18CSC207J – Advanced Programming Practice

C.Arun, Asst. Prof. Dept of Software Engineering, School of Computing, SRMIST

GUI & Event Handling Programming Paradigm

Event Driven Programming Paradigm

- Event-driven programming is a programming paradigm in which the flow of program execution is determined by events - for example a user action such as a mouse click, key press, or a message from the operating system or another program.
- An event-driven application is designed to detect events as they occur, and then deal with them using an appropriate event-handling procedure.
- In a typical modern event-driven program, there is no discernible flow of control. The main routine is an event-loop that waits for an event to occur, and then invokes the appropriate event-handling routine.
- Event callback is a function that is invoked when something significant happens like when click event is performed by user or the result of database query is available.

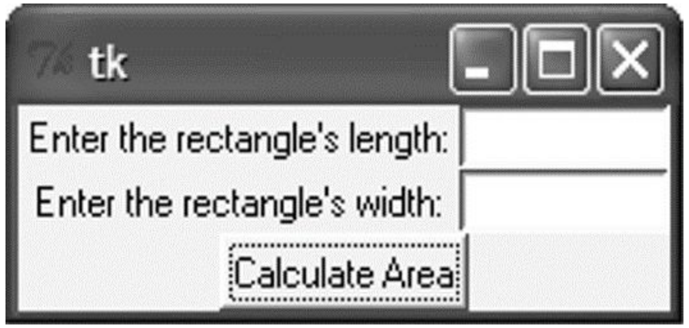
Event Handlers: Event handlers is a type of function or method that run a specific action when a specific event is triggered. For example, it could be a button that when user click it, it will display a message, and it will close the message when user click the button again, this is an event handler.

Trigger Functions: Trigger functions in event-driven programming are a functions that decide what code to run when there are a specific event occurs, which are used to select which event handler to use for the event when there is specific event occurred.

Events: Events include mouse, keyboard and user interface, which events need to be triggered in the program in order to happen, that mean user have to interacts with an object in the program, for example, click a button by a mouse, use keyboard to select a button and etc.

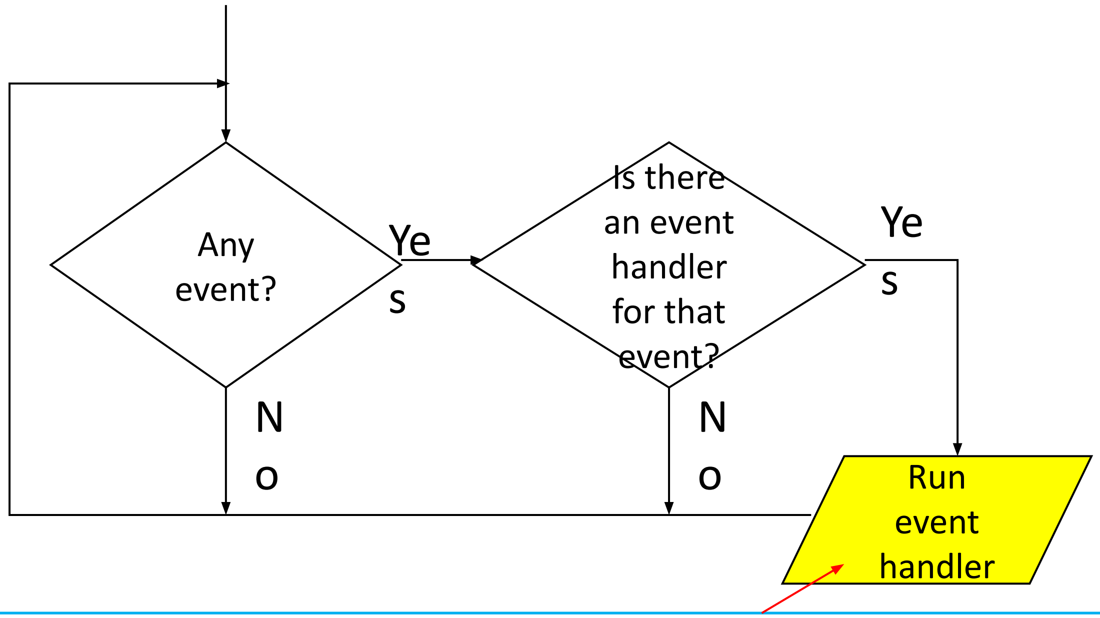
Introduction

- A graphical user interface allows the user to interact with the operating system and other programs using graphical elements such as icons, buttons, and dialog boxes.
- GUIs popularized the use of the mouse.
- GUIs allow the user to point at graphical elements and click the mouse button to activate them.
- GUI Programs Are Event-Driven
- User determines the order in which things happen
- GUI programs respond to the actions of the user, thus they are event driven.
- The tkinter module is a wrapper around tk, which is a wrapper around tcl, which is what is used to create windows and graphical user interfaces.



Introduction

- A major task that a GUI designer needs to do is to determine what will happen when a GUI is invoked
- Every GUI component may generate different kinds of “events” when a user makes access to it using his mouse or keyboard
- E.g. if a user moves his mouse on top of a button, an event of that button will be generated to the Windows system
- E.g. if the user further clicks, then another event of that button will be generated (actually it is the click event)
- For any event generated, the system will first check if there is an event handler, which defines the action for that event
- For a GUI designer, he needs to develop the event handler to determine the action that he wants Windows to take for that event.



GUI Using Python

- Tkinter: Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.
- wxPython: This is an open-source Python interface for wxWindows
- 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 Java class libraries on the local machine

Tkinter Programming

- Tkinter is the standard GUI library for Python.
- Creating a GUI application using Tkinter

Steps

- Import the Tkinter module.

  Import tKinter as tk

- Create the GUI application main window.

  root = tk.Tk()

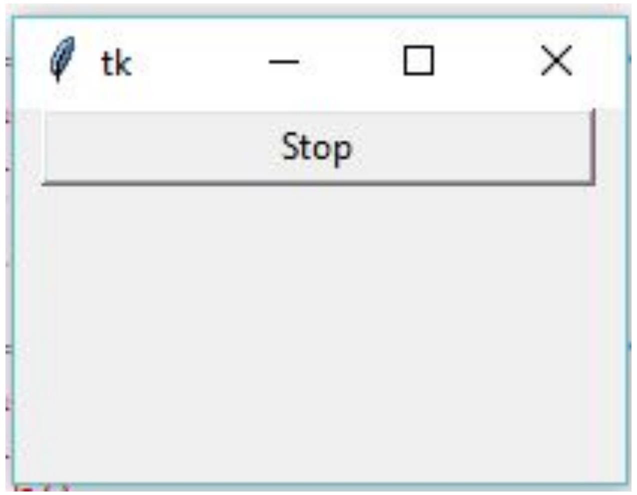
- Add one or more of the above-mentioned widgets to the GUI application.

  button = tk.Button(root, text='Stop', width=25, command=root.destroy)

  button.pack()

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

  root.mainloop()

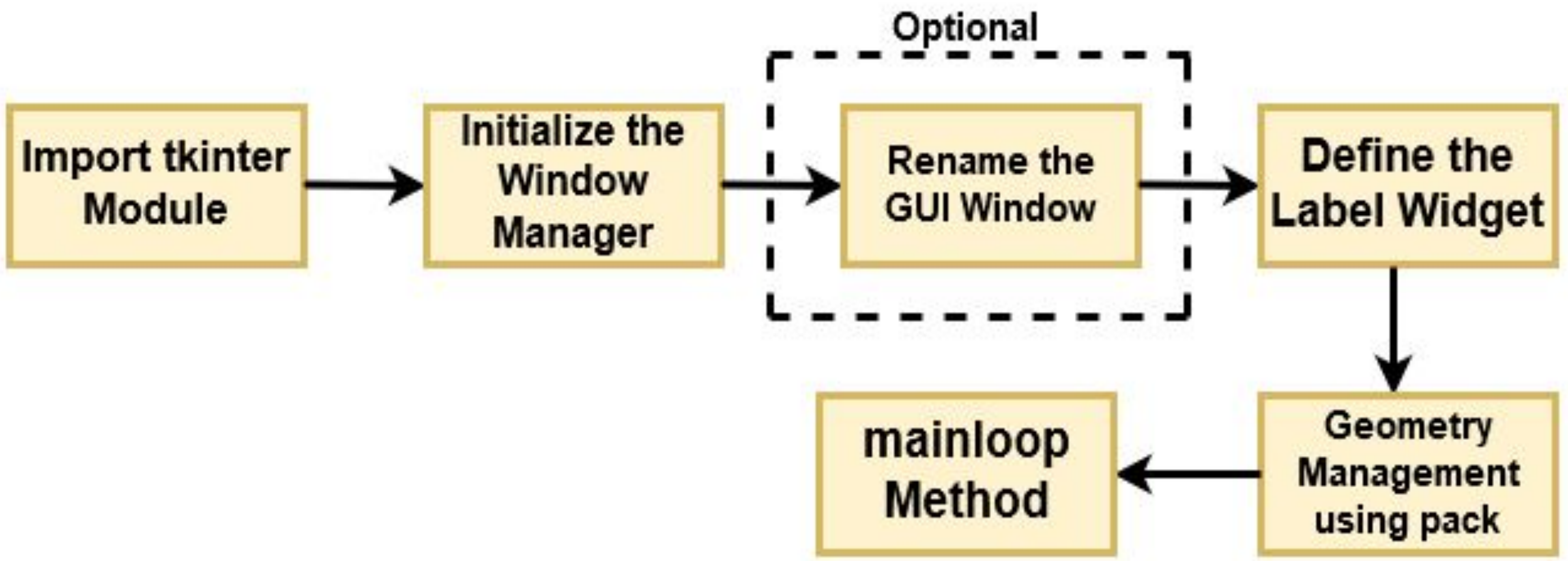


Tkinter widgets

- Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

Widget	**Description**
Label	Used to contain text or images
Button	Similar to a Label but provides additional functionality for mouse overs, presses, and releases as well as keyboard activity/events
Canvas	Provides ability to draw shapes (lines, ovals, polygons, rectangles); can contain images or bitmaps
Radiobutton	Set of buttons of which only one can be "pressed" (similar to HTML radio input)
Checkbutton	Set of boxes of which any number can be "checked" (similar to HTML checkbox input)
Entry	Single-line text field with which to collect keyboard input (similar to HTML text input)
Frame	Pure container for other widgets
Listbox	Presents user list of choices to pick from
Menu	Actual list of choices "hanging" from a Menubutton that the user can choose from
Menubutton	Provides infrastructure to contain menus (pulldown, cascading, etc.)
Message	Similar to a Label, but displays multi-line text
Scale	Linear "slider" widget providing an exact value at current setting; with defined starting and ending values
Text	Multi-line text field with which to collect (or display) text from user (similar to HTML TextArea)
Scrollbar	Provides scrolling functionality to supporting widgets, i.e., Text, Canvas, Listbox, and Entry
Toplevel	Similar to a Frame, but provides a separate window container

Operation Using Tkinter Widget



Geometry Managers

- The pack() Method – This geometry manager organizes widgets in blocks before placing them in the parent widget.

  widget.pack(pack_options)

  options

  - expand – When set to true, widget expands to fill any space not otherwise used in widget's parent.
  - 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).
  - side – Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.
- The grid() Method – This geometry manager organizes widgets in a table-like structure in the parent widget.

  widget.grid(grid_options)

  options –

  - Column/row – The column or row to put widget in; default 0 (leftmost column).
  - Columnspan, rowsapn– How many columns or rows to widgetoccupies; default 1.
  - ipadx, ipady – How many pixels to pad widget, horizontally and vertically, inside widget's borders.
  - padx, pady – How many pixels to pad widget, horizontally and vertically, outside v's borders.

Geometry Managers

- The place() Method − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

  widget.place(place_options)

  options −

  - anchor − The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)
  - bordermode − INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.
  - height, width − Height and width in pixels.
  - relheight, relwidth − Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
  - relx, rely − Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
  - x, y − Horizontal and vertical offset in pixels.

Common Widget Properties

Common attributes such as sizes, colors and fonts are specified.

- Dimensions
- Colors
- Fonts
- Anchors
- Relief styles
- Bitmaps
- Cursors

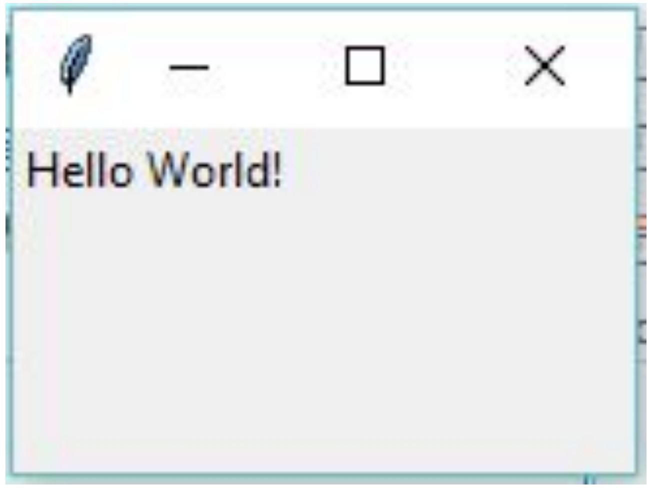
Label Widgets

- A label is a widget that displays text or images, typically that the user will just view but not otherwise interact with. Labels are used for such things as identifying controls or other parts of the user interface, providing textual feedback or results, etc.
- Syntax

  tk.Label(parent,text="message")

Example:

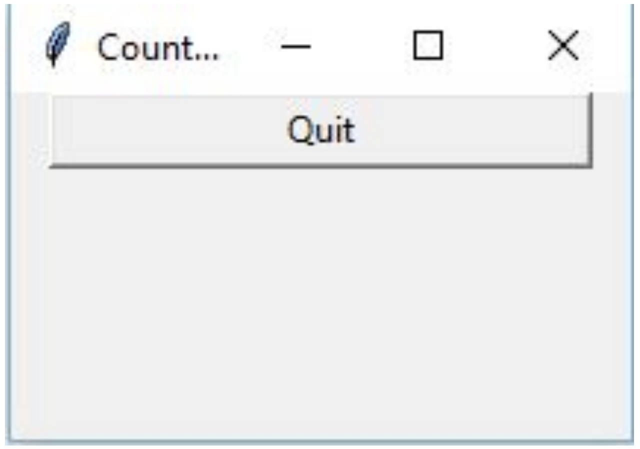
```
import tkinter as tk
root = tk.Tk()
label = tk.Label(root, text='Hello World!')
label.grid()
root.mainloop()
```



Hello World!

Button Widgets

```
import tkinter as tk

r = tk.Tk()

r.title('Counting Seconds')

button = tk.Button(r, text='Stop', width=25, command=r.destroy)

button.pack()

r.mainloop()
```



Button Widgets

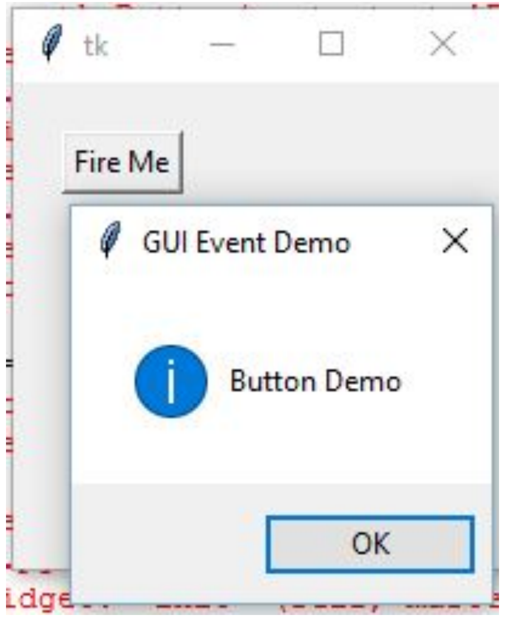
- A button, unlike a frame or label, is very much designed for the user to interact with, and in particular, press to perform some action. Like labels, they can display text or images, but also have a whole range of new options used to control their behavior.

Syntax

button = ttk.Button(parent, text=‘ClickMe', command=submitForm)

Example:

```
import tkinter as tk
from tkinter import messagebox
def hello():
  msg = messagebox.showinfo( "GUI Event Demo","Button Demo")
root = tk.Tk()
root.geometry("200x200")
b = tk.Button(root, text='Fire Me',command=hello)
b.place(x=50,y=50)
root.mainloop()
```



Button Demo

Button Widgets

- Button:To add a button in your application, this widget is used.

Syntax :

```
w=Button(master, text=“caption” option=value)
```

- master is the parameter used to represent the parent window.
- activebackground: to set the background color when button is under the cursor.
- activeforeground: to set the foreground color when button is under the cursor.
- bg: to set he normal background color.
- command: to call a function.
- font: to set the font on the button label.
- image: to set the image on the button.
- width: to set the width of the button.
- height: to set the height of the button.

Entry Widgets

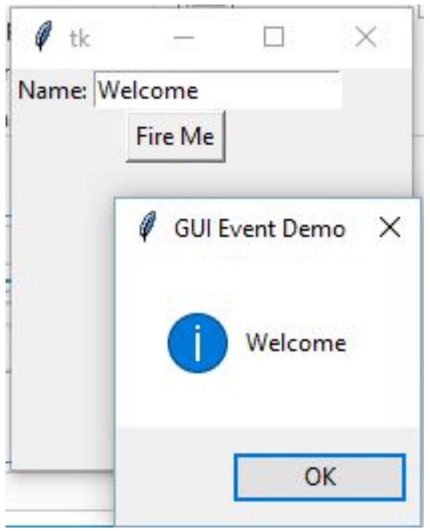
- An entry presents the user with a single line text field that they can use to type in a string value. These can be just about anything: their name, a city, a password, social security number, and so on.

Syntax

name = ttk.Entry(parent, textvariable=username)

Example:

```
def hello():
   msg = messagebox.showinfo( "GUI Event Demo",t.get())
root = tk.Tk()
root.geometry("200x200")
l1=tk.Label(root,text="Name:")
l1.grid(row=0)
t=tk.Entry(root)
t.grid(row=0,column=1)
b = tk.Button(root, text='Fire Me',command=hello)
b.grid(row=1,columnspan=2);
root.mainloop()
```



Canvas

- The Canvas is a rectangular area intended for drawing pictures or other complex layouts. You can place graphics, text, widgets or frames on a Canvas.
- It is used to draw pictures and other complex layout like graphics, text and widgets.

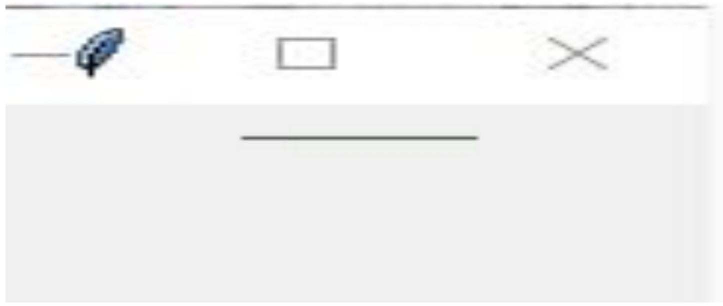
Syntax:

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

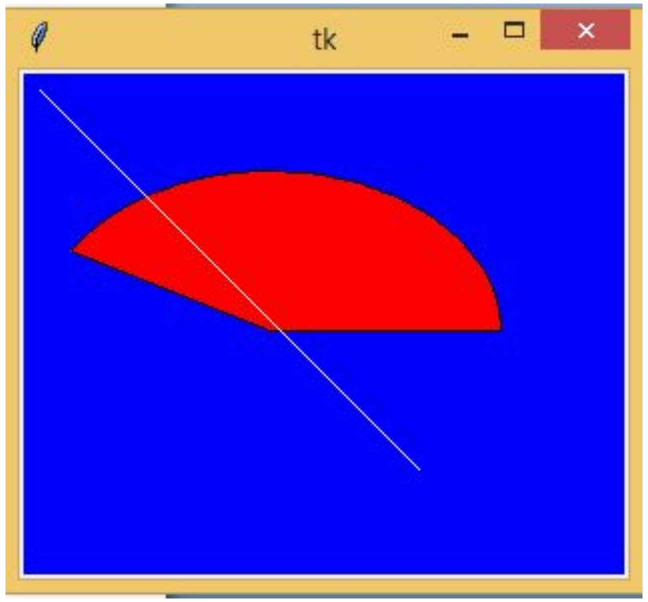
- master is the parameter used to represent the parent window.
- bd: to set the border width in pixels.
- bg: to set the normal background color.
- cursor: to set the cursor used in the canvas.
- highlightcolor: to set the color shown in the focus highlight.
- width: to set the width of the widget.
- height: to set the height of the widget.

Canvas

```
from tkinter import *
master = Tk()
w = Canvas(master, width=40, height=60)
w.pack()
canvas_height=20
canvas_width=200
y = int(canvas_height / 2)
w.create_line(0, y, canvas_width, y )
mainloop()
```



```
from tkinter import *
from tkinter import messagebox
top = Tk()
C = Canvas(top, bg = "blue", height = 250, width = 300)
coord = 10, 50, 240, 210
arc = C.create_arc(coord, start = 0, extent = 150, fill = "red")
line = C.create_line(10,10,200,200,fill = 'white')
C.pack()
top.mainloop()
```



Checkbutton

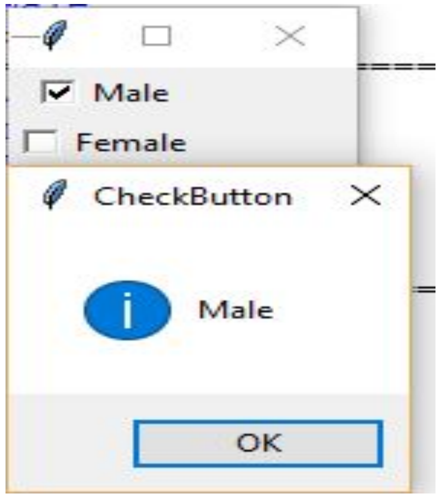
- A checkbutton is like a regular button, except that not only can the user press it, which will invoke a command callback, but it also holds a binary value of some kind (i.e. a toggle). Checkbuttons are used all the time when a user is asked to choose between, e.g. two different values for an option.

Syntax

w = CheckButton(master, option=value)

Example:

```
from tkinter import *
root= Tk()
root.title('Checkbutton Demo')
v1=IntVar()
v2=IntVar()
cb1=Checkbutton(root,text='Male', variable=v1,onvalue=1, offvalue=0, command=test)
cb1.grid(row=0)
cb2=Checkbutton(root,text='Female', variable=v2,onvalue=1, offvalue=0, command=test)
cb2.grid(row=1)
root.mainloop()
```



```
def test():
    if(v1.get()==1 ):
        v2.set(0)
        print("Male")
    if(v2.get()==1):
        v1.set(0)
        print("Female")
```

radiobutton

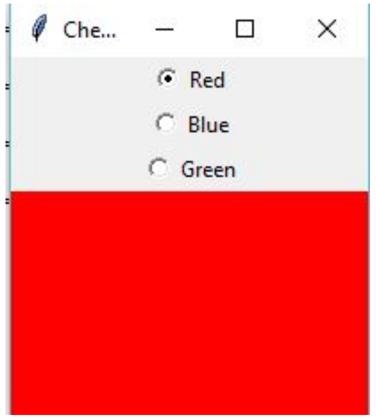
- A radiobutton lets you choose between one of a number of mutually exclusive choices; unlike a checkbutton, it is not limited to just two choices. Radiobuttons are always used together in a set and are a good option when the number of choices is fairly small
- **Syntax**

  w = CheckButton(master, option=value)

Example:

```
root= Tk()
root.geometry("200x200")
radio=IntVar()
rb1=Radiobutton(root,text='Red', variable=radio,width=25,value=1, command=choice)
rb1.grid(row=0)
rb2=Radiobutton(root,text='Blue', variable=radio,width=25,value=2, command=choice)
rb2.grid(row=1)
rb3=Radiobutton(root,text='Green', variable=radio,width=25,value=3, command=choice)
rb3.grid(row=3)
root.mainloop()
```

```
def choice():
  if(radio.get()==1):
    root.configure(background='red')
  elif(radio.get()==2):
    root.configure(background='blue')
  elif(radio.get()==3):
    root.configure(background='green')
```



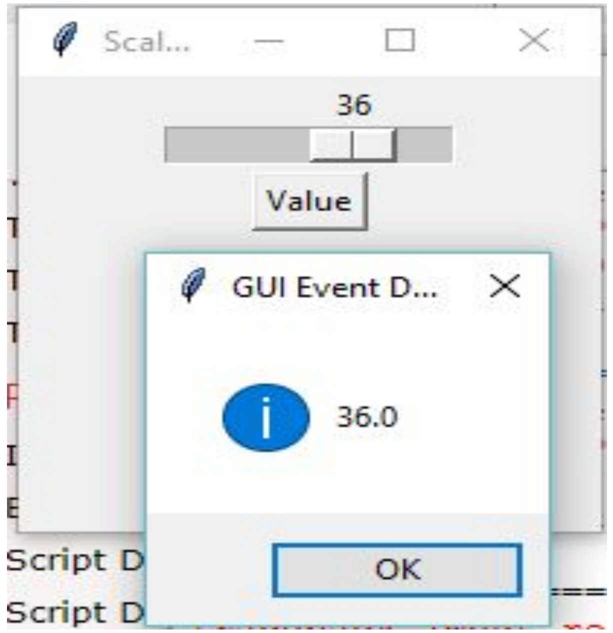
Scale

- Scale widget is used to implement the graphical slider to the python application so that the user can slide through the range of values shown on the slider and select the one among them. We can control the minimum and maximum values along with the resolution of the scale. It provides an alternative to the Entry widget when the user is forced to select only one value from the given range of values.
- **Syntax**

  w = Scale(top, options)

Example:

```
from tkinter import messagebox
root= Tk()
root.title('Scale Demo')
root.geometry("200x200")
def slide():
   msg = messagebox.showinfo( "GUI Event Demo",v.get())
v = DoubleVar()
scale = Scale( root, variable = v, from_ = 1, to = 50, orient = HORIZONTAL)
scale.pack(anchor=CENTER)
btn = Button(root, text="Value", command=slide)
btn.pack(anchor=CENTER)
root.mainloop()
```



```
36.0
```

Spinbox

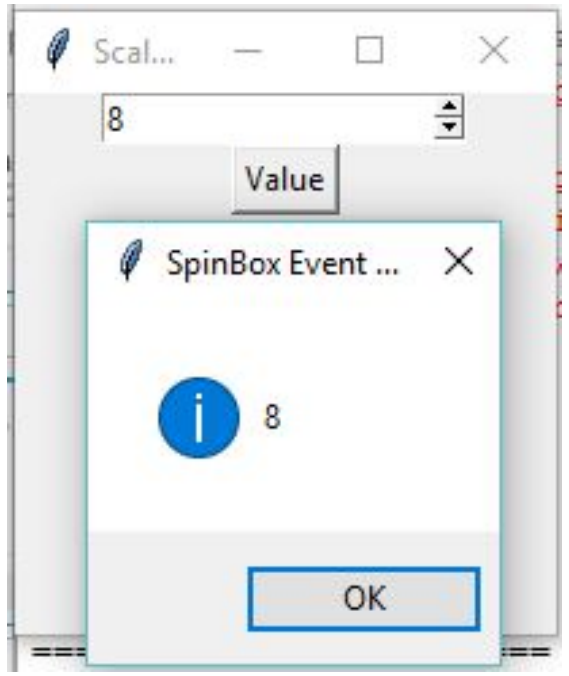
- The Spinbox widget is an alternative to the Entry widget. It provides the range of values to the user, out of which, the user can select the one.

Syntax

w = Spinbox(top, options)

Example:

```
from tkinter import *
from tkinter import messagebox

root= Tk()
root.title('Scale Demo')
root.geometry("200x200")

def slide():
   msg = messagebox.showinfo( "SpinBox Event Demo",spin.get())

spin = Spinbox(root, from_= 0, to = 25)
spin.pack(anchor=CENTER)
btn = Button(root, text="Value", command=slide)
btn.pack(anchor=CENTER)
root.mainloop()
```



Menubutton

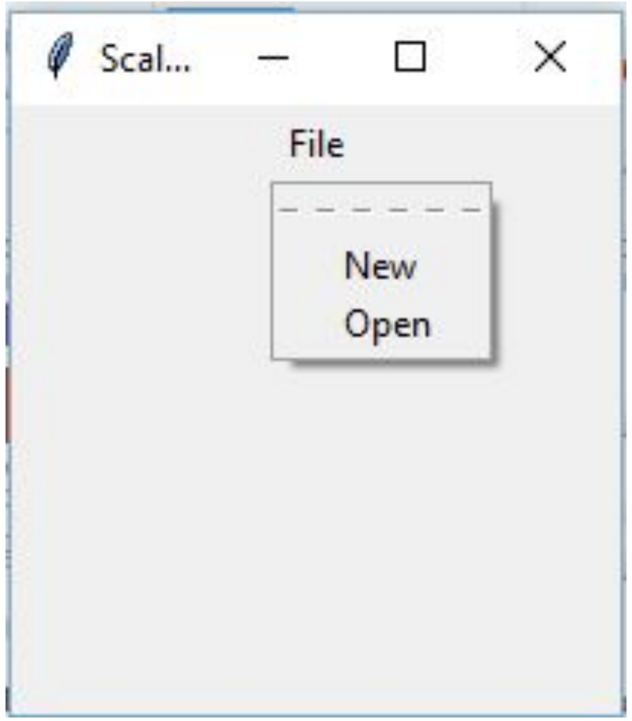
- Menubutton widget can be defined as the drop-down menu that is shown to the user all the time. It is used to provide the user a option to select the appropriate choice exist within the application.

Syntax

w = Menubutton(Top, options)

Example:

```
from tkinter import *
from tkinter import messagebox
root= Tk()
root.title('Scale Demo')
root.geometry("200x200")
menubutton = Menubutton(root, text = "File", relief = FLAT)
menubutton.grid()
menubutton.menu = Menu(menubutton)
menubutton["menu"]=menubutton.menu
menubutton.menu.add_checkbutton(label = "New", variable=IntVar(),command=)
menubutton.menu.add_checkbutton(label = "Open", variable = IntVar())
menubutton.pack()
root.mainloop()
```



Menubutton

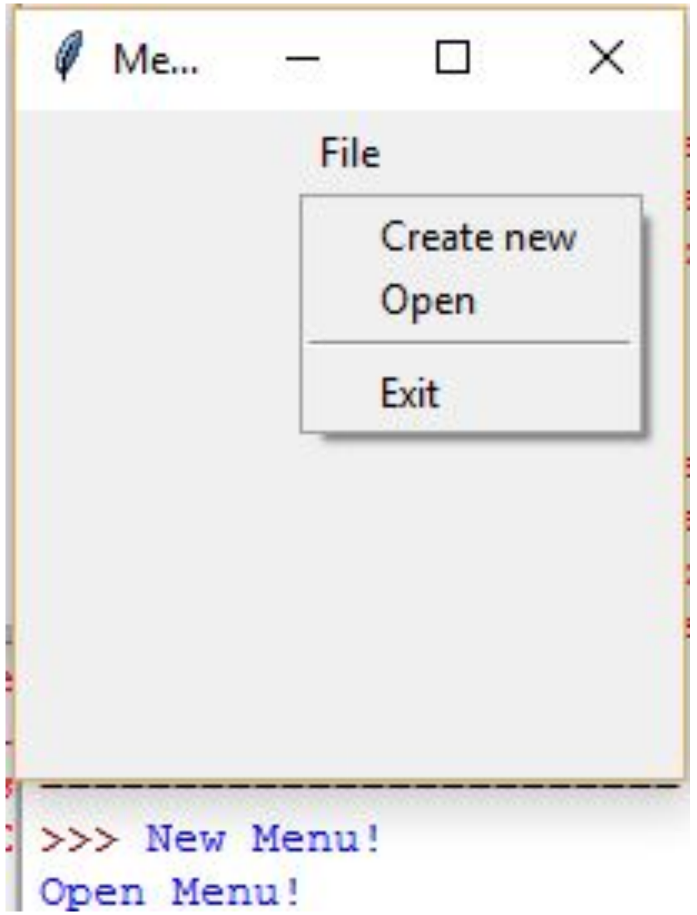
- Menubutton widget can be defined as the drop-down menu that is shown to the user all the time. It is used to provide the user a option to select the appropriate choice exist within the application.

Syntax

```
w = Menubutton(Top, options)
```

Example:

```
from tkinter import *
from tkinter import messagebox
root= Tk()
root.title('Menu Demo')
root.geometry("200x200")
def new():
   print("New Menu!")
def disp():
   print("Open Menu!")
menubutton = Menubutton(root, text="File")
menubutton.grid()
menubutton.menu = Menu(menubutton, tearoff = 0)
menubutton["menu"] = menubutton.menu
menubutton.menu.add_command(label="Create new",command=new)
menubutton.menu.add_command(label="Open",command=disp)
menubutton.menu.add_separator()
menubutton.menu.add_command(label="Exit",command=root.quit)
menubutton.pack()
```



```
>>> New Menu!
Open Menu!
```