**A Project Report**

**On**

**Python 3.6 With GUI Based Development**

**SUBMITTED TO**

**Bhagwan Mahavir College of Engineering**

**and management Sonipat**

****

**In partial fulfillments of the requirements for the award in the degree of**

**B.Tech Information Technology & Engineering**

**By**

**Himanshu**

**Under the guidance**

**of**

**Bhagwan Mahavir College of Engineering and Management**

**NOVEMBER 2019**

**DECLARATION**

I, **Himanshu Dixit***,* student of B.Tech IT 5th Semester, hereby declare that the project titled “**Python Management GUI BASED**”which is submitted by me to the Department of Information Technology and Engineering, BMCEM Jagdishpur, Sonipat , Haryana in partial fulfilment of requirement for the award of the degree of B.Tech in Information Technology and Engineering, has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition.

The Author attests that permission has been obtained for the use of any copy righted material appearing in the project report other than excerpt s requiring only proper acknowledgement in scholarly writing and all such use is acknowledged.

July 15th, 2019 Himanshu Dixit

**CERTIFICATE**

On the basis of declaration submitted by **Himanshu Dixit***,* student of B.Tech IT 5th Semester, I hereby certify that the project titled “**Python Development GUI BASED**” which is submitted to, Department of Information Technology and Engineering, BMCEM Jagdishpur, Sonipat , Haryana in partial fulfilment of the requirement for the award of the degree of B.Tech in Computer Science and Engineering , is an original contribution with existing knowledge and faithful record of work carried out by him/them under my guidance and supervision.

To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Date (Name and Signature of Guide)

Bhagwan Mahavir College of Engineering and Management

Jagdishpur, Sonipat

**ACKNOWLEDGEMENT**

I have taken efforts in this project. However, it would not have been possible without the kind support and help of **“”**. I would like to extend my sincere thanks to him.

I am highly indebted to “” for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to industry persons for giving me such attention and time.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

I also thanks my team member for giving me support.

**Table of Contents**

***1. Introduction.***

***2. PYHTON AND ITS APPLICATION.***

***3. Making a Python GUI with Tkinter.***

***4. Introduction to GUI programming with tkinter.***

***5. tkinter BASICS.***

***6. Visual Studio, Git Bash and its features.***

***7. Implementation.***

***8. Results.***

***9. Conclusion.***

***10. References.***

**1. INTRODUCTION:**

With the start of every school year comes a number of undergraduates interested in working in our physics lab. In the past, new students had to learn how to interact with our MySQL databases before they could begin any data analysis. This year, I decided to write a Python GUI that will make accessing and cleaning our data easier for new students. The GUI will have a number of predefined query fields which the students can use to obtain data before saving that data to a local csv file. This approach will also reduce the number of students directly interacting with our MySQL database, and thereby reduce the risk of unintentional alterations to the data.

**2.Python: -**

**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language), [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python has a design philosophy that emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability), notably using [significant whitespace](https://en.wikipedia.org/wiki/Significant_whitespace). It provides constructs that enable clear programming on both small and large scales.In July 2018, Van Rossum stepped down as the leader in the language community.

Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural](https://en.wikipedia.org/wiki/Procedural_programming), and has a large and comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

Python interpreters are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). [CPython](https://en.wikipedia.org/wiki/CPython" \o "CPython), the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/Open-source_software) software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation).

**Applications of Python**

* GUI-Based Desktop **Applications**:
* Web Frameworks and Web **Applications**:
* Enterprise and Business **Applications**:
* Operating Systems:
* Language Development:
* Prototyping:

# 3.Making a Python GUI with Tkinter

Python’s Tkinter library provides an easy to use interface to the Tk GUI toolkit. We’ll be using it to build our customized GUI. To install Tkinter, type pip installs tkinter at the command line.

Before we include any functionality in our GUI, let’s build the basic layout. First, we’ll need to create a Tk GUI window in Python. We’ll pass this window to a customized class object (which I’m naming App) that will set up the layout of our GUI in its \_\_init\_\_ method. Note that in this post any code that belongs to the \_\_init\_\_ method will be indicated by two indents, just as it would appear in our real Python code. The mainloop method will open the window and display our GUI.

# 4.Introduction to GUI programming

# with tkinter

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.

**Note**

in some Linux distributions, like Ubuntu and Debian, the tkinter module is packaged separately to the rest of Python, and must be installed separately.

## Event-driven programming

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.

**5. tkinter BASICS: -**

**Tkinter** is a [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) [binding](https://en.wikipedia.org/wiki/Language_binding) to the [Tk](https://en.wikipedia.org/wiki/Tk_(software)) [GUI](https://en.wikipedia.org/wiki/Graphical_user_interface) toolkit. It is the standard Python interface to the Tk GUI toolkit,[[1]](https://en.wikipedia.org/wiki/Tkinter#cite_note-1) and is Python's [*de facto* standard](https://en.wikipedia.org/wiki/De_facto_standard) GUI.[[2]](https://en.wikipedia.org/wiki/Tkinter#cite_note-2) Tkinter is included with standard [Linux](https://en.wikipedia.org/wiki/Linux), [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) and [Mac OS X](https://en.wikipedia.org/wiki/Mac_OS_X) installs of Python.

The name *Tkinter* comes from *Tk interface*. Tkinter was written by Fredrik Lundh.

Tkinter is [free software](https://en.wikipedia.org/wiki/Free_software) released under a [Python license](https://en.wikipedia.org/wiki/Python_license).

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.

Our first GUI will be a window with a label and two buttons:

from tkinter import Tk, Label, Button

class MyFirstGUI:

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

self.master = master

master.title("A simple GUI")

self.label = Label(master, text="This is our first GUI!")

self.label.pack()

self.greet\_button = Button(master, text="Greet", command=self.greet)

self.greet\_button.pack()

self.close\_button = Button(master, text="Close", command=master.quit)

self.close\_button.pack()

def greet(self):

print("Greetings!")

root = Tk()

my\_gui = MyFirstGUI(root)

root.mainloop()

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.

**Note**

The window manager is the part of your operating system which handles windows. All the widgets inside a window, like buttons and other controls, may look different in every GUI toolkit, but the way that the window frames and title bars look and behave is determined by your window manager and should always stay the same.

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.

## Layout options

The GUI in the previous example has a relatively simple layout: we arranged the three widgets in a single column inside the window. To do this, we used the pack method, which is one of the three different geometry managers available in tkinter. We have to use one of the available geometry managers to specify a position for each of our widgets, otherwise the widget will not appear in our window.

By default, pack arranges widgets vertically inside their parent container, from the top down, but we can change the alignment to the bottom, left or right by using the optional side parameter. We can mix different alignments in the same container, but this may not work very well for complex layouts. It should work reasonably well in our simple case, however:

from tkinter import LEFT, RIGHT

# (...)

self.label.pack()

self.greet\_button.pack(side=LEFT)

self.close\_button.pack(side=RIGHT)

We can create quite complicated layouts with pack by grouping widgets together in frames and aligning the groups to our liking – but we can avoid a lot of this complexity by using the gridmethod instead. It allows us to position widgets in a more flexible way, using a grid layout. This is the geometry manager recommended for complex interfaces:

from tkinter import W

# (...)

self.label.grid(columnspan=2, sticky=W)

self.greet\_button.grid(row=1)

self.close\_button.grid(row=1, column=1)

We place each widget in a cell inside a table by specifying a row and a column – the default row is the first available empty row, and the default column is 0.

If a widget is smaller than its cell, we can customise how it is aligned using the sticky parameter – the possible values are the cardinal directions (N, S, E and W), which we can combine through addition. By default, the widget is centered both vertically and horizontally, but we can make it stickto a particular side by including it in the sticky parameter. For example, sticky=W will cause the widget to be left-aligned horizontally, and sticky=W+E will cause it to be stretched to fill the whole cell horizontally. We can also specify corners using NE, SW, etc..

To make a widget span multiple columns or rows, we can use the columnspan and rowspan options – in the example above, we have made the label span two columns so that it takes up the same space horizontally as both of the buttons underneath it.

**Note**

Never use both pack and grid inside the same window. The algorithms which they use to calculate widget positions are not compatible with each other, and your program will hang forever as tkinter tries unsuccessfully to create a widget layout which satisfies both of them.

The third geometry manager is place, which allows us to provide explicit sizes and positions for widgets. It is seldom a good idea to use this method for ordinary GUIs – it’s far too inflexible and time consuming to specify an absolute position for every element. There are some specialised cases, however, in which it can come in useful.

## Custom events

So far we have only bound event handlers to events which are defined in tkinter by default – the Button class already knows about button clicks, since clicking is an expected part of normal button behaviour. We are not restricted to these particular events, however – we can make widgets listen for other events and bind handlers to them, using the bind method which we can find on every widget class.

Events are uniquely identified by a sequence name in string format – the format is described by a mini-language which is not specific to Python. Here are a few examples of common events:

* "<Button-1>", "<Button-2>" and "<Button-3>" are events which signal that a particular mouse button has been pressed while the mouse cursor is positioned over the widget in question.Button 1 is the left mouse button, Button 3 is the right, and Button 2 the middle button – but remember that not all mice have a middle button.
* "<ButtonRelease-1>" indicates that the left button has been released.
* "<B1-Motion>" indicates that the mouse was moved while the left button was pressed (we can use B2 or B3 for the other buttons).
* "<Enter>" and "<Leave>" tell us that the mouse curson has entered or left the widget.
* "<Key>" means that any key on the keyboard was pressed. We can also listen for specific key presses, for example "<Return>" (the enter key), or combinations like "<Shift-Up>" (shift-up-arrow). Key presses of most printable characters are expressed as the bare characters, without brackets – for example, the letter a is just "a".
* "<Configure>" means that the widget has changed size.

We can now extend our simple example to make the label interactive – let us make the label text cycle through a sequence of messages whenever it is clicked:

from tkinter import Tk, Label, Button, StringVar

class MyFirstGUI:

LABEL\_TEXT = [

"This is our first GUI!",

"Actually, this is our second GUI.",

"We made it more interesting...",

"...by making this label interactive.",

"Go on, click on it again.",

]

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

self.master = master

master.title("A simple GUI")

self.label\_index = 0

self.label\_text = StringVar()

self.label\_text.set(self.LABEL\_TEXT[self.label\_index])

self.label = Label(master, textvariable=self.label\_text)

self.label.bind("<Button-1>", self.cycle\_label\_text)

self.label.pack()

self.greet\_button = Button(master, text="Greet", command=self.greet)

self.greet\_button.pack()

self.close\_button = Button(master, text="Close", command=master.quit)

self.close\_button.pack()

def greet(self):

print("Greetings!")

def cycle\_label\_text(self, event):

self.label\_index += 1

self.label\_index %= len(self.LABEL\_TEXT) # wrap around

self.label\_text.set(self.LABEL\_TEXT[self.label\_index])

root = Tk()

my\_gui = MyFirstGUI(root)

root.mainloop()

Updating a label’s text is a little convoluted – we can’t simply update the text using a normal Python string. Instead, we have to provide the label with a special tkinter string variable object, and set a new value on the object whenever we want the text in the label to change.

We have defined a handler which cycles to the next text string in the sequence, and used the bindmethod of the label to bind our new handler to left clicks on the label. It is important to note that this handler takes an additional parameter – an event object, which contains some information about the event. We could use the same handler for many different events (for example, a few similar events which happen on different widgets), and use this parameter to distinguish between them. Since in this case we are only using our handler for one kind of event, we will simply ignore the event parameter.

**8. Visual Studio: -**

Microsoft Visual Studio is an [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) from [Microsoft](https://en.wikipedia.org/wiki/Microsoft). It is used to develop [computer programs](https://en.wikipedia.org/wiki/Computer_program), as well as [websites](https://en.wikipedia.org/wiki/Web_site), [web apps](https://en.wikipedia.org/wiki/Web_app), [web services](https://en.wikipedia.org/wiki/Web_service) and [mobile apps](https://en.wikipedia.org/wiki/Mobile_app). Visual Studio uses Microsoft software development platforms such as [Windows API](https://en.wikipedia.org/wiki/Windows_API), [Windows Forms](https://en.wikipedia.org/wiki/Windows_Forms), [Windows Presentation Foundation](https://en.wikipedia.org/wiki/Windows_Presentation_Foundation), [Windows Store](https://en.wikipedia.org/wiki/Windows_Store) and [Microsoft Silverlight](https://en.wikipedia.org/wiki/Microsoft_Silverlight). It can produce both [native code](https://en.wikipedia.org/wiki/Machine_code) and [managed code](https://en.wikipedia.org/wiki/Managed_code).

Visual Studio includes a [code editor](https://en.wikipedia.org/wiki/Code_editor) supporting [IntelliSense](https://en.wikipedia.org/wiki/IntelliSense) (the [code completion](https://en.wikipedia.org/wiki/Code_completion) component) as well as [code refactoring](https://en.wikipedia.org/wiki/Code_refactoring). [The integrated debugger](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger) works both as a source-level debugger and a machine-level debugger. Other built-in tools include a [code profiler](https://en.wikipedia.org/wiki/Profiling_(computer_programming)), designer for building [GUI](https://en.wikipedia.org/wiki/GUI) applications, [web designer](https://en.wikipedia.org/wiki/Web_designer), [class](https://en.wikipedia.org/wiki/Class_(computing)) designer, and [database schema](https://en.wikipedia.org/wiki/Database_schema) designer.

## **Features: -**

* Code editor. Like any other IDE, it includes a code editor that supports syntax highlighting and code completion using IntelliSense for variables, functions, methods, loops, and LINQ queries. ...
* Debugger.
* Designer.
* Other tools.
* Extensibility.
* Previous products.
* Community.
* Professional.

**8.1 Git Bash: -**

Git Bash is an application for Microsoft Windows environments which provides an emulation layer for a Git command line experience. Bash is an acronym for Bourne Again Shell. A shell is a terminal application used to interface with an operating system through written commands. Bash is a popular default shell on Linux and macOS. Git Bash is a package that installs Bash, some common bash utilities, and Git on a Windows operating system.

## **How to install**

Git Bash comes included as part of the [Git For Windows](https://gitforwindows.org/) package. Download and install Git For Windows like other Windows applications. Once downloaded find the included .exe file and open to execute Git Bash.

## **How to Integrate Git Bash with Visual Studio**

// Git Bash

"terminal.integrated.shell.windows": "C:\\Program Files\\Git\\bin\\bash.exe"

Writing this into user settings a json file in visual studio and adding this as alternative window.

**9. IMPLEMENTATION: -**

**import tkinter as tk**

**from tkinter import ttk**

**from tkinter import colorchooser,font,messagebox,filedialog**

**import os**

**root = tk.Tk()**

**root.geometry('1200x800')**

**root.title("Editor")**

**root.wm\_iconbitmap('icon.ico')**

**main\_menu=tk.Menu(root)**

**##################### main menu ###############################**

**# Cascade**

**file=tk.Menu(main\_menu,tearoff=False)**

**edit=tk.Menu(main\_menu,tearoff=False)**

**view=tk.Menu(main\_menu,tearoff=False)**

**color\_theme=tk.Menu(main\_menu,tearoff=False)**

**main\_menu.add\_cascade(label="File",menu=file)**

**main\_menu.add\_cascade(label="Edit",menu=edit)**

**main\_menu.add\_cascade(label="View",menu=view)**

**main\_menu.add\_cascade(label="Color theme",menu=color\_theme)**

**#file icons**

**new\_icon= tk.PhotoImage(file='icons2/new.png')**

**open\_icon= tk.PhotoImage(file='icons2/open.png')**

**save\_icon= tk.PhotoImage(file='icons2/save.png')**

**save\_as\_icon= tk.PhotoImage(file='icons2/save\_as.png')**

**exit\_icon= tk.PhotoImage(file='icons2/exit.png')**

**## Edit icon**

**copy\_icon=tk.PhotoImage(file='icons2/copy.png')**

**paste\_icon=tk.PhotoImage(file='icons2/paste.png')**

**cut\_icon=tk.PhotoImage(file='icons2/cut.png')**

**clear\_all\_icon=tk.PhotoImage(file='icons2/clear\_all.png')**

**find\_icon=tk.PhotoImage(file='icons2/find.png')**

**#View icon**

**tool\_bar\_icon=tk.PhotoImage(file='icons2/tool\_bar.png')**

**status\_bar\_icon=tk.PhotoImage(file='icons2/status\_bar.png')**

**## Color Theme icon**

**light\_default\_icon=tk.PhotoImage(file='icons2/light\_default.png')**

**light\_plus\_icon=tk.PhotoImage(file='icons2/light\_plus.png')**

**dark\_icon=tk.PhotoImage(file='icons2/dark.png')**

**red\_icon=tk.PhotoImage(file='icons2/red.png')**

**monokai\_icon=tk.PhotoImage(file='icons2/monokai.png')**

**night\_plus\_icon=tk.PhotoImage(file='icons2/night\_blue.png')**

**color\_icons=(light\_default\_icon,light\_plus\_icon,dark\_icon,red\_icon,monokai\_icon,night\_plus\_icon)**

**theme\_choice = tk.StringVar()**

**color\_dict={'Light Default':('#000000','#ffffff'),**

**'Light Plus':('#474747','#e0e0e0'),'Dark':('#c4c4c4','#2d2d2d'),**

**'Red':('#2d2d2d','#ffe8e8'),'Monokai':('#d3b774','#474747'),**

**'Night Plus':('#ededed','#6b9dc2')}**

**##################### Toolbar ###############################**

**#font box**

**tool\_bar=ttk.Label(root)**

**tool\_bar.pack(side=tk.TOP,fill=tk.X)**

**font\_tuple=tk.font.families()**

**font\_family=tk.StringVar()**

**font\_box=ttk.Combobox(tool\_bar,width=30,textvariable=font\_family,state='readonly')**

**font\_box['values']=font\_tuple**

**font\_box.current(11)**

**font\_box.grid(row=0,column=0,padx=5)**

**# size box**

**size\_var=tk.IntVar()**

**font\_size=ttk.Combobox(tool\_bar,width=15,textvariable=size\_var,state='readonly')**

**font\_size['values']=tuple(range(8,80,2))**

**font\_size.current(2)**

**font\_size.grid(row=0,column=1,padx=5)**

**## Bold button**

**bold\_icon=tk.PhotoImage(file='icons2/bold.png')**

**bold\_btn=ttk.Button(tool\_bar,image=bold\_icon)**

**bold\_btn.grid(row=0,column=2,padx=5)**

**# Italic Button**

**italic\_icon=tk.PhotoImage(file='icons2/italic.png')**

**italic\_btn=ttk.Button(tool\_bar,image=italic\_icon)**

**italic\_btn.grid(row=0,column=3,padx=5)**

**# Underline Button**

**underline\_icon=tk.PhotoImage(file='icons2/underline.png')**

**underline\_btn=ttk.Button(tool\_bar,image=underline\_icon)**

**underline\_btn.grid(row=0,column=4,padx=5)**

**# Font Color Button**

**font\_color\_icon=tk.PhotoImage(file='icons2/font\_color.png')**

**font\_color\_btn=ttk.Button(tool\_bar,image=font\_color\_icon)**

**font\_color\_btn.grid(row=0,column=5,padx=5)**

**## Align Left Button**

**align\_left\_icon=tk.PhotoImage(file='icons2/al.png')**

**align\_left\_btn=ttk.Button(tool\_bar,image=align\_left\_icon)**

**align\_left\_btn.grid(row=0,column=6,padx=5)**

**## Ailgn Center Button**

**align\_center\_icon=tk.PhotoImage(file='icons2/ac.png')**

**align\_center\_btn=ttk.Button(tool\_bar,image=align\_center\_icon)**

**align\_center\_btn.grid(row=0,column=7,padx=5)**

**## Align Right Button**

**align\_right\_icon=tk.PhotoImage(file='icons2/ar.png')**

**align\_right\_btn=ttk.Button(tool\_bar,image=align\_right\_icon)**

**align\_right\_btn.grid(row=0,column=8,padx=5)**

**##################### Text Editor ###############################**

**text\_editor=tk.Text(root)**

**text\_editor.config(wrap='word',relief=tk.FLAT)**

**scroll\_bar=tk.Scrollbar(root)**

**text\_editor.focus\_set()**

**scroll\_bar.pack(side=tk.RIGHT,fill=tk.Y)**

**text\_editor.pack(fill=tk.BOTH,expand=True)**

**scroll\_bar.config(command=text\_editor.yview)**

**text\_editor.config(yscrollcommand=scroll\_bar.set)**

**# Font functionality**

**current\_font\_family='Arial'**

**current\_font\_size=12**

**def change\_font(root):**

**global current\_font\_family**

**current\_font\_family=font\_family.get()**

**text\_editor.configure(font=(current\_font\_family,current\_font\_size))**

**def change\_font\_size(root):**

**global current\_font\_size**

**current\_font\_size=size\_var.get()**

**text\_editor.configure(font=(current\_font\_family,current\_font\_size))**

**font\_size.bind("<<ComboboxSelected>>",change\_font\_size)**

**font\_box.bind("<<ComboboxSelected>>",change\_font)**

**text\_editor.configure(font=('Arial',12))**

**# Buttons Functionality**

**# Bold**

**def bold():**

**b=tk.font.Font(font=text\_editor['font']).actual()**

**if b['weight']=='normal':**

**text\_editor.configure(font=(current\_font\_family,current\_font\_size,'bold'))**

**if b['weight']=='bold':**

**text\_editor.configure(font=(current\_font\_family,current\_font\_size,'normal'))**

**bold\_btn.configure(command=bold)**

**#Italic**

**def italic():**

**b=tk.font.Font(font=text\_editor['font']).actual()**

**if b['slant']=='roman':**

**text\_editor.configure(font=(current\_font\_family,current\_font\_size,'italic'))**

**if b['slant']=='italic':**

**text\_editor.configure(font=(current\_font\_family,current\_font\_size,'roman'))**

**italic\_btn.configure(command=italic)**

**# Under Line**

**def underline():**

**b=tk.font.Font(font=text\_editor['font']).actual()**

**if b['underline']==0:**

**text\_editor.configure(font=(current\_font\_family,current\_font\_size,'underline'))**

**if b['underline']==1:**

**text\_editor.configure(font=(current\_font\_family,current\_font\_size,'normal'))**

**underline\_btn.configure(command=underline)**

**# Font Color**

**def change\_font\_color():**

**color\_var=tk.colorchooser.askcolor()**

**text\_editor.configure(fg=color\_var[1])**

**font\_color\_btn.configure(command=change\_font\_color)**

**## Align Left**

**def align\_left():**

**text\_content=text\_editor.get(1.0,'end')**

**text\_editor.tag\_config('left',justify=tk.LEFT)**

**text\_editor.delete(1.0,tk.END)**

**text\_editor.insert(tk.INSERT,text\_content,'left')**

**align\_left\_btn.configure(command=align\_left)**

**## Align Center**

**def align\_center():**

**text\_content=text\_editor.get(1.0,'end')**

**text\_editor.tag\_config('center',justify=tk.CENTER)**

**text\_editor.delete(1.0,tk.END)**

**text\_editor.insert(tk.INSERT,text\_content,'center')**

**align\_center\_btn.configure(command=align\_center)**

**## Align Right**

**def align\_right():**

**text\_content=text\_editor.get(1.0,'end')**

**text\_editor.tag\_config('right',justify=tk.RIGHT)**

**text\_editor.delete(1.0,tk.END)**

**text\_editor.insert(tk.INSERT,text\_content,'right')**

**align\_right\_btn.configure(command=align\_right)**

**##################### Status Bar ###############################**

**status\_bar=ttk.Label(root,text='Status Bar')**

**status\_bar.pack(side=tk.BOTTOM)**

**text\_changed= False**

**def changed(event=None):**

**global text\_changed**

**if text\_editor.edit\_modified():**

**text\_changed=True**

**words=len(text\_editor.get(1.0,'end-1c').split())**

**characters=len(text\_editor.get(1.0,'end-1c'))**

**status\_bar.config(text=f' Characters : {characters} Words : {words}')**

**text\_editor.edit\_modified(False)**

**text\_editor.bind('<<Modified>>',changed)**

**##################### Addition Functionality ###############################**

**# Global Variable**

**url=''**

**# new**

**def new\_file(event=None):**

**global url**

**url=''**

**text\_editor.delete(1.0,tk.END)**

**# open**

**def open\_file(event=None):**

**global url**

**url=filedialog.askopenfilename(initialdir=os.getcwd(),title='Select File',filetypes=(('Text File','\*.txt'),('Pdf File','\*.pdf'),('All Files','\*.\*')))**

**try:**

**with open(url,'r') as fr:**

**text\_editor.delete(1.0,tk.END)**

**text\_editor.insert(1.0,fr.read())**

**except FileNotFoundError:**

**return**

**except:**

**return**

**root.title(os.path.basename(url))**

**# Save**

**def save\_file(event=None):**

**global url**

**try:**

**if url:**

**content=str(text\_editor.get(1.0,tk.END))**

**with open(url,'w',encoding='utf-8') as fw:**

**fw.write(content)**

**else:**

**url=filedialog.asksaveasfile(mode='w',defaultextension='.txt',filetypes=(('Text File','\*.txt'),('Pdf File','\*.pdf'),('All Files','\*.\*')))**

**content2=text\_editor.get(1.0,tk.END)**

**url.write(content2)**

**url.close()**

**except:**

**return**

**# Save As**

**def save\_as(event=None):**

**global url**

**try:**

**content=str(text\_editor.get(1.0,tk.END))**

**url=filedialog.asksaveasfile(mode='w',defaultextension='.txt',filetypes=(('Text File','\*.txt'),('Pdf File','\*.pdf'),('All Files','\*.\*')))**

**url.write(content)**

**url.close()**

**except:**

**return**

**## Exit**

**def exit\_file(event=None):**

**global url,text\_changed**

**try:**

**if text\_changed:**

**mbox = messagebox.askyesnocancel('Warning','Do you want to save the file ?')**

**if mbox is True:**

**if url:**

**content=text\_editor.get(1.0,tk.END)**

**with open(url,'w','utf-8') as fw:**

**fw.write(content)**

**root.destroy()**

**else:**

**content2=str(text\_editor.get(1.0,tk.END))**

**url=filedialog.asksaveasfile(mode='w',defaultextension='.txt',filetypes=(('Text File','\*.txt'),('Pdf File','\*.pdf'),('All Files','\*.\*')))**

**url.write(content2)**

**url.close()**

**root.destroy()**

**elif mbox is False:**

**root.destroy()**

**else:**

**root.destroy()**

**except:**

**return**

**#File Commands**

**file.add\_command(label='New',image=new\_icon,compound=tk.LEFT,accelerator = 'Ctrl+N',command=new\_file)**

**file.add\_command(label='Open',image=open\_icon,compound=tk.LEFT,accelerator = 'Ctrl+O',command=open\_file)**

**file.add\_command(label='Save',image=save\_icon,compound=tk.LEFT,accelerator = 'Ctrl+S',command=save\_file)**

**file.add\_command(label='Save As',image=save\_as\_icon,compound=tk.LEFT,accelerator = 'Ctrl+ALT+S',command=save\_as)**

**file.add\_command(label='Exit',image=exit\_icon,compound=tk.LEFT,accelerator = 'Ctrl+Q',command=exit\_file)**

**# Find**

**def find\_func(event=None):**

**def find():**

**word=find\_input.get()**

**text\_editor.tag\_remove('match','1.0',tk.END)**

**matches=0**

**if word:**

**start\_pos='1.0'**

**while True:**

**start\_pos=text\_editor.search(word,start\_pos,stopindex=tk.END)**

**if not start\_pos:**

**break**

**end\_pos=f'{start\_pos}+{len(word)}c'**

**text\_editor.tag\_add('match',start\_pos,end\_pos)**

**matches+=1**

**start\_pos=end\_pos**

**text\_editor.tag\_config('match',foreground='red',background='yellow')**

**# Replace**

**def replace():**

**word=find\_input.get()**

**replace\_text=replace\_input.get()**

**content=text\_editor.get(1.0,tk.END)**

**new\_content=content.replace(word,replace\_text)**

**text\_editor.delete(1.0,tk.END)**

**text\_editor.insert(1.0,new\_content)**

**find\_dialog=tk.Toplevel()**

**find\_dialog.geometry("450x200+500+200")**

**find\_dialog.title("Find")**

**find\_dialog.resizable(0,0)**

**#frame**

**find\_frame=ttk.LabelFrame(find\_dialog,text='Find/Replace')**

**find\_frame.pack(pady=20)**

**text\_find\_label=ttk.Label(find\_frame,text='Find : ')**

**text\_replace\_label=ttk.Label(find\_frame,text='Replace : ')**

**find\_input=ttk.Entry(find\_frame,width=30)**

**replace\_input=ttk.Entry(find\_frame,width=30)**

**find\_btn=ttk.Button(find\_frame,text='Find',command=find)**

**replace\_btn=ttk.Button(find\_frame,text='Replace',command=replace)**

**text\_find\_label.grid(row=0,column=0,padx=4,pady=4)**

**text\_replace\_label.grid(row=1,column=0,padx=4,pady=4)**

**find\_input.grid(row=0,column=1,padx=4,pady=4)**

**replace\_input.grid(row=1,column=1,padx=4,pady=4)**

**find\_btn.grid(row=2,column=0 , padx=8,pady =4)**

**replace\_btn.grid(row=2,column=1, padx=8,pady =4)**

**find\_dialog.mainloop()**

**# Edit Commands**

**edit.add\_command(label='Copy',image=copy\_icon,compound=tk.LEFT,accelerator='Ctrl+C',command=lambda:text\_editor.event\_generate('<Control c>'))**

**edit.add\_command(label='Paste',image=paste\_icon,compound=tk.LEFT,accelerator='Ctrl+V',command=lambda:text\_editor.event\_generate('<Control v>'))**

**edit.add\_command(label='Cut',image=cut\_icon,compound=tk.LEFT,accelerator='Ctrl+X',command=lambda:text\_editor.event\_generate('<Control x>'))**

**edit.add\_command(label='Clear All',image=clear\_all\_icon,compound=tk.LEFT,accelerator='Ctrl+Alt+X',command=lambda:text\_editor.delete(1.0,tk.END))**

**edit.add\_command(label='Find',image=find\_icon,compound=tk.LEFT,accelerator='Ctrl+F',command=find\_func)**

**#View Commands**

**show\_toolbar=tk.BooleanVar()**

**show\_statusbar=tk.BooleanVar()**

**show\_statusbar.set(True)**

**show\_toolbar.set(True)**

**def hide\_toolbar():**

**global show\_toolbar**

**if show\_toolbar:**

**tool\_bar.pack\_forget()**

**show\_toolbar=False**

**else:**

**text\_editor.pack\_forget()**

**status\_bar.pack\_forget()**

**tool\_bar.pack(side=tk.TOP,fill=tk.X)**

**text\_editor.pack(fill=tk.BOTH,expand=True)**

**status\_bar.pack(side=tk.BOTTOM)**

**show\_toolbar=True**

**def hide\_statusbar():**

**global show\_statusbar**

**if show\_statusbar:**

**status\_bar.pack\_forget()**

**show\_statusbar=False**

**else:**

**status\_bar.pack(side=tk.BOTTOM)**

**show\_statusbar=True**

**view.add\_checkbutton(label="ToolBar",image=tool\_bar\_icon,onvalue=True,variable=show\_toolbar,offvalue=0,compound=tk.LEFT,command=hide\_toolbar)**

**view.add\_checkbutton(label="StatusBar",image=status\_bar\_icon,onvalue=True,variable=show\_statusbar,offvalue=0,compound=tk.LEFT,command=hide\_statusbar)**

**## Color\_theme Commands**

**count=0**

**def change\_theme():**

**chosen\_theme=theme\_choice.get()**

**color\_tuple=color\_dict.get(chosen\_theme)**

**fg\_color=color\_tuple[0]**

**bg\_color=color\_tuple[1]**

**text\_editor.config(background=bg\_color,fg=fg\_color)**

**for i in color\_dict:**

**color\_theme.add\_radiobutton(label=i,image=color\_icons[count],variable=theme\_choice,compound=tk.LEFT,command=change\_theme)**

**count+=1**

**##\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##**

**root.config(menu=main\_menu)**

**## Bind Shortcut Keys**

**root.bind('<Control-n>',new\_file)**

**root.bind('<Control-o>',open\_file)**

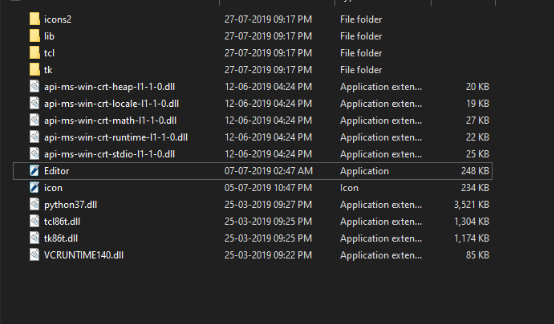
**root.bind('<Control-s>',save\_file)**

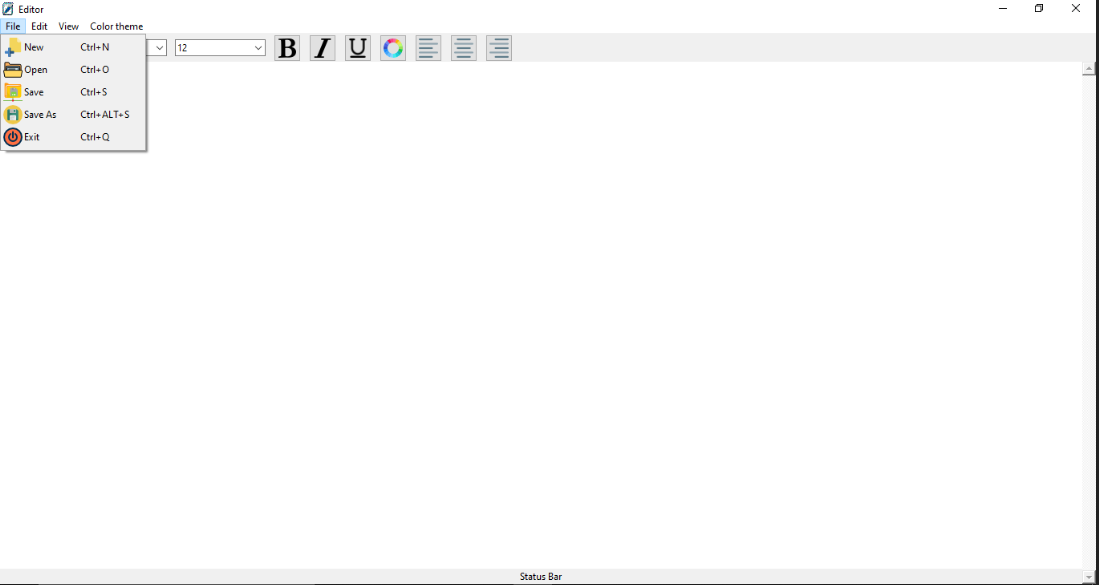
**root.bind('<Control-Alt-s>',save\_as)**

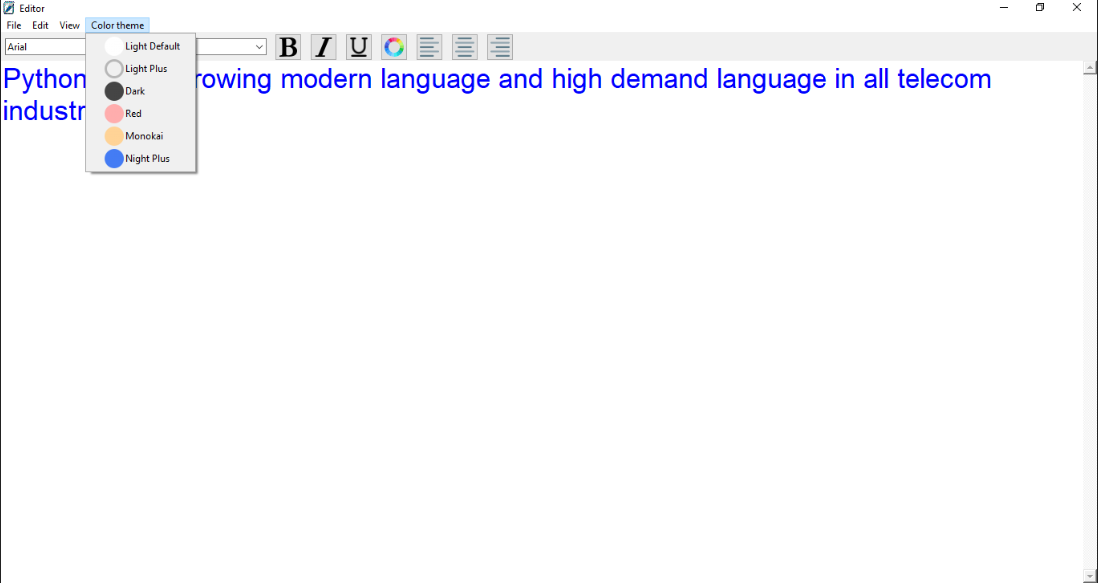
**root.bind('<Control-q>',exit\_file)**

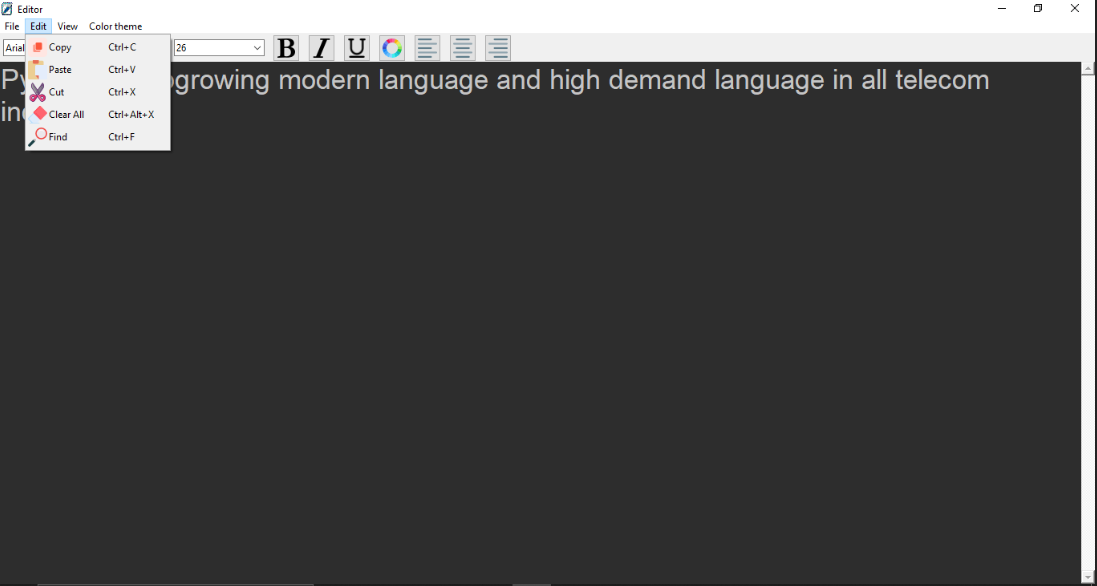
**root.mainloop()**

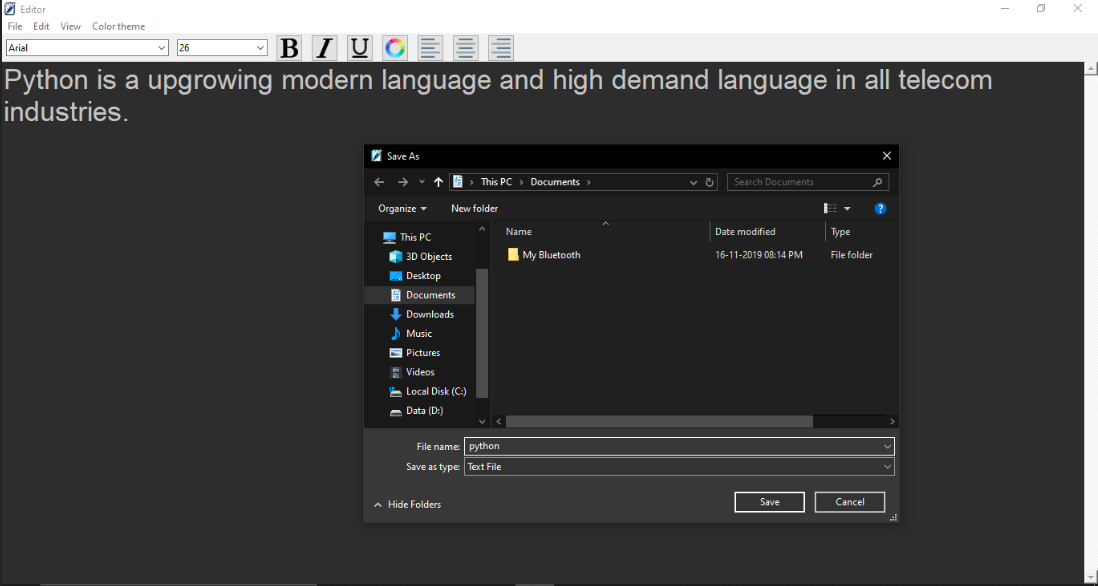
**10.RESULT: -**

****

****

****

****

****

**11. CONCLUSION:**

The entire project has been developed and deployed as per the requirements stated by the user, it is found to be bug free as per the testing standards that are implemented. And by specification-untraced errors concentrated in the coming versions, which are planned to be developed in near future.

Finally, we like to conclude that we put all our efforts throughout the development of our project and tired to fulfill most of the requirements of the user.

**12.REFERENCE:**

**Websites**

* Tutorials Point.
* Google's **Python** Class eBook.
* **Python**.org.
* **Learn Python** the Hard Way eBook.