

# GUI PROGRAMMING

- Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.
- **Tkinter** is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. **Tkinter** is included with the standard Microsoft Windows and Mac OS X install of Python. The name **Tkinter** comes from Tk interface.
- **wxPython**: Open source python interface
- **Jpython**: It is a python port for java which gives Python scripts seamless access to Java class libraries



# ADVANTAGES

- python and tkinter are excellent for developing GUIs. While this is true of most scripting languages, I think these two make a particularly good combination.
- For most GUIs, functionality is more important than form (ie: eye candy is not top priority). Even then, it's possible to make very nice looking GUIs with Tkinter. Tk has been my go-to toolkit for close to twenty years. It is most easily used with Tcl since that is its native language, but there's no denying the advantage python has over Tcl as far as built-in functionality goes.
- That being said, all of the toolkits available to a python programmer are good. Tkinter's main advantages are that it is extremely easy to use and you probably already have it installed, so the barrier for entry is very low.



## DISADVANTAGES:

- Difficult to develop and high cost:-
- Slower than command line tools:-
- Extra attention required:-
- Memory resources:-



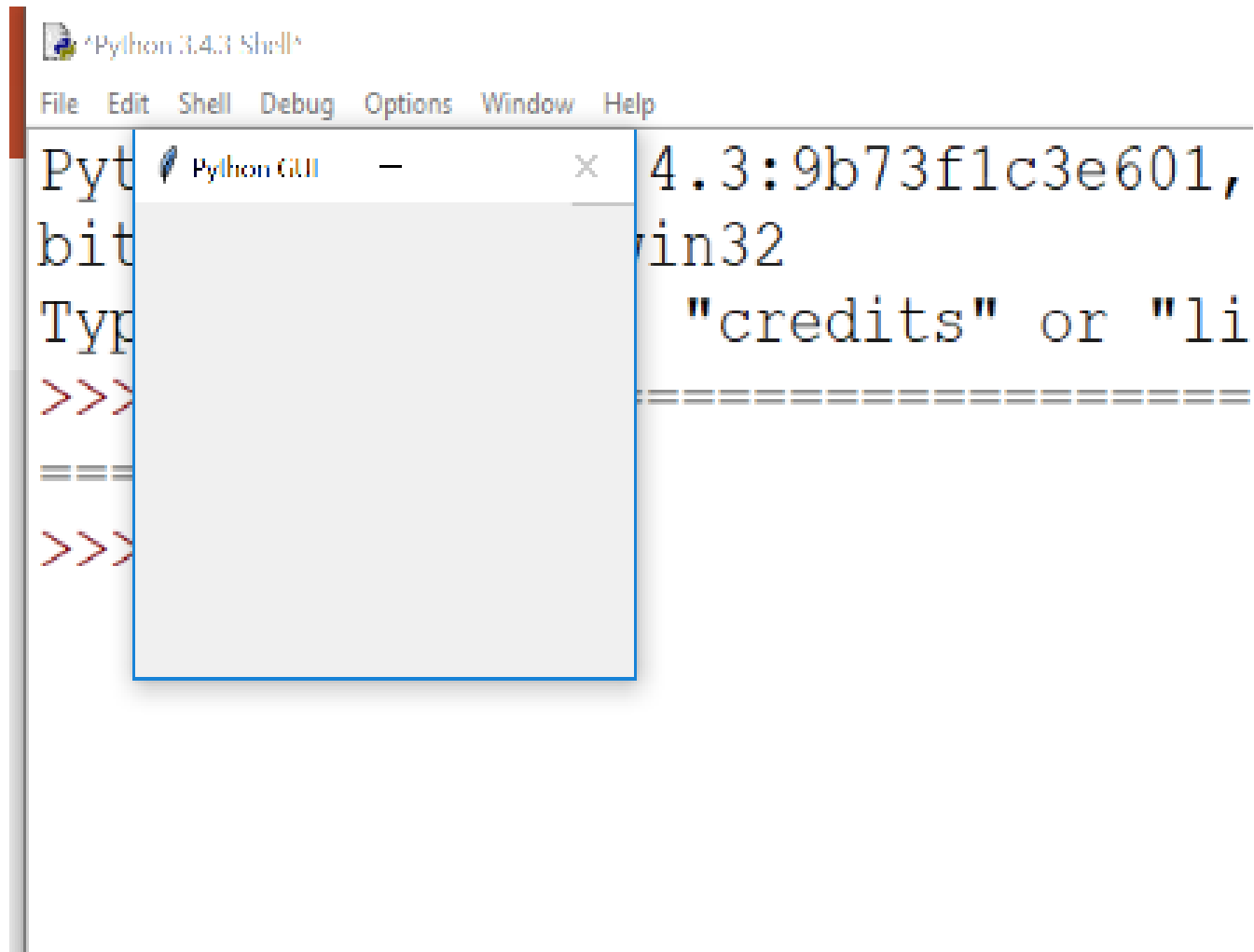
# TKINTER PROGRAMMING (LIBRARY)

- 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 the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.



```
import tkinter as tk # 1  
win = tk.Tk() # 2  
win.title("Python GUI") # 3  
win.mainloop() # 4
```





A screenshot of a Python 3.4.3 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following text: 'Pyt', 'bit', 'Typ', '4.3:9b73f1c3e601,', 'win32', '"credits" or "li', '=====', and '>>>'. A modal dialog box titled 'Python GUI' is open in the foreground, partially obscuring the text. The dialog box has a title bar with a feather icon, the text 'Python GUI', and a close button. The dialog box is empty.

```
Pyt Python GUI 4.3:9b73f1c3e601,  
bit win32  
Typ "credits" or "li  
>>>=====  
=====  
>>>
```



- We are preventing the GUI from being resized.

```
import tkinter as tk # 1 imports  
win = tk.Tk() # 2 Create instance  
win.title("Python GUI") # 3 Add a title  
win.resizable(0, 0) # 4 Disable resizing the GUI  
win.mainloop() # 5 Start GUI
```

Why is this important? Because, once we add widgets to our form, resizing can make our GUI look not as good as we want it to be. We



# GEOMETRY MANAGEMENT

- All widgets in the **tkinter** will have some geometry measurements. These measurements give you to organize the widgets and their parent frames, windows, etc.,
- **Tkinter** has the following three Geometry Manager classes.
- **pack()**:- It organizes the widgets in the block, which mean it occupies the entire available width. It's a standard method to show the widgets in the window
- **grid()**:- It organizes the widgets in table-like structure. You will see details about **grid** later in this tutorial.
- **place()**:- It's used to place the widgets at a specific position you want.





# BUTTON

- The 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:** list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.



Option	Description
activebackground	Background color when the button is under the cursor.
activeforeground	Foreground color when the button is under the cursor.
bd	Border width in pixels. Default is 2.
bg	Normal background color.
command	Function or method to be called when the button is clicked.
fg	Normal foreground (text) color.
font	Text font to be used for the button's label.

## Methods

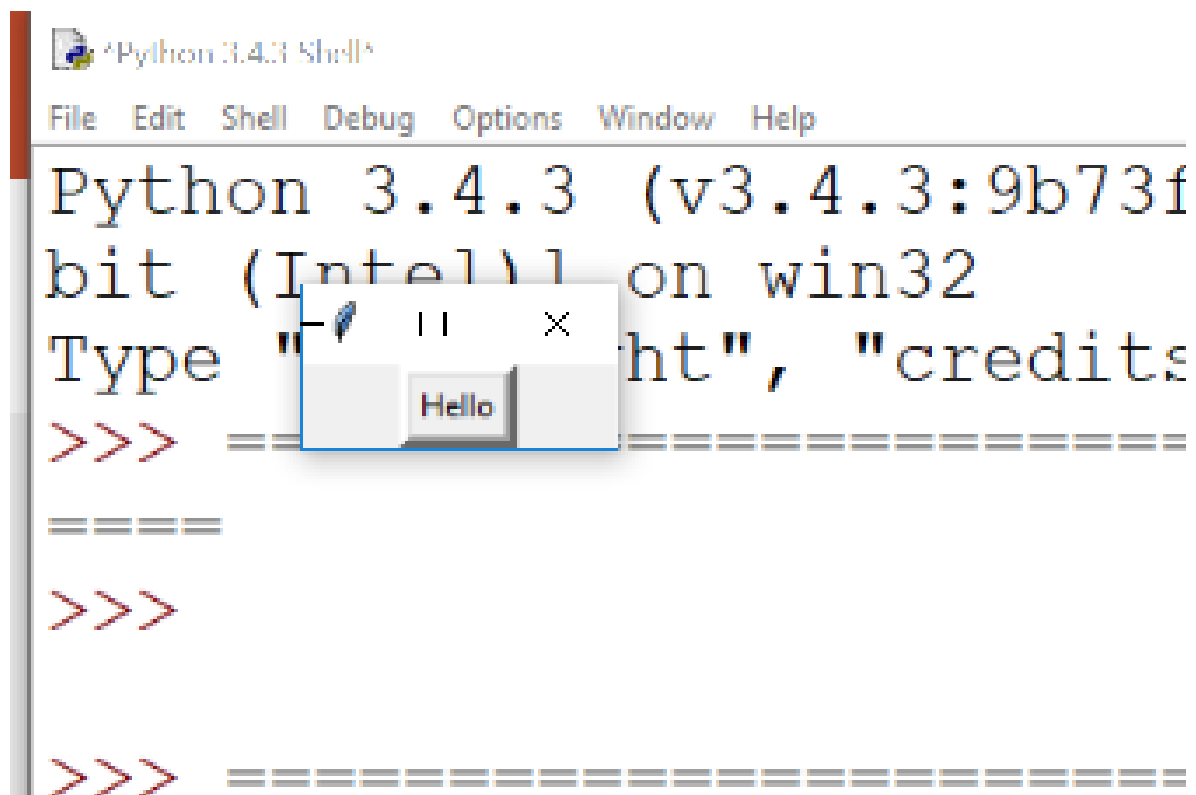
Following are commonly used methods for this widget –

Method	Description
flash()	Causes the button to flash several times between active and normal colors. Leaves the button in the state it was in originally. Ignored if the button is disabled.



```
import tkinter as tk  
top = tk.Tk()  
B = tk.Button(top, text = "Hello", activebackground = "  
red", activeforeground = "blue", bd = 5)  
B.pack()  
top.mainloop()
```





A screenshot of a Python 3.4.3 Shell window. The window title is "Python 3.4.3 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The text in the shell reads: "Python 3.4.3 (v3.4.3:9b73f", "bit (Intel) on win32", "Type \"ht\", \"credits\"", and three red prompt characters ">>>". A small dialog box with a blue border and a white background is overlaid on the shell. It has a title bar with a blue icon, a close button (X), and a maximize button (two vertical bars). The dialog box contains the text "Hello".

```
Python 3.4.3 (v3.4.3:9b73f
bit (Intel) on win32
Type "ht", "credits"
>>>
>>>
>>>
```



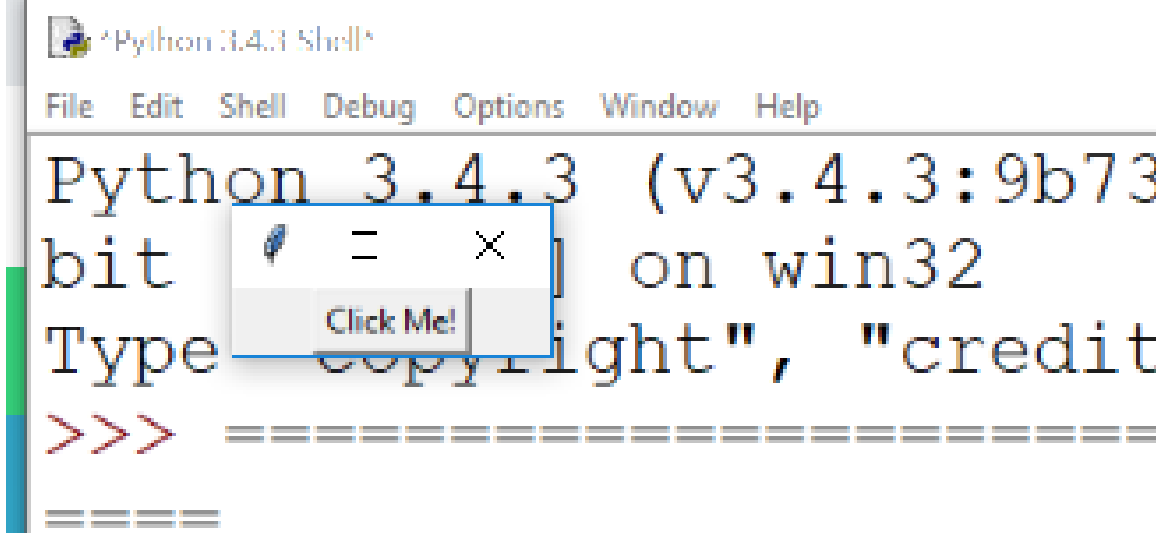
- Calling functions whenever an event occurs refers to a **binding function**.
- In the below example, when you click the button, it calls a function called **say\_hi**.
- Function **say\_hi** creates a new label with the text **Hi**.

```
import tkinter
```

```
window = tkinter.Tk()  
window.title("GUI")
```

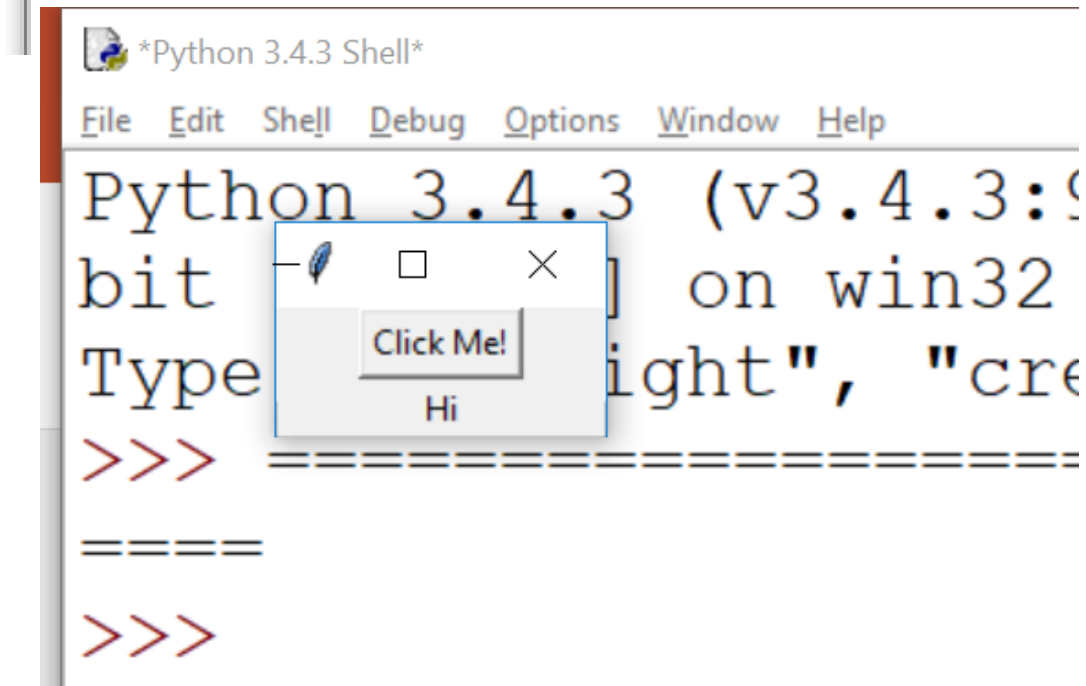
```
# creating a function called say_hi()  
def say_hi():  
    tkinter.Label(window, text = "Hi").pack()
```

```
tkinter.Button(window, text = "Click Me!", command = say_hi).pack()  
# 'command' is executed when you click the button  
# in this above case we're calling  
the function 'say_hi'.  
window.mainloop()
```



A screenshot of a Python 3.4.3 Shell window. The title bar reads '\*Python 3.4.3 Shell\*'. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main text area displays the following text:  
Python 3.4.3 (v3.4.3:9b73  
bit on win32  
Type "copyright", "credit  
>>> =====  
=====

A small dialog box is overlaid on the text. It has a feather icon, a minus sign, and a close button (X). The text 'Click Me!' is centered in the dialog box.



A screenshot of a Python 3.4.3 Shell window, similar to the one above. The title bar reads '\*Python 3.4.3 Shell\*'. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main text area displays the following text:  
Python 3.4.3 (v3.4.3:9  
bit on win32  
Type "copyright", "cre  
>>> =====  
=====

A small dialog box is overlaid on the text. It has a feather icon, a square icon, and a close button (X). The text 'Click Me!' is centered in the dialog box, and the text 'Hi' is displayed below it.



# EVENTS AND BINDINGS

- As was mentioned earlier, a Tkinter application spends most of its time inside an event loop (entered via the **mainloop** method). 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).
- Tkinter provides a powerful mechanism to let you deal with events yourself. For each widget, you can **bind** Python functions and methods to events.
- `widget.bind(event, handler)` If an event matching the *event* description occurs in the widget, the given *handler* is called with an object describing the event.



- `from tkinter import *`
- `root = Tk()`
- `def callback(event):`
  - `print ("clicked at", event.x, event.y)`
- `frame = Frame(root, width=100, height=100)`
- `frame.bind("<Button-1>", callback)`
- `frame.pack()`
- `root.mainloop()`





# CAPTURING CLICKS IN A WINDOW

```
from tkinter import *
```

```
root = Tk()
```

```
def callback(event):  
    print( "clicked at", event.x, event.y)
```

```
frame = Frame(root, width=100, height=100)  
frame.bind("<Button-1>", callback)  
frame.pack()
```

```
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.

## Syntax

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



The Canvas widget can support the following standard items:

**arc** . Creates an arc item, which can be a chord, a pieslice or a simple arc.

```
coord = 10, 50, 240, 210  
arc = canvas.create_arc(coord, start=0, extent=150, fill="blue")
```

**image** . Creates an image item, which can be an instance of either the BitmapImage or the PhotoImage classes.

```
filename = PhotoImage(file = "sunshine.gif")  
image = canvas.create_image(50, 50, anchor=NE, image=filename)
```

**line** . Creates a line item.

```
line = canvas.create_line(x0, y0, x1, y1, ..., xn, yn, options)
```

**oval** . 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)
```

**polygon** . Creates a polygon item that must have at least three vertices.

```
oval = canvas.create_polygon(x0, y0, x1, y1,...xn, yn, options)
```



Option	Description
bd	Border width in pixels. Default is 2.
bg	Normal background color.
confine	If true (the default), the canvas cannot be scrolled outside of the scrollregion.
cursor	Cursor used in the canvas like <i>arrow</i> , <i>circle</i> , <i>dot</i> etc.
height	Size of the canvas in the Y dimension.



```
import tkinter as tk  
top = tk.Tk()
```

```
C = tk.Canvas(top, bg="blue", height=250, width=300)
```

```
coord = 10, 50, 240, 210
```

```
arc = C.create_arc(coord, start=0, extent=150, fill="red")
```

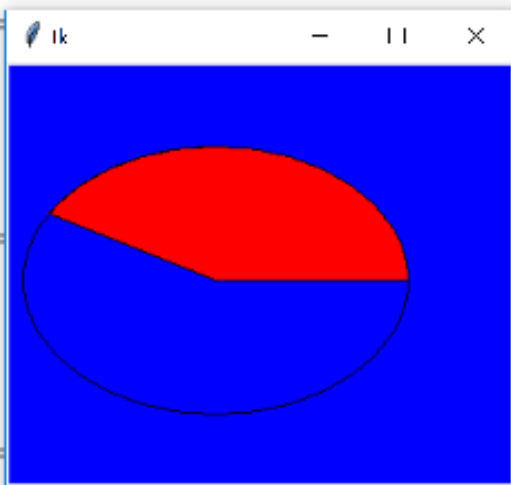
```
oval = C.create_oval(coord)
```

```
C.pack()
```

```
top.mainloop()
```



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:
bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
>>> ===== RESTART =====
>>>
>>> ===== RESTART =====
>>>
>>> ===== RESTART =====
>>>
```



# CHECKBUTTON

- 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.

## Syntax

- `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.



Option	Description
activebackground	Background color when the checkbutton is under the cursor.
activeforeground	Foreground color when the checkbutton is under the cursor.
bg	The normal background color displayed behind the label and indicator.
bitmap	To display a monochrome image on a button.
bd	The size of the border around the indicator. Default is 2 pixels.

## Methods

Following are commonly used methods for this widget –

Method	Description
deselect()	Clears (turns off) the checkbutton.
flash()	Flashes the checkbutton a few times between its active and normal colors, but leaves it the way it started.
invoke()	You can call this method to get the same actions that would occur if the user clicked on the checkbutton to change its state.
select()	Sets (turns on) the checkbutton.
toggle()	Clears the checkbutton if set, sets it if cleared.

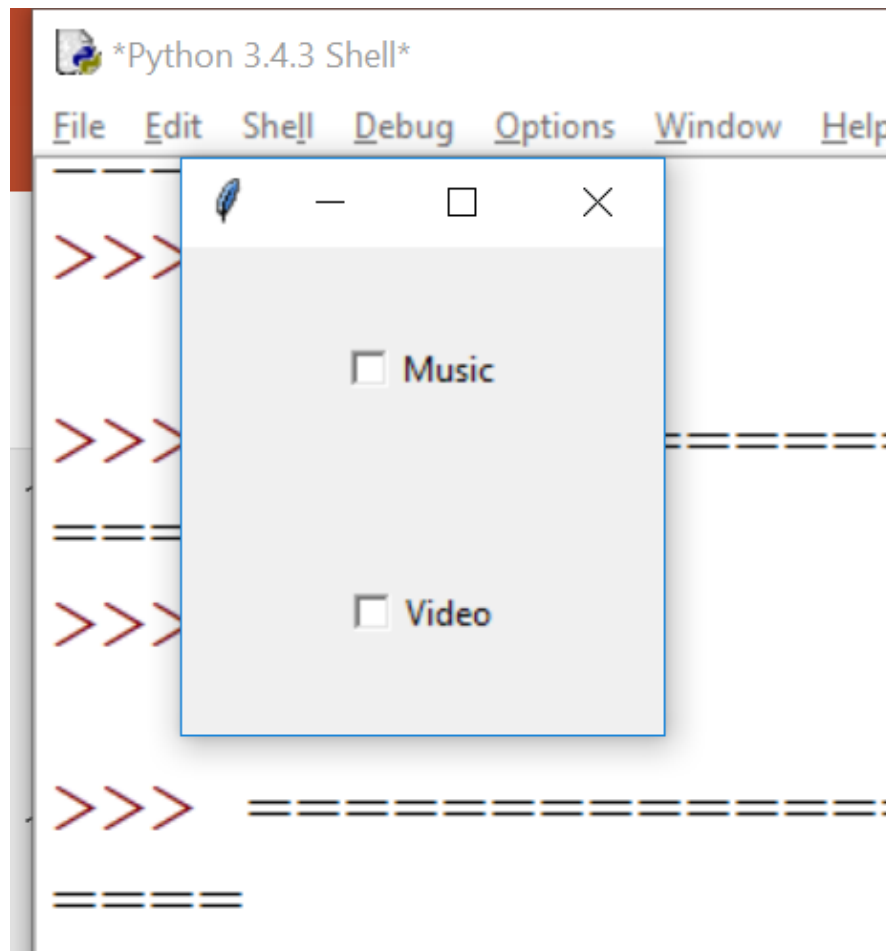




```
from tkinter import *  
import tkinter as tk
```

```
top = tk.Tk()  
CheckVar1 = IntVar()  
CheckVar2 = IntVar()  
C1 = Checkbutton(top, text = "Music", variable =  
CheckVar1, \  
                  onvalue = 1, offvalue = 0, height=5, \  
                  width = 20)  
C2 = Checkbutton(top, text = "Video", variable =  
CheckVar2, \  
              onvalue = 1, offvalue = 0, height=5, \  
              width = 20)  
C1.pack()  
C2.pack()  
top.mainloop()
```





# ENTRY

- 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

- `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.



Option	Description
bg	The normal background color displayed behind the label and indicator.
bd	The size of the border around the indicator. Default is 2 pixels.
command	A procedure to be called every time the user changes the state of this checkbutton.
cursor	If you set this option to a cursor name ( <i>arrow, dot etc.</i> ), the mouse cursor will change to that pattern when it is over the checkbutton.



```
from tkinter import *
```

```
top = Tk()
```

```
L1 = Label(top, text="User Name")
```

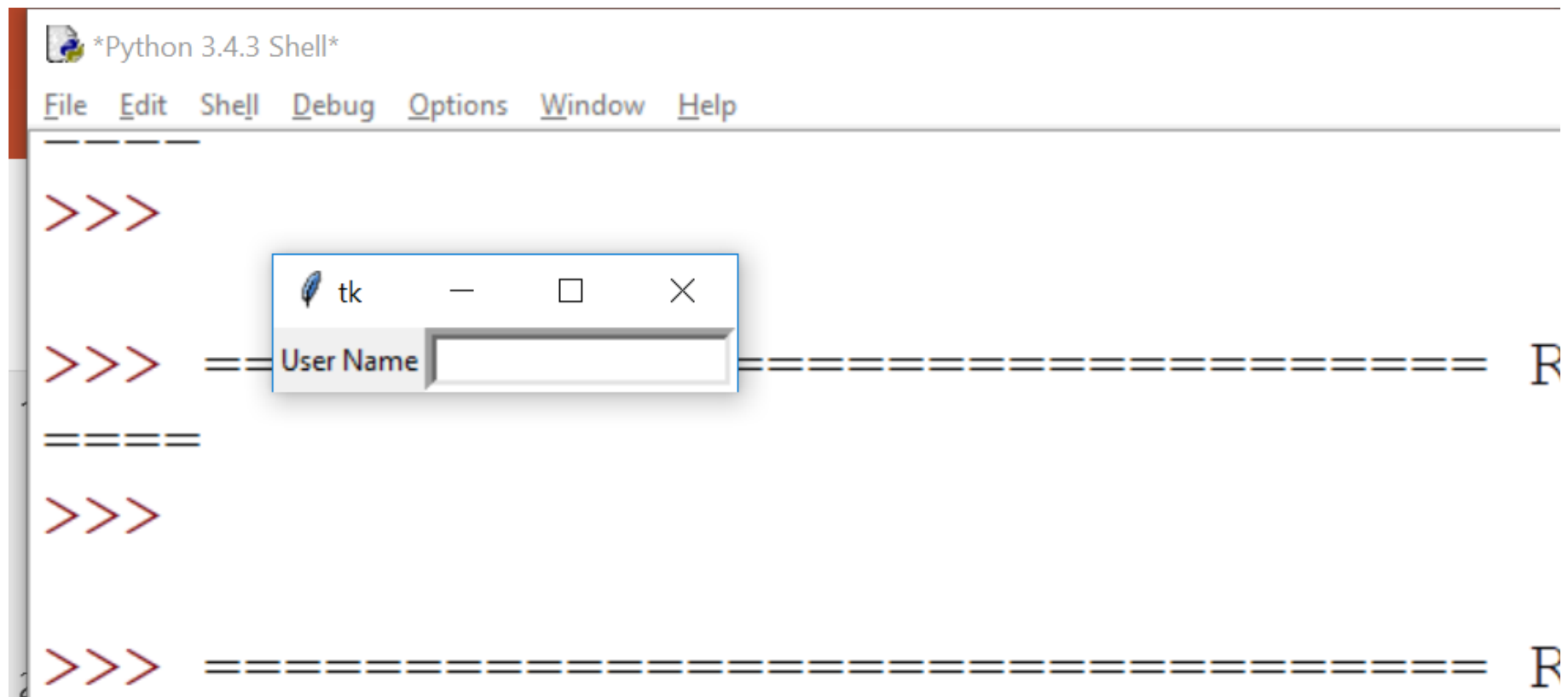
```
L1.pack( side = LEFT)
```

```
E1 = Entry(top, bd =5)
```

```
E1.pack(side = RIGHT)
```

```
top.mainloop()
```





# FRAME

- The Frame widget is very important for the process of grouping and organizing other widgets in a somehow friendly way.
- 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

```
w = Frame ( 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
bg	The normal background color displayed behind the label and indicator.
bd	The size of the border around the indicator. Default is 2 pixels.
command	A procedure to be called every time the user changes the state of this checkbutton.
cursor	If you set this option to a cursor name ( <i>arrow, dot etc.</i> ), the mouse cursor will change to that pattern when it is over the checkbutton.





```
from tkinter import *
```

```
root = Tk()  
frame = Frame(root,bd=10,bg="yellow")  
frame.pack()
```

```
bottomframe = Frame(root,bg="blue",bd=10)  
bottomframe.pack( side = BOTTOM )
```

```
redbutton = Button(frame, text="Red", fg="red")  
redbutton.pack( side = LEFT)
```

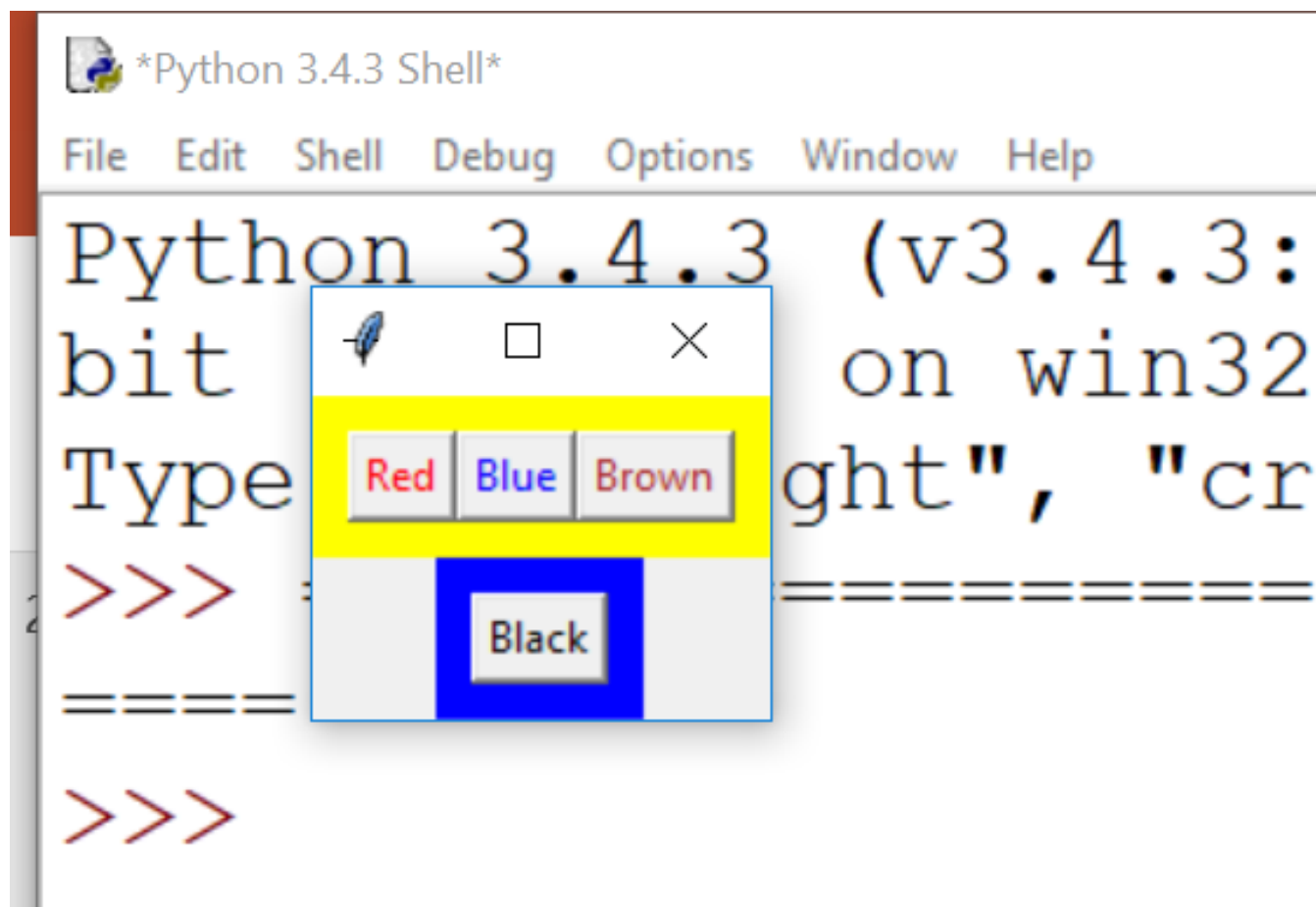
```
greenbutton = Button(frame, text="Green", fg="green")  
greenbutton.pack( side = RIGHT )
```

```
bluebutton = Button(frame, text="Blue", fg="blue")  
bluebutton.pack( side = LEFT )
```

```
blackbutton = Button(bottomframe, text="Black", fg="black")  
blackbutton.pack( side = BOTTOM)
```

```
root.mainloop()
```





# LABEL

- 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.

- Syntax

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
bg	The normal background color displayed behind the label and indicator.
bd	The size of the border around the indicator. Default is 2 pixels.
cursor	The cursor that appears when the mouse is over the listbox.
font	The font used for the text in the listbox.



## LISTBOX

- 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, ...
```
- **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()
```

```
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()  
x=Lb1.get(2)  
print(x)  
top.mainloop()
```





# MENU

- The goal of this widget is to allow us to create all kinds of menus that can be used by our applications. The core functionality provides ways to create three menu types: pop-up, toplevel and pull-down.
- 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.
- Syntax
- Here is the simple syntax to create this widget –
- `w = Menu ( 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
activebackground	The background color that will appear on a choice when it is under the mouse.
activeborderwidth	Specifies the width of a border drawn around a choice when it is under the mouse. Default is 1 pixel.
activeforeground	The foreground color that will appear on a choice when it is under the mouse.
bg	The background color for choices not under the mouse.
bd	The width of the border around all the choices. Default is 1.
cursor	The cursor that appears when the mouse is over the choices, but only when the menu has been torn off.



# METHODS OF MENU

Option	Description
<code>add_command (options)</code>	Adds a menu item to the menu.
<code>add_radiobutton( options )</code>	Creates a radio button menu item.
<code>add_checkbutton( options )</code>	Creates a check button menu item.
<code>add_cascade(options)</code>	Creates a new hierarchical menu by associating a given menu to a parent menu
<code>add_separator()</code>	Adds a separator line to the 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()
```

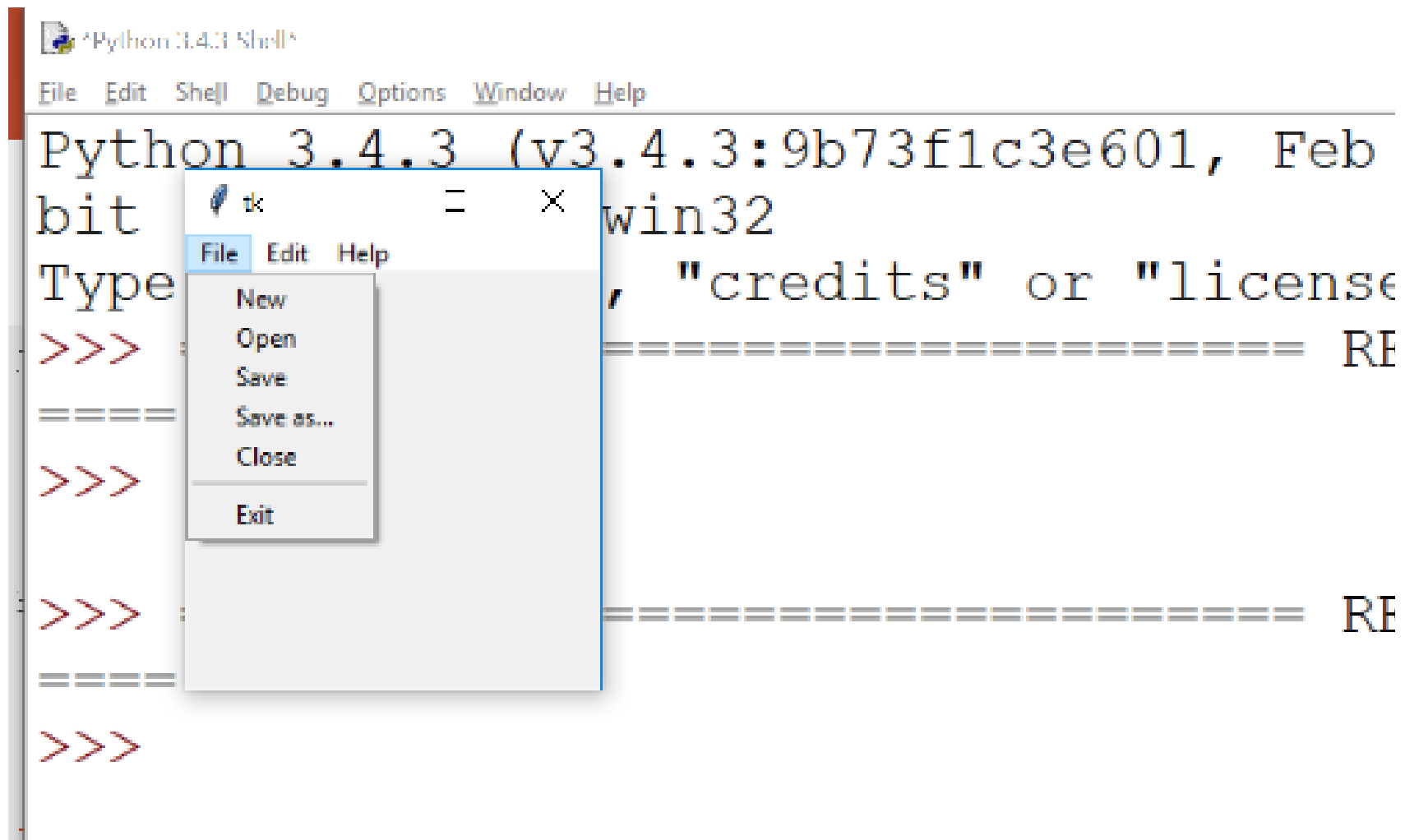
```
editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
editmenu.add_command(label="Paste", command=donothing)
editmenu.add_command(label="Delete", command=donothing)
editmenu.add_command(label="Select All", command=donothing)
```

```
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()
```



# YOU CAN WRITE FOR 1 MENU FILE IN CODE



# MESSAGE

- This widget provides a multiline and noneditable object that displays texts, automatically breaking lines and justifying their contents.
- Its functionality is very similar to the one provided by the Label widget, except that it can also automatically wrap the text, maintaining a given width or aspect ratio.
- Syntax
- Here is the simple syntax to create this widget –
- `w = Message ( 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 *
```

```
root = Tk()
```

```
var = StringVar()
```

```
label = Message( root, textvariable=var, relief=RAISED )
```

```
var.set("Hey!? How are you doing?")
```

```
label.pack()
```

```
root.mainloop()
```



\*Python 3.4.3 Shell\*

File Edit Shell Debug Options Window Help

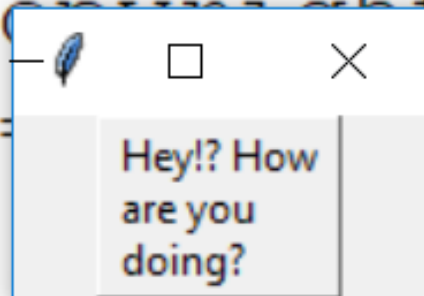
Python 3.4.3 (v3.4.3:9b73f1c3  
bit (Intel)] on win32

Type "copyright", "credits" o

>>> =====

=====

>>>





# RADIOBUTTON

- 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.



## OPTIONS

activebackground	The background color when the mouse is over the radiobutton.
activeforeground	The foreground color when the mouse is over the radiobutton.
anchor	If the widget inhabits a space larger than it needs, this option specifies where the radiobutton will sit in that space. The default is anchor=CENTER.
bg	The normal background color behind the indicator and label.
bitmap	To display a monochrome image on a radiobutton, set this option to a bitmap.
borderwidth	The size of the border around the indicator part itself. Default is 2 pixels.
command	A procedure to be called every time the user changes the state of this radiobutton.



# METHODS

Methods	Description
deselect()	Clears (turns off) the radiobutton.
flash()	Flashes the radiobutton a few times between its active and normal colors, but leaves it the way it started.
select()	Sets (turns on) the radiobutton.



```
from tkinter import *
```

```
def sel():
```

```
    selection = "You selected the option " + str(var.get())
```

```
    label.config(text = selection)
```

```
root = Tk()
```

```
var = IntVar()
```

```
R1 = Radiobutton(root, text="Option 1", variable=var, value=1,  
                  command=sel)
```

```
R1.pack( anchor = W )
```

```
R2 = Radiobutton(root, text="Option 2", variable=var, value=2,  
                  command=sel)
```

```
R2.pack( anchor = W )
```

```
R3 = Radiobutton(root, text="Option 3", variable=var, value=3,  
                  command=sel)
```

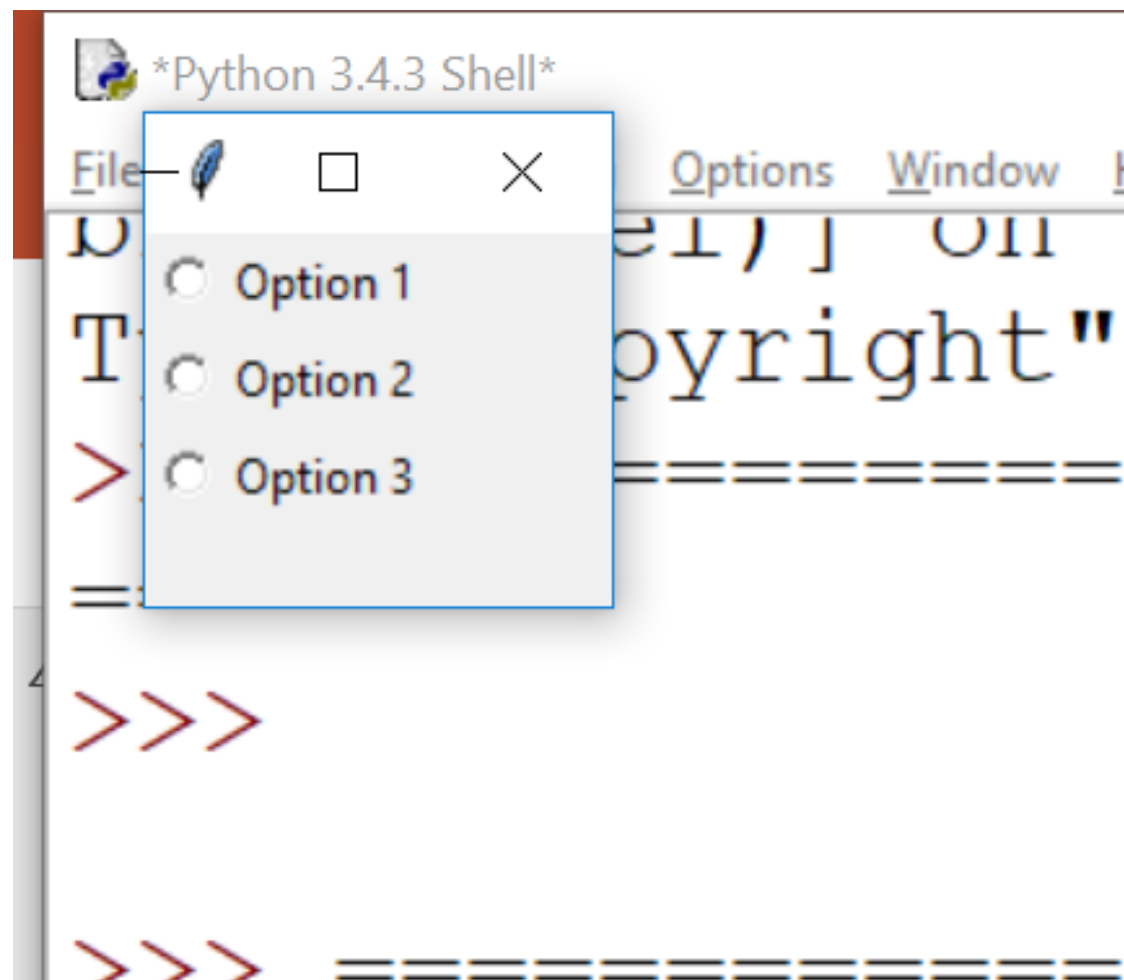
```
R3.pack( anchor = W)
```

```
label = Label(root)
```

```
label.pack()
```

```
root.mainloop()
```





# SCALE

- The Scale widget provides a graphical slider object that allows you to select values from a specific scale.
- Syntax
- Here is the simple syntax to create this widget –
- `w = Scale ( 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
activebackground	The background color when the mouse is over the scale.
bg	The background color of the parts of the widget that are outside the trough.
bd	Width of the 3-d border around the trough and slider. Default is 2 pixels.
command	A procedure to be called every time the slider is moved. This procedure will be passed one argument, the new scale value. If the slider is moved rapidly, you may not get a callback for every possible position, but you'll certainly get a callback when it settles.
cursor	If you set this option to a cursor name ( <i>arrow, dot etc.</i> ), the mouse cursor will change to that pattern when it is over the scale.

## Methods

Scale objects have these methods –

Methods	Description
get()	This method returns the current value of the scale.
set ( value )	Sets the scale's value.



```
from tkinter import *
```

```
def sel():
```

```
    selection = "Value = " + str(var.get())
```

```
    label.config(text = selection)
```

```
root = Tk()
```

```
var = DoubleVar()
```

```
scale = Scale( root, variable = var )
```

```
scale.pack(anchor=CENTER)
```

```
button = Button(root, text="Get Scale Value", command=sel)
```

```
button.pack(anchor=CENTER)
```

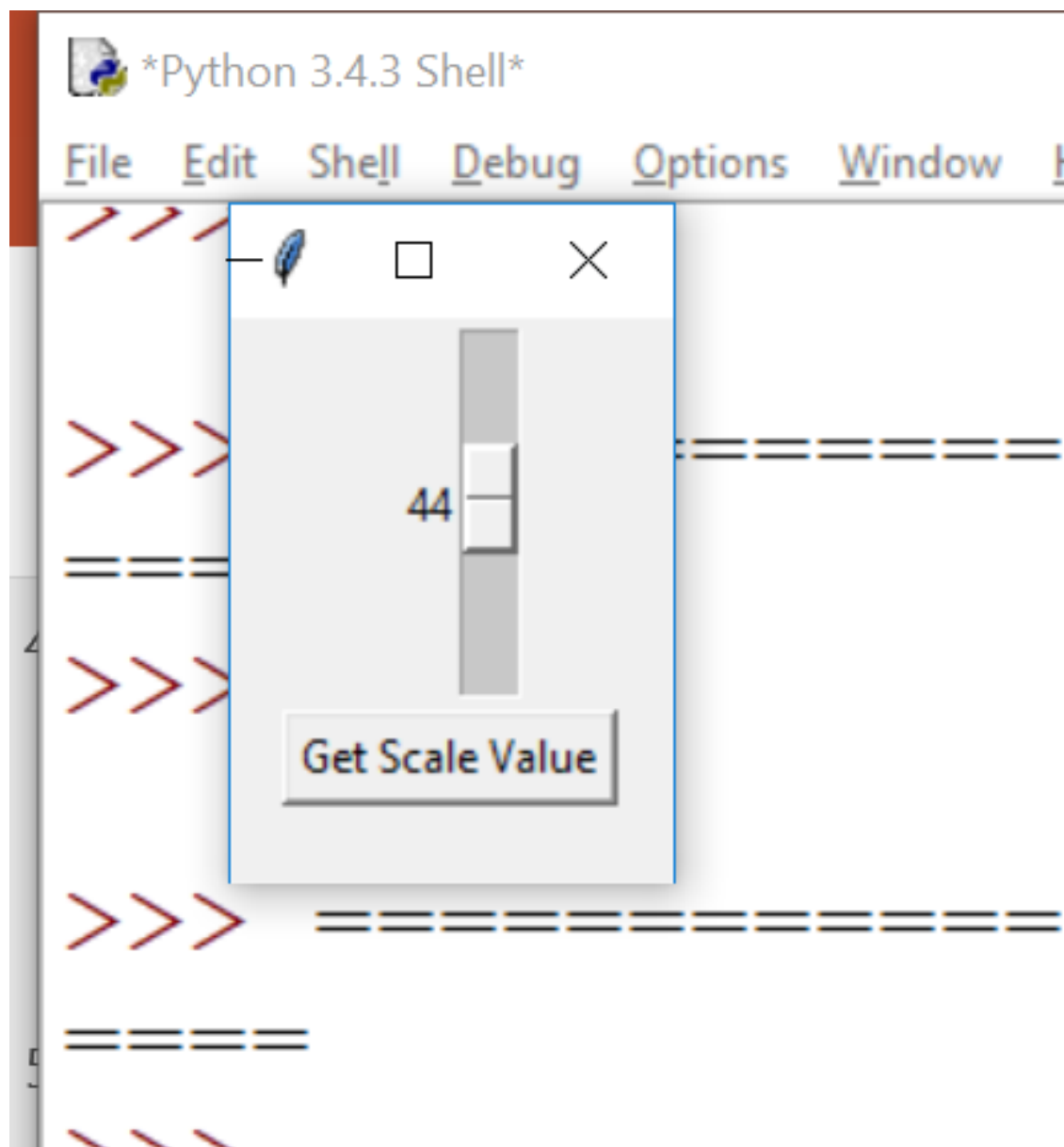
```
label = Label(root)
```

```
label.pack()
```

```
root.mainloop()
```







# SCROLLBAR

- This widget provides a slide controller that is used to implement vertical scrolled widgets, such as Listbox, Text and Canvas. Note that you can also create horizontal scrollbars on Entry widgets.
- Syntax
- Here is the simple syntax to create this widget –
- `w = Scrollbar ( 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
activebackground	The color of the slider and arrowheads when the mouse is over them.
bg	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.
command	A procedure to be called whenever the scrollbar is moved.
cursor	The cursor that appears when the mouse is over the scrollbar.



```
from tkinter import *
```

```
root = Tk()
```

```
scrollbar = Scrollbar(root)
```

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

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

```
for line in range(100):
```

```
    mylist.insert(END, "This is line number " + str(line))
```

```
mylist.pack( side = LEFT, fill = BOTH )
```

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

```
root.mainloop()
```



\*Python 3.4.3 Shell\*

File Edit Shell Debug Options Window Help

>>>

>>>

>>>

>>>

>>>

>>>

This is line number 0  
This is line number 1  
This is line number 2  
This is line number 3  
This is line number 4  
This is line number 5  
This is line number 6  
This is line number 7  
This is line number 8  
This is line number 9



## HORIZONTAL SCROLLBAR

```
from tkinter import *
```

```
root = Tk()
```

```
scrollbar = Scrollbar(root, orient = HORIZONTAL)
```

```
scrollbar.pack( side = BOTTOM, fill=X )
```

```
mylist = Listbox(root, xscrollcommand = scrollbar.set )
```

```
for line in range(100):
```

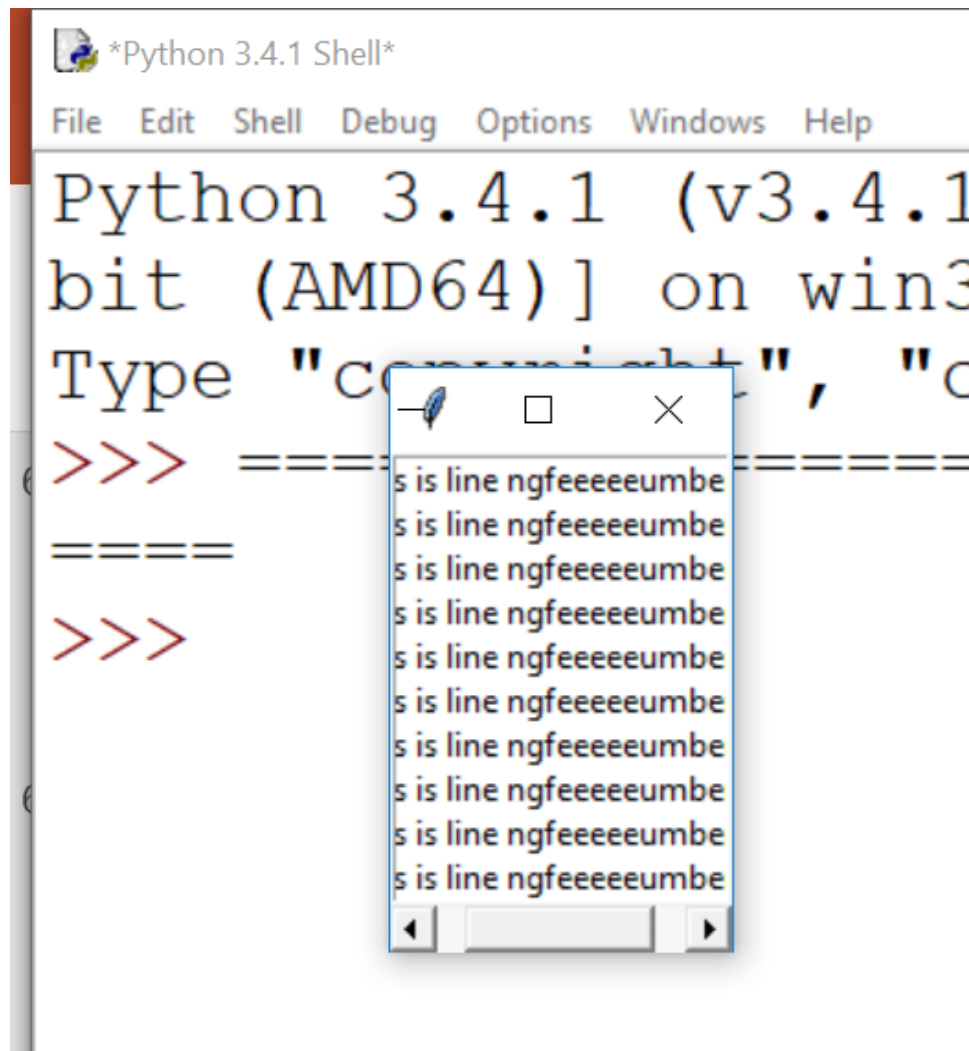
```
    mylist.insert(END, "This is line number " + str(line))
```

```
mylist.pack( side = LEFT, fill = BOTH )
```

```
scrollbar.config( command = mylist.xview )
```

```
root.mainloop()
```





The image shows a screenshot of a Python 3.4.1 Shell window. The window has a title bar that reads '\*Python 3.4.1 Shell\*'. Below the title bar is a menu bar with the following options: File, Edit, Shell, Debug, Options, Windows, and Help. The main text area of the shell displays the following text: 'Python 3.4.1 (v3.4.1', 'bit (AMD64)] on win3', and 'Type "copyright", "c'. Below this text, there are two lines of red text: '>>> =====' and '=====>>>'. A small, semi-transparent text box is overlaid on the shell window, containing the text 's is line ngfeeeeumbe' repeated ten times. The text box has a small icon of a feather in the top left corner and a close button (X) in the top right corner. The text box is positioned over the red text in the shell window.

```
*Python 3.4.1 Shell*
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1
bit (AMD64)] on win3
Type "copyright", "c
>>> =====
=====>>>
s is line ngfeeeeumbe
s is line ngfeeeeumbe
s is line ngfeeeeumbe
s is line ngfeeeeumbe
s is line ngfeeeeumbe
s is line ngfeeeeumbe
s is line ngfeeeeumbe
s is line ngfeeeeumbe
s is line ngfeeeeumbe
s is line ngfeeeeumbe
```



# MOUSE CLICKING EVENTS

- Clicking events are of 3 different types namely **leftClick**, **middleClick**, and **rightClick**.
- Now, you will learn how to call a particular function based on the event that occurs.
- Run the following program and click the **left**, **middle**, **right** buttons to calls a specific **function**.
- That **function** will create a new label with the mentioned text.





```
import tkinter
```

```
window = tkinter.Tk()  
window.title("GUI")
```

```
#creating 3 different functions for 3 events
```

```
def left_click(event):  
    tkinter.Label(window, text = "Left Click!").pack()
```

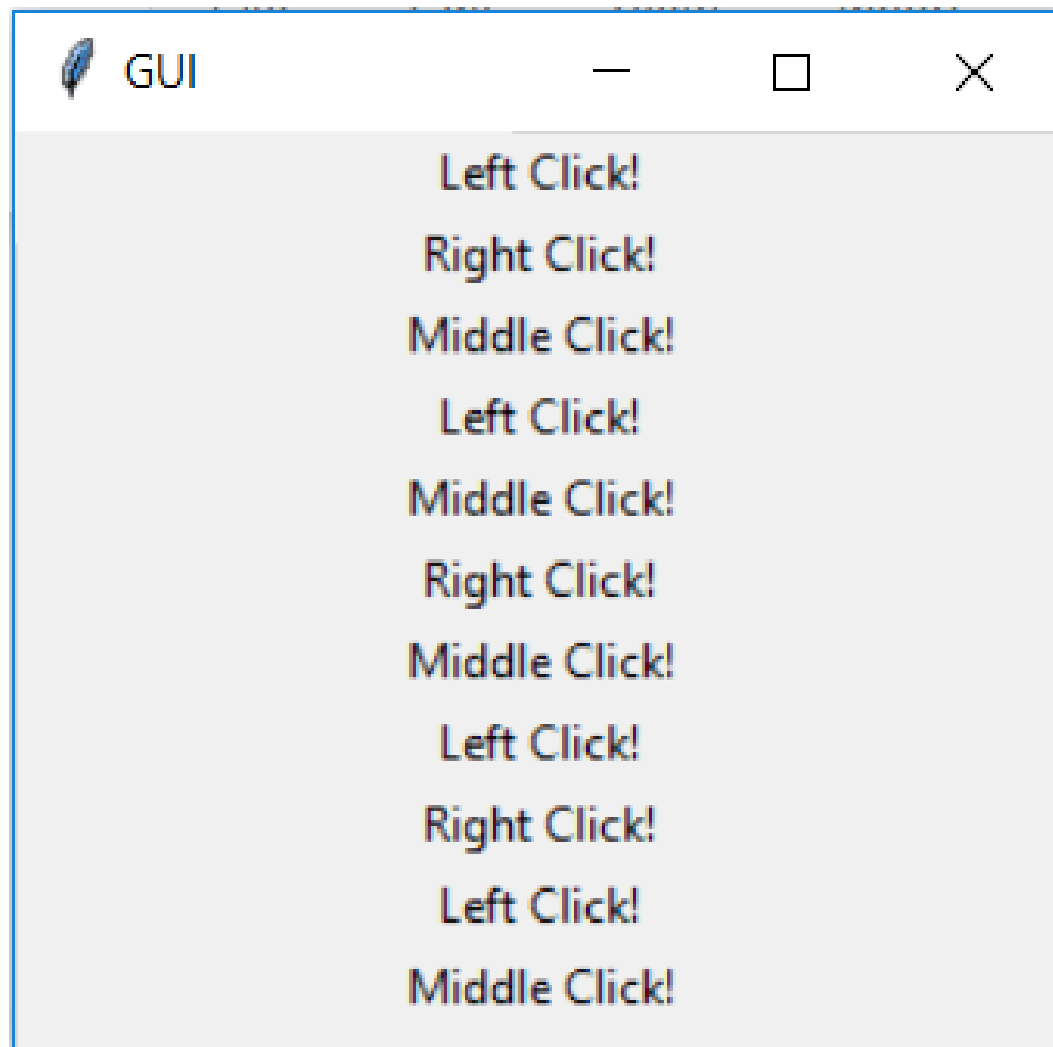
```
def middle_click(event):  
    tkinter.Label(window, text = "Middle Click!").pack()
```

```
def right_click(event):  
    tkinter.Label(window, text = "Right Click!").pack()
```

```
window.bind("<Button-1>", left_click)  
window.bind("<Button-2>", middle_click)  
window.bind("<Button-3>", right_click)
```

```
window.mainloop()
```





## MESSAGE BOX

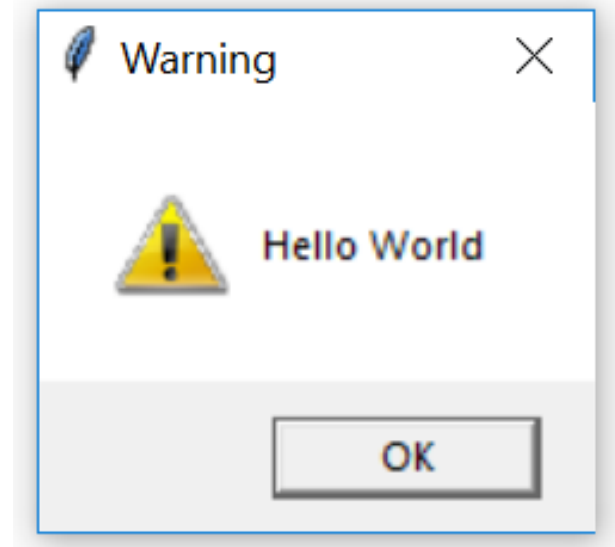
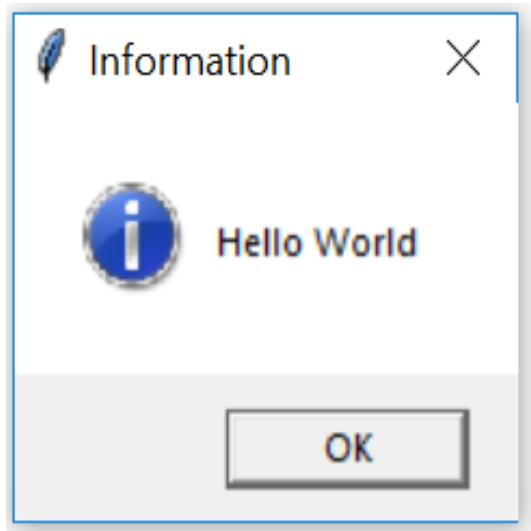
```
from tkinter import *
from tkinter import messagebox

top = Tk()
top.geometry("100x100")
def hello():
    messagebox.showinfo("Information", "Hello World") #1
    messagebox.showwarning("Warning", "Hello World") #2
    messagebox.showerror("Error", "Hello World") #3
    messagebox.askquestion("Question", "Are you sure?") #4
    messagebox.askyesno("Say Hello", "Hello world") #5
    messagebox.askretrycancel("Retry", "Want to retry") #6
    messagebox.askyesnocancel("Ask", "Are you sure?") #7
B1 = Button(top, text = "Say Hello", command = hello)
B1.place(x = 35,y = 50)

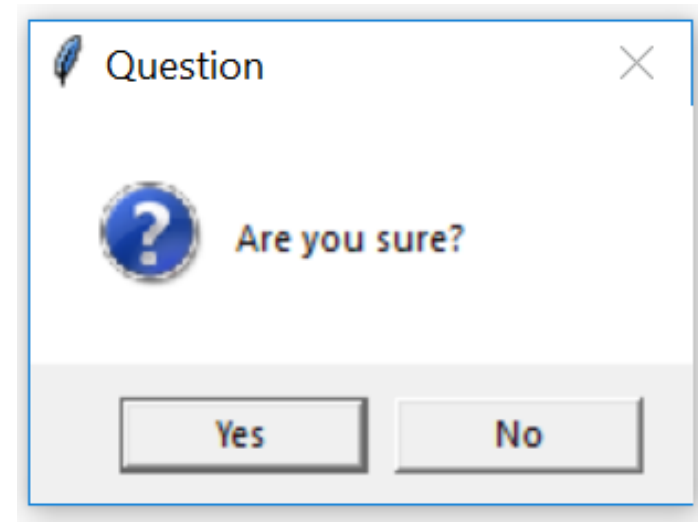
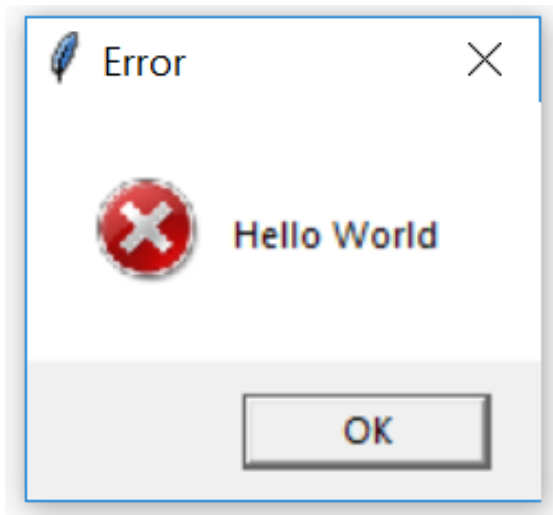
top.mainloop()
```



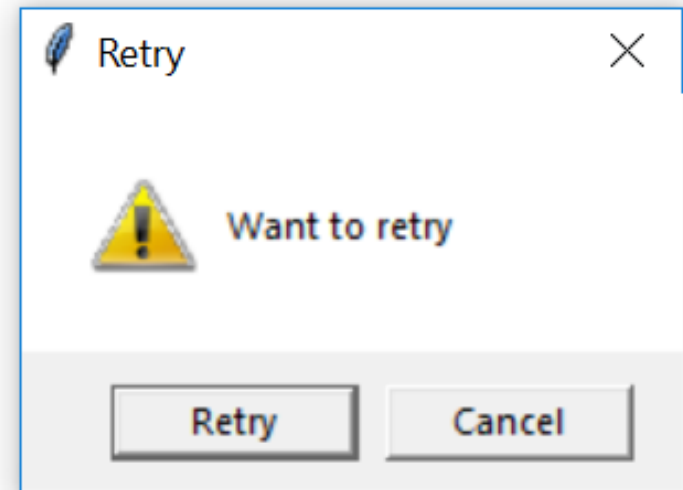
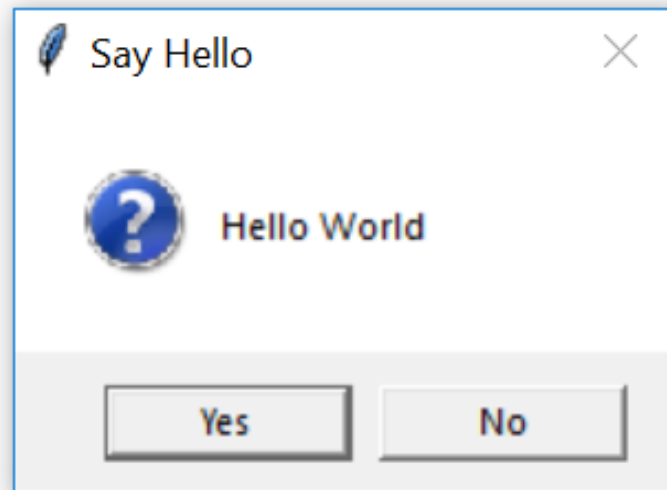
1



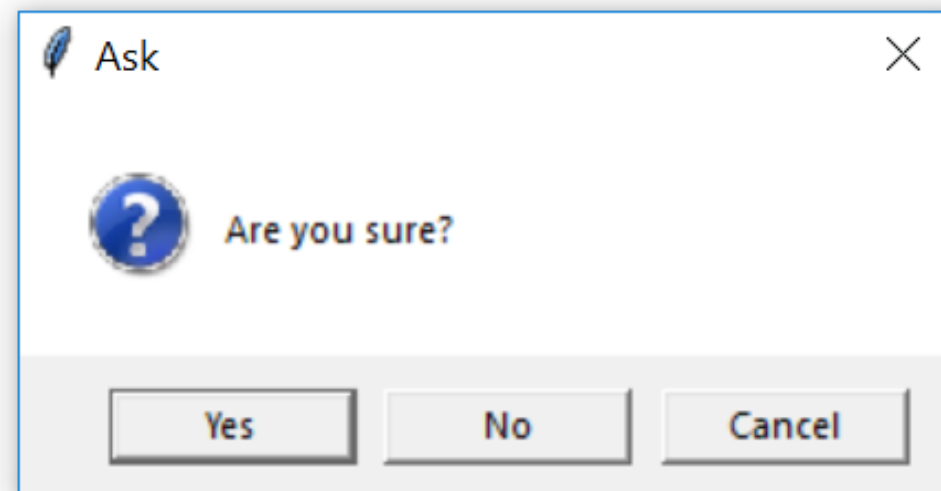
3



5



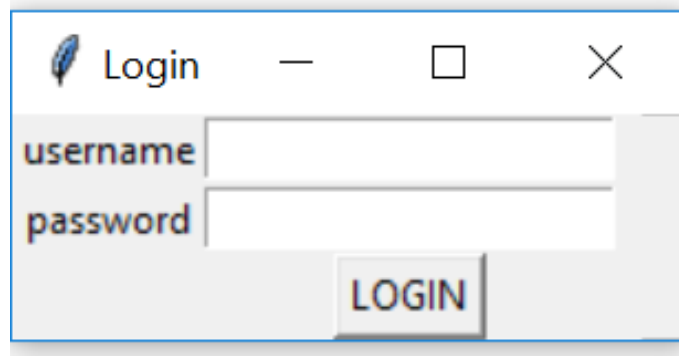
7



# LOGIN SCREEN

```
from tkinter import *
gui = Tk()
us1=StringVar()
pw1=StringVar()
def login():
    if us1.get() == 'Abc' and pw1.get()=='AAA':
        i=Label(gui,text='Login success').grid(row=6,column=0)
        print("login success")
    else:
        print("wrong password")
        j=Label(gui,text='Login failed').grid(row=6,column=0)
gui.title("Login")
a = Label(gui ,text="username").grid(row=0,column = 0)
b = Label(gui ,text="password").grid(row=1,column=0)
e = Entry(gui,textvariable=us1).grid(row=0,column=1)
f = Entry(gui,textvariable=pw1,show="*").grid(row=1,column=1)
c1 = Button(gui, text="LOGIN",command=login).grid(row=5,
column=1)
gui.mainloop()
```





A login window titled "Login" with a feather icon. It contains two input fields labeled "username" and "password", and a "LOGIN" button.

username

password

LOGIN



# MENU

```
import tkinter as tk
from tkinter import ttk
from tkinter import Menu

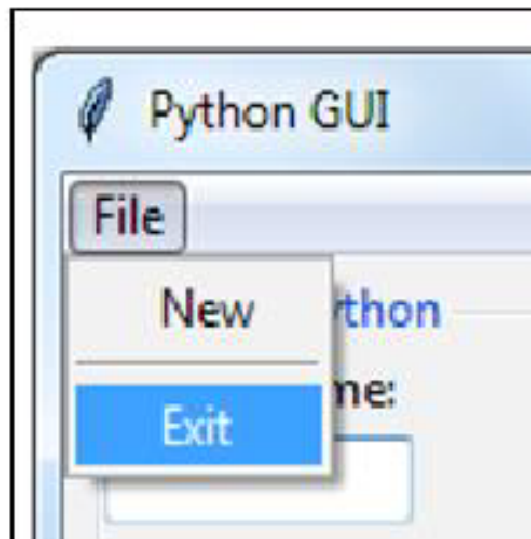
win = tk.Tk()
win.title("Python GUI")
menuBar = Menu(win)
win.config(menu=menuBar)
fileMenu = Menu(menuBar)
fileMenu.add_command(label="New")
fileMenu.add_separator()
fileMenu.add_command(label="Exit")
menuBar.add_cascade(label="File", menu=fileMenu)
fileMenu = Menu(menuBar, tearoff=0)

helpMenu = Menu(menuBar, tearoff=0)
helpMenu.add_command(label="About")
menuBar.add_cascade(label="Help", menu=helpMenu)

win.mainloop()
```







- All the codes done in the class also need to be taken care of

