SARVAJANIK COLLEGE OF ENGINEERING & TECHNOLOGY

Information Technology

Python Programming

Topic

Building a basic GUI application step-by-step in Python with Tkinter with example.

Python - GUI Programming (Tkinter)

- Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below:

- **Tkinter:** Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.

- **wxPython:** This is an open-source Python interface for wxWindows http://wxpython.org.

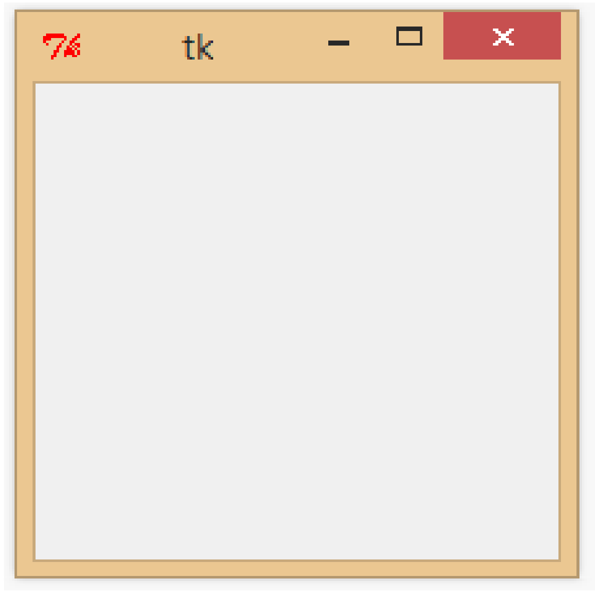
- **JPython:** JPython is a Python port for Java, which gives Python scripts seamless access to Java class libraries on the local machine http://www.jython.org.

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

Example :

```
import Tkinter

top = Tkinter.Tk()

# Code to add widgets will go here...

top.mainloop()
```



Tkinter Widgets

- Tkinter provides various controls that are used in GUI application.
- These controls are commonly called widgets.
- There are various types of widgets in Tkinter.

•Button
•Canvas
•Checkbutton
•Entry
•Frame
•Label
•Listbox
•Menubutton
•Menu
•Message

- Radiobutton
- Scale
- Scrollbar
- Text
- Toplevel
- Spinbox
- PanedWindow
- LabelFrame
- tkMessagebox

Tkinter Label

- This widget implements a display box where you can place text. The text displayed by this widget can be updated at any time you want.
- It is also possible to underline part of the text (like to identify a keyboard shortcut), and span the text across multiple lines.

Syntax:

Here is the simple syntax to create this widget:

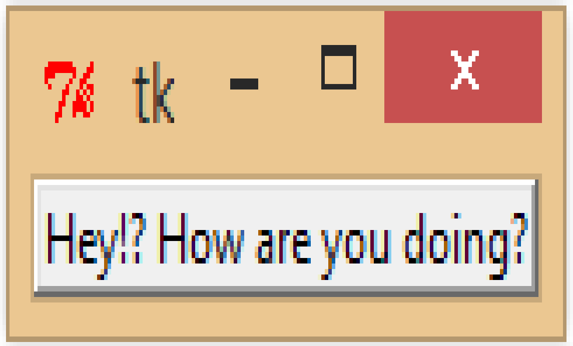
```
w = Label ( master, option, ... )
```

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.

Example:

```
from Tkinter import *

root = Tk()

var = StringVar()

label = Label( root, textvariable=var, relief=RAISED )

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

label.pack()

root.mainloop()
```


tk
Hey!? How are you doing?

Button Widgets

- 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

Syntax:

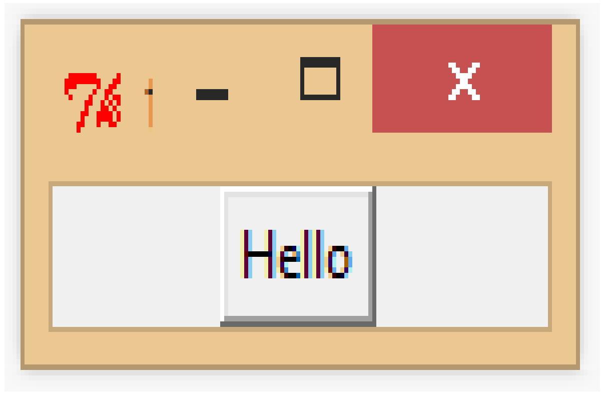
```
w = Button ( master, option=value, ... )
```

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. which is called automatically when you click the button.

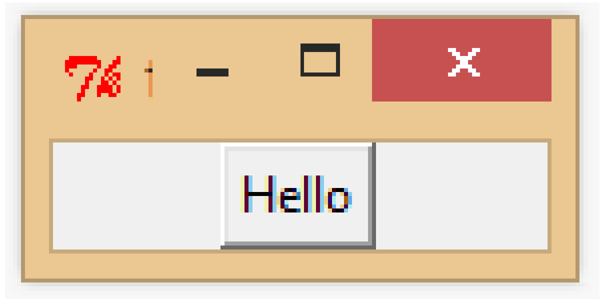
Example:

```
import Tkinter

root = Tk()

button = Button(root, text ="Hello")

button.pack()

top.mainloop()
```

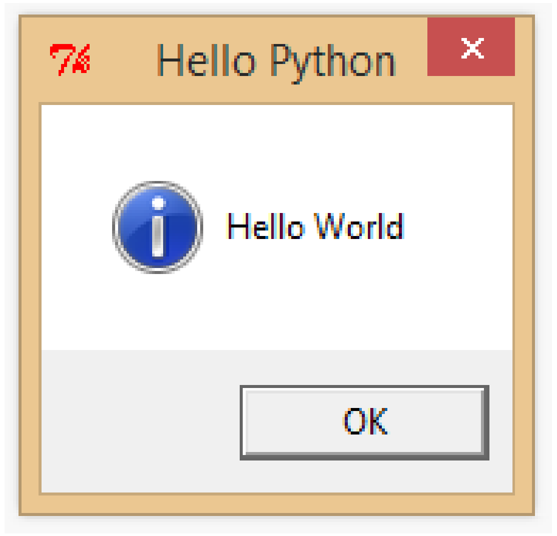

Hello

Example:

```
import Tkinter

import tkMessageBox

root = Tk()

def helloCallBack():

   tkMessageBox.showinfo( "Hello Python", "Hello World")

button = Button(root, text ="Hello", command = helloCallBack)

button.pack()

root.mainloop()
```



Click Button



Hello World

tkMessagebox

Tkinter 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 usethe *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:

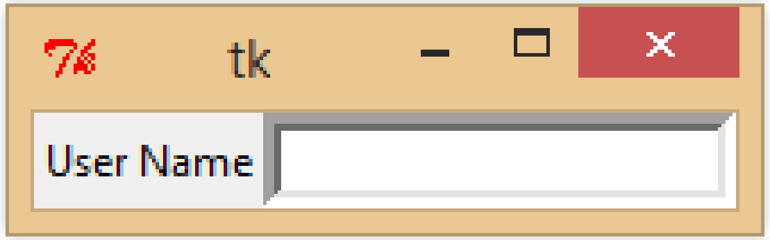
w = Entry(master, option, ...)

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)
top.mainloop()
```


tk
User Name

Tkinter 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.
- You can also display images in place of text.

Syntax:

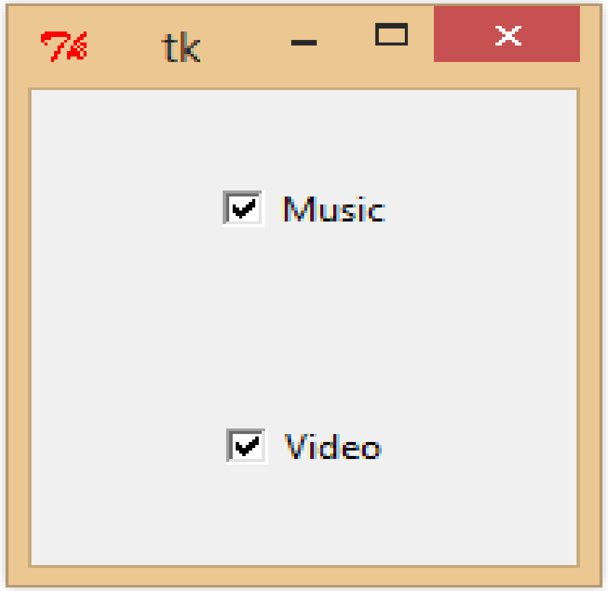
```
w = Checkbutton ( master, option, ... )
```

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.

Example:

```
from Tkinter import *
import tkMessageBox
import Tkinter
top = Tkinter.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()
```


tk
Music
Video

Radio button

- This widget implements a multiple-choice button, which is a way to offer many possible selections to the user, and let 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.

Syntax:

Here is the simple syntax to create this widget:

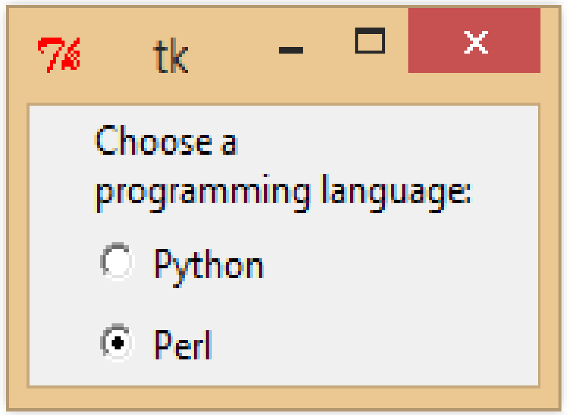
w = Radiobutton (master, option, ...)

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.

Example:

```
from Tkinter import *
root = Tk()
var = IntVar()
tk.Label(root, text="Choose a programming language:",  justify = tk.LEFT, padx = 20).pack()
tk.Radiobutton(root, text="Python", padx = 20, variable=var, value=1).pack(anchor=tk.W)
tk.Radiobutton(root, text="Perl", padx = 20, variable=var, value=2).pack(anchor=tk.W)
root.mainloop()
```



Choose a
programming language:

( ) Python

(●) Perl

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:

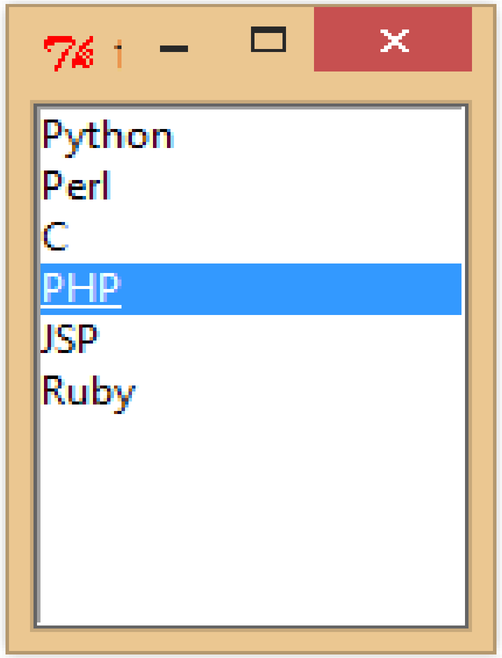
  w = Listbox (master, option, ...)

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.

Example:

```
from Tkinter import *
import tkMessageBox
import Tkinter
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()
```



Python
Perl
C
PHP
JSP
Ruby

Tkinter 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:

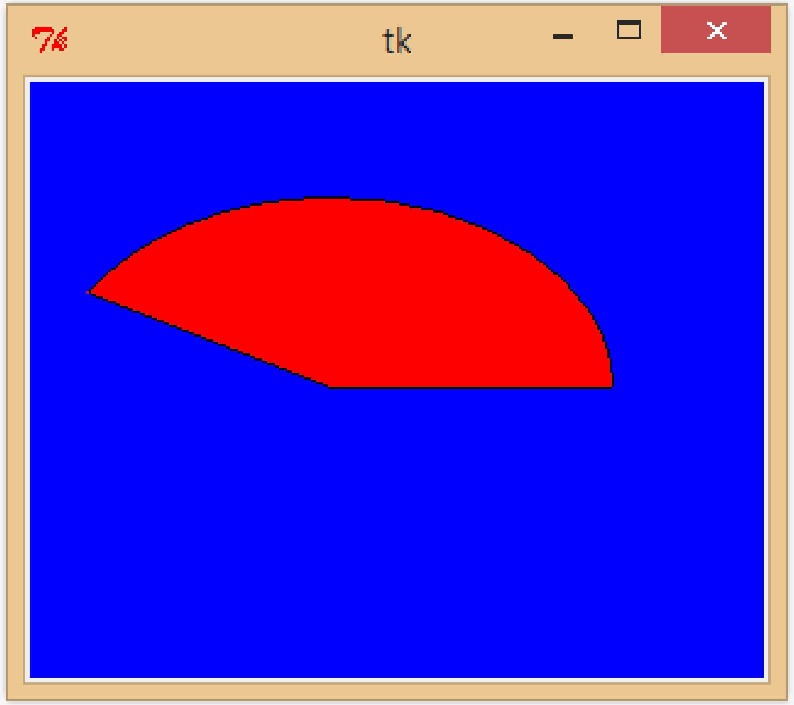
```
w = Canvas ( master, option=value, ... )
```

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:

```
import Tkinter

import tkMessageBox

top = Tkinter.Tk()

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

coord = 10, 50, 240, 210

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

C.pack()

top.mainloop()
```


tk

Tkinter Frame

- 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:

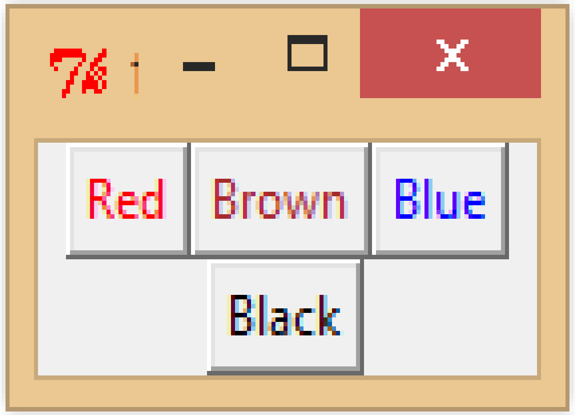
```
w = Frame ( master, option, ... )
```

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 *
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( side = BOTTOM)
root.mainloop()
```


Red
Brown
Blue
Black

Tkinter **Standard Widgets**

There are currently 15 types of widgets in Tkinter

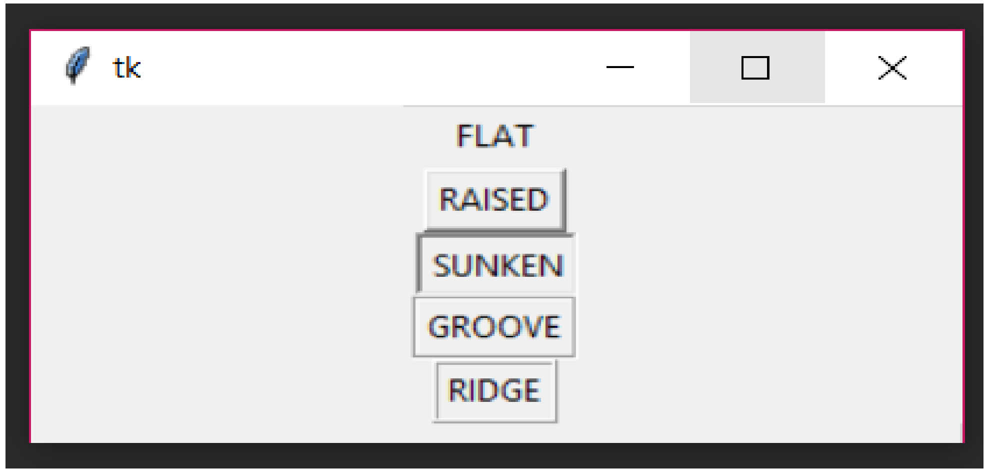
widget	Description
Button	The Button widget is used to display buttons in your application.
Canvas	The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
Check button	The Checkbutton widget is used to display a number of options as checkboxes.
Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
Frame	The Frame widget is used as a container widget to organize other widgets.
Label	The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
Listbox	The Listbox widget is used to provide a list of options to a user.

Standard attributes

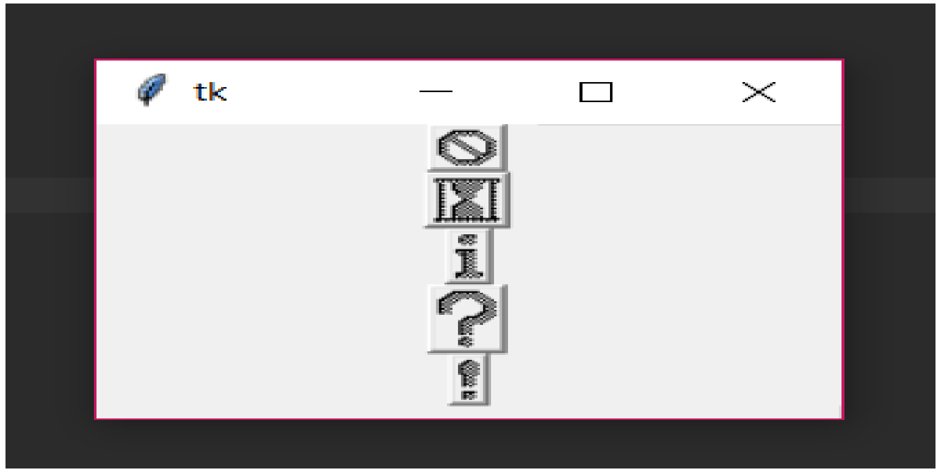
of these widgets

- Dimensions- c,i,m,p
- Colors- hexadecimal digits:#000000, standard color name: red
- Fonts- ("Times", "24", "bold italic") For regular null
- Anchors- N,E,S,W, NW, SE, etc
- Relief styles- refers to certain simulated 3-D effects around the outside of the widget.
- Bitmaps- attribute to displays a bitmap
- Cursors- supports quite a number of different mouse cursors available. Ex: arrow, circle, plus, heart, etc

Relief styles



Bitmaps



Geometry Management

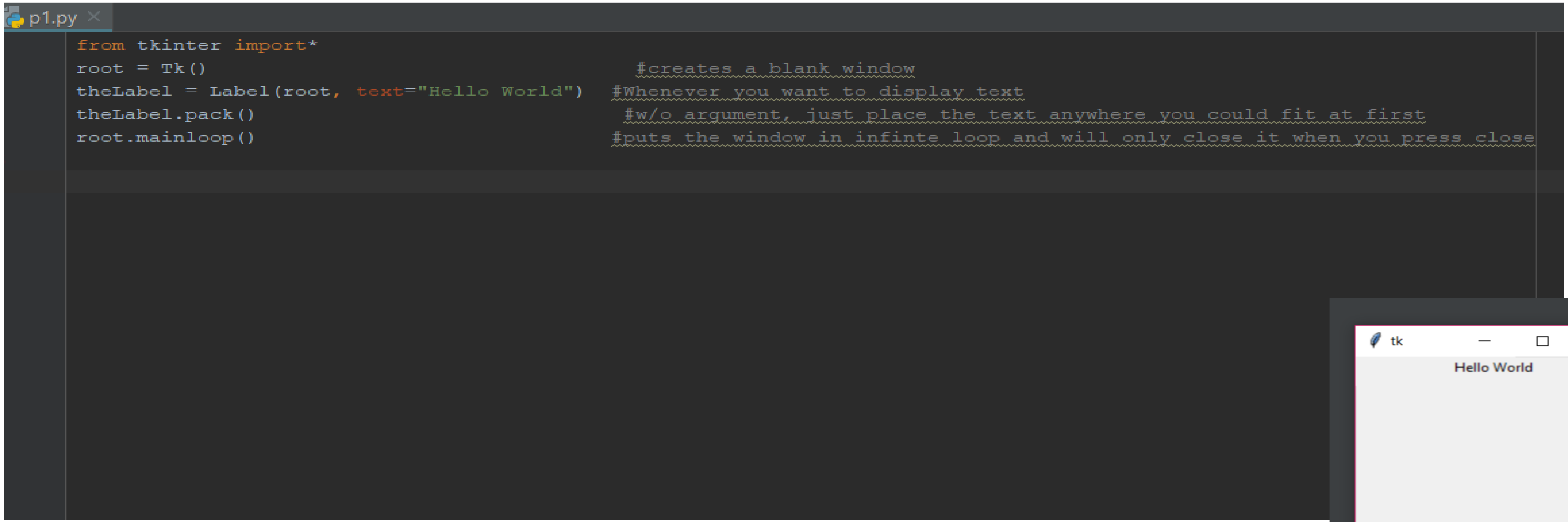
Tkinter exposes the following geometry manager classes

- The *pack()* Method – This geometry manager organizes widgets in blocks before placing them in the parent widget.
- The *grid()* Method – This geometry manager organizes widgets in a table-like structure in the parent widget.
- The *place()* Method – This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Containers

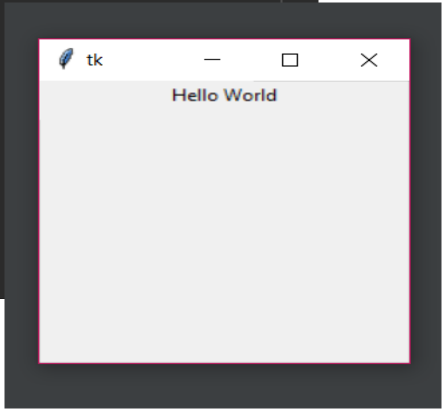
- **Canvas-**is a rectangular area intended for drawing pictures or other complex layouts. You can place graphics, text, widgets or frames on a Canvas.
- **Frame-** does the process of grouping and organizing other widgets in a somehow friendly way.
- **Toplevel-**work as windows that are directly managed by the window manager. They do not necessarily have a parent widget on top of them.
- **PanedWindow-**is a container widget that may contain any number of panes, arranged horizontally or vertically. uses rectangular areas in the screen to organize the layout and to provide padding of these widgets.

Every serious tutorial should start with a "hello world" example.



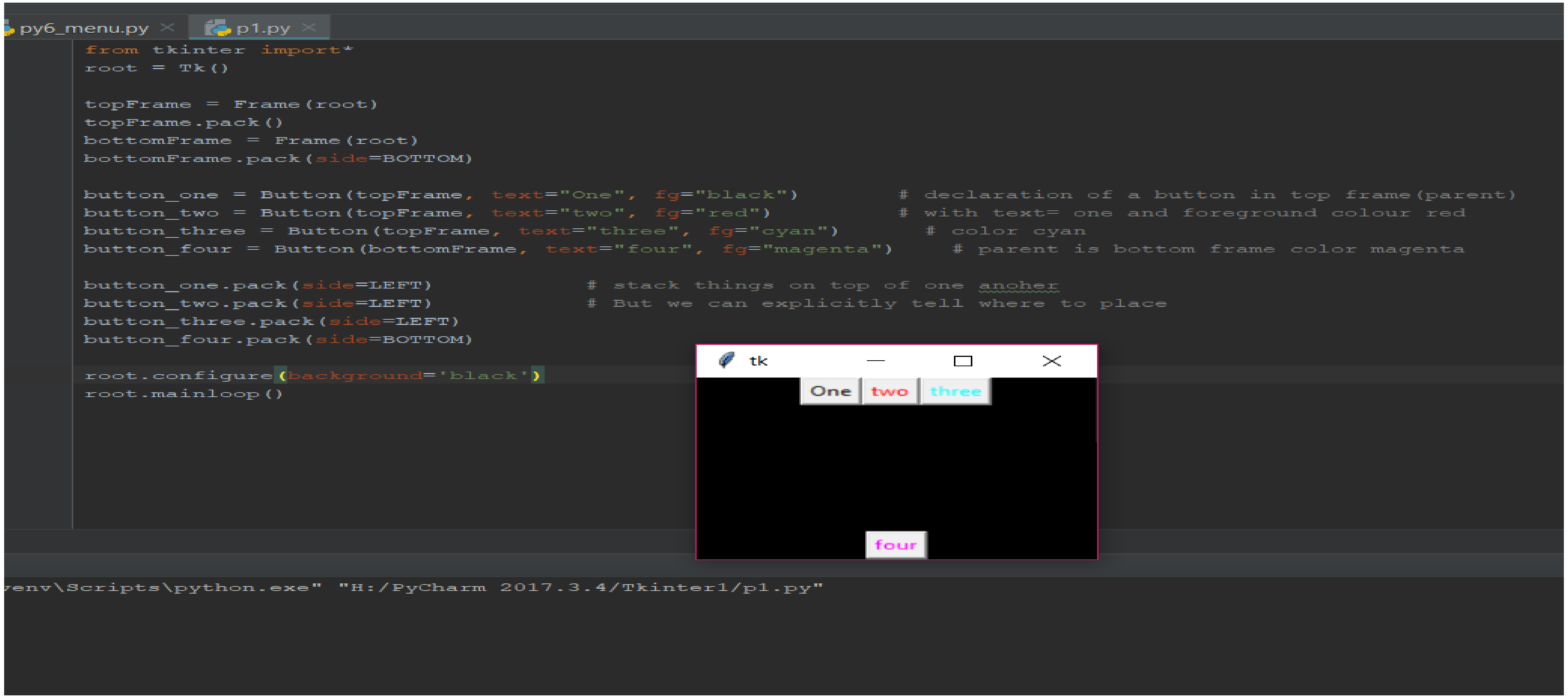
p1.py

```
from tkinter import*
root = Tk()                                             #creates a blank window
theLabel = Label(root, text="Hello World")  #Whenever you want to display text
theLabel.pack()                                          #w/o argument, just place the text anywhere you could fit at first
root.mainloop()                                         #puts the window in infinte loop and will only close it when you press close
```



Hello World

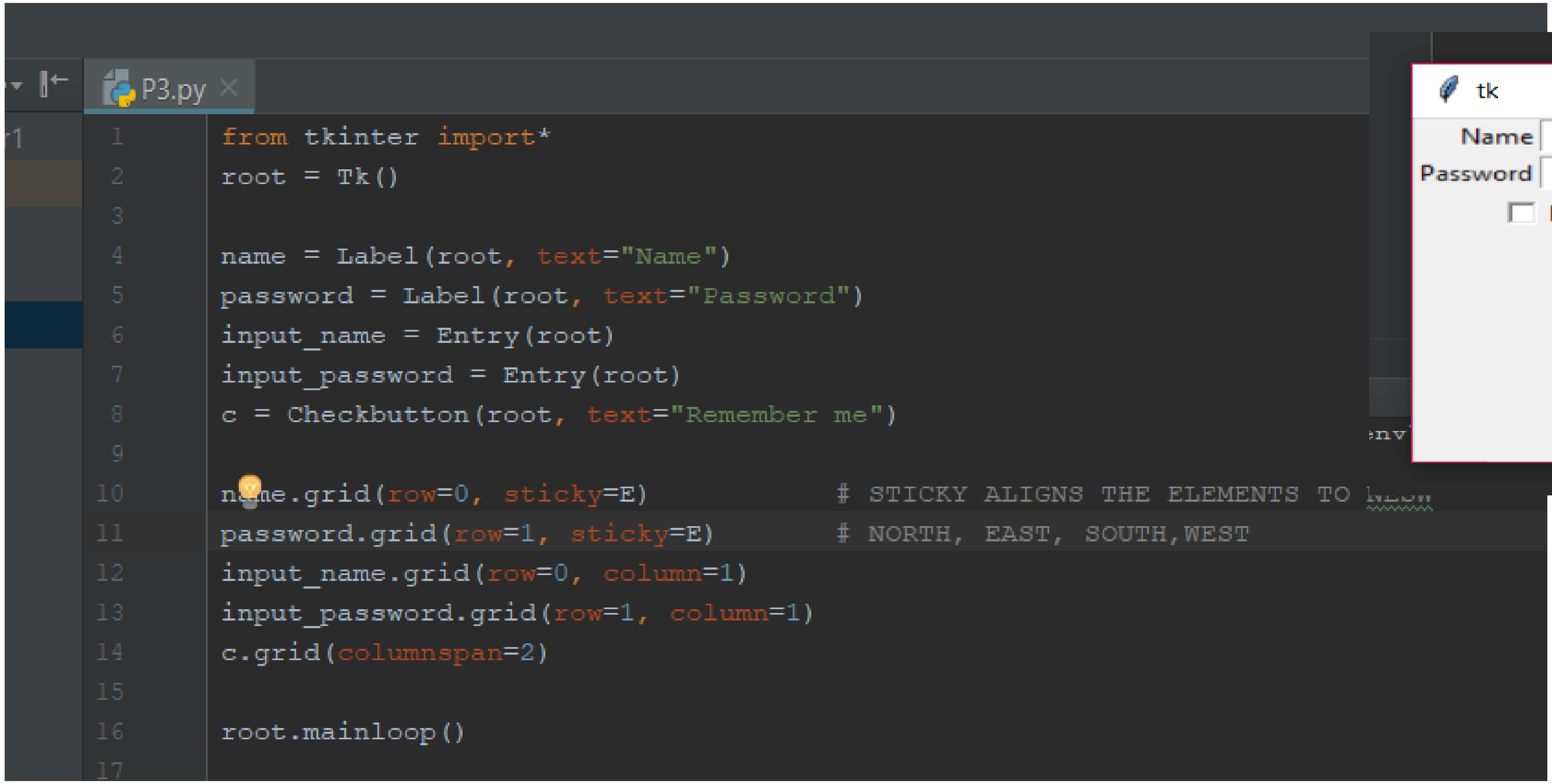
Creating frames and adding buttons to them



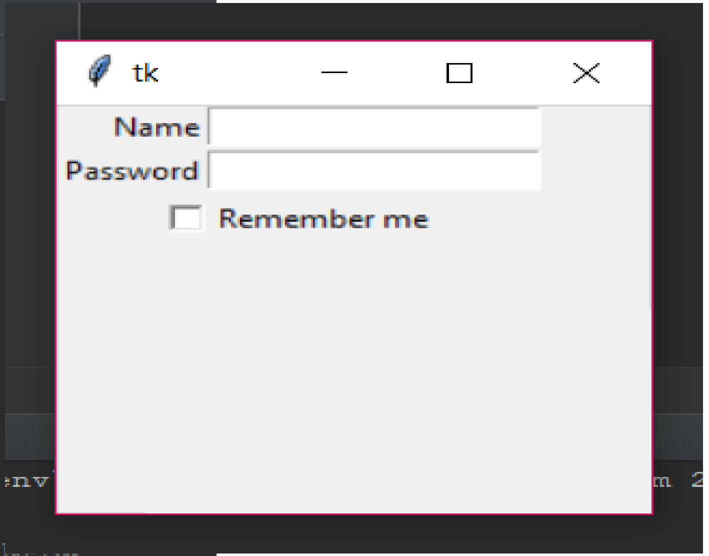
```
py6_menu.py ×    p1.py ×
from tkinter import*
root = Tk()

topFrame = Frame(root)
topFrame.pack()
bottomFrame = Frame(root)
bottomFrame.pack(side=BOTTOM)

button_one = Button(topFrame, text="One", fg="black")         # declaration of a button in top frame(parent)
button_two = Button(topFrame, text="two", fg="red")           # with text= one and foreground colour red
button_three = Button(topFrame, text="three", fg="cyan")       # color cyan
button_four = Button(bottomFrame, text="four", fg="magenta")     # parent is bottom frame color magenta

button_one.pack(side=LEFT)                      # stack things on top of one anoher
button_two.pack(side=LEFT)                      # But we can explicitly tell where to place
button_three.pack(side=LEFT)
button_four.pack(side=BOTTOM)

root.configure(background='black')
root.mainloop()
```

```
venv\Scripts\python.exe" "H:/PyCharm 2017.3.4/Tkinter1/p1.py"
```

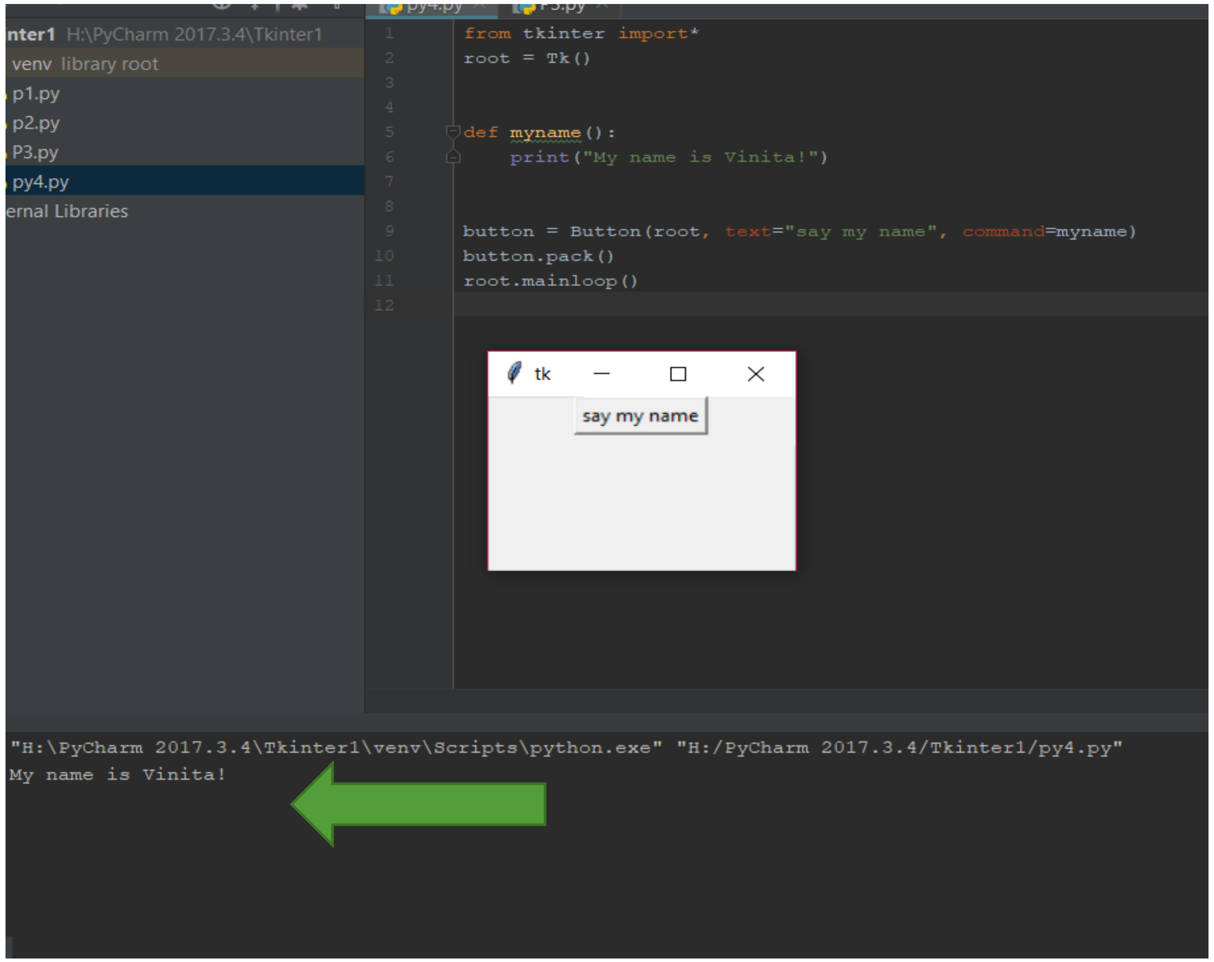
Creating a user login screen



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

name = Label(root, text="Name")
password = Label(root, text="Password")
input_name = Entry(root)
input_password = Entry(root)
c = Checkbutton(root, text="Remember me")

name.grid(row=0, sticky=E)          # STICKY ALIGNS THE ELEMENTS TO NESW
password.grid(row=1, sticky=E)      # NORTH, EAST, SOUTH,WEST
input_name.grid(row=0, column=1)
input_password.grid(row=1, column=1)
c.grid(columnspan=2)

root.mainloop()
```

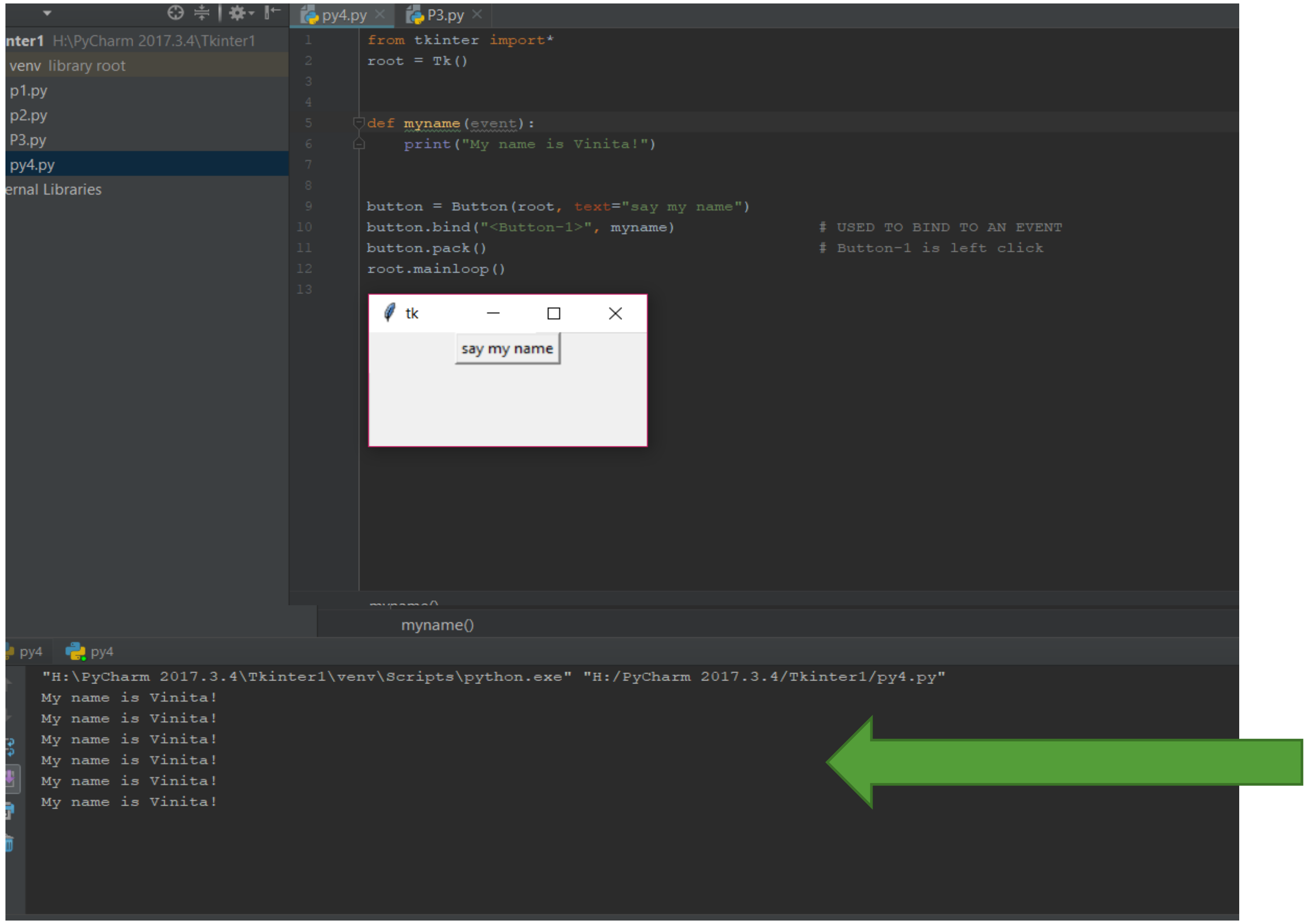


Onclick Events: Command function



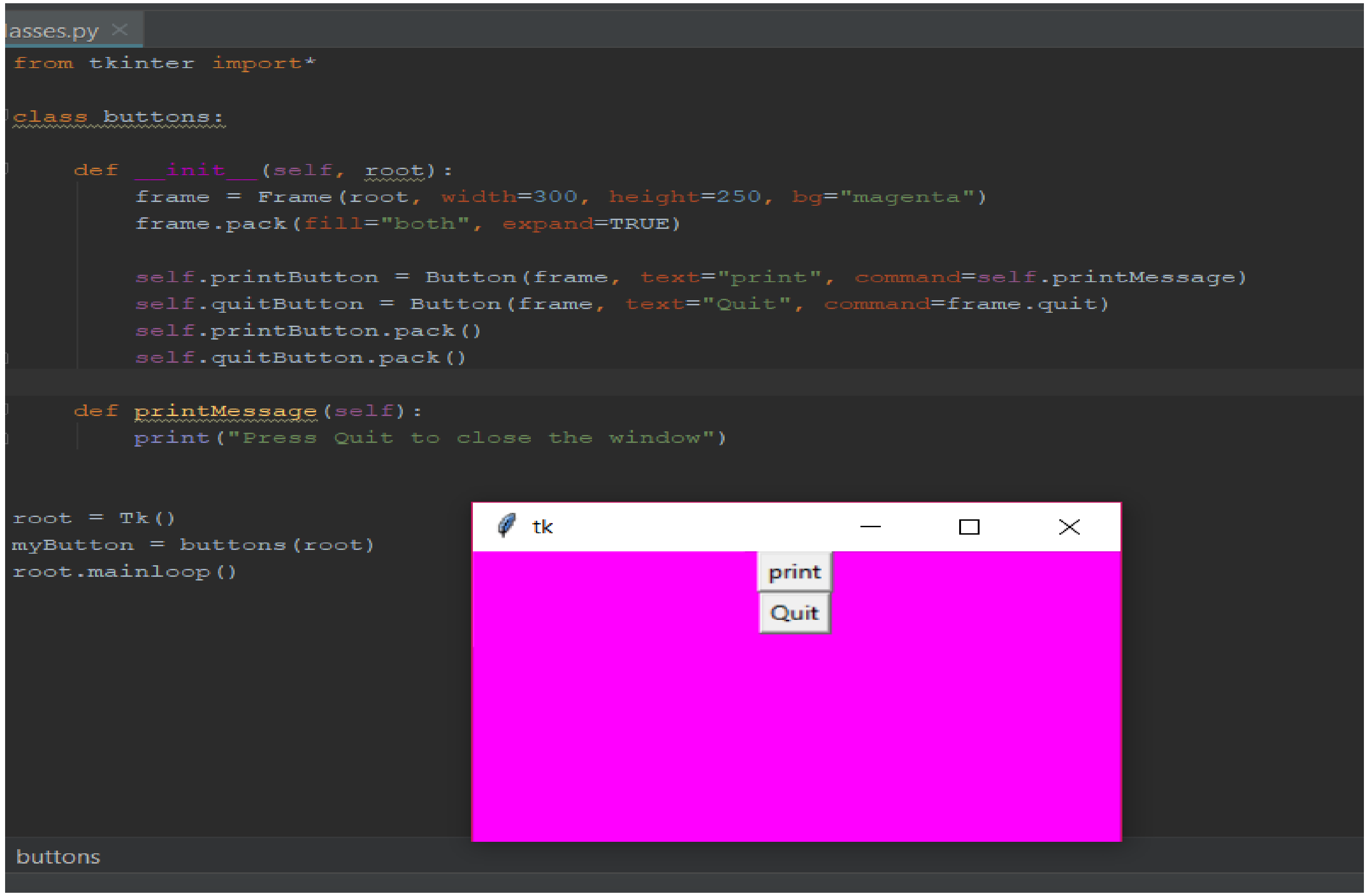
```
from tkinter import*
root = Tk()


def myname():
    print("My name is Vinita!")


button = Button(root, text="say my name", command=myname)
button.pack()
root.mainloop()
```

```
"H:\PyCharm 2017.3.4\Tkinter1\venv\Scripts\python.exe" "H:/PyCharm 2017.3.4/Tkinter1/py4.py"
My name is Vinita!
```


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


def myname(event):
    print("My name is Vinita!")


button = Button(root, text="say my name")
button.bind("<Button-1>", myname)                  # USED TO BIND TO AN EVENT
button.pack()                                        # Button-1 is left click
root.mainloop()
```

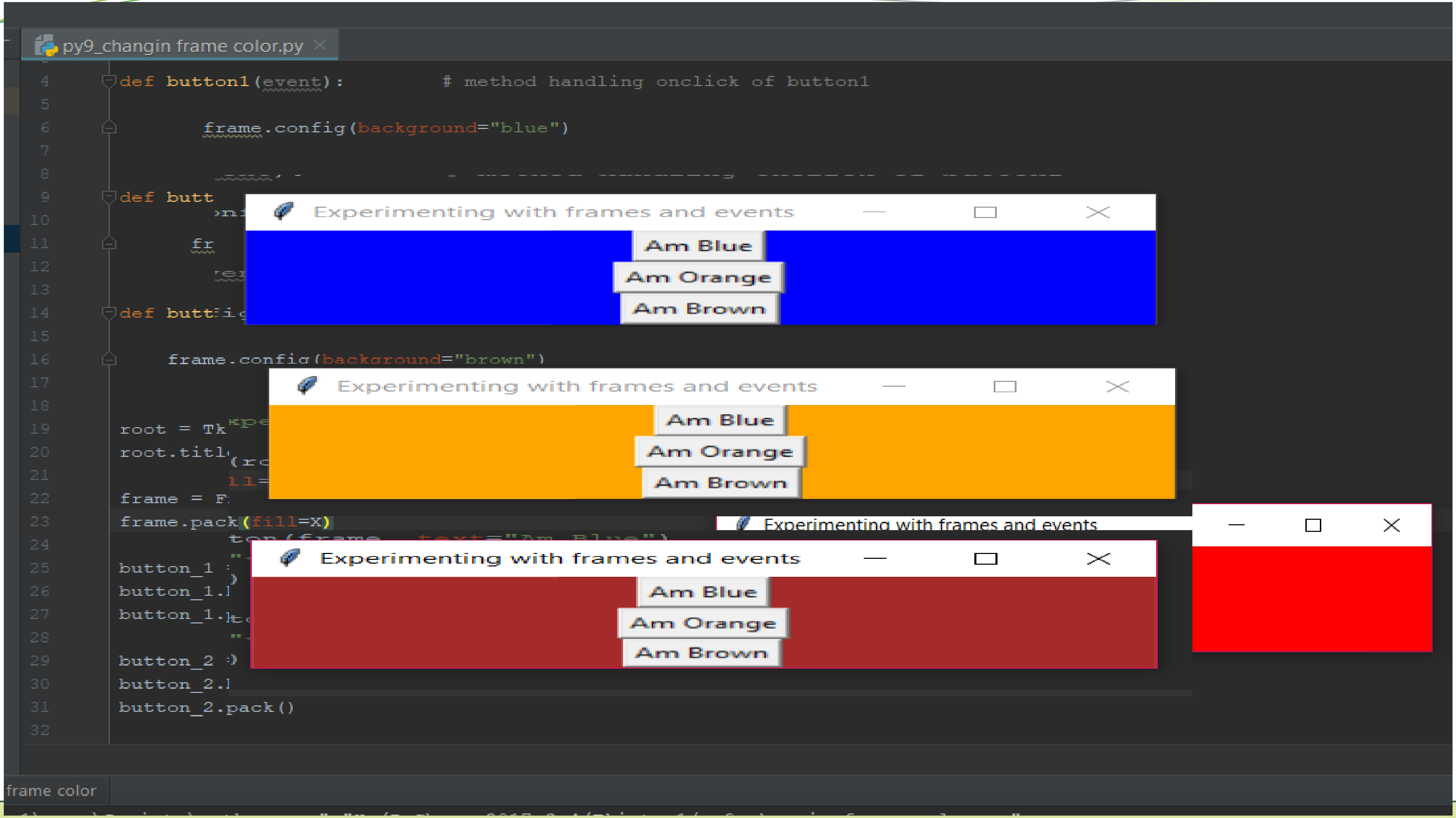
```
"H:\PyCharm 2017.3.4\Tkinter1\venv\Scripts\python.exe" "H:/PyCharm 2017.3.4/Tkinter1/py4.py"
My name is Vinita!
My name is Vinita!
My name is Vinita!
My name is Vinita!
My name is Vinita!
My name is Vinita!
```

Classes



lasses.py

```
from tkinter import*

class buttons:

    def __init__(self, root):
        frame = Frame(root, width=300, height=250, bg="magenta")
        frame.pack(fill="both", expand=TRUE)

        self.printButton = Button(frame, text="print", command=self.printMessage)
        self.quitButton = Button(frame, text="Quit", command=frame.quit)
        self.printButton.pack()
        self.quitButton.pack()

    def printMessage(self):
        print("Press Quit to close the window")


root = Tk()
myButton = buttons(root)
root.mainloop()
```

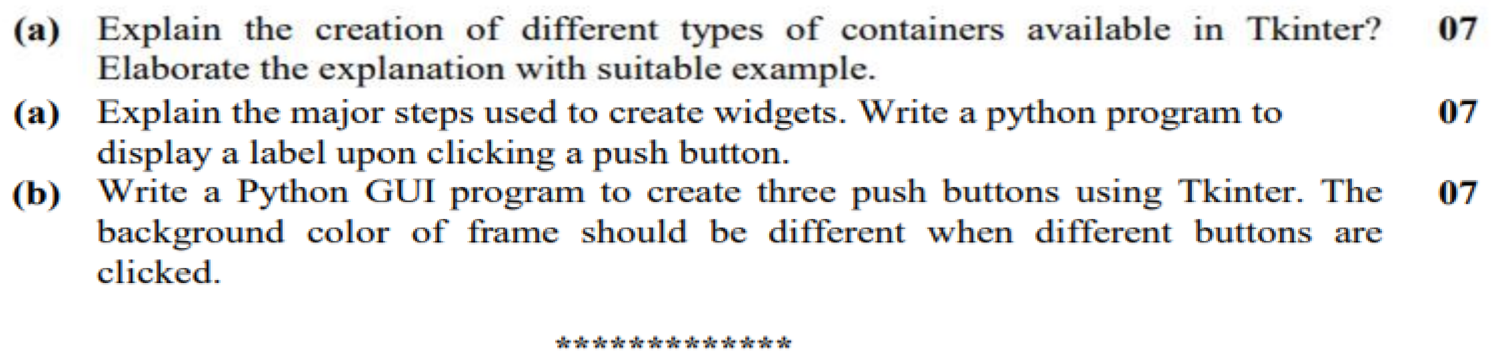


py9_changin frame color.py

```
def button1(event):          # method handling onclick of button1

        frame.config(background="blue")


def butt

       fr

def butt

      frame.config(background="brown")


root = Tk
root.titl

frame = F
frame.pack(fill=X)

button_1
button_1.
button_1.

button_2
button_2.
button_2.pack()
```

Experimenting with frames and events
Am Blue
Am Orange
Am Brown
Experimenting with frames and events
Am Blue
Am Orange
Am Brown
Experimenting with frames and events
Experimenting with frames and events
Am Blue
Am Orange
Am Brown
frame color

Questions from Tkinter



(a)	Explain the creation of different types of containers available in Tkinter? Elaborate the explanation with suitable example.	**07**
(a)	Explain the major steps used to create widgets. Write a python program to display a label upon clicking a push button.	**07**
(b)	Write a Python GUI program to create three push buttons using Tkinter. The background color of frame should be different when different buttons are clicked.	**07**

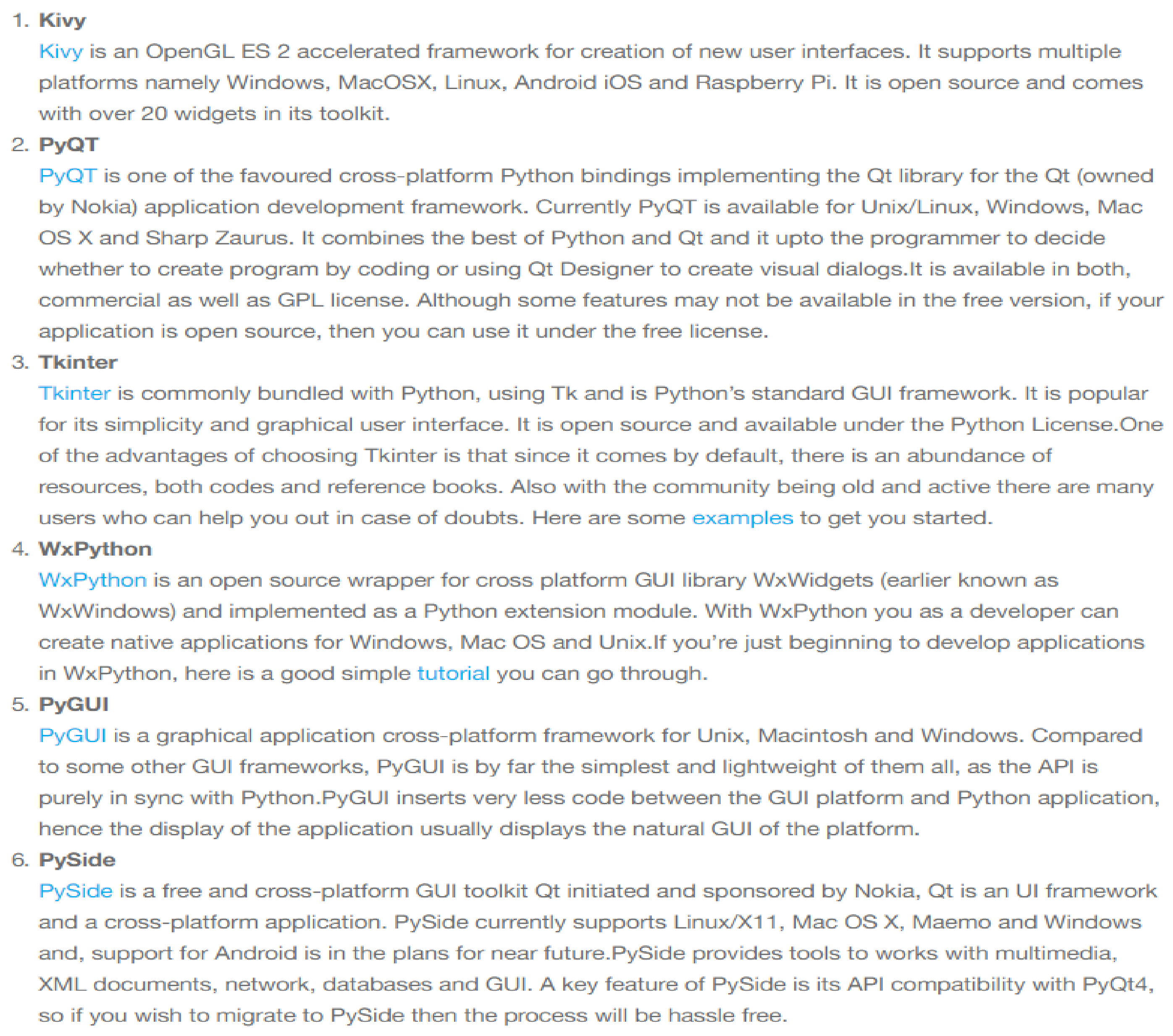
****

Advantages

- Popular for its simplicity and graphical user interface
- It is open source and available under the Python License
- Since it comes by default, there is an abundance of resources, both codes and reference books.
- Also with the community being old and active there are many users who can help you out in case of doubts.
- simple to use and intuitive in nature, making it suitable for programmers and non-programmers alike.

Limitations

- Tkinter is nice to write small programs (then it doesn't require a GUI Designer), but is not really appropriate for large application development.
- Tkinter and/or apps written using it tend to be simple enough to not really need that much 'help'. So no drag and drop



1. **Kivy**
   Kivy is an OpenGL ES 2 accelerated framework for creation of new user interfaces. It supports multiple platforms namely Windows, MacOSX, Linux, Android iOS and Raspberry Pi. It is open source and comes with over 20 widgets in its toolkit.
2. **PyQT**
   PyQT is one of the favoured cross-platform Python bindings implementing the Qt library for the Qt (owned by Nokia) application development framework. Currently PyQT is available for Unix/Linux, Windows, Mac OS X and Sharp Zaurus. It combines the best of Python and Qt and it upto the programmer to decide whether to create program by coding or using Qt Designer to create visual dialogs.It is available in both, commercial as well as GPL license. Although some features may not be available in the free version, if your application is open source, then you can use it under the free license.
3. **Tkinter**
   Tkinter is commonly bundled with Python, using Tk and is Python's standard GUI framework. It is popular for its simplicity and graphical user interface. It is open source and available under the Python License.One of the advantages of choosing Tkinter is that since it comes by default, there is an abundance of resources, both codes and reference books. Also with the community being old and active there are many users who can help you out in case of doubts. Here are some examples to get you started.
4. **WxPython**
   WxPython is an open source wrapper for cross platform GUI library WxWidgets (earlier known as WxWindows) and implemented as a Python extension module. With WxPython you as a developer can create native applications for Windows, Mac OS and Unix.If you're just beginning to develop applications in WxPython, here is a good simple tutorial you can go through.
5. **PyGUI**
   PyGUI is a graphical application cross-platform framework for Unix, Macintosh and Windows. Compared to some other GUI frameworks, PyGUI is by far the simplest and lightweight of them all, as the API is purely in sync with Python.PyGUI inserts very less code between the GUI platform and Python application, hence the display of the application usually displays the natural GUI of the platform.
6. **PySide**
   PySide is a free and cross-platform GUI toolkit Qt initiated and sponsored by Nokia, Qt is an UI framework and a cross-platform application. PySide currently supports Linux/X11, Mac OS X, Maemo and Windows and, support for Android is in the plans for near future.PySide provides tools to works with multimedia, XML documents, network, databases and GUI. A key feature of PySide is its API compatibility with PyQt4, so if you wish to migrate to PySide then the process will be hassle free.

These are some of the widely used and best Python GUI frameworks available.

These frameworks help developers create GUI applications in an easy and secure manner.

Depending on your requirements you can choose the Python GUI framework that is best suited for you.

References

- www.tutorialspoint.com
- https://thenewboston.com/

Thank You