**Name:- Tausif Khan**

**Roll No:- 12**

**Div:- B**

**Class:- SE/Comps**

**Subject:- OSTL**

**Experiment No: 9.**

**AIM:** A python program to create three push buttons and change the background of the frame according to the button clicked by the user.

**Tools Used:** Python 3.4.3, Terminal

**Theory:**

1. **Explain GUI in python and different GUIs in Python**

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

**To create a tkinter app:**

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

**Different type of GUI in Python**

**1. Kivy -**  
[**Kivy**](https://kivy.org/#home)is an OpenGL ES 2 accelerated framework for the 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**](https://riverbankcomputing.com/news)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 up to the programmer to decide whether to create a 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**](https://wiki.python.org/moin/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**](https://docs.python.org/2/library/tkinter.html)to get you started.

**4. WxPython -**  
[**WxPython**](https://wiki.python.org/moin/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**](http://zetcode.com/wxpython/)you can go through.

1. **PyGUI -**

[**PyGUI**](http://www.cosc.canterbury.ac.nz/greg.ewing/python_gui/)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.

1. **PySide -**

[**PySide**](https://wiki.qt.io/About_PySide)is a free and cross-platform GUI toolkit Qt initiated and sponsored by Nokia, Qt is a 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 the 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.

1. **Explain Tkinter and its basic working**

We have previously seen how to write text-only programs which have a *command-line interface*, or CLI. Now we will briefly look at creating a program with a *graphical user interface*, or GUI. In this chapter we will use tkinter, a module in the Python standard library which serves as an interface to Tk, a simple *toolkit*. There are many other toolkits available, but they often vary across platforms. If you learn the basics of tkinter, you should see many similarities should you try to use a different toolkit.

We will see how to make a simple GUI which handles user input and output. GUIs often use a form of OO programming which we call *event-driven*: the program responds to *events*, which are actions that a user takes.

Anything that happens in a user interface is an *event*. We say that an event is *fired* whenever the user does something – for example, clicks on a button or types a keyboard shortcut. Some events could also be triggered by occurrences which are not controlled by the user – for example, a background task might complete, or a network connection might be established or lost.

Our application needs to monitor, or *listen* for, all the events that we find interesting, and respond to them in some way if they occur. To do this, we usually associate certain functions with particular events. We call a function which performs an action in response to an event an *event handler* – we *bind* handlers to events.

**Tkinter basics-**

Tkinter provides us with a variety of common GUI elements which we can use to build our interface – such as buttons, menus and various kinds of entry fields and display areas. We call these elements *widgets*. We are going to construct a *tree* of widgets for our GUI – each widget will have a parent widget, all the way up to the *root window* of our application. For example, a button or a text field needs to be *inside* some kind of containing window.

The widget classes provide us with a lot of default functionality. They have methods for configuring the GUI’s appearance – for example, arranging the elements according to some kind of *layout* – and for handling various kinds of user-driven events. Once we have constructed the backbone of our GUI, we will need to customise it by integrating it with our internal application class.Try executing this code for yourself. You should be able to see a window with a title, a text label and two buttons – one which prints a message in the console, and one which closes the window. The window should have all the normal properties of any other window you encounter in your window manager – you are probably able to drag it around by the titlebar, resize it by dragging the frame, and maximise, minimise or close it using buttons on the titlebar.

We are using three widgets: Tk is the class which we use to create the *root* window – the main window of our application. Our application should only have one root, but it is possible for us to create other windows which are separate from the main window.

Button and Label should be self-explanatory. Each of them has a parent widget, which we pass in as the first parameter to the constructor – we have put the label and both buttons inside the main window, so they are the main window’s children in the tree. We use the pack method on each widget to position it inside its parent – we will learn about different kinds of layout later.

All three of these widgets can display text (we could also make them display images). The label is a static element – it doesn’t *do* anything by default; it just displays something. Buttons, however, are designed to cause something to happen when they are clicked. We have used the command keyword parameter when constructing each button to specify the function which should handle each button’s click events – both of these functions are object methods.

We didn’t have to write any code to make the buttons fire click events or to bind the methods to them explicitly. That functionality is already built into the button objects – we only had to provide the handlers. We also didn’t have to write our own function for closing the window, because there is already one defined as a method on the window object. We did, however, write our own method for printing a message to the console.

There are many ways in which we could organise our application class. In this example, our class doesn’t inherit from any tkinter objects – we use *composition* to associate our tree of widgets with our class. We could also use *inheritance* to extend one of the widgets in the tree with our custom functions.

root.mainloop() is a method on the main window which we execute when we want to run our application. This method will loop forever, waiting for events from the user, until the user exits the program – either by closing the window, or by terminating the program with a keyboard interrupt in the console.

**Widget classes-**

There are many different widget classes built into tkinter – they should be familiar to you from other GUIs:

* A Frame is a container widget which is placed inside a window, which can have its own border and background – it is used to group related widgets together in an application’s layout.
* Toplevel is a container widget which is displayed as a separate window.
* Canvas is a widget for drawing graphics. In advanced usage, it can also be used to create custom widgets – because we can draw anything we like inside it, and make it interactive.
* Text displays formatted text, which can be editable and can have embedded images.
* A Button usually maps directly onto a user action – when the user clicks on a button, something should happen.
* A Label is a simple widget which displays a short piece of text or an image, but usually isn’t interactive.
* A Message is similar to a Label, but is designed for longer bodies of text which need to be wrapped.
* A Scrollbar allows the user to scroll through content which is too large to be visible all at once.
* Checkbutton, Radiobutton, Listbox, Entry and Scale are different kinds of input widgets – they allow the user to enter information into the program.
* Menu and Menubutton are used to create pull-down menus

1. **Explain how to create the following:**
2. **root window**

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. Here, we show how simple it is to create a very basic window in just 8 lines. We get a window that we can resize, minimize, maximize, and close! The tkinter module's purpose is to generate GUIs. Python is not very popularly used for this purpose, but it is more than capable of doing it.

Let's walk through each step to making a tkinter window:

Simple enough, just import everything from tkinter.

from tkinter import \*

Here, we are creating our class, Window, and inheriting from the Frame class. Frame is a class from the tkinter module. (see Lib/tkinter/\_\_init\_\_)

Then we define the settings upon initialization. This is the master widget.

class Window(Frame):

def \_\_init\_\_(self, master=None):

Frame.\_\_init\_\_(self, master)

self.master = master

The above is really all we need to do to get a window instance started.

Root window created. Here, that would be the only window, but you can later have windows within windows.

root = Tk()

Then we actually create the instance.

app = Window(root)

Finally, show it and begin the mainloop.

root.mainloop()

1. **set window title**
2. Sample Solution:
3. **Python** Code: import **tkinter** as tk # **Create** instance parent = tk.Tk() # Add a **title** parent.**title**("-Welcome to **Python tkinter** Basic exercises-") # Start GUI parent.mainloop()
4. Flowchart:
5. **Python** Code Editor:

import tkinter as tk

# Create instance

parent = tk.Tk()

# Add a title

parent.title("-Welcome to Python tkinter Basic exercises-")

# Start GUI

parent.mainloop()

1. **set window size**

In Tkinter, **minsize()** method is used to set the minimum size of the [Tkinter](https://www.geeksforgeeks.org/python-gui-tkinter/) window. Using this method user can set window’s initialized size to its minimum size, and still be able to maximize and scale the window larger.

**Syntax:**

master.minsize(height, width)

1. **Explain Canvas and Frames in Tkinter and its applications**

The Canvas widget supplies graphics facilities for Tkinter. Among these graphical objects are lines, circles, images, and even other widgets. With this widget it's possible to draw graphs and plots, create graphics editors, and implement various kinds of custom widgets.  
  
We demonstrate in our first example, how to draw a line.  
The method create\_line(coords, options) is used to draw a straight line. The coordinates "coords" are given as four integer numbers: x1, y1, x2, y2 This means that the line goes from the point (x1, y1) to the point (x2, y2). After these coordinates follows a comma separated list of additional parameters, which may be empty. We set, for example, the colour of the line to the special green of our website: fill="#476042"  
  
We kept the first example intentionally very simple. We create a canvas and draw a straight horizontal line into this canvas. This line vertically cuts the canvas into two areas.  
  
The casting to an integer value in the assignment "y = int(canvas\_height / 2)" is superfluous, because create\_line can work with float values as well. They are automatically turned into integer values.

**CODE:**

from tkinter import \*

class MyButton:

def \_\_init\_\_(self,root):

self.f=Frame(root,height=400,width=500)

self.f.propagate(0)

self.f.pack()

self.b1=Button(self.f, text='Red',width=15,height=2,command=lambda: self.buttonClick(1))

self.b2=Button(self.f, text='Blue',width=15,height=2,command=lambda: self.buttonClick(2))

self.b3=Button(self.f, text='Green',width=15,height=2,command=lambda: self.buttonClick(3))

self.b1.pack()

self.b2.pack()

self.b3.pack()

def buttonClick(self,num):

if num==1:

self.f["bg"]='red'

if num==2:

self.f["bg"]='blue'

if num==3:

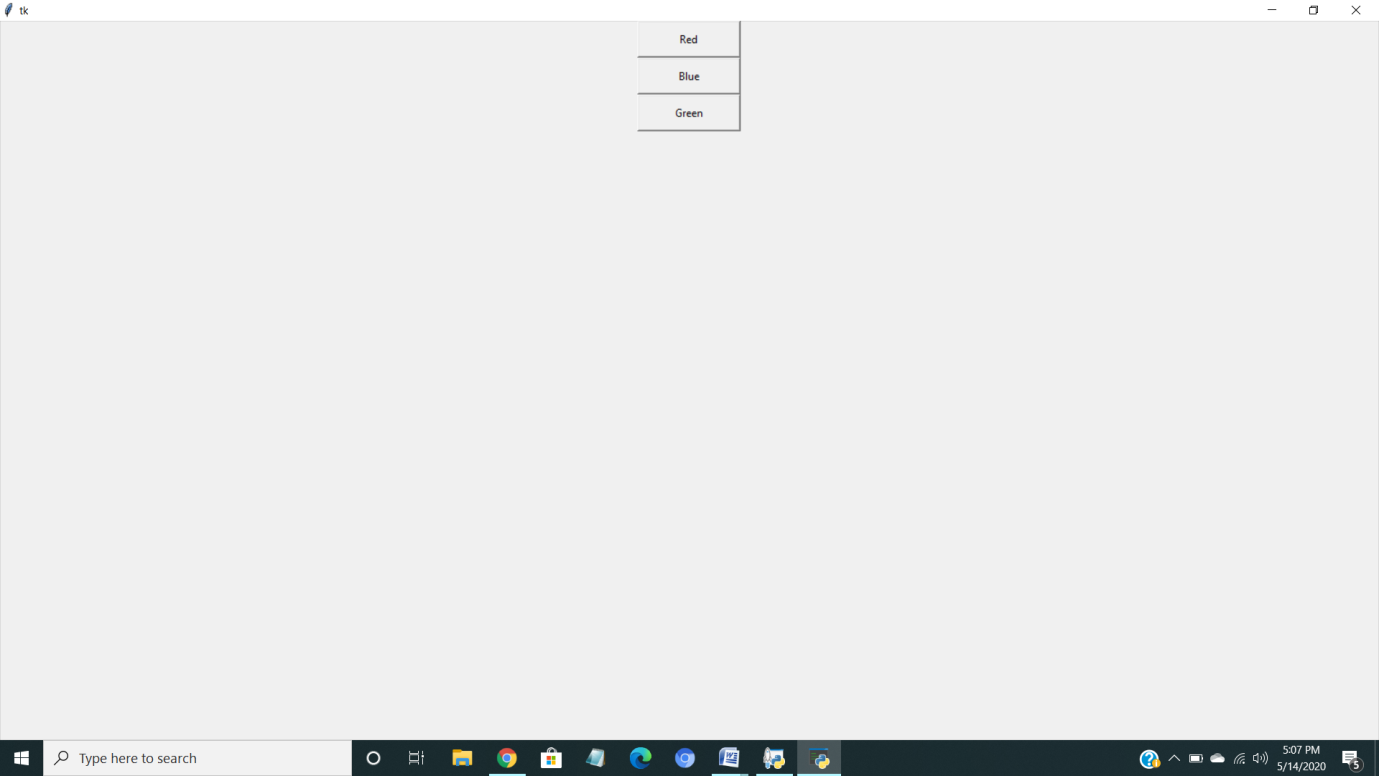
self.f["bg"]='green'

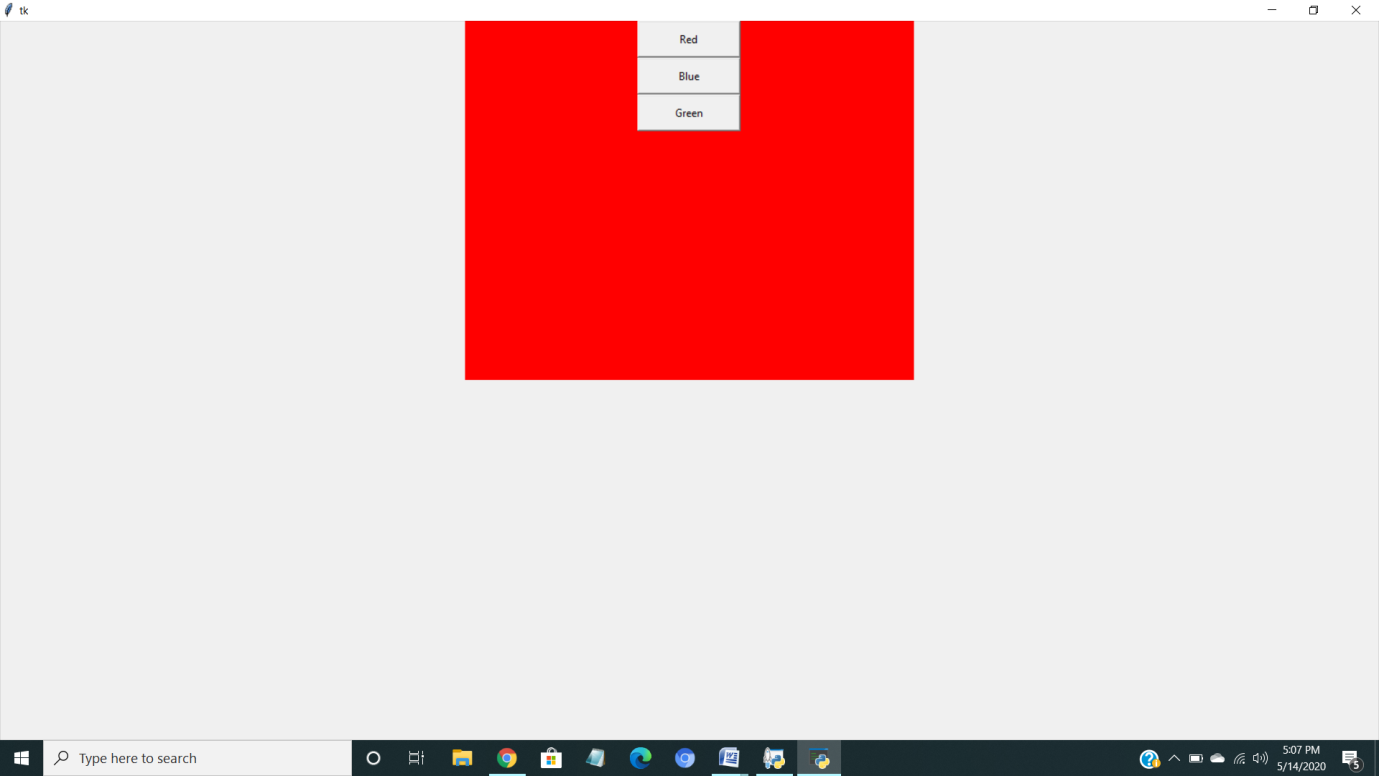
root=Tk()

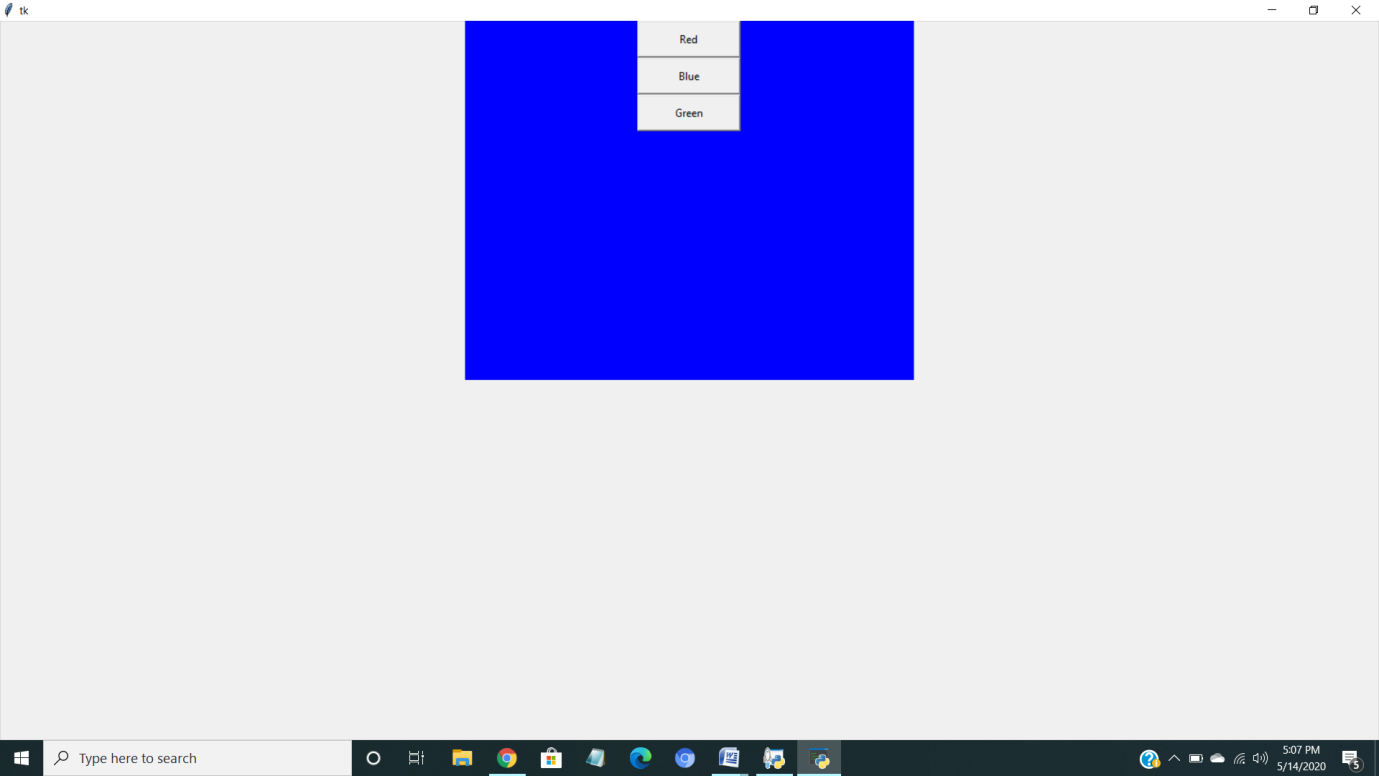
mb=MyButton(root)

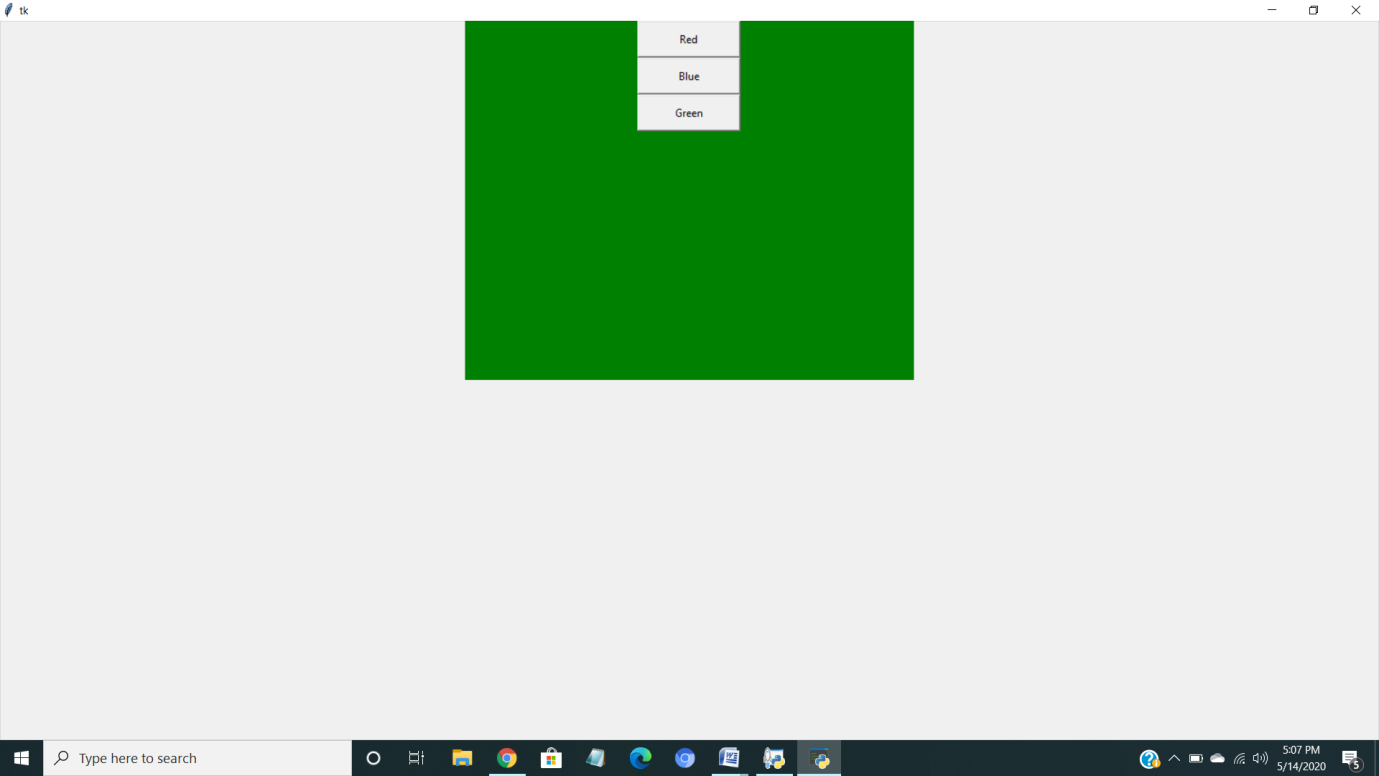
root.mainloop()

**OUTPUT:**

****

****

****

****

**CONCLUSION:** Thus we have Successfully studied and implementedA python program to create three push buttons and change the background of the frame according to the button clicked by the user.