEN" so that the status bar looks like seamlessly one part of the window. The alignment of the text inside the label is set to "w" (west) using "anchor" attribute.
- The `Label()` variable "sbar" is packed at the BOTTOM of the window and the label is filled along X dimension.

```
sbar = Label(root, textvariable=statusvar, relief=SUNKEN, anchor="w")
sbar.pack(side=BOTTOM, fill=X)
```

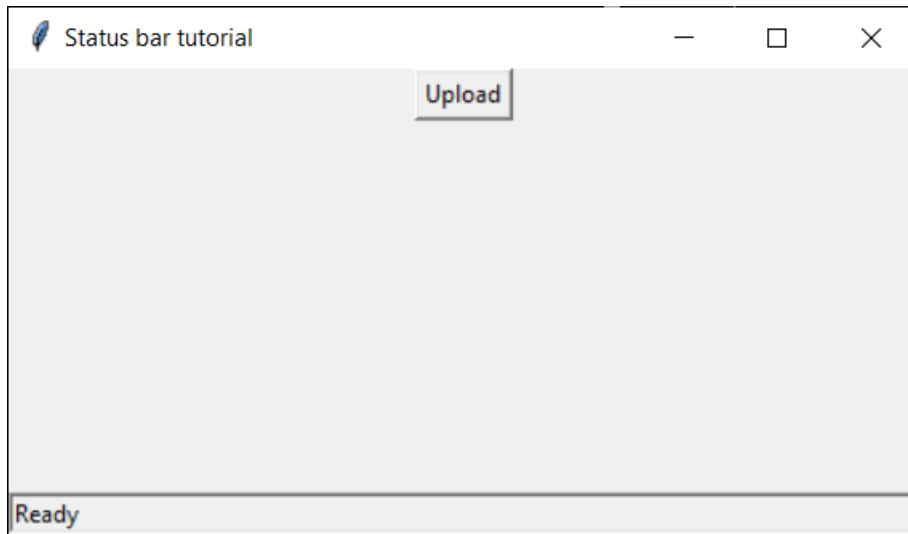
- We extend our little script by button "Upload". We bind the function `upload()` to the *Upload* button. So, every time this button is clicked, it will call the `upload()` function. Then we need to use `pack()` method to pack the button.

```
Button(root, text="Upload", command=upload).pack()
```

- There is a method known by the name `mainloop()` which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
root.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *

def upload():
    statusvar.set("Busy..")
    sbar.update()
    import time
    time.sleep(2)
    statusvar.set("Ready")

root = Tk()
root.geometry("455x233")
root.title("Status bar tutorial")

statusvar = StringVar()
statusvar.set("Ready")
sbar = Label(root, textvariable=statusvar, relief=SUNKEN, anchor="w")
sbar.pack(side=BOTTOM, fill=X)
Button(root, text="Upload", command=upload).pack()
root.mainloop()
```

Using Classes And Objects To Create GUIs

Python is often treated purely as a scripting language, but it is fundamentally an OOP language, actually. With OOP, the user basically states the structure of his program, and user's classes quite literally return "objects," which is why it is called "object" oriented. The objects serve as "instances" of your classes. For writing any lengthy GUI program, it's better to use OOP (classes and objects).

Code is described below:

```
from tkinter import *

class GUI(Tk):
    def __init__(self):
        super().__init__()
        self.geometry("744x377")

    def status(self):
        self.var = StringVar()
        self.var.set("Ready")
        self.statusbar = Label(self, textvar=self.var, relief=SUNKEN,
                                anchor="w")
        self.statusbar.pack(side=BOTTOM, fill=X)

    def click(self):
        print("Button clicked")

    def createbutton(self, inptext):
        Button(text=inptext, command=self.click).pack()

if __name__ == '__main__':
    window = GUI()
    window.status()
    window.createbutton("Click me")
    window.mainloop()
```

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

```
from tkinter import *
```

- A class GUI(Tk) is created which is inherited from Tk (Tkinter). Within this class we have to make a constructor using 'def __init__' and "self" option is used in this constructor rather than root.
- Within this constructor, we have to call the super class constructor using super().__init__().
- We set the geometry of the GUI using geometry() function. As in example: the width is 744 pixels and height is 377 pixels so we can write the function as *geometry(744x377)*.

```
class GUI(Tk):
    def __init__(self):
        super().__init__()
```

```
self.geometry("744x377")
```

- Another function is defined using 'def' ('status' is the function name) and self is passed within this function.
- A variable var is taken as StringVar() and that is set to "Ready".
- A statusbar is created using Label() widget and the var is taken as the textvar of the statusbar.
- The statusbar is packed using pack() method at the BOTTOM filled in the X dimension of the window.

```
def status(self):  
    self.var = StringVar()  
    self.var.set("Ready")  
    self.statusbar = Label(self, textvar=self.var, relief=SUNKEN,  
anchor="w")  
    self.statusbar.pack(side=BOTTOM, fill=X)
```

- A function click() is defined using 'def' which will print "Button clicked" on terminal whenever the function will be called.

```
def click(self):  
    print("Button clicked")
```

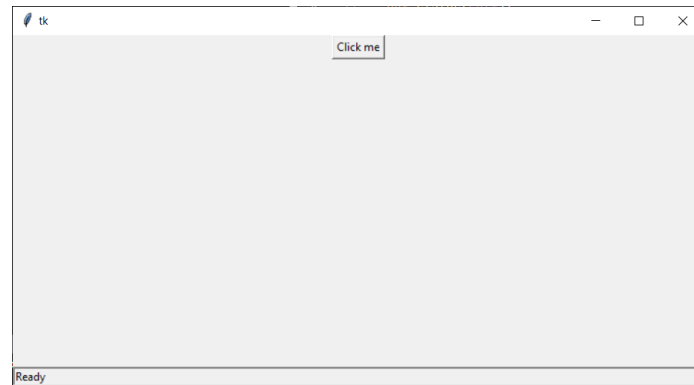
- We extend our little script by defined one more function "createbutton". Within this function a button is created using Button() widget. We bind the button to the *click()* function. So, every time this button is clicked, the text "Button clicked" will be printed on the terminal from which we had called the script. Then we need to use pack() method to pack it.

```
def createbutton(self, inptext):  
    Button(text=inptext, command=self.click).pack()
```

- The main function is created as `__name__ == '__main__'`. We have to make an object named "window" (equivalent to root) of the GUI() class. Similarly, we have to call other functions aswell.
- There is a method known by the name *mainloop()* which is used when your application is ready to run. This is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
if __name__ == '__main__':  
    window = GUI()  
    window.status()  
    window.createbutton("Click me")  
    window.mainloop()
```

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *

class GUI(Tk):
    def __init__(self):
        super().__init__()
        self.geometry("744x377")

    def status(self):
        self.var = StringVar()
        self.var.set("Ready")
        self.statusbar = Label(self, textvar=self.var, relief=SUNKEN,
                                anchor="w")
        self.statusbar.pack(side=BOTTOM, fill=X)

    def click(self):
        print("Button clicked")

    def createbutton(self, inptext):
        Button(text=inptext, command=self.click).pack()

if __name__ == '__main__':
    window = GUI()
    window.status()
    window.createbutton("Click me")
    window.mainloop()
```

More Tkinter Tips, Tricks & Functions

In this tutorial, we'll discuss about some important tips, tricks and functions which one, as a programmer, should always know. But it is suggested to explore more tips and tricks to gain more knowledge!!!

```
from tkinter import *
root = Tk()
root.geometry("655x444")
root.title("CodeWithHarry - Title Of My GUI")
root.wm_iconbitmap("1.ico")
root.configure(background="grey")

width = root.winfo_screenwidth()
height = root.winfo_screenheight()

print(f"{width}x{height}")
Button(text="Close", command = root.destroy).pack()

root.mainloop()
```

The tips, tricks & functions are discussed below:

- **wm_iconbitmap:** It is used to set the window icon. Here the icon "1.ico" will be showed as the window icon.
- **configure():** It is used to change the formatting of the GUI window. Here we set the configuration passing "background" attribute as grey so that the window becomes grey in color.
- **winfo_screenwidth():** This is used to know the width of the current window.
- **winfo_screenheight():** This is used to know the height of the current window.
- **destroy:** This method is used to destroy or close the particular GUI window. Here this method is used with a button. So when the button "Close" will be clicked the GUI window will be closed.

Output: The output of the code (or the GUI window) is given below:



Code as described/written in the video

```
from tkinter import *
root = Tk()
root.geometry("655x444")
root.title("CodeWithHarry - Title Of My GUI")
root.wm_iconbitmap("1.ico")
root.configure(background="grey")

width = root.winfo_screenwidth()
height = root.winfo_screenheight()

print(f"{width}x{height}")
Button(text="Close", command = root.destroy).pack()

root.mainloop()
```


Creating A Calculator Using Tkinter

```
from tkinter import *

def click(event):
    global scvalue
    text = event.widget.cget("text")
    if text == "=":
        if scvalue.get().isdigit():
            value = int(scvalue.get())
        else:
            try:
                value = eval(screen.get())

            except Exception as e:
                print(e)
                value = "Error"

        scvalue.set(value)
        screen.update()

    elif text == "C":
        scvalue.set("")
        screen.update()

    else:
        scvalue.set(scvalue.get() + text)
        screen.update()

root = Tk()
root.geometry("644x970")
root.title("Calclator By CodeWithHarry")
root.wm_iconbitmap("1.ico")

scvalue = StringVar()
scvalue.set("")
screen = Entry(root, textvar=scvalue, font="lucida 40 bold")
screen.pack(fill=X, ipadx=8, pady=10, padx=10)

f = Frame(root, bg="grey")
b = Button(f, text="9", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="8", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="7", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

f.pack()
```

```

f = Frame(root, bg="grey")
b = Button(f, text="6", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="5", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="4", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

f.pack()

f = Frame(root, bg="grey")
b = Button(f, text="3", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="2", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="1", padx=28, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

f.pack()

f = Frame(root, bg="grey")
b = Button(f, text="0", padx=31, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="-", padx=31, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="*", padx=31, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

f.pack()

f = Frame(root, bg="grey")
b = Button(f, text="/", padx=33, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="%", padx=21, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="=", padx=27, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

```

```
f.pack()

f = Frame(root, bg="grey")
b = Button(f, text="C", padx=26, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text=".", padx=26, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

b = Button(f, text="00", padx=26, pady=18, font="lucida 35 bold")
b.pack(side=LEFT, padx=18, pady=5)
b.bind("", click)

f.pack()

root.mainloop()
```

Creating a GUI Notepad In Tkinter

```
from tkinter import *
from tkinter.messagebox import showinfo
from tkinter.filedialog import askopenfilename, asksaveasfilename
import os

def newFile():
    global file
    root.title("Untitled - Notepad")
    file = None
    TextArea.delete(1.0, END)

def openFile():
    global file
    file = askopenfilename(defaultextension=".txt",
                           filetypes=[("All Files", "*.*"),
                                       ("Text Documents", "*.txt")])

    if file == "":
        file = None
    else:
        root.title(os.path.basename(file) + " - Notepad")
        TextArea.delete(1.0, END)
        f = open(file, "r")
        TextArea.insert(1.0, f.read())
        f.close()

def saveFile():
    global file
    if file == None:
        file = asksaveasfilename(initialfile = 'Untitled.txt',
                                defaultextension=".txt",
                                filetypes=[("All Files", "*.*"),
                                            ("Text Documents", "*.txt")])

        if file == "":
            file = None
        else:
            #Save as a new file
            f = open(file, "w")
            f.write(TextArea.get(1.0, END))
            f.close()

            root.title(os.path.basename(file) + " - Notepad")
            print("File Saved")
    else:
        # Save the file
        f = open(file, "w")
        f.write(TextArea.get(1.0, END))
        f.close()

def quitApp():
```

```

        root.destroy()

def cut():
    TextArea.event_generate(("<>"))

def copy():
    TextArea.event_generate(("<>"))

def paste():
    TextArea.event_generate(("<>"))

def about():
    showinfo("Notepad", "Notepad by Code With Harry")

if __name__ == '__main__':
    #Basic tkinter setup
    root = Tk()
    root.title("Untitled - Notepad")
    root.wm_iconbitmap("1.ico")
    root.geometry("644x788")

    #Add TextArea
    TextArea = Text(root, font="lucida 13")
    file = None
    TextArea.pack(expand=True, fill=BOTH)

    # Lets create a menubar
    MenuBar = Menu(root)

    #File Menu Starts
    FileMenu = Menu(MenuBar, tearoff=0)
    # To open new file
    FileMenu.add_command(label="New", command=newFile)

    #To Open already existing file
    FileMenu.add_command(label="Open", command = openFile)

    # To save the current file

    FileMenu.add_command(label = "Save", command = saveFile)
    FileMenu.add_separator()
    FileMenu.add_command(label = "Exit", command = quitApp)
    MenuBar.add_cascade(label = "File", menu=FileMenu)
    # File Menu ends

    # Edit Menu Starts
    EditMenu = Menu(MenuBar, tearoff=0)
    #To give a feature of cut, copy and paste
    EditMenu.add_command(label = "Cut", command=cut)
    EditMenu.add_command(label = "Copy", command=copy)
    EditMenu.add_command(label = "Paste", command=paste)

    MenuBar.add_cascade(label="Edit", menu = EditMenu)

    # Edit Menu Ends

    # Help Menu Starts
    HelpMenu = Menu(MenuBar, tearoff=0)
    HelpMenu.add_command(label = "About Notepad", command=about)
    MenuBar.add_cascade(label="Help", menu=HelpMenu)

```

```
# Help Menu Ends

root.config(menu=MenuBar)

#Adding Scrollbar using rules from Tkinter lecture no 22
Scroll = Scrollbar(TextArea)
Scroll.pack(side=RIGHT, fill=Y)
Scroll.config(command=TextArea.yview)
TextArea.config(yscrollcommand=Scroll.set)

root.mainloop()
```

Tkinter notes