Day-8: GUI
29/06/2018

Department Of
Computer Science
Engineering

SKFGI, Mankundu

Python 3 - GUI Programming (Tkinter)

Python provides various options for developing graphical user interfaces (GUIs). The most important features are listed below.

- **Tkinter** – Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look this option in this chapter.
- **wxPython** – WxPython brings the wxWidgets cross-platform GUI library from its native C++ to Python. WxPython is a slightly more modern approach to, which looks a little more native than Tkinter across different operating systems as it does not attempt to create its own set of widgets (although these can be themed to look much like native components). It's fairly easy to get started with as well, and has a growing developer community. You may need to bundle wxPython with your applications, as it is not automatically installed with Python.
- **PyQt** –This is also a Python interface for a popular cross-platform Qt GUI library.
- **JPython** – JPython is a Python port for Java, which gives Python scripts seamless access to the Java class libraries on the local machine

1.Tkinter Programming

Tkinter is the standard GUI library for Python. 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. All you need to do is perform the following steps –

Import the *Tkinter* module.

Create the GUI application main window.

Add one or more of widgets to the GUI application.

Enter the main event loop to take action against each event triggered by the user.

Example

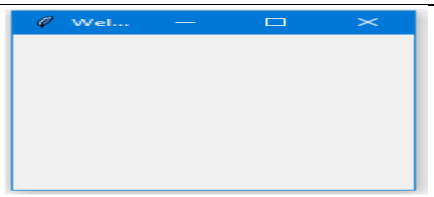
```
From tkinter import *
 # note that module name has changed from Tkinter in Python 2 to
tkinter in Python 3
top = Tk()
# Code to add widgets will go here...
top.mainloop()
```

The last line calls the mainloop function. This function calls the endless loop of the window, so the window will wait for any user interaction till we close it.

If you forget to call the mainloop function, nothing will appear to the user.

1.1To rename the widgets

```
from tkinter import *
window = Tk()
window.title("Welcome ")
window.mainloop()
```



1.2Setting Window Size

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

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

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

1.3Print the screen size

```
from tkinter import*
root=Tk()
w=root.winfo_screenwidth()
h=root.winfo_screenheight()
print(w)
print(h)
```

1.4To make Fullscreen

```
from tkinter import*
root=Tk()
root.attributes('-fullscreen',True)
```

2.Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets. There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table –

2.1Label: This widget implements a display box where you can place text or images. The text displayed by this widget can be updated at any time you want.

Create a Label Widget

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

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

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

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

So the complete code will be like this:

```
from tkinter import *
window = Tk()
window.title("Welcome to SKFGI")
lbl = Label(window, text="Hello")
lbl.grid(column=0, row=0)
window.mainloop()
```

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

Syntax

Here is the simple syntax to create this widget –

```
w = Label ( master, option, ... )
```

Parameters

master – This represents the parent window.

options – Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Option	Description	Example
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.	lbl = Label(window, text="Hello")
Bg	The normal background color displayed behind the label and indicator.	lbl = Label(window, text="Hello",bg='red')
bd	The size of the border around the indicator. Default is 2 pixels.	lbl = Label(window, text="Hello",bd=20)
font	If you are displaying text in this label with the text or textvariable option, the font option specifies in what font that text will be displayed.	lbl = Label(window, text="Hello", font='times 50 underline bold italic')
fg	If you are displaying text in this label, this option specifies the color of the text.	lbl = Label(window, text="Hello",fg='red')
height	The vertical dimension of the new	lbl = Label(window,

	frame.	text="Hello",height= 100)
image	To display a static image in the label widget, set this option to an image object.	from tkinter import * main = Tk() imagePath = PhotoImage(file="Untitled.png") widgetf = Label(main, image=imagePath).pack(side="right") comments = """hi""" widgets = Label(main, text=comments).pack(side="left") main.mainloop()
padx	Extra space added to the left and right of the text within the widget. Default is 1.	lbl = Label(window, text="Hello",bg="red", fg="white", padx=100)
pady	Extra space added above and below the text within the widget. Default is 1.	lbl = Label(window, text="Hello",bg="red", fg="white", pady=100)
underline	You can display an underline (_) below the nth letter of the text, counting from 0, by setting this option to n. The default is underline = -1, which means no underlining.	lbl = Label(window,text="Hello",bg="red", fg="white", underline=4)
width	Width of the label in characters (not pixels!). If this option is not set, the label will be sized to fit its contents.	lbl = Label(window,text="Hello",bg="red", fg="white", width=100)

font	Font has 3 argument, 1st font name and 2nd one is font size,and 3rd one italic bold underlin	lbl = Label(window, text="Hello", font=("Arial", 16))
relief	The relief style of a widget refers to certain simulated 3-D effects around the outside of the widget. Here is a screenshot of a row of buttons exhibiting all the possible relief styles – Here is list of possible constants which can be used for relief attribute – FLAT RAISED SUNKEN GROOVE RIDGE	lbl = Label(window,text="Hello",relief=RAISED)
cursors	Python Tkinter supports quite a number of different mouse cursors available. The exact graphic may vary according to your operating system. Here is the list of interesting ones "arrow", "circle", "clock", "cross", "dotbox", "exchange", "fleur", "heart", "heart", "man", "mouse" , "pirate", "plus", "shuttle" , "sizing" , "spider", "spraycan", "star", "target", "tcross", "trek", "watch"	lbl = Label(window,text="Hello",relief=RAISED,cursor=”heart”)

2.2Button widget :

The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons. You can attach a function or a method to a button which is called automatically when you click the button.

Syntax

Here is the simple syntax to create this widget –

```
w = Button ( master, option = value, ... )
```

Parameters

master – This represents the parent window.

options – Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Example

```
from tkinter import *
top = Tk()
L1 = Label(top, text = "User Name")
L1.pack( side = LEFT)
E1 = Entry(top, bd = 5)
E1.pack(side = RIGHT)
```

Option	Description	Example
activebackground	Background color are changed when you press the button	lbl = Button(window,text=s,bd=5,activebackground="red")

activeforeground	foreground color are changed when you press the button	lbl = Button(window,text=s,bd=5,activeforeground="red")
command	Function or method to be called when the button is clicked.	from tkinter import * window = Tk() window.title("Welcome to python gui") window.geometry('350x200') lbl = Label(window, text="Hello") lbl.grid(column=0, row=0) def clicked(): lbl.configure(text="Button was clicked !!") btn = Button(window, text="Click Me", command=clicked) btn.grid(column=1, row=0) window.mainloop()
state	Set this option to DISABLED to gray out the button and make it unresponsive. Has the value ACTIVE when the mouse is over it. Default is NORMAL.	lbl = Button(window,text=s,bd=5,state=DISABLED)

All the options of Label are applicable for Button. Other option of Button are given below

2.3Entry widget:

The Entry widget is used to accept single-line text strings from a user.

If you want to display multiple lines of text that can be edited, then you should use the *Text* widget.

If you want to display one or more lines of text that cannot be modified by the user, then you should use the *Label* widget.

Syntax

Here is the simple syntax to create this widget –

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

Parameters

master – This represents the parent window.

options – Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Example:

```
from tkinter import *
top = Tk()
L1 = Label(top, text = "User Name")
L1.pack( side = LEFT)
E1 = Entry(top, bd = 5)
E1.pack(side = RIGHT)
```

Options: bg, bd, command, font , fg, relief, state,cursors are same as before

Option	Description	Example
selectbackground	The background color to use displaying selected text.	E1 = Entry(top, bd = 5,selectbackground='Red')
selectforeground	The foreground (text) color of selected text.	E1 = Entry(top, bd = 5,selectforeground='Red')

show	Normally, the characters that the user types appear in the entry. To make a .password. entry that echoes each character as an asterisk, set show = "*".	E1 = Entry(top, bd = 5, show="*")

Methods of Entry

delete (first, last = n)

Deletes characters from the widget, starting with the one at index first, up to but not including the character at position last. If the second argument is omitted, only the single character at position first is deleted. If you want to delete all character then use last='end'

```
from tkinter import*
master=Tk()
def e1_delete():
    e1.delete(first=0,last='end')  #if last=20 then it delete 20 character from first
e1=Entry(master, width=20)
e1.pack()
B=Button(master, text="del", command=e1_delete)
B.pack()
master.mainloop
```

get(): Returns the entry's current text as a string.

```
from tkinter import*
master=Tk()
def e1_data():
    s=e1.get()
    l=Label(master,text=s)
    l.pack()
e1=Entry(master, width=20)
e1.pack()
B=Button(master, text="click", command=e1_data)
B.pack()
master.mainloop
```

icursor (index)

Set the insertion cursor just before the character at the given index.

```
from tkinter import*
master=Tk()
def move_cursor():
    e1.icursor ( 5 )
e1=Entry(master, width=20)
e1.pack()
B=Button(master, text="click", command=move_cursor)
B.pack()
master.mainloop
```

insert (index, s)

Inserts string s before the character at the given index.

```
from tkinter import*
master=Tk()
def insert1():
    s="skfgi"
    e1.insert(5,s)
e1=Entry(master, width=20)
e1.pack()
B=Button(master, text="click", command=insert1)
B.pack()
master.mainloop
```

Set the Focus of the Entry Widget

That's super easy, all we need to do is to call the focus function like this:

```
e1.focus()
```

And when you run your code, you will notice that the entry widget has the focus so you can write your text right away.

2.4Checkbutton widget : The Checkbutton widget is used to display a number of options to a user as toggle buttons. The user can then select one or more options by clicking the button corresponding to each option. You can also display images in place of text.

Syntax

Here is the simple syntax to create this widget –

```
w = Checkbutton ( master, option, ... )
```

Parameters

master – This represents the parent window.

options – Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

```
from tkinter import *
top = Tk()
C1 = Checkbutton(top, text = "Music" )
C2 = Checkbutton(top, text = "Video")
C1.pack()
C2.pack()
top.mainloop()
```

Activebackground, activeforeground, ,bd, bg, command, cursor, font, fg, height, image, padx, pady , relief, state, text, underline, width option are already discussed

Methods

Following are commonly used methods for this widget –

S.No.	Medthod & Description
1	**deselect()** Clears (turns off) the checkbutton. Ex: C1.deselect()
2	**select()** Sets (turns on) the checkbutton. Ex:C1.select()
3	**toggle()** Clears the checkbutton if set, sets it if cleared.

2.5Radiobutton

This widget implements a multiple-choice button, which is a way to offer many possible selections to the user and lets user choose only one of them.

In order to implement this functionality, each group of radiobuttons must be associated to the same variable and each one of the buttons must symbolize a single value. You can use the Tab key to switch from one radionbutton to another.

Syntax

Here is the simple syntax to create this widget –

```
w = Radiobutton ( master, option, ... )
```

Parameters

master – This represents the parent window.

options – Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

The options and deslect(), select() methods are same as check box

Option	Description	Example
value	The value property on a radio button is used server-side to determine which button was selected. It's important to set different values for radio buttons in the same group for server-side implementation . if there are three radio button then value=1 for 1st, value=2 for 2nd, value=3 for 3rd	rad1 = Radiobutton(window,text='First', value=1)

```
from tkinter import *
#from tkinter.ttk import *
window = Tk()
```

```
window.title("Welcome to SKFGI")
window.geometry('350x200')
rad1 = Radiobutton(window,text='First', value=1)
rad1.select()
rad2 = Radiobutton(window,text='Second', value=2)
rad3 = Radiobutton(window,text='Third', value=3)
rad1.grid(column=0, row=0)
rad2.grid(column=1, row=0)
rad3.grid(column=2, row=0)
window.mainloop()
```

example2: Get Radio Button Values

```
from tkinter import *
window = Tk()
window.title("Welcome to LikeGeeks app")
selected = IntVar()
rad1 = Radiobutton(window,text='First', value=1, variable=selected)
rad2 = Radiobutton(window,text='Second', value=2, variable=selected)
rad3 = Radiobutton(window,text='Third', value=3, variable=selected)
def clicked():
      print(selected.get())
btn = Button(window, text="Click Me", command=clicked)
rad1.grid(column=0, row=0)
rad2.grid(column=1, row=0)
rad3.grid(column=2, row=0)
btn.grid(column=3, row=0)
window.mainloop()
```

2.6Message Box

The MessageBox module is used to display message boxes in your applications. This module provides a number of functions that you can use to display an appropriate message.

Some of these functions are showinfo, showwarning, showerror, askquestion, askokcancel, askyesno, and askretrycancel.

showinfo	messagebox.showinfo('Message title', 'Message content')
showwarning	messagebox.showwarning('Message title', 'Message content')
showerror	messagebox.showerror('Message title', 'Message content') #shows error message
askquestion	res = messagebox.askquestion('Message title','Message content')
askokcancel	res = messagebox.askokcancel('Message title','Message content')
askyesno	res = messagebox.askyesno('Message title','Message content')
askretrycancel	res = messagebox.askretrycancel('Message title','Message content')
askyesnocancel	res = messagebox.askyesnocancel('Message title','Message content')

Syntax

Here is the simple syntax to create this widget –

```
tkMessageBox.FunctionName(title, message [, options])
```

Parameters

FunctionName – 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.

options – options are alternative choices that you may use to tailor a standard message box. Some of the options that you can use are default and parent. The default option is used to specify the default button, such as ABORT, RETRY, or IGNORE in the message box. The parent option is

used to specify the window on top of which the message box is to be displayed.

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

```
from tkinter import messagebox
messagebox.showinfo('Message title','Message content')
```

example:

```
from tkinter import *
from tkinter import messagebox
window = Tk()
window.title("Welcome to LikeGeeks app")
window.geometry('350x200')
def clicked():
    messagebox.showinfo('hello', 'how r u?')
btn = Button(window,text='Click here', command=clicked)
btn.grid(column=0,row=0)
window.mainloop()
```

2.8Frame:

The Frame widget is very important for the process of grouping and organizing other widgets in a somehow friendly way. It works like a container, which is responsible for arranging the position of other widgets.

It uses rectangular areas in the screen to organize the layout and to provide padding of these widgets. A frame can also be used as a foundation class to implement complex widgets.

Syntax

Here is the simple syntax to create this widget –

```
w = Frame ( master, option, ... )
```

the options are bg, bd, cursor, height, relief, width same as others.

Example:

```
from tkinter import *
root = Tk()
frame = Frame(root)
```

```
frame.pack()
bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )
redbutton = Button(frame, text = "Red", fg = "red")
redbutton.pack( side = LEFT)
greenbutton = Button(frame, text = "Brown", fg="brown")
greenbutton.pack( side = LEFT )
bluebutton = Button(frame, text = "Blue", fg = "blue")
bluebutton.pack( side = LEFT )
blackbutton = Button(bottomframe, text = "Black", fg = "black")
blackbutton.pack( )
root.mainloop()
```

2.9List box:

The Listbox widget is used to display a list of items from which a user can select a number of items

Syntax

Here is the simple syntax to create this widget −

```
w = Listbox ( master, option, ... )
```

```
from tkinter import *
top = Tk()
Lb1 = Listbox(top)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.insert(4, "PHP")
Lb1.insert(5, "JSP")
Lb1.insert(6, "Ruby")
Lb1.pack()
top.mainloop()
```

2.10Combobox Widget

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

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

```
from tkinter import *
from tkinter.ttk import *
window = Tk()
def clicked():
   print(combo.get()) #to get the data from combo box
window.title("Welcome ")
window.geometry('350x200')
combo = Combobox(window)
combo['values']= (1, 2, 3, 4, 5, "Text")
combo.current(1) #set the selected item
combo.grid(column=0, row=0)
btn = Button(window, text="Click Me", command=clicked)
btn.grid(row=1,column=1)
window.mainloop()
```

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

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

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

```
combo.get()
```

2.11 ScrolledText Widget (Tkinter textarea)

To add a ScrolledText widget, you can use the ScrolledText class like this:

```
from tkinter import scrolledtext
txt = scrolledtext.ScrolledText(window,width=40,height=10)
```

Here we specify the width and the height of the ScrolledText widget, otherwise, it will fill the entire window.

```
from tkinter import *
from tkinter import scrolledtext
window = Tk()
```

```
window.title("Welcome to LikeGeeks app")
window.geometry('350x200')
txt = scrolledtext.ScrolledText(window,width=40,height=10)
txt.grid(column=0,row=0)
window.mainloop()
```

Set Scrolledtext Content

To set scrolledtext content, you can use the insert method like this:

```
txt.insert(INSERT,'You text goes here')
```

Delete/Clear Scrolledtext Content

To clear the contents of a scrolledtext widget, you can use the delete method like this:

```
txt.delete(1.0,END)
```

2.12SpinBox (Numbers Widget)

To create a Spinbox widget, you can use the Spinbox class like this:

```
spin = Spinbox(window, from_=0, to=100)
```

Here we create a Spinbox widget and we pass the from_ and to parameters to specify the numbers range for the Spinbox.

Also, you can specify the width of the widget using the width parameter:

```
spin = Spinbox(window, from_=0, to=100, width=5)
```

Check the complete example:

```
from tkinter import *
window = Tk()
window.title("Welcome ")
window.geometry('350x200')
spin = Spinbox(window, from_=0, to=100, width=5)
spin.grid(column=0,row=0)
window.mainloop()
```

You can specify the numbers for the Spinbox instead of using the whole range like this:

```
spin = Spinbox(window, values=(3, 8, 11), width=5)
```

Here the Spinbox widget only shows these 3 numbers: 3, 8, and 11.

Set a Default Value for Spinbox

To set the Spinbox default value, you can pass the value to the textvariable parameter like this:

```
var =IntVar()
var.set(36)
spin = Spinbox(window, from_=0, to=100, width=5, textvariable=var)
```

Now, if you run the program, it will show 36 as a default value for the Spinbox.

3.Geometry Management

All Tkinter widgets have access to the specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

3.1The pack() Method – Pack is the easiest to use of the three geometry managers of Tk and Tkinter. Instead of having to declare precisely where a widget should appear on the display screen, we can declare the positions of widgets with the pack command relative to each other. The pack command takes care of the details. Though the pack command is easier to use, this layout managers is limited in its possibilities compared to the grid and place mangers. For simple applications it is definitely the manager of choice. For example simple applications like placing a number of widgets side by side, or on top of each other.

fill Option

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

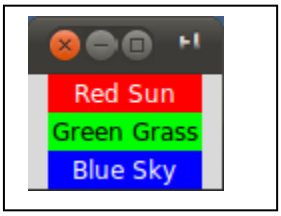
label has been given the size of the text. If you want to make the widgets as wide as the parent widget, you have to use the fill=X option:

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

Padding

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

padx External padding, horizontally




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

pady External padding, vertically

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

Placing widgets side by side

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

```
from Tkinter import *
root = Tk()
w = Label(root, text="red", bg="red", fg="white")
w.pack(padx=5, pady=10, side=LEFT)
w = Label(root, text="green", bg="green", fg="black")
w.pack(padx=5, pady=20, side=LEFT)
w = Label(root, text="blue", bg="blue", fg="white")
w.pack(padx=5, pady=20, side=LEFT)
mainloop()
```

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

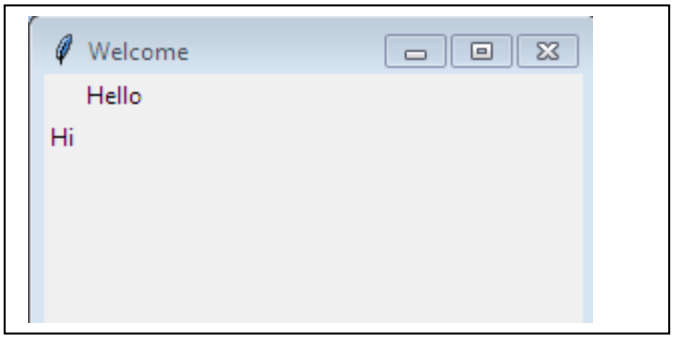
3.2<u>The grid() Method</u> – The first geometry manager of Tk had been pack. The algorithmic behaviour of pack is not easy to understand and it can be difficult to change an existing design. Grid was introduced in 1996 as an alternative to pack. Though grid is easier to learn and to use and produces nicer layouts, lots of developers keep using pack. Grid is in many cases the best choice for general use. While pack is sometimes not sufficient for changing details in the layout, place gives you complete control of positioning each element, but this makes it a lot

more complex than pack and grid.

The Grid geometry manager places the widgets in a 2-dimensional table, which consists of a number of rows and columns. The position of a widget is defined by a row and a column number. Widgets with the same column number and different row numbers will be above or below each other. Correspondingly, widgets with the same row number but different column numbers will be on the same "line" and will be beside of each other, i.e. to the left or the right.

Using the grid manager means that you create a widget, and use the grid method to tell the manager in which row and column to place them. The size of the grid doesn't have to be defined, because the manager automatically determines the best dimensions for the widgets used.

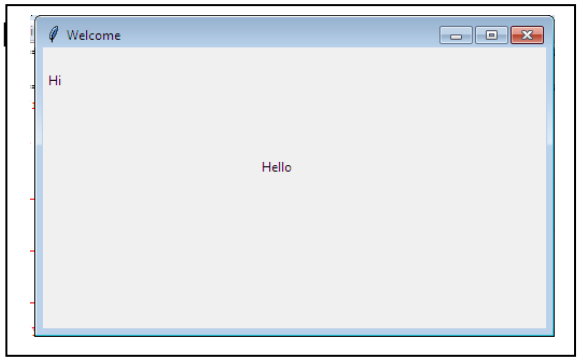
```
from tkinter import *
window = Tk()
window.title("Welcome ")
lbl = Label(window,
text="Hello")
lbl1 = Label(window,
text="Hi")
lbl.grid(row=0,column=2)
lbl1.grid(row=1,column=1)
window.mainloop()
```



3.3<u>The place() Method</u> – The Place geometry manager allows you explicitly set the position and size of a window, either in absolute terms, or relative to another window. The place manager can be accessed through the place method. It can be a

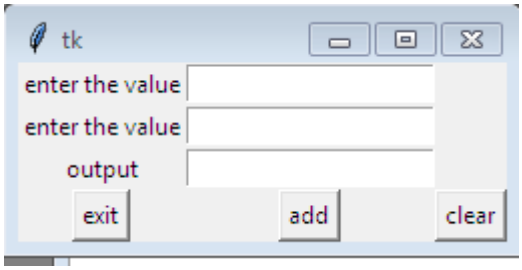
```
from tkinter import *
window = Tk()
window.title("Welcome ")
lbl = Label(window, text="Hello")
lbl1 = Label(window, text="Hi")
lbl.place(x=200,y=100)
lbl1.place(x=2,y=20)
window.mainloop()
```

er:

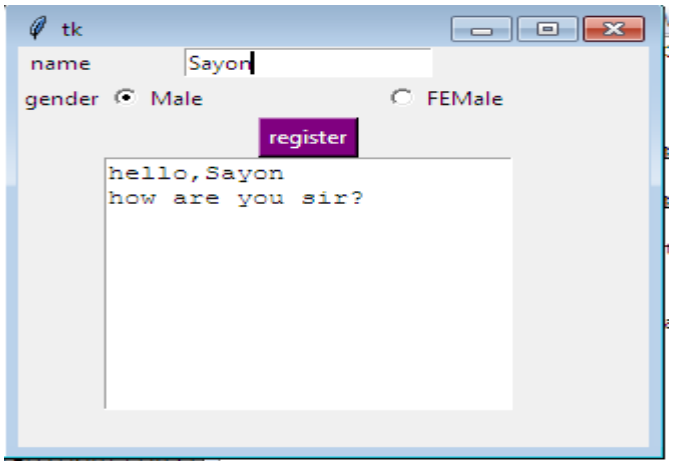


```
from tkinter import*
from tkinter import messagebox
root=Tk()

def clear_n():
        e1.delete(first=0,last='end')
        e2.delete(first=0,last='end')
        e3.delete(first=0,last='end')
        e1.focus()

def add_n():
        try:
        x=float(e1.get())
        y=float(e2.get())
        s=x+y
        e3.insert(0,s)
        except ValueError:
        messagebox.showinfo('Wrong input', 'enter only number')

l1=Label(root,text="enter the value")
l2=Label(root,text="enter the value")
l3=Label(root,text="output")
e1=Entry(root)
e1.focus()
e2=Entry(root)
e3=Entry(root)
l1.grid(row=0,column=0)
l2.grid(row=1,column=0)
l3.grid(row=2,column=0)
e1.grid(row=0,column=1)
e2.grid(row=1,column=1)
```

```
e3.grid(row=2,column=1)
b1=Button(root,text="add",command=add_n)
b2=Button(root,text="clear",command=clear_n)
b3=Button(root,text="exit",command=root.destroy)
b1.grid(row=4,column=1)
b2.grid(row=4,column=3)
b3.grid(row=4,column=0)
root.mainloop()
```



P2.




```
from tkinter import *
def show_data():
txt.delete(0.0,'end')
txtname=ent.get()
gender=var_chk.get()
if gender==1:
gender="sir"
else:
```

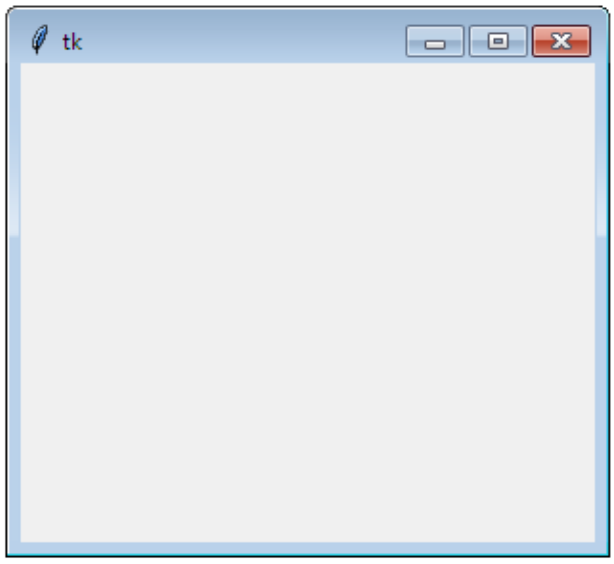
```
gender="mam"
s="hello,"+str(txtname)+"\nhow are you "+gender+"?"
txt.insert(0.0,s)
root=Tk()
root.geometry("200x250")
l1=Label(root,text="name")
l2=Label(root,text="gender")
ent= Entry(root)
var_chk=IntVar()
rd1=Radiobutton(root,text="Male", variable=var_chk,value=1)
rd2=Radiobutton(root,text="FEMale", variable=var_chk,value=2)
l1.grid(row=0)
l2.grid(row=1)
ent.grid(row=0,column=1)
rd1.grid(row=1,column=1,sticky=W)
rd2.grid(row=1,column=1,sticky=E)
btn=Button(root,text="register",bg="purple",
fg="white",command=show_data)
btn.grid(row=2,column=1)
txt=Text(root, width=25,height=10,wrap=WORD)
txt.grid(row=3,column=1)
```

P3: Mouse event

```
from tkinter import *
def left (event):
print("left button")
def right (event):
print("right button")
def middle (event):
print("middle button")
```

```
root=Tk()
mf=Frame(root, width=300, height=250)
mf.bind("<Button-1>",left)
mf.bind("<Button-2>",middle)
mf.bind("<Button-3>",right)

mf.pack()
root.mainloop()
```

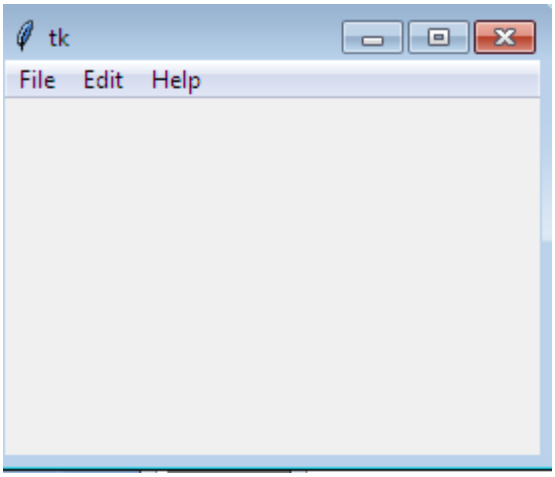
```
= RESTART: C:\Users\PC\AppData\Local\Programs\Python\Python36-32\project.py =
>>>
 RESTART: C:\Users\PC\AppData\Local\Programs\Python\Python36-32\project2_mouse.p
y
left button
right button
middle button
```



P4:Create a Menu

```
from tkinter import *
def donothing():
filewin = Toplevel(root)
```

```
button = Button(filewin, text="Do nothing button")
button.pack()

root = Tk()
menubar = Menu(root)
filemenu = Menu(menubar, tearoff = 0)
filemenu.add_command(label="New", command = donothing)
filemenu.add_command(label = "Open", command = donothing)
filemenu.add_command(label = "Save", command = donothing)
filemenu.add_command(label = "Save as...", command = donothing)
filemenu.add_command(label = "Close", command = donothing)
filemenu.add_separator()
filemenu.add_command(label = "Exit", command = root.quit)
menubar.add_cascade(label = "File", menu = filemenu)
editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label = "Undo", command = donothing)
editmenu.add_separator()
editmenu.add_command(label = "Cut", command = donothing)
editmenu.add_command(label = "Copy", command = donothing)
editmenu.add_command(label = "Paste", command = donothing)
editmenu.add_command(label = "Delete", command = donothing)
editmenu.add_command(label = "Select All", command = donothing)
menubar.add_cascade(label = "Edit", menu = editmenu)
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label = "Help Index", command = donothing)
helpmenu.add_command(label = "About...", command = donothing)
menubar.add_cascade(label = "Help", menu = helpmenu)
root.config(menu = menubar)
root.mainloop()
```


tk
File
Edit
Help