PySimpleGUI

Python GUIs for Humans

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

PySimpleGui is an open source, cross-platform GUI library for Python. It aims to provide a uniform API for creating desktop GUIs based on Python's Tkinter, PySide and WxPython toolkits.

PySimpleGUI also has a port for Remi which is useful for building GUIs for the web. PySimpleGui lets you build GUIs quicker than by directly using the libraries it uses.

Audience

This tutorial is designed for Python developers who want to learn how to build cross-platform desktop as well as web based GUI designs using PySimpleGui library.

Prerequisites

Before you proceed, make sure that you understand the basics of procedural and object-oriented programming in Python. For understanding the advanced topics such as integration of PySimpleGui with Matplotlib and OpenCV packages, their understanding is essential.

Disclaimer & Copyright

© Copyright 2022 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial i

Audience i

Prerequisites i

Disclaimer & Copyright i

Table of Contents ii

1. PYSIMPLEGUI – INTRODUCTION 1

Python GUIs for Humans 1

Comparison with other GUI Frameworks 1

2. PYSIMPLEGUI – ENVIRONMENT SETUP 3

3. PYSIMPLEGUI – HELLO WORLD 5

First Window using PySimpleGUI 5

Equivalent Tkinter Code 6

PySimpleGUIQt 7

Equivalent PySide2 Code 9

PySimpleGUIWx 10

PySimpleGUIWeb 12

4. PYSIMPLEGUI – POPUP WINDOWS 15

Scrolled Popup 19

Progress Meter 20

Debug Popup 21

tutorialspoint
SIMPLYEASYLEARNING

5. PYSIMPLEGUI – WINDOW CLASS 23

Layout Structure **23**

Persistent Window **25**

Window Methods **26**

Update Window with Key **27**

Borderless Window **31**

Window with Disabled Close **32**

Transparent Window **32**

Multiple Windows **32**

Asynchronous Window **35**

6. PYSIMPLEGUI – ELEMENT CLASS 37

Properties of Element Class **38**

Methods of Element Class **38**

7. PYSIMPLEGUI – EVENTS 39

Window Closed Event **39**

Button Events **41**

Events of Other Elements **42**

8. PYSIMPLEGUI – TEXT ELEMENT 44

9. PYSIMPLEGUI – INPUT ELEMENT 46

Multiline Element **48**

10. PYSIMPLEGUI – BUTTON ELEMENT 51

FileBrowse **52**

tutorialspoint
SIMPLYEASYLEARNING

FilesBrowse **53**

FolderBrowse **55**

FileSaveAs **56**

ColorChooserButton **58**

CalendarButton **60**

Image Button **61**

11. PYSIMPLEGUI – LISTBOX ELEMENT 63

12. PYSIMPLEGUI – COMBO ELEMENT 66

13. PYSIMPLEGUI – RADIO ELEMENT 69

14. PYSIMPLEGUI – CHECKBOX ELEMENT 72

15. PYSIMPLEGUI – SLIDER ELEMENT 76

16. PYSIMPLEGUI – SPIN ELEMENT 79

17. PYSIMPLEGUI – PROGRESSBAR ELEMENT 82

18. PYSIMPLEGUI – FRAME ELEMENT 84

19. PYSIMPLEGUI – COLUMN ELEMENT 86

20. PYSIMPLEGUI – TAB ELEMENT 89

21. PYSIMPLEGUI – CANVAS ELEMENT 92

22. PYSIMPLEGUI – GRAPH ELEMENT 95

tutorialspoint
SIMPLYEASYLEARNING

23. PYSIMPLEGUI – MENUBAR 99

Menu button with Hot Key **101**

Right-click Menu **102**

ButtonMenu **103**

24. PYSIMPLEGUI – TABLE ELEMENT 105

25. PYSIMPLEGUI – TREE ELEMENT 108

26. PYSIMPLEGUI – IMAGE ELEMENT 111

Using Graph Element **112**

27. PYSIMPLEGUI – MATPLOTLIB INTEGRATION 114

Example: Draw a Sinewave Line graph **115**

28. PYSIMPLEGUI – WORKING WITH PIL 117

29. PYSIMPLEGUI – DEBUGGER 119

30. PYSIMPLEGUI – SETTINGS 122

Global Settings **122**

User Settings **122**

tutorialspoint
SIMPLYEASYLEARNING

1. PySimpleGUI – Introduction

Python GUIs for Humans

The **PySimpleGui** project started as a wrapper around TKinter package, which is bundled with Python's standard library, with the objective to simplify the GUI building process.

PySimpleGui subsequently added the ability to design desktop GUIs based on PySide library (which itself ports Qt GUI toolkit, originally written in C++, to Python) and WxPython (which ports another popular GUI toolkit called WxWidgets). These libraries are called **PySimpleGUIQt** and **PySimpleGUIWx** respectively.

The latest addition to the PySimpleGui family is the **PySimpleGUIWeb** package which uses the Remi (REMote Interface Library) to construct GUI design that is rendered in a web page.

All the packages in the PySimpleGui group follow the similar API, which means the names of GUI elements, their properties and methods are same in all the four packages. As a result, just by replacing the import statement (and keeping the rest of the code unchanged), one can get the corresponding GUI design rendered. This is in fact the most important feature of PySimpleGui. That's why, it is known as **Python GUIs for Humans**.

Comparison with other GUI Frameworks

A Python programmer has a variety of GUI frameworks to choose from, to develop a GUI application. TKinter is the one which is officially included in Python's standard library. Others, most of them are open source, have to be explicitly installed.

TkInter	Included in Python standard library
PyQt	Python 3 bindings for the Qt application framework.
PySide	Qt for Python (formerly known as PySide) offers the official Python bindings for the Qt cross-platform application and UI framework.

tutorialspoint
SIMPLYEASYLEARNING

PySimpleGUI	Wraps tkinter, Qt (pyside2), wxPython and Remi (for browser support) in a non-OOP API
wxPython	Supports Windows/Unix/Mac. Supports Python 2.7 and >=3.4. Wraps & extends the wxWidgets toolchain.
PyGObject	PyGObject is a Python package which provides bindings for GObject based libraries such as GTK Replacement for PyGtk.
PyForms	A Python framework to develop GUI application, which promotes modular software design and code reusability with minimal effort.

tutorialspoint
SIMPLY EASY LEARNING

2. PySimpleGUI – Environment Setup

PySimpleGui supports both Python 3.x versions as well as Python 2.7 version. The main port, PySimpleGui doesn't have any external dependencies, as Tkinter – on which it is based – is a part of Python's standard library, and hence it needn't be installed separately. Install it in the current Python3 environment by the PIP installer as follows

```
pip3 install PySimpleGUI
```

To verify if the library is correctly installed, enter the following statement:

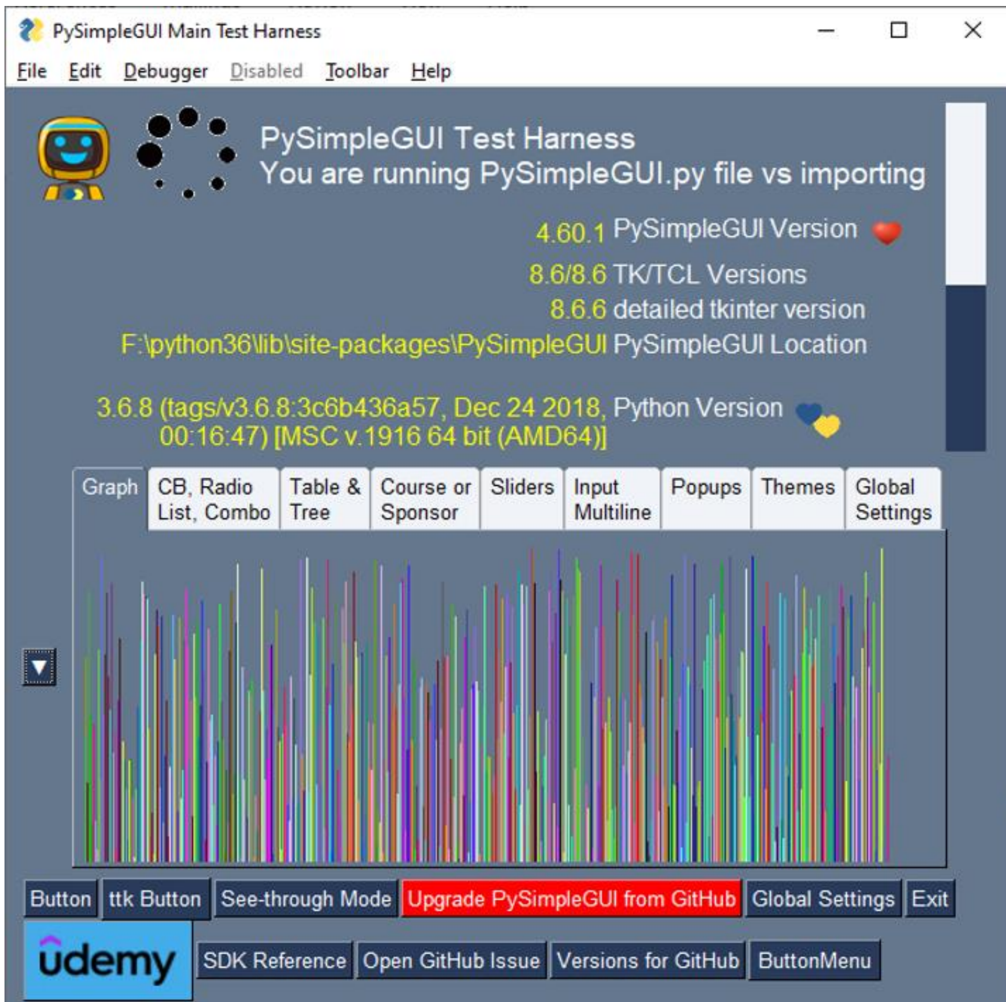
```
>>> import PySimpleGUI
>>> PySimpleGUI.version
'4.60.1 Released 22-May-2022'
```

In case, the PIP installation doesn't work, you can download "pysimplegui.py" from the Github repository (https://github.com/PySimpleGUI/PySimpleGUI) and place it in your folder along with the application that is importing it.

The **pysimplegui.py** file has the "main()" function. When called from Python prompt, it generates the following window to affirm that the package is correctly installed.

```
>>> import PySimpleGUI as psg
>>> psg.main()
Starting up PySimpleGUI Diagnostic & Help System
PySimpleGUI long version =  4.60.1 Released 22-May-2022
PySimpleGUI Version  4.60.1
tcl ver = 8.6 tkinter version = 8.6
Python Version 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)]
tcl detailed version = 8.6.6
PySimpleGUI.py location F:\python36\lib\site-packages\PySimpleGUI\PySimpleGUI.py
```

The GUI window appears as below:



If you are using Python3 version earlier than 3.4, you may need to install the "typing" module since it is not shipped in the corresponding standard library

```
pip3 install typing
```

For Python 2.7, change the name to `PySimpleGUI27`.

```
pip3 install PySimpleGUI27
```

You may need to also install "future" for version 2.7

```
pip install future
```

However, it is important to note that Python Software Foundation doesn't officially support Python 2.x branches.

tutorialspoint
SIMPLYEASYLEARNING

3. PySimpleGUI – Hello World

First Window using PySimpleGUI

To check whether PySimpleGUI along with its dependencies are properly installed, enter the following code and save it as "hello.py", using any Python-aware editor.

```
import PySimpleGUI as psg
layout = [[psg.Text(text='Hello World',
                    font=('Arial Bold', 20),
                    size=20,
                    expand_x=True,
                    justification='center')],
          ]
window = psg.Window('HelloWorld', layout, size=(715,250))

while True:
    event, values = window.read()
    print(event, values)
    if event in (None, 'Exit'):
        break

window.close()
```

The above code constructs a window with a Text element (equivalent of a Label in TKinter) and displays the "Hello World" message placed centrally across the width of the window.

Run this program from the command terminal as:

```
Python hello.py
```


tutorialspoint
SIMPLYEASYLEARNING

The **output** generated by the program should be similar to the one displayed below:

Equivalent Tkinter Code

To obtain similar output using pure Tkinter code, we would require the following Python script:

```
from tkinter import *

window=Tk()

lbl=Label(window, text="Hello World",
          fg='white', bg='#64778D',
          font=("Arial Bold", 20))
lbl.place(x=300, y=15)

window.title('HelloWorld Tk')
window['bg']='#64778D'
window.geometry("715x250+10+10")

window.mainloop()
```

All other functionalities remain same, except we use the **serve()** function off **waitress** module to start the WSGI server. On visiting the '/' route in

the browser after running the program, the Hello World message is displayed as before.

Instead of a function, a callable class can also be used as a View. A callable class is the one which overrides the **__call__()** method.

```
from pyramid.response import Response


class MyView(object):
    def __init__(self, request):
        self.request = request


    def __call__(self):
        return Response('hello world')
```

PySimpleGUIQt

The object model of PySimpleGUI API has been made compatible with the widgets as defined in PySide2 package (which is the Python port for Qt graphics toolkit). The Qt version of PySimpleGui is called PySimpleGUIQt. It can be similarly installed with following PIP command:

```
pip3 install PySimpleGUIQt
```

Since this package depends on PySide2, the same will also be installed.

```
>>> import PySide2
>>> PySide2.__version__
'5.15.2.1'
>>> import PySimpleGUIQt
>>> PySimpleGUIQt.version
'0.35.0 Released 6-Jun-2020'
```

As mentioned earlier, the most important feature of PySimpleGui projects is that the code written for one package is completely compatible with the other. Hence, the hello.py program used earlier can be used as it is for the

tutorialspoint
SIMPLYEASYLEARNING

Qt version. The only change needed is import PySimpleGUIQt instead of PySimpleGui.

```
import PySimpleGUIQt as psg


layout = [[psg.Text(text='Hello World',
                    font=('Arial Bold', 20),
                    justification='center')],
          ]


window = psg.Window('HelloWorldQt', layout, size=(715,250))


while True:
    event, values = window.read()
    print(event, values)
    if event in (None, 'Exit'):
        break


window.close()
```

The **output** is fairly similar.

Equivalent PySide2 Code

The pure PySide2 code to achieve the same result is as follows:

```
import sys

from PySide2.QtCore import *
from PySide2.QtGui import *
from PySide2.QtWidgets import *

def window():
    app = QApplication(sys.argv)
    w = QWidget()
    w.setStyleSheet("background-color: #64778D;")

    b = QLabel(w)
    b.setText("Hello World!")
    b.setFont(QFont('Arial Bold', 20))
    b.setAlignment(Qt.AlignCenter)
    b.setStyleSheet("color: white;")
    b.setGeometry(100, 100, 715, 250)
    b.move(50, 20)

    w.setWindowTitle("HelloWorldQt")
    w.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    window()
```

It will produce the same output window.

tutorialspoint
SIMPLYEASYLEARNING

PySimpleGUIWx

This module encapsulates the functionality of GUI widgets as defined in WxPython toolkit. WxPython is a Python port for the widely used WxWidgets library originally written in C++. Obviously, PySimpleGUIWx depends on WxPython package, and hence the latter will get automatically installed by the following PIP command:

```
pip3 install PySimpleGUIWx
```

To confirm that both PySimpleGUIWx and WxPython are properly installed, enter following statements in Python terminal.

```
>>> import PySimpleGUIWx
>>> PySimpleGUIWx.version
'0.17.1 Released 7-Jun-2020'
>>> import wx
>>> wx.__version__
'4.0.7'
```

Not much of change is required in the "hello.py" script. We need to just replace PySimpleGUI with PySimpleGUIWx module in the "import" statement.

```
import PySimpleGUIWx as psg


layout = [[psg.Text(text='Hello World',
                    font=('Arial Bold', 20),
                    size=(500, 5),
                    justification='center')],
          ]


window = psg.Window('HelloWorldWx', layout,
                    size=(715, 250))


while True:
```

```
    event, values = window.read()
    print(event, values)
    if event in (None, 'Exit'):
        break

window.close()
```

It will produce the following **output**:

Note that you'll need a little more complex code to obtain the similar output with **pure WxPython code** as follows:

```
import wx


app = wx.App()
window = wx.Frame(None, title="WxPython", size=(715, 250))


panel = wx.Panel(window)
panel.SetBackgroundColour((100, 119, 141))


label = wx.StaticText(panel, -1, style=wx.ALIGN_CENTER)
label.SetLabel("Hello World")
label.SetForegroundColour((255, 255, 255))


font = wx.Font()
```

```
font.SetFaceName("Arial Bold")
font.SetPointSize(30)

label.SetFont(font)
window.Show(True)

app.MainLoop()
```

It will display a top level window with a text label having Hello World as the caption.

PySimpleGUIWeb

Remi (REMote Interface) is a GUI library for Python applications that are rendered in a web browser. PySimpleGUIWeb package ports the original PySimpleGui library to Remi so that its apps can be run in a browser. Following PIP command installs both PySimpleGUIWeb and Remi in the current Python environment:

```
pip3 install PySimpleGUIWeb
```

Check for their proper installation before writing an app.

```
>>> import PySimpleGUIWeb
>>> PySimpleGUIWeb.version
'0.39.0  Released 6-Jun-2020'
```

Following script is the PySimpleGUIWeb version of the original Hello World program.

```
import PySimpleGUIWeb as psg

layout = [[psg.Text(text='Hello World',
                    font=('Arial Bold', 20),
                    justification='center')]]

window = psg.Window('HelloWorldWeb', layout)
```

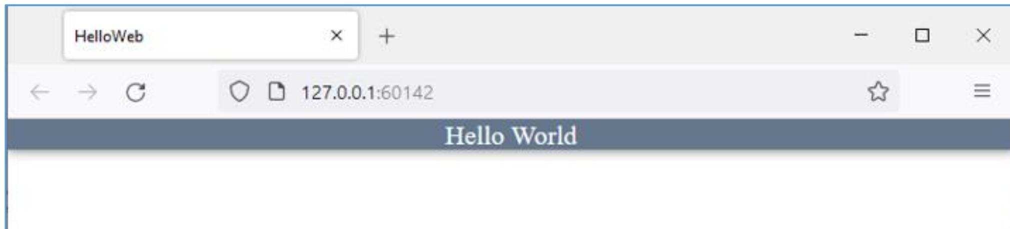
```
while True:
    event, values = window.read()
    print(event, values)
    if event in (None, 'Exit'):
        break

window.close()
```

To obtain similar output using pure Remi library's functionality is a little complex, as the following code shows:

```
import remi.gui as gui
from remi import start, App

class HelloWeb(App):
    def __init__(self, *args):
        super(HelloWeb, self).__init__(*args)

    def main(self):
        wid = gui.VBox(style={"background-color": "#64778D"})

        self.lbl = gui.Label('Hello World',
                             width='100%',
                             height='100%',
                             style={ "color":"white",
                                     "text-align": "center",
                                     "font-family": "Arial Bold",
                                     "font-size": "20px"}
                             )
        wid.append(self.lbl)
        return wid
```

```
if __name__ == "__main__":
    start(HelloWeb, debug=True, address='0.0.0.0', port=0)
```

When we run these programs, the Remi server starts, a browser window automatically opens and the Hello World message is displayed.



Here we have seen the Hello World program written in the PySimpleGUI, PySimpleGUIQt, PySimpleGUIWx and PySimpleGUIWeb libraries. We can see that the widget library remains the same. Moreover, the same Hello world program, when written in pure Tkinter, PySide, WxPython and Remi respectively, becomes far more complex and tedious than the PySimpleGUI versions.

4. PySimpleGUI – Popup Windows

A function in PySimpleGUI module that start with the prefix popup* generates window of a predefined appearance. The name of the popup function indicates is purpose and configuration of buttons present on it. These popups are created with just one line of code. Each popup serves a certain purpose, and then closes immediately.

A most basic popup is created by the **popup()** function. It can be used like a print() function to display more than one parameters on the window, and an OK button. It acts like a message box, that disappears immediately on pressing the OK button

```
>>> import PySimpleGUI as psg
>>> psg.popup("Hello World")
```

It displays a popup window with Hello World text and OK button. Note that more than one strings can be displayed. Following popups with different button configurations are available:

- **popup_ok**: Display Popup with OK button only
- **popup_ok_cancel**: Display popup with OK and Cancel buttons
- **popup_cancel**: Display Popup with "cancelled" button text
- **popup_yes_no**: Display Popup with Yes and No buttons
- **popup_error**: Popup with colored button and 'Error' as button text

These functions return the text of the button pressed by the user. For example, if the user presses OK button of the ok-cancel popup, it returns Ok which can be used in further programming logic.

Following popups accept input from the user in the form of text or let the user select file/folder/date from the selectors.

- **popup_get_text**: Display Popup with text entry field. Returns the text entered or None if closed / cancelled

- **popup_get_file**: Display popup window with text entry field and browse button so that a file can be chosen by user.

- **popup_get_folder**: Display popup with text entry field and browse button so that a folder can be chosen.
- **popup_get_date**: Display a calendar window, get the user's choice, return as a tuple (mon, day, year)

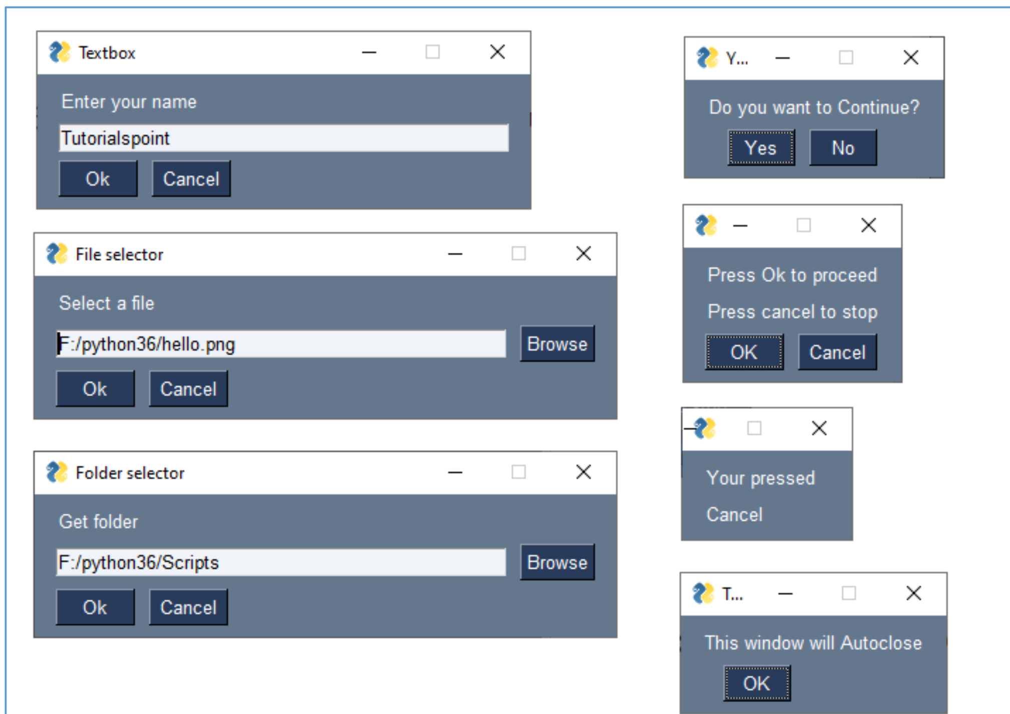
When user has made the selection and Ok button is pressed, the return value of the popup is the text, which can be used further in the program.

Following script shows the use of some of the above popups:

```
import PySimpleGUI as psg

text = psg.popup_get_text('Enter your name',
                          title="Textbox")
print ("You entered: ", text)

file=psg.popup_get_file('Select a file',
                        title="File selector")
print ("File selected", file)

folder=psg.popup_get_folder('Get folder',
                            title="Folder selector")
print ("Folder selected",folder)

ch = psg.popup_yes_no("Do you want to Continue?",
                      title="YesNo")
print ("You clicked", ch)

ch = psg.popup_ok_cancel("Press Ok to proceed",
                         "Press cancel to stop",
                         title="OkCancel")
if ch=="OK":
    print ("You pressed OK")
```

```
if ch=="Cancel":
    print ("You pressed Cancel")
psg.popup_no_buttons('You pressed', ch,  non_blocking=True)

psg.popup_auto_close('This window will Autoclose')
```

Output: The popups generated by the above code are shown below:



The following **output** is displayed on the Python **console**:

```
You entered:  Tutorialspoint
File selected F:/python36/hello.png
Folder selected F:/python36/Scripts
You clicked Yes
You pressed Cancel
```

All types of popups are objects of respective classes inherited from popup class. All of them have a common set of properties. These properties have a certain default value, and can be used to customize the appearance and

behaviour of the popup objects. Following table lists the common parameters:

Type	Parameter	Description
Any	*args	Values to be displayed on the popup
Str	title	Optional title for the window.
(str, str) or None	button_color	Color of the buttons shown (text color, button color)
Str	background_color	Window's background color
Str	text_color	text color
Bool	auto_close	If True the window will automatically close
Int	auto_close_duration	time in seconds to keep window open before closing it automatically
Bool	non_blocking	If True then will immediately return from the function without waiting for the user's input.
Tuple[font_name, size, modifiers]	font	specifies the font family, size, etc. Tuple or Single string format 'name size styles'.
Bool	grab_anywhere	If True can grab anywhere to move the window.
(int, int)	Location	Location on screen to display the top left corner of window. Defaults to window centered on screen
Bool	keep_on_top	If True the window will remain above all current windows
Bool	modal	If True, then makes the popup will behave like a Modal window. Default = True

tutorialspoint
SIMPLYEASYLEARNING

Scrolled Popup

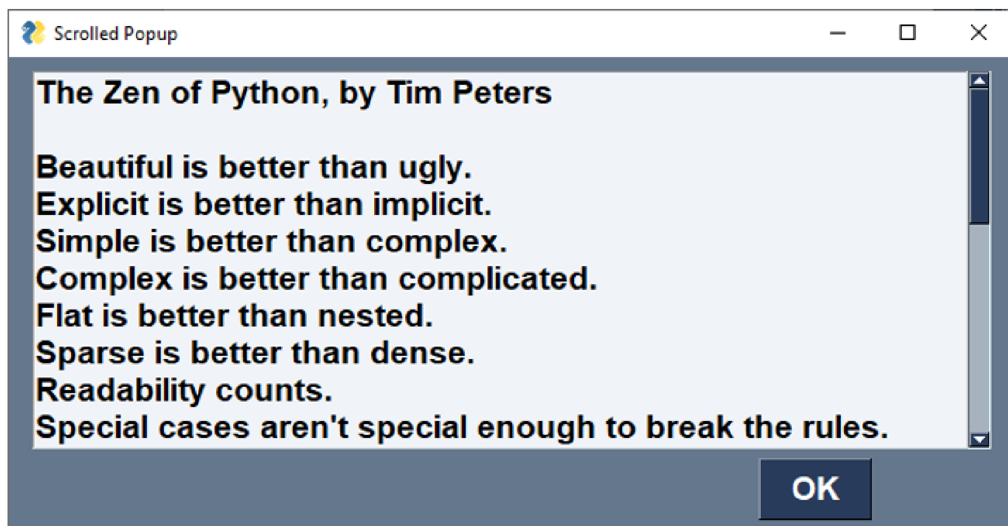
The **popup_scrolled()** function generates a popup with a scrollable text box in it. Use this to display a large amount of text, consisting of many lines with number of characters more than the width.

The size property is a tuple (w, h) with "w" being the number of characters in one line, and "h" being the lines displayed at a time. The horizontal/vertical scrollbar to the text box will become active if the number of characters/no of lines of text are more than "w" or "h".

In the following example, a big file zen.txt is displayed in a popup with scrollable text box. The file contains the design principles of Python called the "Zen of Python".

```
import PySimpleGUI as psg

file=open("zen.txt")
text=file.read()

psg.popup_scrolled(text, title="Scrolled Popup",
                        font=("Arial Bold", 16), size=(50,10))
```

It will produce the following **output**:



Progress Meter

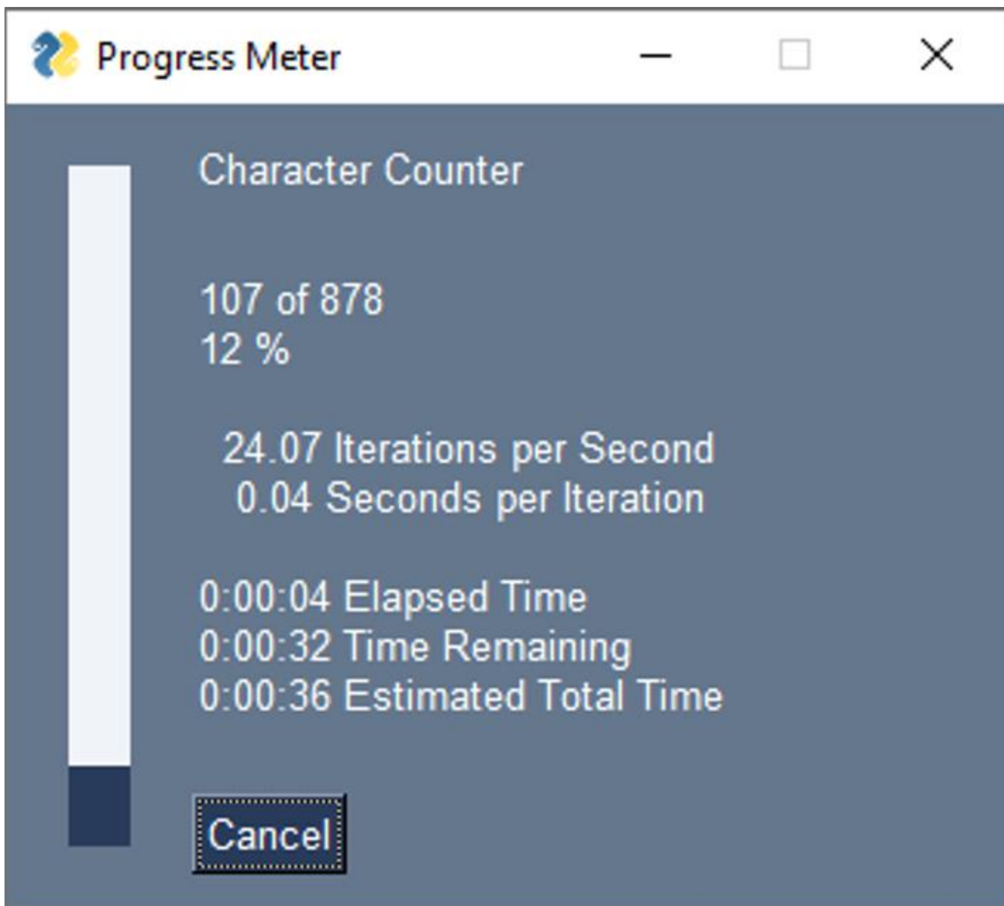
The "one_line_progress_meter" is a popup that displays the visual representation of an ongoing long process, such as a loop. It shows the instantaneous value of a certain parameter, estimated time to complete the process, and the elapsed time.

In the following example, a text file is read character by character. The Progress meter shows the progress of the process in the form of a progress bar, estimated time required for completion, and the instantaneous value of the count.

```
import PySimpleGUI as psg
import os

size = os.path.getsize('zen.txt')
file=open("zen.txt")
i=0

while True:
    text=file.read(1)
    i=i+1
    if text=="":
        file.close()
        break
    print (text,end='')
    psg.one_line_progress_meter(
        'Progress Meter', i, size,
        'Character Counter'
    )
```

It will produce the following **output** window:

tutorialspoint
SIMPLYEASYLEARNING



Debug Popup

During the execution of a program, it is usually required to keep track of intermediate values of certain variables, although not required in the following output. This can be achieved by the **Print()** function in PySimpleGUI library.

Note: Unlike Python's built-in **print()** function, this function has "P" in uppercase).

As the program encounters this function for the first time, the debug window appears and all the subsequent Prints are echoed in it. Moreover, we can use **EasyPrint** or **eprint** that also have same effect.

The following program computes the factorial value of the number input by the user. Inside the for loop, we want to keep track of the values of f (for factorial) on each iteration. That is done by the Print function and displayed in the debug window.

```
import PySimpleGUI as psg


f=1
num=int(psg.popup_get_text("enter a number: "))


for x in range(1, num+1):
   f=f*x
   psg.Print (f,x)


print ("factorial of {} = {}".format(x,f))
```

Assuming that the user inputs 5, the debug window shows the following **output**:

5. PySimpleGUI – Window Class

Popups have a predefined configuration of buttons, text labels and text input fields. The Window class allows you to design a GUI of more flexible design. In addition to these elements, other elements like listbox, checkbox, radio buttons, etc., are available. You can also provide a menu system to the GUI. Certain specialized widgets such as spinner, sliders, etc., can also be used to make the design more effective.

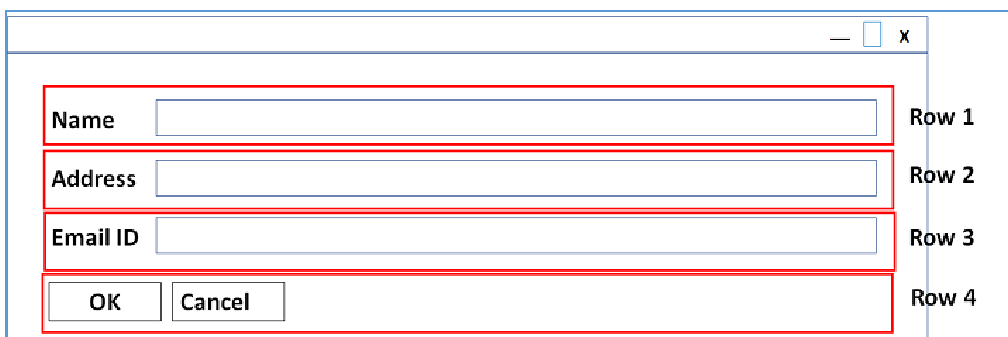
A window can be a non-persistent window, similar to the popups. It blocks the program flow till the user closes it by clicking a button on the client area or the close (X) button in the title bar.

A persistent window on the other hand continues to be visible till the event causing it to be closed occurs. The asynchronous window is the one whose contents are periodically updated.

Layout Structure

The placement of elements or widgets in the window's client area is controlled by list of list objects. Each list element corresponds to one row on the window surface, and may contain one or more GUI elements available in PySimpleGUI library.

The first step is to visualize the placement of elements by making a drawing as follows:



The elements on the window are placed in four rows. First three rows have a Text element (displays a static text) and an InputText element (in which user can enter). Last row has two buttons, Ok and Cancel.

This is represented in the list of lists as below:

```
import PySimpleGUI as psg
```


tutorialspoint
SIMPLYEASYLEARNING

```
layout = [
    [psg.Text('Name '),psg.Input()],
    [psg.Text('Address '), psg.Input()],
    [psg.Text('Email ID '), psg.Input()],
    [psg.OK(), psg.Cancel()]
]
```

This list object is used as the value of layout parameter for the constructor of the Window class.

```
window = psg.Window('Form', layout)
```

This will display the desired window. The user inputs are stored in a dictionary named as values. The **read()** method of Window class is called as the user presses the Ok button, and the window closes immediately.

The complete code to render the window is as follows:

```
import PySimpleGUI as psg

psg.set_options(font=('Arial Bold', 16))

layout = [
    [psg.Text('Name ', size=(15,1)),psg.Input(expand_x=True)],
    [psg.Text('Address ', size=(15,1)), psg.Input(expand_x=True)],
    [psg.Text('Email ID ', size=(15,1)), psg.Input(expand_x=True)],
    [psg.OK(), psg.Cancel()]
]

window = psg.Window('Form', layout, size=(715,207))
event, values = window.read()
print (event, values)

window.close()
```

Here is the **output** as displayed:

Enter the data as shown and press the "OK" button. The values will be printed as below:

```
OK {0: 'Kiran Gupta', 1: 'Mumbai', 2: 'kiran@gmail.com'}
```

If, after filling the data, you press the "Cancel" button, the result printed will be:

```
Cancel {0: 'Kiran Gupta', 1: 'Mumbai', 2: 'kiran@gmail.com'}
```

Persistent Window

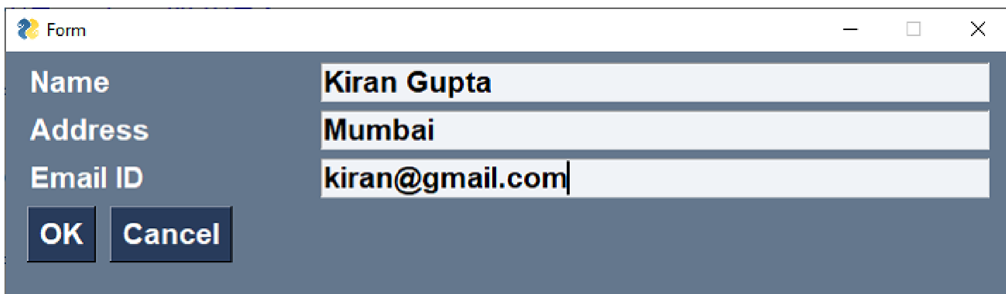
Note that this window gets closed as soon as any button (or the "X" button in the title bar) is clicked. To keep the window alive till a special type of button called Exit is pressed or if the window is closed by pressing "X", the **read()** method is placed in an infinite loop with provision to break when WIN_CLOSED event occurs (when Exit button is pressed) or Exit event occurs (when "X" button is pressed).

Let us change the Cancel button in the above code with Exit button.

```
import PySimpleGUI as psg


layout = [
    [psg.Text('Name      '), psg.Input()],
    [psg.Text('Address   '), psg.Input()],
    [psg.Text('Email ID '), psg.Input()],
    [psg.OK(), psg.Exit()]
]
```

```
window = psg.Window('Form', layout)

while True:
    event, values = window.read()
    if event == psg.WIN_CLOSED or event == 'Exit':
        break
    print (event, values)

window.close()
```

The appearance of the window will be similar as before, except that instead of Cancel, it has Exit button.



The entered data will be printed in the form of a tuple. First element is the event, i.e., the caption of button, and second is a dictionary whose key is incrementing number and value is the text entered.

```
OK {0: 'kiran', 1: 'Mumbai', 2: 'kiran@gmail.com'}
OK {0: 'kirti', 1: 'Hyderabad', 2: 'kirti@gmail.com'}
OK {0: 'karim', 1: 'Chennai', 2: 'karim@gmail.com'}
```

Window Methods

The important method defined in the Window class is the **read()** method, to collect the values entered in all input elements. The Window class has other methods to customize the appearance and behaviour. They are listed below:

AddRow	Adds a single row of elements to a window's "self.Rows" variable
AddRows	Loops through a list of lists of elements and adds each row, list, to the layout.
close	Closes the window so that resources are properly freed up.
disable	Disables window from taking any input from the user
disappear	Causes a window to "disappear" from the screen, but remain on the taskbar.
enable	Re-enables window to take user input
fill	Fill in elements that are input fields with data provided as dictionary.
find_element	Find element object associated with the provided key. It is equivalent to "element = window[key]"
get_screen_dimensions	Get the screen dimensions.
hide	Hides the window from the screen and the task bar
load_from_disk	Restore values from a Pickle file created by the "SaveToDisk" function
layout	Populates the window with a list of widget lists.
read	Get all of your data from your Window. Pass in a timeout (in milliseconds) to wait.
reappear	Causes a disappeared window to display again.
save_to_disk	Saves the values contained in each of the input elements to a pickle file.
set_title	Change the title of the window in taskbar

Update Window with Key

The data entered by the user in different input elements on the window layout is stored in the dictionary format. The dictionary keys are numbered (starting from 0) corresponding to input elements from left to right and top to bottom. We can refer to the input data by dictionary operator. That means, the data in the first element is returned by "values[0]".

tutorialspoint
SIMPLYEASYLEARNING

```
values = {0: 'kiran', 1: 'Mumbai', 2: 'kiran@gmail.com'}


data = [values[k] for k in values.keys()]
print (data)
```

It will print the following on the console:

```
['kiran', 'Mumbai', 'kiran@gmail.com']
```

However, if you want to manipulate the value of an element programmatically, the element must be initialized by assigning a unique string value to its key parameter. The key of an element is like the name of the variable or identifier, which makes it convenient to handle reading or assigning a value to it programmatically.

The key parameter should be a string. The convention is that it should be an uppercase string preceded and followed by a "-" character (Example: "-NAME-"). However, any string can be used.

Let us assign keys to the Input elements in the above example as shown below:

```
layout = [
    [psg.Text('Name  '),psg.Input(key='-NM-')],
    [psg.Text('Address '), psg.Input(key='-AD-')],
    [psg.Text('Email ID '), psg.Input(key='-ID-')],
    [psg.OK(), psg.Exit()],
]
```

As a result, the values dictionary returned after the **read()** method will contain the key identifiers instead of integers previously.

```
OK {'-NM-': 'Kiran', '-AD-': 'Mumbai', '-ID-': 'kiran@gmail.com'}
```

Now, values[-NM-'] will fetch 'Kiran'. The key can be assigned to any element and not just the input element. You can use the same key to call Update on an element. We can use "find_element(key)" of the Window object, or use window['key'] to refer to the element.

Let us extend our previous example to add a row before the Ok and Cancel buttons and have an empty Text element with "-OUT-" key. On the OK event, this Text label shows the concatenation of data entered in three input elements having keys "-NM-", "-AD-" and "-ID-".

```
import PySimpleGUI as psg

psg.set_options(font=('Arial Bold', 16))

layout = [
    [psg.Text('Name  ', size=(15, 1)),
     psg.Input(key='-NM-', expand_x=True)],

    [psg.Text('Address ', size=(15, 1)),
     psg.Input(key='-AD-', expand_x=True)],

    [psg.Text('Email ID ', size=(15, 1)),
     psg.Input(key='-ID-', expand_x=True)],

    [psg.Text('You Entered '), psg.Text(key='-OUT-')],

    [psg.OK(), psg.Exit()],
]

window = psg.Window('Form', layout, size=(715, 200))

while True:
    event, values = window.read()
    print(event, values)
    out = values['-NM-'] + ' ' + values['-AD-'] + ' ' + values['-ID-']
    window['-OUT-'].update(out)

    if event == psg.WIN_CLOSED or event == 'Exit':
        break
```

```
window.close()
```

Run the above code, enter text in three input elements and press OK. The -OUT- text label will be updated as shown here:

Another example of use of key attribute is given below. To Input elements are assigned key parameters -FIRST- and -SECOND-. There are two buttons captioned Add and Sub. The Text element displays addition or subtraction of two numbers depending on the button pressed.

```
import PySimpleGUI as psg
import PySimpleGUI as psg

psg.set_options(font=('Arial Bold', 16))

layout = [
    [psg.Text('Enter a num:  '), psg.Input(key='-FIRST-')],
    [psg.Text('Enter a num:  '), psg.Input(key='-SECOND-')],
    [psg.Text('Result      :  '), psg.Text(key='-OUT-')],
    [psg.Button("Add"), psg.Button("Sub"), psg.Exit()],
]

window = psg.Window('Calculator', layout, size=(715, 180))

while True:
    event, values = window.read()
    print(event, values)
```


tutorialspoint
SIMPLYEASYLEARNING

```
    if event == "Add":
        result = int(values['-FIRST-']) + int(values['-SECOND-'])

    if event == "Sub":
        result = int(values['-FIRST-']) - int(values['-SECOND-'])

    window['-OUT-'].update(result)

    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

The following screenshot shows the result when the "Add" button is pressed.

Borderless Window

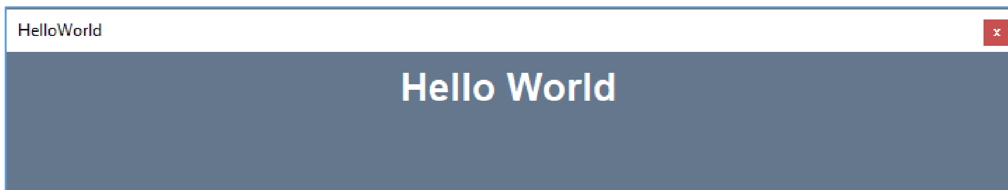
By default, the application window is created with a title bar above the client area wherein all other elements are placed in the layout. The titlebar consists of a window title on the left, and the control buttons (minimize, restore/maxmimize and close) on the right. However, particularly for a kiosk-like application, there is no need of a title bar. You can get rid of the title bar by setting the "no_titlebar" property of the Window object to "True".

Hello World

To terminate such an application, the event loop must be terminated on the occurrence of Exit button event.

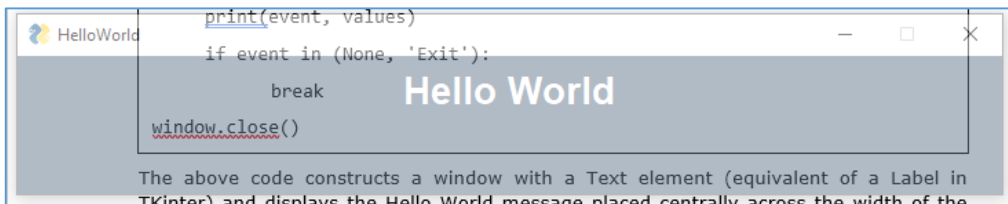
Window with Disabled Close

If you wish to prevent the user to minimize the application window, the "disable_minimize" property of the Window object should be set to True. Similarly, the True value to "disable_close" property the Close button is displayed but it doesn't create the WINDOW_CLOSED event.



Transparent Window

The "alpha_channel" property of the Window object decides the transparency of the window. Its value is between 0 to 1. By default, it is 0, which means that the window appears as opaque. Set it to 1 to make it completely transparent. Any float value between 0 to 1 makes the transparency proportional.



Multiple Windows

PySimpleGUI allows more than one windows to be displayed simultaneously. The static function in PySimpleGUI module reads all the active windows when called. To make the window active, it must be finalized. The function returns a tuple of (window, event, values) structure.

```
window, event, values = PySimpleGUI.read_all_windows()
```

If no window is open, its return value is (None, WIN_CLOSED, None)

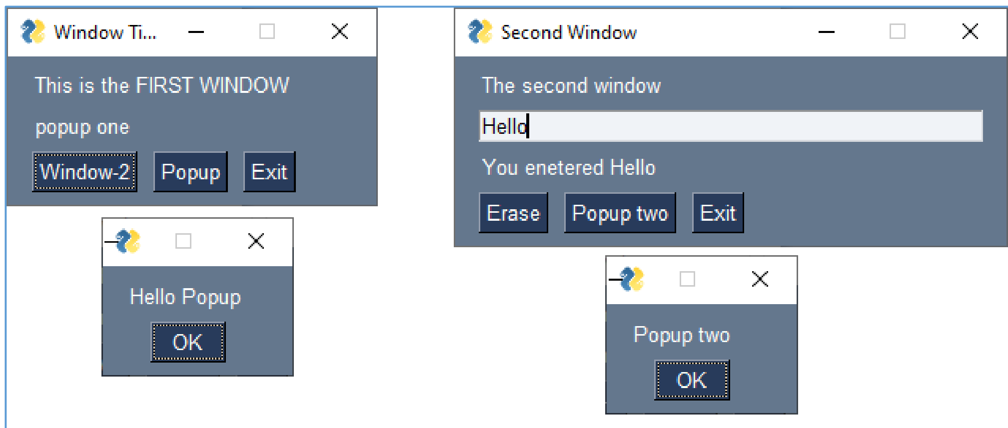
In the following code, two functions "win1()" and "win2()" create a window each when called. Starting with the first window, the button captioned Window-2 opens another window, so that both are active. When CLOSED event on first window takes place, both are closed and the program ends. If the "X" button on second window is pressed, it is marked as closed, leaving the first open.

```
import PySimpleGUI as psg


def win1():
    layout = [
        [psg.Text('This is the FIRST WINDOW'), psg.Text('
', key='-OUTPUT-')],
        [psg.Text('popup one')],
        [psg.Button('Window-2'), psg.Button('Popup'),
psg.Button('Exit')]
    ]
    return psg.Window('Window Title', layout, finalize=True)


def win2():
    layout = [
        [psg.Text('The second window')],
        [psg.Input(key='-IN-', enable_events=True)],
        [psg.Text(size=(25, 1), key='-OUTPUT-')],
        [psg.Button('Erase'), psg.popup('Popup two'),
psg.Button('Exit')]]
    return psg.Window('Second Window', layout, finalize=True)



window1 = win1()
window2 = None


while True:
```

```
    window, event, values = psg.read_all_windows()
    print(window.Title, event, values)

    if event == psg.WIN_CLOSED or event == 'Exit':
        window.close()

        if window == window2:
            window2 = None

        elif window == window1:
            break

    elif event == 'Popup':
        psg.popup('Hello Popup')

    elif event == 'Window-2' and not window2:
        window2 = win2()

    elif event == '-IN-':
        window['-OUTPUT-'].update('You entered
{}'.format(values["-IN-"]))

    elif event == 'Erase':
        window['-OUTPUT-'].update('')
        window['-IN-'].update('')

window.close()
```

It will produce the following **output** windows:

tutorialspoint
SIMPLYEASYLEARNING



Asynchronous Window

The **read()** method of the Window class has the following additional parameters:

```
window.read(timeout = t, timeout_key=TIMEOUT_KEY, close=False)
```

The **timeout** parameter lets your GUIs to use in a non-blocking read situation. It is the miliseconds your device can wait before returning. It makes a window that runs on a periodic basis.

The longer you're able to add to the timeout value, the less CPU time you'll be taking. During the timeout time, you are "yielding" the processor to do other tasks. your GUI will be more responsive than if you used a non-blocking read.

The timeout_key parameter helps in deciding whether there has been any user action within the stipulated time. The default value of "timeout_key" is "__timeout__".

```
while True:
    event, value = window.read(timeout=10)
    if event == sg.WIN_CLOSED:
        break

    if event == sg.TIMEOUT_KEY:
        print("Nothing happened")
```

To make the window movable, set the "grab_anywhere" property of the Window object to true. If the "keep_on_top" property is set to True, the window will remain above the current windows.

tutorialspoint
SIMPLYEASYLEARNING

6. PySimpleGUI – Element Class

The PySimpleGUI library consists of a number of GUI widgets that can be placed on top of the Window object. For instance, the buttons or the textboxes that we have used in the above examples. All these widgets are in fact objects of classes defined in this library, in which Element class acts as the base for all other widget classes.

An object of this Element class is never declared explicitly. It defines the common properties like size, color, etc. Here is the list of the available widgets (also called elements)

Text element	Display some text in the window. Usually this means a single line of text.
Input element	Display a single text input field.
Multiline element	Display and/or read multiple lines of text. This is both an input and output element.
Combo element	A combination of a single-line input and a drop-down menu.
OptionMenu element	Similar to Combo. Only on TKinter port
Checkbox element	Displays a checkbox and text next to it.
Radio element	Used in a group of other Radio Elements to provide user with ability to select only one choice in a list of choices.
Spin element	A spinner with up/down buttons and a single line of text.
Button element	Defines all possible buttons. The shortcuts such as Submit, FileBrowse, ... each create a Button
ButtonMenu element	Creates a button that when clicked will show a menu similar to right click menu.
Slider element	Horizontal or vertical slider to increment/decrement a value.
Listbox element	Provide a list of values for the user to choose one or more of. Returns a list of selected rows when a **window.read()** is executed.
Image element	Show an image in the window. Should be a GIF or a PNG only.

Graph element	Creates area to draw graph
Canvas element	An area to draw shapes
ProgressBar element	Displays a colored bar that is shaded as progress of some operation is made.
Table element	Display data in rows and columns
Tree element	Presents data in a tree-like manner, much like a file/folder browser.
Sizer element	This element is used to add more space.
StatusBar element	A StatusBar Element creates the sunken text-filled strip at the bottom.

Properties of Element Class

Following are the properties of the Element Class:

- **size**: (w=characters-wide, h=rows-high)
- **font**: specifies the font family, size.
- **background_color**: color of background.
- **text_color**: element's text color.
- **key**: Identifies an Element.
- **visible**: set visibility state of the element (Default = True)

Methods of Element Class

Following are the methods of the Element Class:

- **set_tooltip()**: Called by application to change the tooltip text for an Element.
- **set_focus():** Sets the current focus to be on this element
- **set_size():** Changes the size of an element to a specific size.
- **get_size():** Return the size of an element in Pixels.
- **expand():** Causes the Element to expand to fill available space in the X and Y directions.
- **set_cursor():** Sets the cursor for the current Element.
- **set_right_click_menu():** Sets right click menu to be invoked when clicked.

7. PySimpleGUI – Events

Any GUI application is event driven, having the ability to respond to the various possible events occurring on the GUI elements. In PySimpleGUI, the event handling is done inside an infinite loop below the constitution of GUI design, continuously checking whether an event occurs and perform the actions based on the event.

There are two types of events:

- Window events, and
- Element events.

The window events are enabled by default, and include the button events (occur when any button is clicked) and the event of the "X" button on the titlebar clicked.

The element events are not enabled by default. Element-specific events can be detected only when the "enable_events" parameter is set to True when an element is created.

Window Closed Event

The infinite event loop that makes the PySimpleGUI window persistent, is terminated when the user presses the "X" button, or the **close()** method of Window class is executed. The standard way of terminating the loop is as follows:

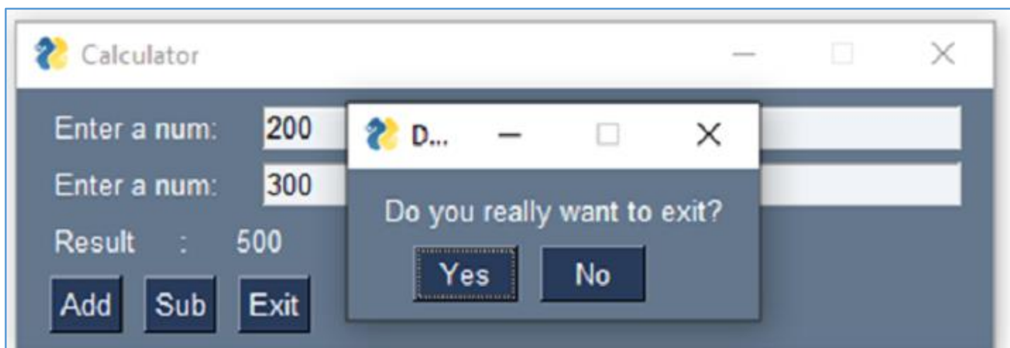
```
import PySimpleGUI as psg
...
while True:
    ...

    if event == psg.WIN_CLOSED:
        break
    ...

window.close()
```

The Widow class also emits an "enable_close_attempted_event" if this parameter is set to True. It is a good practice to call yes-no popup when it is detected inside the loop.

```
window = psg.Window('Calculator', layout,
                    enable_close_attempted_event=True)


while True:
    event, values = window.read()
    print(event, values)

    if event == "Add":
        result = int(values['-FIRST-']) + int(values['-SECOND-'])
    if event == "Sub":
        result = int(values['-FIRST-']) - int(values['-SECOND-'])

    window['-OUT-'].update(result)

    if event == psg.WINDOW_CLOSE_ATTEMPTED_EVENT and
psg.popup_yes_no('Do you really want to exit?') == 'Yes':
        break

    if event == psg.WIN_CLOSED or event == 'Exit':
        break
```

In this case, as the "X" button is pressed, the Popup with Yes/No button appears and the program exits when the "Yes" button is clicked.

It will produce the following output window:



The event value also returns the "-WINDOW CLOSE ATTEMPTED-" value.

```
-WINDOW CLOSE ATTEMPTED- {'-FIRST-': '200', '-SECOND-': '300'}
```

Button Events

The button click event is enabled by default. To disable, use "Button.update(disabled=True)". You can also set "enable_events=True" in Button's constructor, it will enable the Button Modified event. This event is triggered when something 'writes' to a button.

When we read the contents of the window (using "window.read()"), the button value will be either its caption (if key is not set) or key if it is set.

In the above example, since the key parameter is not set on the Add and Sub buttons, their captions are returned when the window is read.

```
Add {'-FIRST-': '200', '-SECOND-': '300'}
```

Add key parameters to Add and Sub buttons in the program.

```
import PySimpleGUI as psg

layout = [
    [psg.Text('Enter a num:  '), psg.Input(key='-FIRST-')],
    [psg.Text('Enter a num:  '), psg.Input(key='-SECOND-')],
    [psg.Text('Result      :  '), psg.Text(key='-OUT-')],
    [psg.Button("Add", key='-ADD-'), psg.Button("Sub", key='-
SUB-'), psg.Exit()],
]
```

```
window = psg.Window('Calculator', layout)

while True:
    event, values = window.read()
    print(event, values)

    if event == "-ADD-":
        result = int(values['-FIRST-']) + int(values['-SECOND-'])

    if event == "-SUB-":
        result = int(values['-FIRST-']) - int(values['-SECOND-'])

    window['-OUT-'].update(result)

    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

The tuple returned by the read() method will now show the key of button pressed.

```
-ADD- {'-FIRST-': '200', '-SECOND-': '300'}
```

Events of Other Elements

Many of the elements emit events when some type of user interaction takes place. For example, when a slider is moved, or an item from the list is selected on or a radio button is clicked on.

Unlike Button or Window, these events are not enabled by default. To enable events for an Element, set the parameter "enable_events=True".

The following table shows the elements and the events generated by them.

Name	Events
InputText	any key pressed

Combo	item selected
Listbox	selection changed
Radio	selection changed
Checkbox	selection changed
Spinner	new item selected
Multiline	any key pressed
Text	Clicked
Status Bar	Clicked
Graph	Clicked
Graph	Dragged
Graph	drag ended (mouse up)
TabGroup	tab clicked
Slider	slider moved
Table	row selected
Tree	node selected
ButtonMenu	menu item chosen
Right click menu	menu item chosen

tutorialspoint
SIMPLYEASYLEARNING

8. PySimpleGUI – Text Element

The Text element is one of the basic and most commonly used elements. An object of the Text class displays a non-editable, single line of text containing Unicode characters. Although most of the times, it is not used to respond to events, it can emit the event having its key as the name.

The Text element has the following properties in addition to those derived from the Element class:

text	The text to display. Can include /n to achieve multiple lines.
justification	How string should be aligned within space provided by size. Valid choices = "left", "right", "center"
pad	Amount of padding to put around element in pixels
expand_x	If True the element will automatically expand in the "X" direction to fill available space
expand_y	If True the element will automatically expand in the "Y" direction to fill available space
tooltip	Text that will appear when mouse hovers over the element

The most important method defined in the Text class is the **get()** method that retrieves the current value of the displayed text, to be used programmatically elsewhere. You can also change the displayed text programmatically by capturing the click event, which should be enabled in the constructor.

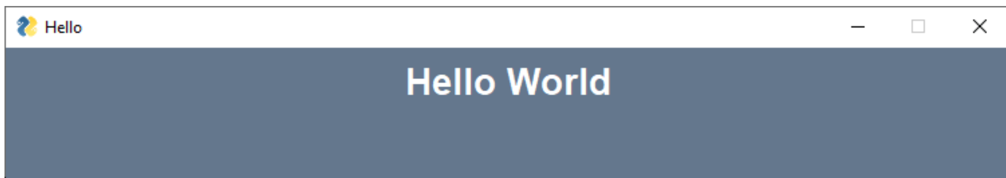
The following example initially displays "Hello World" on the Text element, which changes to "Hello Python", when clicked.

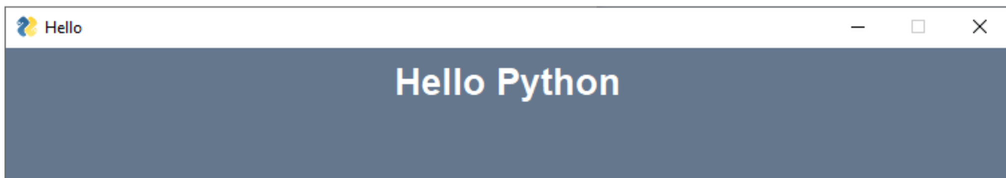
```
import PySimpleGUI as psg


layout = [[psg.Text('Hello World', enable_events=True,
                    key='-TEXT-', font=('Arial Bold', 20),
                    expand_x=True, justification='center')],
]
window = psg.Window('Hello', layout, size=(715, 100))
```


tutorialspoint
SIMPLYEASYLEARNING

```
while True:
    event, values = window.read()
    print(event, values)

    if event == '-TEXT-':
        window['-TEXT-'].update("Hello Python")

    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

Run the above program. Click the label to change its text as shown below:





9. PySimpleGUI – Input Element

This type of widget is most popular in any GUI toolkit. The Input element is based on the Entry widget in TKinter. The object of this class gives a input text field of single line.

In addition to the common properties, those specific to the Input element are as follows:

default_text	Text initially shown in the input box as a default value
disabled	Set disable state for element
use_readonly_for_disable	If True (the default) tkinter state set to 'readonly'. Otherwise state set to 'disabled'
password_char	Password character if this is a password field

The Input class defines the **get()** method which returns the text entered by the user. The **update()** method changes some of the settings for the Input Element. Following properties are defined:

value	new text to display as default text in Input field
disabled	disable or enable state of the element
select	if True, then the text will be selected
visible	change visibility of element
move_cursor_to	Moves the cursor to a particular offset. Defaults to 'end'
password_char	Password character if this is a password field
paste	If True "Pastes" the value into the element rather than replacing the entire element. If anything is selected it is replaced. The text is inserted at the current cursor location.

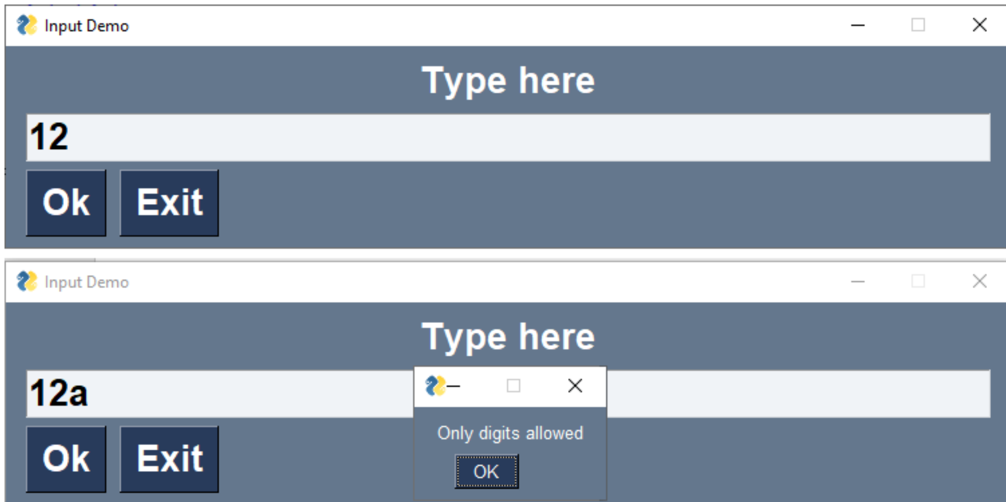
In the example give below, the window has an Input element to accept user input. It is programmed to accept only digits. If any non-digit key is pressed, a message pops up informing that it is not allowed. For that, the last character from the Input is compared with a string made of digit

tutorialspoint
SIMPLYEASYLEARNING

characters. If the last key pressed is not a digit, it is removed from the Input box.

```
import PySimpleGUI as psg

l1 = psg.Text('Type here', key='-OUT-', font=('Arial Bold', 20),
              expand_x=True, justification='center')

t1 = psg.Input('', enable_events=True, key='-INPUT-',
               font=('Arial Bold', 20), expand_x=True,
               justification='left')

b1 = psg.Button('Ok', key='-OK-', font=('Arial Bold', 20))
b2 = psg.Button('Exit', font=('Arial Bold', 20))

layout = [[l1], [t1], [b1, b2]]
window = psg.Window('Input Demo', layout, size=(750, 150))

while True:
    event, values = window.read()
    print(event, values)

    if event == '-INPUT-':
        if values['-INPUT-'][-1] not in ('0123456789'):
            psg.popup("Only digits allowed")
            window['-INPUT-'].update(values['-INPUT-'][:-1])

    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

It will produce the following **output** window:



Multiline Element

If you wish to input a text consisting of mor than one lines, you can use Multiline element instead of Input element. In fact, it is used as an input as well as output element. If the length of the text is more than the height/width of the text entered/displayed, the scroll bars appear to the element.

Following properties are specific to Multiline element:

default_text	Initial text to show
autoscroll	If True the contents of the element will automatically scroll as more data added to the end
auto_size_text	If True will size the element to match the length of the text
horizontal_scroll	Controls if a horizontal scrollbar should be shown. If True a horizontal scrollbar will be shown in addition to vertical
reroute_stdout	If True, then all output to stdout will be output to this element
reroute_cprint	If True, your cprint calls will output to this element.
no_scrollbar	If False, then a vertical scrollbar will be shown (the default)

Like the Input element, the Multiline class also has a **get()** method to retrieve its text content. The **Update()** method changes the values of some properties of this element. For example:

- **value**: new text to display
- **append**: If True, then the new value will be added onto the end of the current value. if False then contents will be replaced.

In the following example, a Multiline textbox is used to display the contents of a text file:

```
import PySimpleGUI as psg


file = open("zen.txt")
text = file.read()


l1 = psg.Text('Multiline Input/Output', font=('Arial Bold', 20),
              expand_x=True, justification='center')
t1 = psg.Multiline(text, enable_events=True, key='-INPUT-',
                   expand_x=True, expand_y=True,
                   justification='left')


b1 = psg.Button('Ok', key='-OK-', font=('Arial Bold', 20))
b2 = psg.Button('Exit', font=('Arial Bold', 20))


layout = [[l1], [t1], [b1, b2]]
window = psg.Window('Multiline Demo', layout, size=(715, 250))


while True:
    event, values = window.read()
    if event == psg.WIN_CLOSED or event == 'Exit':
        break


window.close()
```


tutorialspoint SIMPLY EASY LEARNING

The program will produce the following **output** window:

tutorialspoint
SIMPLYEASYLEARNING

10. PySimpleGUI – Button Element

Almost every GUI window will have at least one button. When a button element is clicked, it initiates a certain action. PySimpleGUI has some button types with predefined caption. They are defined to perform a specific task. Ohers with a user defined caption are capable of doing any required task.

The buttons with predefined caption have a shortcut name. So that a button with OK as a caption can be created in two ways:

```
>>> b1=psg.Button("OK")

# OR

>>> b1=psg.OK()
```

Other such predefined captions are:

- OK
- Ok
- Submit
- Cancel
- Yes
- No
- Exit
- Quit
- Help
- Save
- SaveAs
- Open

In PySimpleGUI, the button event is automatically enabled by default. When clicked, these predefined captions become the name of the event generated.

There are some chooser buttons in PysimpleGUI. When clicked these buttons open a dialog box to let the user choose from it.

- FileBrowse
- FilesBrowse
- FileSaveAs
- FolderBrowse
- CalendarButton
- ColorChooserButton

These special buttons return a string representation of selected object and that value is filled in any other element (such as Input or Multiline) on the window. This element is pointed by the target property.

The value of the target property is represented using (row, col) tuple. The default target is the element in the same row and to the left of this button, represented by (ThisRow, -1) value. ThisRow means the same row, "-1" means the element to the button's immediate left. If a value of target is set to (None, None), then the button itself will hold the information. The value can be accessed by using the button's key.

The target property can also be the key property of target element.

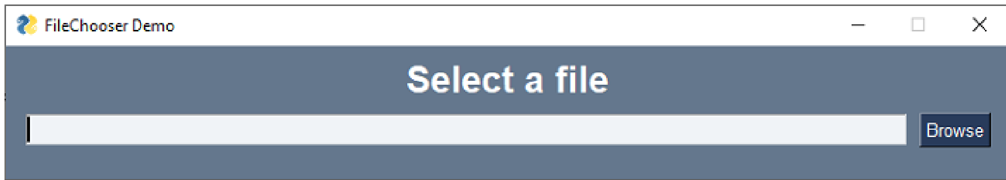
FileBrowse

The FileBrowse button opens a file dialog from which a single file can be selected. In the following code, the path string of the selected file is displayed in the target Input bow in the same row.

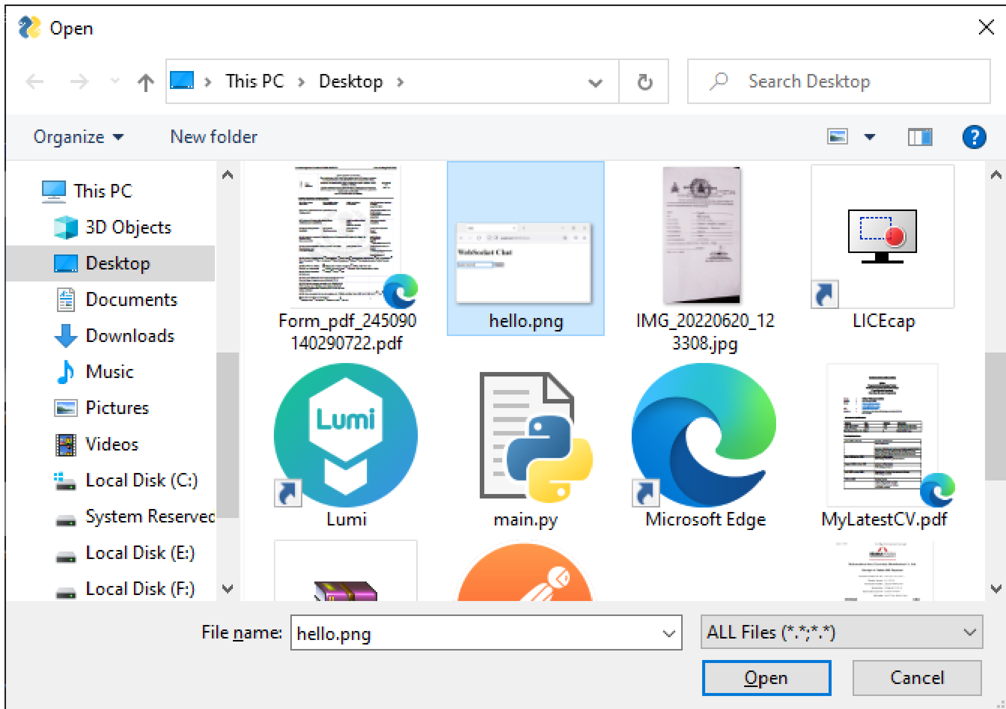
```
import PySimpleGUI as psg


layout = [[psg.Text('Select a file',font=('Arial Bold', 20),

         expand_x=True, justification='center')],

    [psg.Input(enable_events=True, key='-IN-',font=('Arial
Bold', 12),expand_x=True),

           psg.FileBrowse()]]

window = psg.Window('FileChooser Demo', layout,
size=(715,100))


while True:

    event, values = window.read()

    if event == psg.WIN_CLOSED or event == 'Exit':

        break


window.close()
```


tutorialspoint
SIMPLYEASYLEARNING

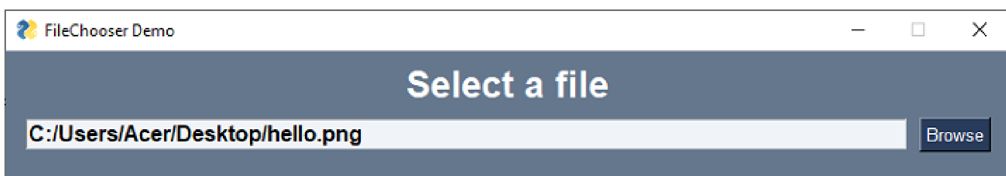
This code renders the following window:



Click on the Browse button to display the **file dialog**:



The selected file name along with its path is displayed in the Input box.

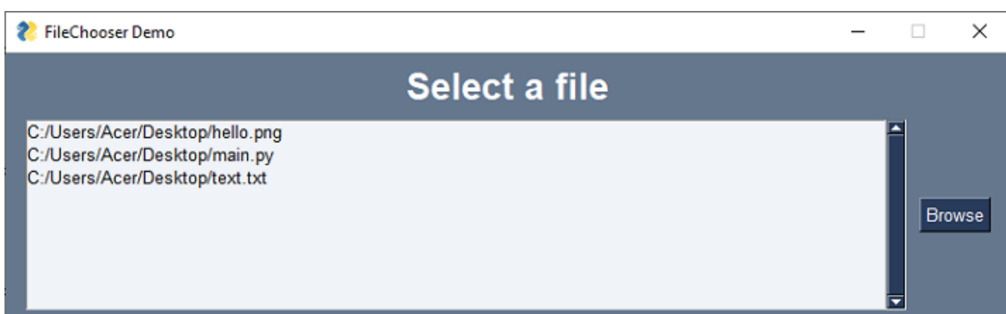


FilesBrowse

This element allows the user to select multiple files. The return string is the concatenation of files, separated by ";" character. We shall populate a list box with the selected files by the following code.

```
import PySimpleGUI as psg

layout = [[psg.Text('Select a file', font=('Arial Bold', 20),
                    expand_x=True, justification='center')],
          [psg.LBox([], size=(20, 10), expand_x=True,
                    expand_y=True, key='-LIST-'),
           psg.Input(visible=False, enable_events=True, key='-IN-',
                     font=('Arial Bold', 10), expand_x=True),
           psg.FilesBrowse()]
          ]
window = psg.Window('FileChooser Demo', layout, size=(715, 200))

while True:
    event, values = window.read()
    if event == '-IN-':
        window['-LIST-'].Update(values['-IN-'].split(';'))
    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

Here, the Input element with "-IN-" key is hidden by setting the "visible" property to False. Still, it contains the ";" separated list of selected files. The string is split at occurrence of ";" character and the list below is file with the file names.



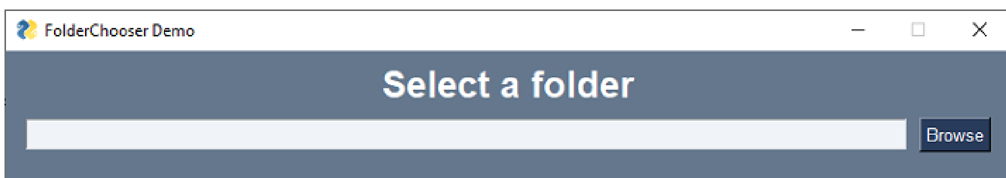
FolderBrowse

This element works similar to the **FileBrowse** element. It is used to select the current folder. It may be used to set the selected folder as default for subsequent file related operations.

You can set the "initial_folder" property of this element to a folder name (along with its path) to open the folder dialog with that folder opened to start with.

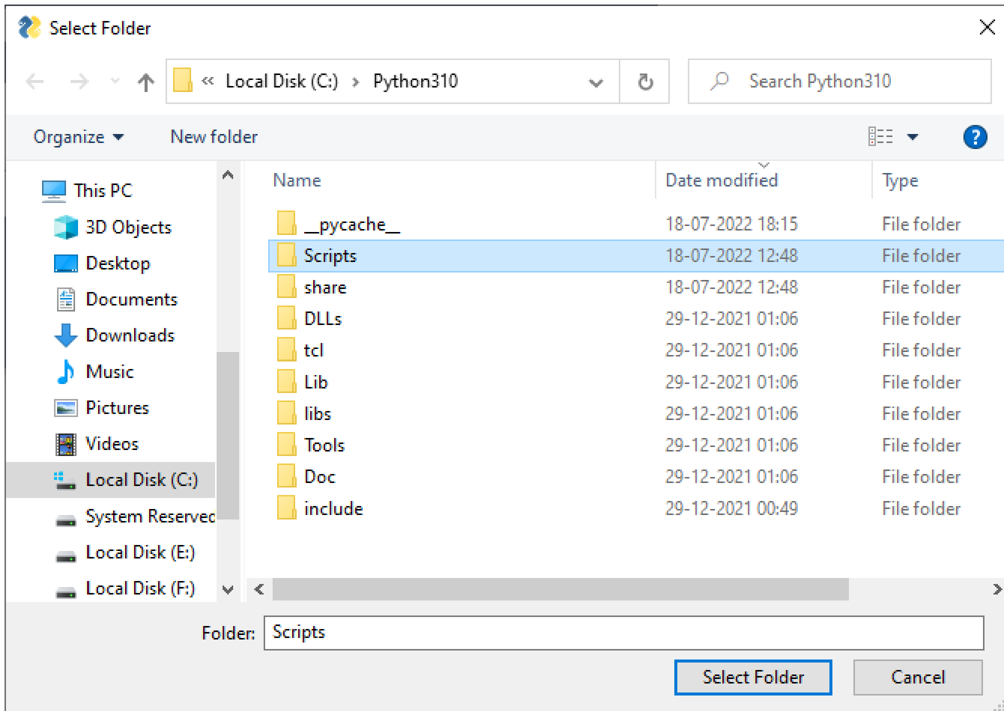
```
import PySimpleGUI as psg


layout = [
    [psg.Text('Select a folder', font=('Arial Bold', 20),
              expand_x=True, justification='center')],
    [psg.Input(enable_events=True, key='-IN-',
               font=('Arial Bold', 12), expand_x=True),
     psg.FolderBrowse(initial_folder="c:/Python310")]
]
window = psg.Window('FolderChooser Demo', layout, size=(715,
100))


while True:
    event, values = window.read()
    if event == psg.WIN_CLOSED or event == 'Exit':
        break


window.close()
```

A window with Browse button is displayed.

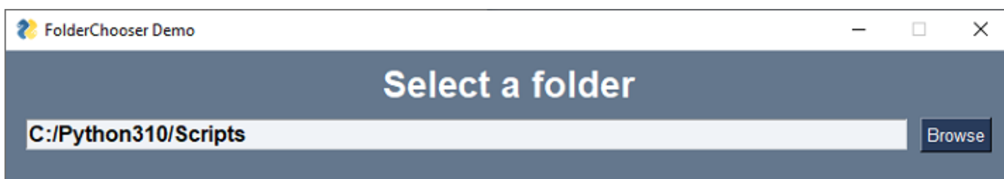


tutorialspoint
SIMPLYEASYLEARNING

The **folder dialog** opens when it is clicked.



The path of the selected folder is displayed in the Input text field.



FileSaveAs

This button also opens a file dialog, but provides a save button so that information on the PySimpleGUI window can be saved by the name given by the user. The SaveAs dialog can be customized by the following properties. We can apply filter on file types to be selected, and set the initial folder for browsing action.

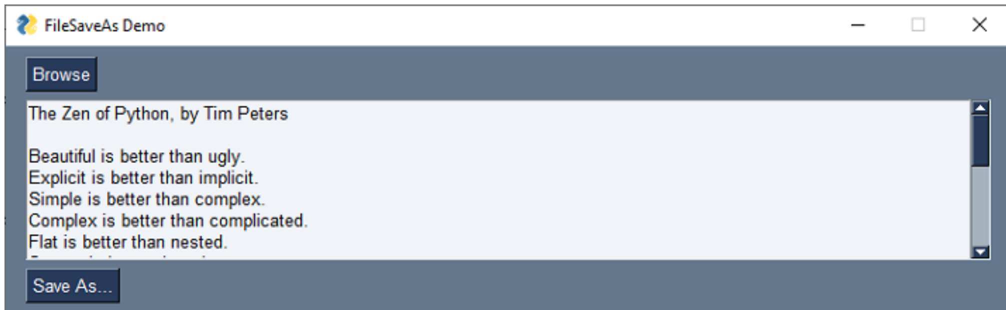
file_types	Default value = (("ALL Files", "*.* *"),)
default_extension	If no extension entered by user, add this to filename
initial_folder	Starting path for folders and files

tutorialspoint
SIMPLYEASYLEARNING

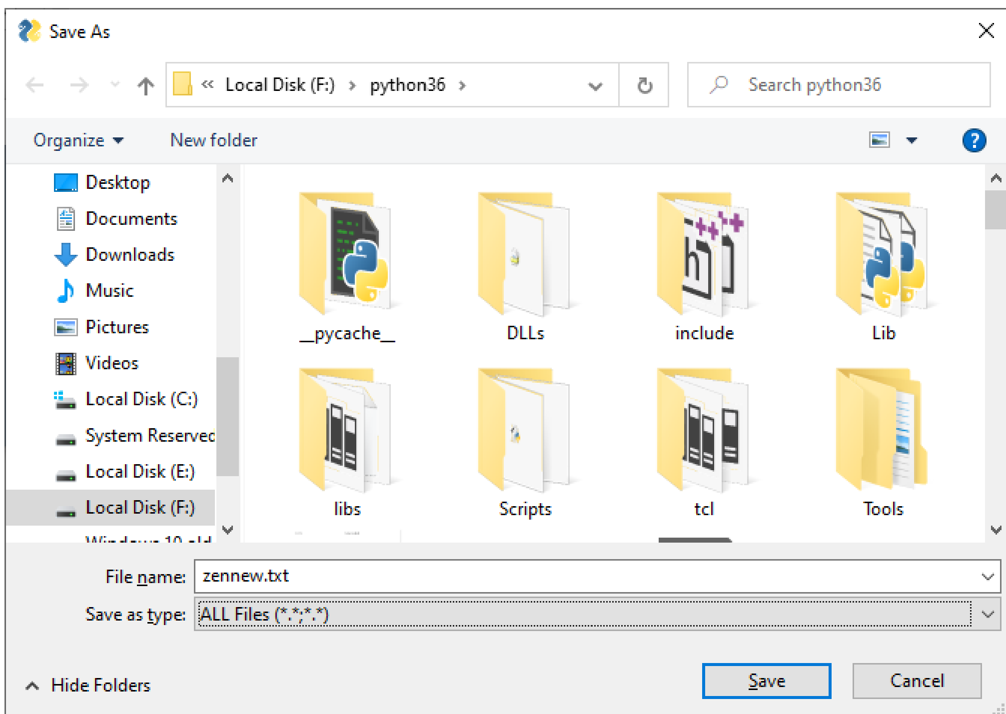
In the following example, a FileBrowse button allows you to read the contents of a file and display in a Multiline text box. Click on the SaveAS button to save the displayed text as a new file name.

```
import PySimpleGUI as psg

t1 = psg.Input(visible=False, enable_events=True, key='-T1-',
               font=('Arial Bold', 10), expand_x=True)

t2 = psg.Input(visible=False, enable_events=True, key='-T2-',
               font=('Arial Bold', 10), expand_x=True)

t3 = psg.Multiline("", enable_events=True, key='-INPUT-',
                   expand_x=True, expand_y=True,
                   justification='left')

layout = [[t1, psg.FilesBrowse()], [t3], [t2, psg.FileSaveAs()]]
window = psg.Window('FileSaveAs Demo', layout, size=(715, 200))

while True:
    event, values = window.read()
    if event == '-T1-':
        file = open(t1.get())
        txt = file.read()
        window['-INPUT-'].Update(value=txt)
    if event == '-T2-':
        file = open(t2.get(), "w")
        file.write(t3.get())
        file.close()
    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

Select a text file. Its contents will be displayed in the textbox.



Choose the name and destination folder to save the text in a new file.



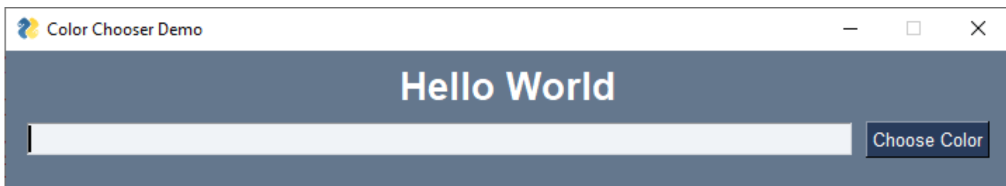
ColorChooserButton

This button brings up a color dialog. You can choose a color from the swatches, or using the slider, or setting the RGB values from the spinner. The dialog box returns the Hex string of the RGB value of the selected colour. It is displayed in the target input control, and it can be further used to change the color relate property of any element.

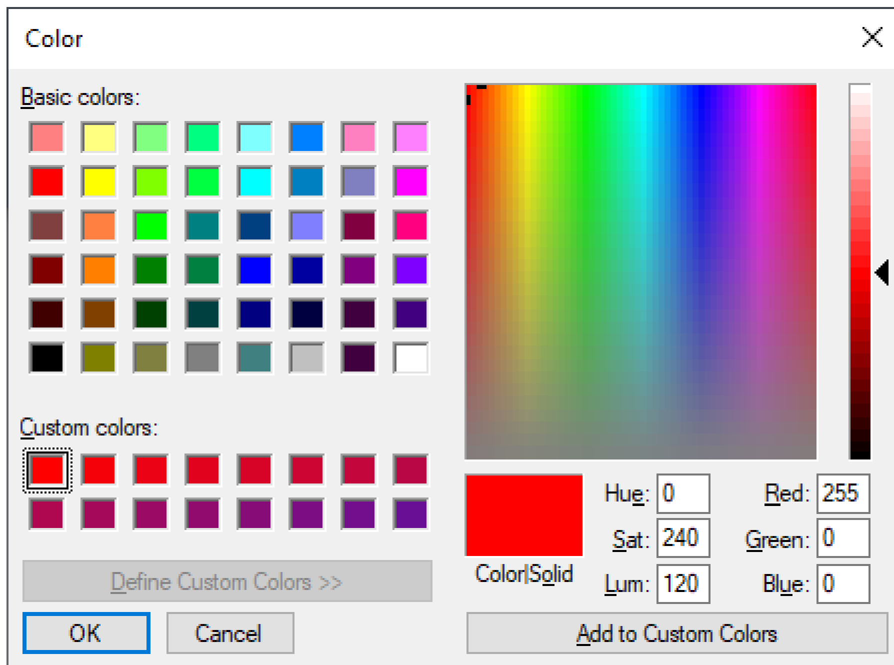
In the following example, the chosen color is used to update the "text_color" property of the Text element displaying Hello World string.

```
import PySimpleGUI as psg

layout = [[psg.Text('Hello World', font=('Arial Bold', 20),
                    expand_x=True, justification='center',
                    key='-T1-')],
          [psg.Input(enable_events=True, key='-IN-',
                     font=('Arial Bold', 12), expand_x=True),
           psg.ColorChooserButton("Choose Color")]
          ]
window = psg.Window('Color Chooser Demo', layout, size=(715, 100))

while True:
    event, values = window.read()
    print(event, values)
    if event == '-IN-':
        window['-T1-'].update(text_color=values['-IN-'])
    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

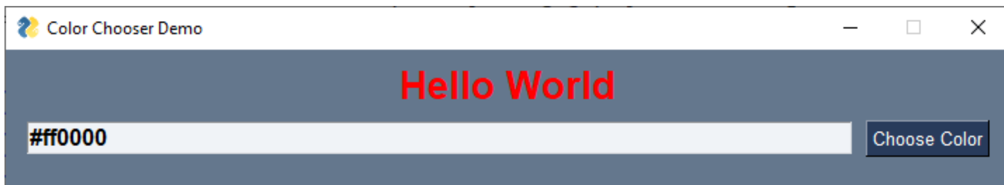
A window with ColorChooserButton with Choose Color caption appears.



Click the button to open the color dialog.



Choose the desired colour and press OK. The Hex string corresponding to it will be returned and displayed in the target Input element. The **get()** method of the Input element is used to fetch it and update the **text_color** property of Hello World text.

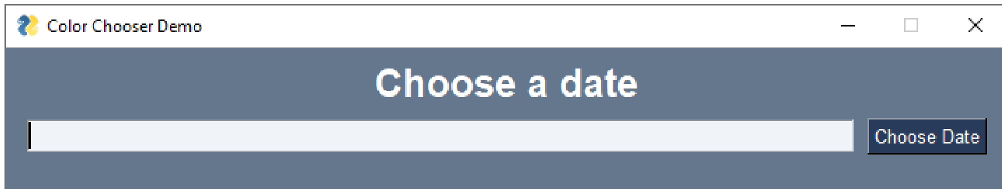


CalendarButton

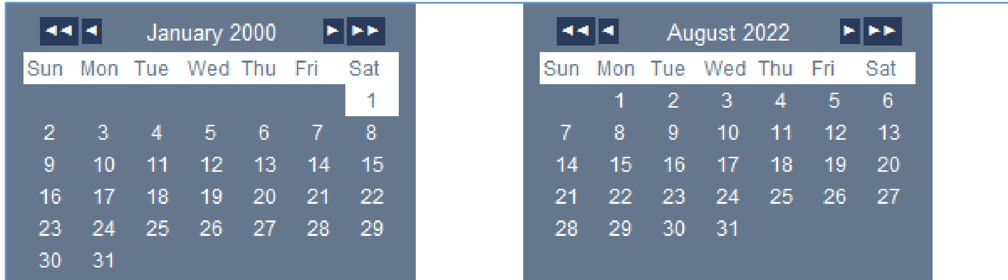
This button shows a calendar chooser window. The target element is filled with return value as a string. Following important properties are defined in CalendarButton class:

button_text	Text in the button
default_date_m_d_y	Beginning date to show
locale	Defines the locale used to get day names
month_names	Optional list of month names to use (should be 12 items)
day_abbreviations	Optional list of abbreviations to display as the day of week
title	Title shown on the date chooser window

Here is a simple example demonstrating how to use the CalendarButton:



It initially shows a Calendar Button. Click it to open the calendar dialog:



Using the selector arrows, choose the required date. It will be displayed in the window.

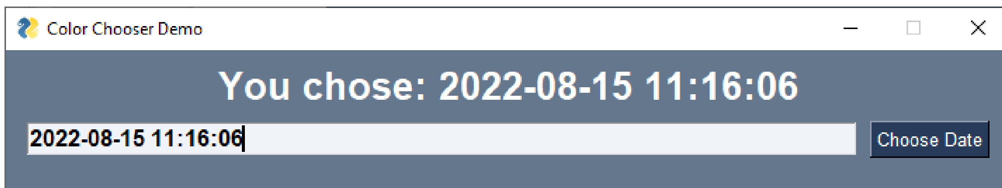


Image Button

Instead of a text caption, an image can be displayed on the face of a button. The button class has an "image_filename" property. Assign the name of the image to be displayed. The image should be PNG or GIF type.

The "image_filename" property of the Button object may be set to an image file that you want to display on the button.

In the following example, the Add, Subtract and Exit buttons have images instead of captions. To capture their click event, their key parameter is used.

```
import PySimpleGUI as psg


layout = [
    [psg.Text('Enter a num:  '), psg.Input(key='-FIRST-')],
    [psg.Text('Enter a num:  '), psg.Input(key='-SECOND-')],
```

```
    [psg.Text('Result     :  '), psg.Text(key='-OUT-')],
    [psg.Button(key="Add", image_filename="plus.png"),
     psg.Button(key="Sub", image_filename="minus.png"),
     psg.Button(key="Exit", image_filename="close.png")],
]
window = psg.Window('Calculator', layout, size=(715, 200),
                    enable_close_attempted_event=True)
while True:
    event, values = window.read()
    print(event, values)
    if event == "Add":
        result = int(values['-FIRST-']) + int(values['-SECOND-'])
    if event == "Sub":
        result = int(values['-FIRST-']) - int(values['-SECOND-'])
    window['-OUT-'].update(result)

    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

Given below is the **result** of the above code:

11. PySimpleGUI – ListBox Element

This GUI element in PySimpleGUI toolkit is a container that can display one or more items, and select from it. You can specify the number of items that can be visible at a time. If the number of items or their length becomes more than the dimensions of the Listbox, a vertical and/or horizontal scrollbar appears towards the right or bottom of the element.

Important properties of the ListBox class are as follows:

Values	List of values to display. Can be any type including mixed types
default_values	Which values should be initially selected
select_mode	Select modes are used to determine if only 1 item can be selected or multiple and how they can be selected.
no_scrollbar	Controls if a scrollbar should be shown. If True, no scrollbar will be shown
horizontal_scroll	Controls if a horizontal scrollbar should be shown. If True a horizontal scrollbar will be shown in addition to vertical

The "select_mode" property can have one of the following enumerated values:

- LISTBOX_SELECT_MODE_SINGLE (default)
- LISTBOX_SELECT_MODE_MULTIPLE
- LISTBOX_SELECT_MODE_BROWSE
- LISTBOX_SELECT_MODE_EXTENDED

The Listbox class inherits the **update()** method from the Element class. It effects changes in some of the properties when the window gets updated. The parameters to the update() method are:

Values	New list of choices to be shown to user
Disabled	Disable or enable state of the element
set_to_index	Highlights the item(s) indicated. If **parm** is an **int**, one entry will be set. If is a list, then each entry in the list is highlighted

scroll_to_index	Scroll the listbox so that this index is the first shown
select_mode	Changes the select mode
Visible	Control visibility of element

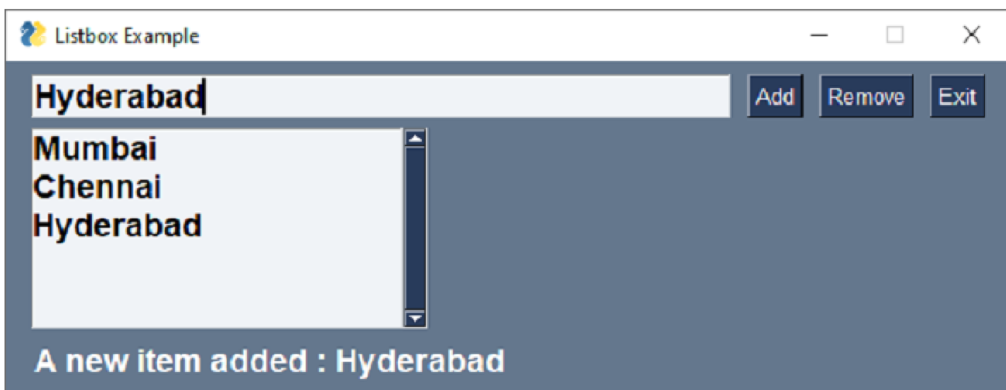
The Listbox element is in action in the following program. The PySimpleGUI window shows an Input element, a Listbox and the buttons with captions Add, Remove and Exit.

```
import PySimpleGUI as psg


names = []
lst = psg.Listbox(names, size=(20, 4),
                  font=('Arial Bold', 14), expand_y=True,
                  enable_events=True, key='-LIST-')


layout = [[psg.Input(size=(20, 1), font=('Arial Bold', 14),
                     expand_x=True, key='-INPUT-'),
           psg.Button('Add'),
           psg.Button('Remove'),
           psg.Button('Exit')],
          [lst],
          [psg.Text("", key='-MSG-',
                    font=('Arial Bold', 14),
                    justification='center')]
          ]
window = psg.Window('Listbox Example', layout, size=(600, 200))


while True:
    event, values = window.read()
    print(event, values)
    if event in (psg.WIN_CLOSED, 'Exit'):
```


tutorialspoint
SIMPLYEASYLEARNING

```
        break
    if event == 'Add':
        names.append(values['-INPUT-'])
        window['-LIST-'].update(names)
        msg = "A new item added : {}".format(values['-INPUT-'])
        window['-MSG-'].update(msg)
    if event == 'Remove':
        val = lst.get()[0]
        names.remove(val)
        window['-LIST-'].update(names)
        msg = "A new item removed : {}".format(val)
        window['-MSG-'].update(msg)

window.close()
```

Run the above code, type some text in the Input box and press Add button. The text will be added in the listbox below it.



The get() method of the Listbox class returns the list of selected items. By default, a single item is selectable. The remove button gets the value of the selected item and removes it from the collection.

12. PySimpleGUI – Combo Element

The Combo element is a drop down list. It initially shows an Input element with an arrow towards its right hand side. When the arrow is clicked, the list box pulls down. So, you can enter text in the Input text box, or select an item from the drop down list, so that the selected item shows up in the Input box.

The Combo element functions more or less similarly to Listbox. It is populated by a collection of string items in a list. You can also specify the default value to be displayed on top.

Following are the important properties of the Combo class:

values	list of values to be displayed and to choose.
default_value	Choice to be displayed as initial value.
size	width, height. Width = characters-wide, height = the number of entries to show in the list.

The get() method returns the current (right now) value of the Combo. The update() method modifies following properties of the Combo object:

value	change which value is current selected based on new list of previous list of choices
values	change list of choices
set_to_index	change selection to a particular choice starting with index = 0
readonly	if True make element readonly (user cannot change any choices).

In the following example, we use the selection changed event of the Combo element. The selected element in the dropdown is removed if the user responds by pressing Yes on the Popup dialog.

```
import PySimpleGUI as psg


names = []
```

```
lst = psg.Combo(names, font=('Arial Bold', 14),
                expand_x=True, enable_events=True,
                readonly=False, key='-COMBO-')

layout = [[lst,
           psg.Button('Add', ),
           psg.Button('Remove'),
           psg.Button('Exit')],
          [psg.Text("", key='-MSG-',
                    font=('Arial Bold', 14),
                    justification='center')]
          ]
window = psg.Window('Combobox Example',
                    layout, size=(715, 200))

while True:
    event, values = window.read()
    print(event, values)
    if event in (psg.WIN_CLOSED, 'Exit'):
        break
    if event == 'Add':
        names.append(values['-COMBO-'])
        print(names)
        window['-COMBO-'].update(values=names, value=values['-COMBO-'])
        msg = "A new item added : {}".format(values['-COMBO-'])
        window['-MSG-'].update(msg)

    if event == '-COMBO-':
        ch = psg.popup_yes_no("Do you want to Continue?",
title="YesNo")
```


tutorialspoint
SIMPLYEASYLEARNING

```
        if ch == 'Yes':
            val = values['-COMBO-']
            names.remove(val)
            window['-COMBO-'].update(values=names, value=' ')
            msg = "A new item removed : {}".format(val)
            window['-MSG-'].update(msg)
window.close()
```

When the Combo object emits the event (identified by its key "-COMBO-") as an item in the dropdown is clicked. A Yes-No popup is displayed asking for the confirmation. If the Yes button is clicked, the item corresponding to the text box of the Combo element is removed from the list and the element is repopulated by the remaining items.

A screenshot of the window is shown below:

tutorialspoint
SIMPLYEASYLEARNING

13. PySimpleGUI – Radio Element

A Radio button is a type of toggle button. Its state changes to True to False and vice versa on every click. A caption appears to the right of a circular clickable region the dot selection indicator in it.

When more than one radio buttons are added as members of a group, they are mutually exclusive, in the sense only one of the buttons can have True state and others become False.

Apart from the common properties inherited from the Element class, the Radio object has following properties important in the context of a Radio button:

- **text**: Text to display next to button
- **group_id**: Groups together multiple Radio Buttons.
- **default**: Set to True for the one element of the group you want initially selected

If the "enable_events" property is set to True for all the buttons having same group_id, the selection changed event is transmitted.

The "get()" method returns True if it is selected, false otherwise. The "update()" method is overridden to modify the properties of the Radio element. These properties are:

- **value**: if True change to selected and set others in group to unselected
- **text**: Text to display next to radio button
- **disabled**: disable or enable state of the element

In the following example, three groups of radio buttons are used. The code computes interest on a loan amount. The interest rate depends on the gender (less by 0.25% for female), period and the type of loan (personal or business – 3% more for business loan) selected by the user.

```
import PySimpleGUI as psg


psg.set_options(font=("Arial Bold", 14))
l1 = psg.Text("Enter amount")
```

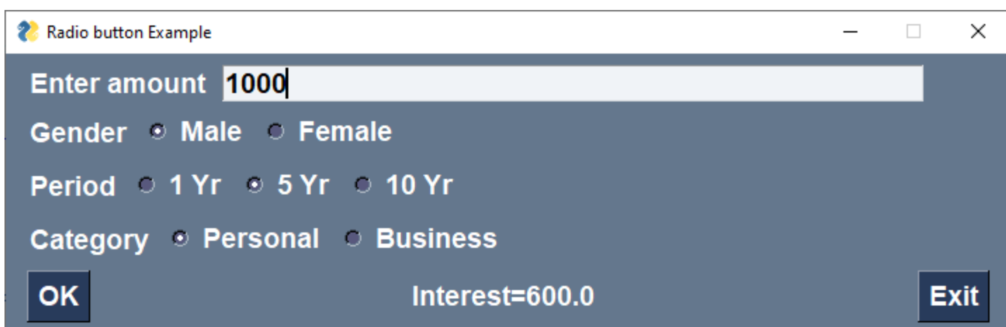

tutorialspoint
SIMPLYEASYLEARNING

```
l2 = psg.Text("Gender")
l3 = psg.Text("Period")
l4 = psg.Text("Category")
l5 = psg.Text(" ", expand_x=True,
              key='-OUT-', justification='center')

t1 = psg.Input("", key='-AMT-')

r11 = psg.Radio("Male", "gen", key='male', default=True)
r12 = psg.Radio("Female", "gen", key='female')
r21 = psg.Radio("1 Yr", "per", key='one')
r22 = psg.Radio("5 Yr", "per", key='five', default=True)
r23 = psg.Radio("10 Yr", "per", key='ten')
r31 = psg.Radio("Personal", "ctg", key='per', default=True)
r32 = psg.Radio("Business", "ctg", key='bus')

b1 = psg.Button("OK")
b2 = psg.Button("Exit")

layout = [[l1, t1], [l2, r11, r12],
          [l3, r21, r22, r23], [l4, r31, r32],
          [b1, l5, b2]
          ]
window = psg.Window('Radio button Example', layout, size=(715, 200))

while True:
    rate = 12
    period = 5
    event, values = window.read()
    print(event, values)
```


tutorialspoint
SIMPLYEASYLEARNING

```
    if event in (psg.WIN_CLOSED, 'Exit'):
        break

    if event == 'OK':
        if values['female'] == True: rate = rate - 0.25
        if values['one'] == True:
            rate = rate + 1
            period = 1
        if values['ten'] == True:
            rate = rate - 1
            period = 10
        if values['bus'] == True: rate = rate + 3
        amt = int(values['-AMT-'])
        print(amt, rate, period)
        interest = amt * period * rate / 100
        window['-OUT-'].update("Interest={}".format(interest))

window.close()
```

It will produce the following **output** window:



14. PySimpleGUI – Checkbox Element

The Checkbox is also a toggle button having two states: **checked** and **unchecked**. It presents a rectangular box which when clicked displays a check mark (or removes it when it already has one) and a caption next to it.

Usually, checkbox controls are provided to let the user select one or more items from the available options. Unlike the Radio button, the Checkboxes on the GUI window do not belong to any group. Hence, user can make multiple selections.

The object of Checkbox class is declared with following specific parameters

```
PySimpleGUI.Checkbox(text, default, checkbox_color)
```

These are the properties specific to Checkbox class:

- **text**: This is a string, representing the text to display next to checkbox
- **default**: Set to True if you want this checkbox initially checked
- **checkbox_color**: You can specify the color of background of the box that has the check mark in it.

Apart from these, other common keyword arguments to set the properties inherited from the Element class can be given to the constructor.

The two important methods inherited but overridden in the Checkbox class are:

- **get()**: It return the current state of this checkbox
- **update()**: The Checkbox emits the selection changed event. One or more properties of the Checkbox element are updated in response to an event on the window. These properties are:
- **value**: if True checks the checkbox, False clears it
- **text**: Text to display next to checkbox

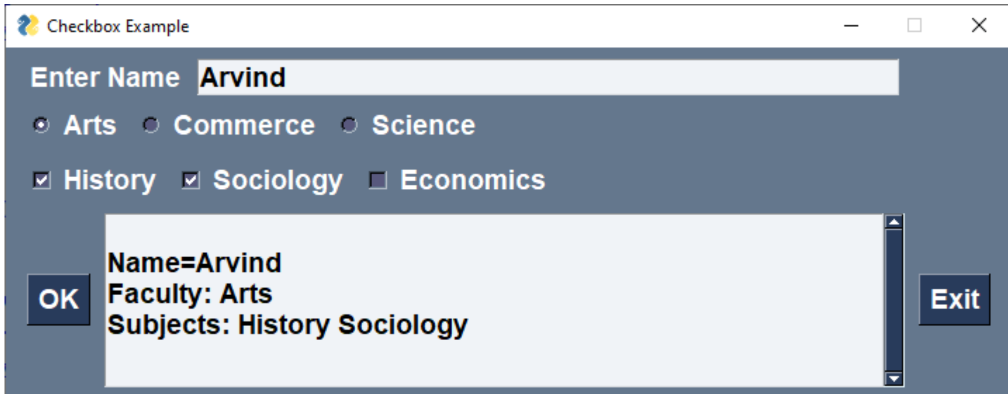
In the following example, a group of three radio buttons represent the faculty streams available in a college. Depending on the faculty chosen, three subjects of that faculty are made available for the user to select one or more from the available options.

```
import PySimpleGUI as psg

psg.set_options(font=("Arial Bold",14))
l1=psg.Text("Enter Name")
l2=psg.Text("Faculty")
l3=psg.Text("Subjects")
l4=psg.Text("Category")
l5=psg.Multiline(" ", expand_x=True, key='-OUT-',
                 expand_y=True,justification='left')
t1=psg.Input("", key='-NM-')

rb=[]
rb.append(psg.Radio("Arts", "faculty", key='arts',
enable_events=True,default=True))
rb.append(psg.Radio("Commerce", "faculty", key='comm',
enable_events=True))
rb.append(psg.Radio("Science", "faculty",
key='sci',enable_events=True))

cb=[]
cb.append(psg.Checkbox("History", key='s1'))
cb.append(psg.Checkbox("Sociology", key='s2'))
cb.append(psg.Checkbox("Economics", key='s3'))

b1=psg.Button("OK")
b2=psg.Button("Exit")

layout=[[l1, t1],[rb],[cb],[b1, l5, b2]]
window = psg.Window('Checkbox Example', layout, size=(715,250))
```

```
while True:
    event, values = window.read()
    print (event, values)

    if event in (psg.WIN_CLOSED, 'Exit'): break
    if values['comm']==True:
        window['s1'].update(text="Accounting")
        window['s2'].update(text="Business Studies")
        window['s3'].update(text="Statistics")
    if values['sci']==True:
        window['s1'].update(text="Physics")
        window['s2'].update(text="Mathematics")
        window['s3'].update(text="Biology")
    if values['arts']==True:
        window['s1'].update(text="History")
        window['s2'].update(text="Sociology")
        window['s3'].update(text="Economics")
    if event=='OK':
        subs=[x.Text for x in cb if x.get()==True]
        fac=[x.Text for x in rb if x.get()==True]
        out="""

Name={}
Faculty: {}
Subjects: {}
""".format(values['-NM-'], fac[0], " ".join(subs))
        window['-OUT-'].update(out)

window.close()
```

Run the above code. Select a faculty name and mark checks in the corresponding check buttons to register the selection. Note that the subjects change as the faculty option is changed.

Press the OK button so that the choices are printed in the Multiline box, as shown below:



15. PySimpleGUI – Slider Element

The Slider widget comprises of a horizontal or vertical bar over which a slider button can be moved across with the help of mouse. The length of the bar indicates a range of a numerical parameter (such as font size, length/width of a rectangle etc.). The manual movement of the slider button changes the instantaneous value of the parameter, which can be further used in the program.

The object of Slider class is declared as follows:

```
PySimpleGUI.Slider(range, default_value, resolution,
orientation, tick_interval)
```

These parameters are specific to the Slider control. The description of these parameters is given below:

- **range:** The slider's bar represents this range (min value, max value)
- **default_value:** starting value to which the slider button is set in the beginning
- **resolution:** the smallest amount by which the value changes when the slider is moved
- **tick_interval:** The frequency of a visible tick should be shown next to slider
- **orientation:** either 'horizontal' or 'vertical'
- **disable_number_display:** if True no number will be displayed by the Slider Element

Other attributes inherited from the Element class, such as color, size, font etc can be used to further customize the Slider object.

The update() method of the Slider class helps in refreshing the following parameters of the Slider object:

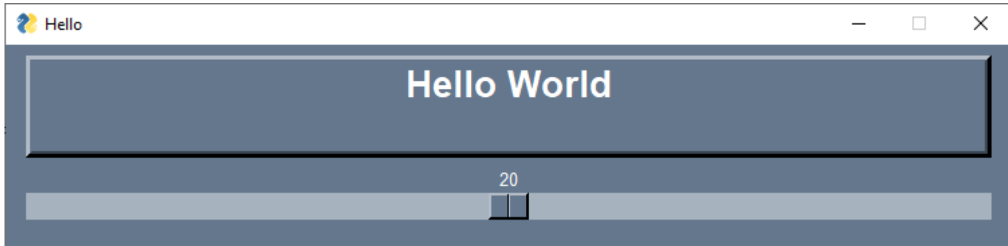
- **value:** sets current slider value
- **range:** Sets a new range for slider

the following code generates a PysimpleGUI window showing a Text Label with Hello World caption. There is a horizontal slider whose value changes from 10 to 30. Its key parameter is "-SL-".

Whenever the slider button is moved across, the "-SL-" event occurs. The instantaneous value of the slider button is used as the font size and the Text caption is refreshed.

```
import PySimpleGUI as psg

layout = [

    [psg.Text('Hello World', enable_events=True,
              key='-TEXT-', font=('Arial Bold', 20),
              size=(50, 2), relief="raised", border_width=5,
              expand_x=True, justification='center')],

    [psg.Slider(range=(10, 30), default_value=12,
                expand_x=True, enable_events=True,
                orientation='horizontal', key='-SL-')]
]
window = psg.Window('Hello', layout, size=(715, 150))

while True:
    event, values = window.read()
    print(event, values)

    if event == psg.WIN_CLOSED or event == 'Exit':
        break
    if event == '-SL-':
        window['-TEXT-'].update(font=('Arial Bold', int(values['-SL-'])))

window.close()
```

Save and run the above code. As you move the slider button, the font size of the Hello World text keeps changing. The output window will appear as follows:



16. PySimpleGUI – Spin Element

The object of Spin class in PysimpleGUI library is also a selection widget. Its visual appearance shows a non-editable text box with up/down buttons to the right side. It is capable of displaying any one item from a list of numbers or strings.

As the up or down button is pressed, the index of the item to be displayed increments or decrements and the next or previous item in the list is displayed in the control's text box. The displayed value may be used as required in the program logic.

The parameters to the Spin() constructor are:

```
PySimpleGUI.Spin(values, initial_value, disabled, readonly, size)
```

Where,

- **values:** List or tuple of valid values - numeric or string
- **initial_value:** Any one item from the supplied list to be initially displayed
- **disabled:** set disable state
- **readonly:** Turns on the Spin element events when up/down button is clicked
- **size:** (w, h) w=characters-wide, h=rows-high.

The **get()** method of the Spin class returns the current item displayed in its text box. On the other hand, the **update()** method is used to dynamically change following properties of the Spin element:

- **value:** Set the current value from list of choices
- **values:** Set new list object as available choices

The Spin element generates the selection changed event identified by the key parameter when the up/down button is clicked.

In the following example, we construct a simple date selector with the help of three Spin elements – for date, name of month and year between 2000 to 2025. Ranges for date and year elements are numeric whereas for the month spin element, the range is of strings.

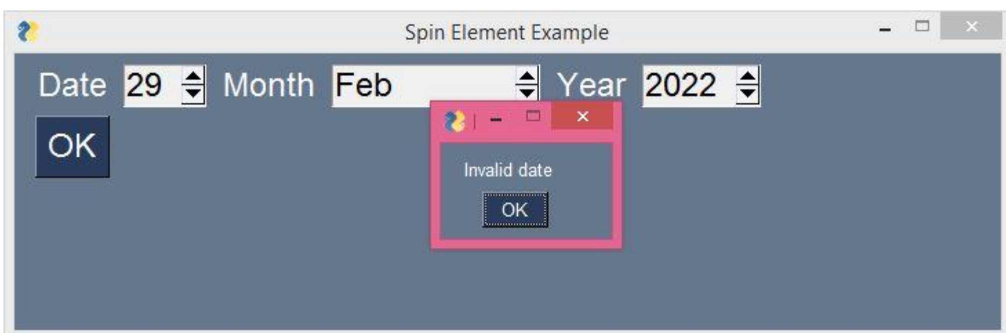
tutorialspoint
SIMPLYEASYLEARNING

```
import PySimpleGUI as psg
import calendar
from datetime import datetime

dates = [i for i in range(1, 32)]
s1 = psg.Spin(dates, initial_value=1, readonly=True,
              size=3, enable_events=True, key='-DAY-')

months = calendar.month_abbr[1:]
s2 = psg.Spin(months, initial_value='Jan', readonly=True,
              size=10, enable_events=True, key='-MON-')

yrs = [i for i in range(2000, 2025)]
s3 = psg.Spin(yrs, initial_value=2000, readonly=True,
              size=5, enable_events=True, key='-YR-')

layout = [
    [psg.Text('Date'), s1, psg.Text("Month"), s2, psg.Text("Year"), s3],
    [psg.OK(), psg.Text("", key='-OUT-')]
]

window = psg.Window('Spin Element Example',
                    layout, font='_ 18', size=(700, 100))

while True:
    event, values = window.read()
    if event == 'OK':
        datestr = str(values['-DAY-']) + " " \
                  + values['-MON-'] + "," \
                  + str(values['-YR-'])
        try:
            d = datetime.strptime(datestr, '%d %b,%Y')
            window['-OUT-'].update("Date: {}".format(datestr))
```

```
        except:
            window['-OUT-'].update("")
            psg.Popup("Invalid date")

    if event == psg.WIN_CLOSED:
        break

window.close()
```

Set the Spin elements to the desired date value and press OK. If the date string represents a valid date, it is displayed in the Text element in the bottom row.

If the date string is incorrect (for example, 29-Feb-2022), then a popup appears indicating that the value is invalid.



tutorialspoint
SIMPLYEASYLEARNING

17. PySimpleGUI – ProgressBar element

Sometimes, a computer operation may be very lengthy, taking a lot of time to complete. Therefore, the user may get impatient. Hence it is important to let him know the state of the application's progress. The ProgressBar element gives a visual indication of the amount of the process completed so far. It is a vertical or horizontal coloured bar that is incrementally shaded by a contrasting colour to show that the process is in progress.

The ProgressBar constructor has the following parameters, in addition to those common parameters inherited from the Element class:

```
PySimpleGUI.ProgressBar(max_value, orientation, size, bar_color)
```

The **max_value** parameter is required to calibrate the width or height of the bar. **Orientation** is either horizontal or vertical. The **size** is (chars long, pixels wide) if horizontal, and (chars high, pixels wide) if vertical. The **bar_color** is a tuple of two colors that make up a progress bar.

The **update()** method modifies one or more of the following properties of the ProgressBar object:

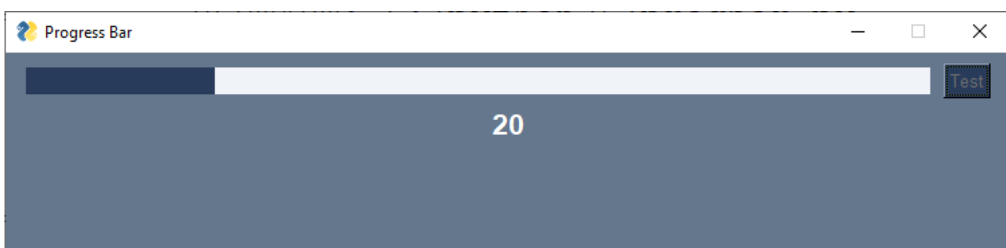
- **current_count:** sets the current value
- **max:** changes the max value
- **bar_color:** The two colors that make up a progress bar. First color shows the progress. Second color is the background.

Given below is a simple demonstration of how a ProgressBar control is used. The layout of the window consists of a progress bar and a Test button. When it is clicked, a for loop ranging from 1 to 100 starts

```
import PySimpleGUI as psg
import time

layout = [
    [psg.ProgressBar(100, orientation='h',
                     expand_x=True, size=(20, 20),
                     key='-PBAR-'), psg.Button('Test')],
    [psg.Text('', key='-OUT-', enable_events=True,
```

```
                font=('Arial Bold', 16), justification='center',
                expand_x=True)]
]
window = psg.Window('Progress Bar', layout, size=(715, 150))

while True:
    event, values = window.read()
    print(event, values)

    if event == 'Test':
        window['Test'].update(disabled=True)
        for i in range(100):
            window['-PBAR-'].update(current_count=i + 1)
            window['-OUT-'].update(str(i + 1))
            time.sleep(1)
        window['Test'].update(disabled=False)

    if event == 'Cancel':
        window['-PBAR-'].update(max=100)

    if event == psg.WIN_CLOSED or event == 'Exit':
        break

window.close()
```

It will produce the following **output** window:



18. PySimpleGUI – Frame Element

The Frame element is a container object that holds on or more elements of other types. It helps in organizing the GUI elements in a logical manner. For example, multiple radio button elements belonging to same group are put inside a frame. It forms a rectangular border around the elements. The frame can have a label and can be placed as per requirement.

```
PySimpleGUI.Frame(title, layout, title_location)
```

The title parameter is the text that is displayed as the Frame's "label" or title. The Frame object can be seen as a child layout of the layout of the main window. It may also be a list of list of elements.

The "title_location" is an enumerated string that decides the position of the label to the frame. The predefined values are TOP, BOTTOM, LEFT, RIGHT, TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, and BOTTOM_RIGHT.

The Frame object is not normally used as an event listener. Still, when clicked on the area of frame, its title can be updated although this feature is rarely used.

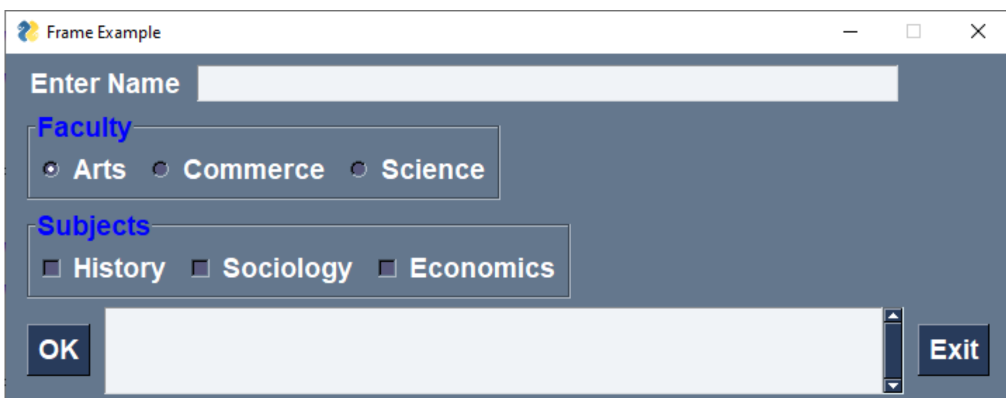
The following code is the same that was used as the example of checkbox. Here, the three radio buttons for choosing the Faculty and the subjects in the chosen faculty as checkboxes are put in separate frames.

```
import PySimpleGUI as psg

psg.set_options(font=("Arial Bold", 14))

l1 = psg.Text("Enter Name")
l2 = psg.Text("Faculty")
l3 = psg.Text("Subjects")
l4 = psg.Text("Category")
l5 = psg.Multiline(" ", expand_x=True, key='-OUT-',
                   expand_y=True, justification='left')

t1 = psg.Input("", key='-NM-')
```


tutorialspoint
SIMPLYEASYLEARNING

```
rb = []
rb.append(psg.Radio("Arts", "faculty", key='arts',
                    enable_events=True, default=True))
rb.append(psg.Radio("Commerce", "faculty", key='comm',
                    enable_events=True))
rb.append(psg.Radio("Science", "faculty", key='sci',
                    enable_events=True))

cb = []
cb.append(psg.Checkbox("History", key='s1'))
cb.append(psg.Checkbox("Sociology", key='s2'))
cb.append(psg.Checkbox("Economics", key='s3'))

b1 = psg.Button("OK")
b2 = psg.Button("Exit")

rlo = psg.Frame("Faculty", [rb], title_color='blue')
clo = psg.Frame("Subjects", [cb], title_color='blue')

layout = [[l1, t1], [rlo], [clo], [b1, l5, b2]]

window = psg.Window('Frame Example', layout, size=(715, 200))
```

The **output** of the program is shown below:



19. PySimpleGUI – Column Element

The Column element is also a container widget. It is very useful if you want to design the GUI window elements represented in one or more vertical columns. Just as a window, the Column area places the other PySimpleGUI elements in a layout consisting of list of lists.

A Column layout is similar to a Frame. However, the Column doesn't have border or title as the Frame. But it is very effective when you want to place group of elements side by side.

The mandatory parameter to be passed to the Column constructor is layout as list of lists, each inner list being a row of elements.

Other parameters may be given as:

```
PySimpleGUI.Column(layout, size, scrollable,
vertical_scroll_only, element_justification)
```

Where,

- **layout**: Layout that will be shown in the Column container
- **size**: (width, height) size in pixels
- **scrollable**: if True then scrollbars will be added to the column
- **vertical_scroll_only**: if True then no horizontal scrollbar will be shown
- **element_justification**: All elements inside the Column will have this justification 'left', 'right', or 'center'

One of the important methods defined in the Column class is **contents_changed()**. If the scrollable property is enabled for the Column, and the layout changes by making some elements visible or invisible, the new scrollable area is computed when this method is called.

Although the container elements like Column normally are not event listeners, its visible property may be dynamically updated.

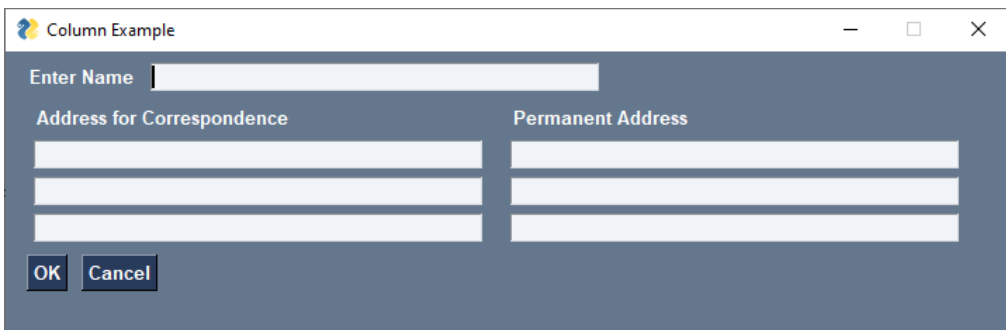
The following code shows how you can use the Column element. The main layout's upper row has a Text and Input element. The last row has "Ok" and "Cancel" buttons. The middle row has two columns, each having input elements for entering the correspondence and permanent address. Their

element layouts are stored as col1 and col2. These are used to declare two Column objects and placed in the list for middle row of the main layout.

```
import PySimpleGUI as psg

psg.set_options(font=("Arial Bold",10))

l=psg.Text("Enter Name")
l1=psg.Text("Address for Correspondence")
l2=psg.Text("Permanent Address")

t=psg.Input("", key='-NM-')

a11=psg.Input(key='-a11-')
a12=psg.Input(key='-a12-')
a13=psg.Input(key='-a13-')

col1=[[l1],[a11], [a12], [a13]]

a21=psg.Input(key='-a21-')
a22=psg.Input(key='-a22-')
a23=psg.Input(key='-a23-')

col2=[[l2],[a21], [a22], [a23]]

layout=[[l,t],[psg.Column(col1), psg.Column(col2)],
        [psg.OK(), psg.Cancel()]]

window = psg.Window('Column Example', layout, size=(715,200))

while True:
```

```
    event, values = window.read()
    print (event, values)
    if event in (psg.WIN_CLOSED, 'Exit'):
        break

window.close()
```

It will produce the following **output** window:



20. PySimpleGUI – Tab Element

Sometimes, the application's GUI design is too big to fit in a single window, and even if we try to arrange all the elements in a layout for the main window, it becomes very clumsy. The use of Tab elements makes the design very convenient, effective and easy for the user to navigate. Tab element is also a container element such as Frame or Column.

First divide the elements in logically relevant groups and put them in separate layouts. Each of them is used to construct a Tab element. These Tab elements are put in a TabGroup as per the following syntax:

```
# l1 is the layout for tab1
# l2 is the layout for tab2
tab1=PySimpleGUI.Tab("title1", l1)
tab2=PySimpleGUI.Tab("title2", l2)

Tg = PySimpleGUI.TabGroup([[tab1, tab2]])
# This titlegroup object may be further used
# in the row of the main layout.
```

The advantage of the tabgroup and tab elements is that only one tab of all the tabs in a group is visible at a time. When you click the title of one tab, it becomes visible and all others are hidden. So that, the entire available area can be utilized to display the elements in a single tab. This makes the GUI design very clean and effective.

Remember that the Tab elements are never placed directly in the main layout. They are always contained in a TabGroup.

To construct a Tab element, use the following **syntax**:

```
PySimpleGUI.Tab(title, layout, title_color)
```

Here, the title parameter is a string displayed on the tab. The layout refers to the nested list of elements to be shown on the top and the title_color the color to be used for displaying the title.

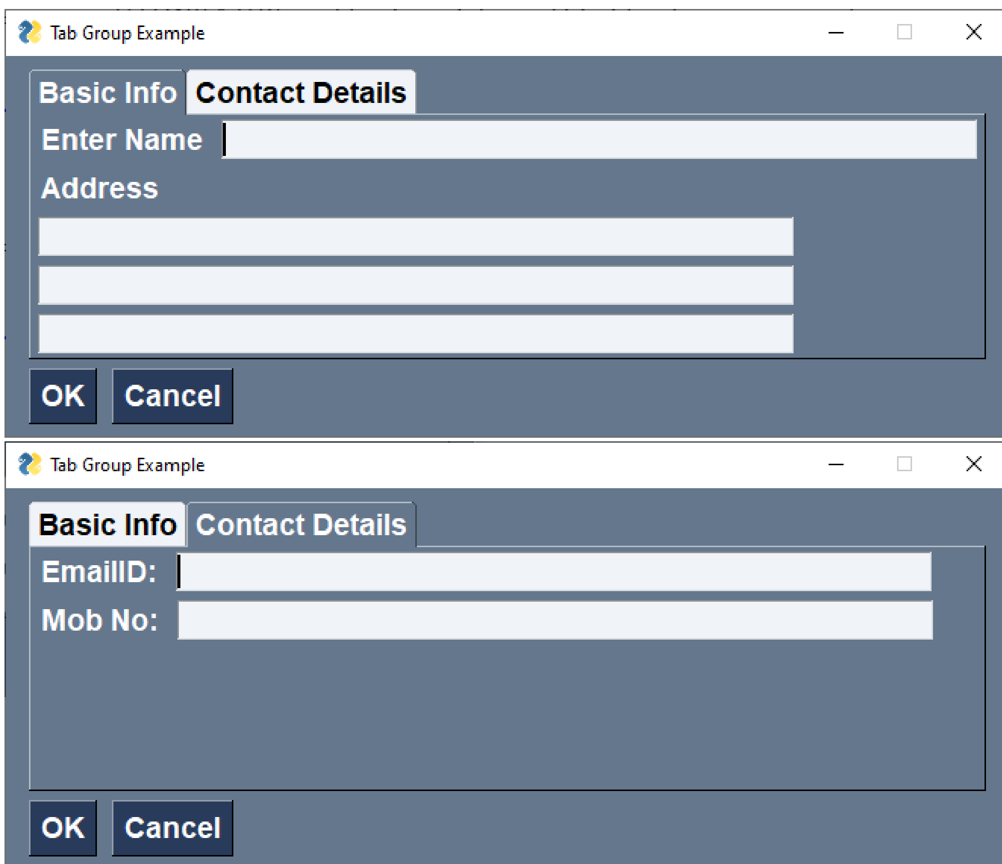
tutorialspoint
SIMPLYEASYLEARNING

In the following example, a typical registration form is designed as a tabgroup with two tabs, one called Basic Info and the other Contact details. Below the tabgroup, two Buttons with OK and Cancel are placed.

```
import PySimpleGUI as psg

psg.set_options(font=("Arial Bold",14))

l1=psg.Text("Enter Name")
lt1=psg.Text("Address")
t1=psg.Input("", key='-NM-')

a11=psg.Input(key='-a11-')
a12=psg.Input(key='-a12-')
a13=psg.Input(key='-a13-')

tab1=[[l1,t1],[lt1],[a11], [a12], [a13]]

lt2=psg.Text("EmailID:")
lt3=psg.Text("Mob No:")

a21=psg.Input("", key='-ID-')
a22=psg.Input("", key='-MOB-')

tab2=[[lt2, a21], [lt3, a22]]

layout = [[psg.TabGroup([
    [psg.Tab('Basic Info', tab1),
     psg.Tab('Contact Details', tab2)]])],
    [psg.OK(), psg.Cancel()]
]

window = psg.Window('Tab Group Example', layout)
```

```
while True:
    event, values = window.read()
    print (event, values)
    if event in (psg.WIN_CLOSED, 'Exit'):
        break

window.close()
```

Run the above code. The main window with two tabs is displayed, with the first tab visible by default.



Click the title of the second tab which will show the two Input controls for entering the EmailID and the Mobile number.

tutorialspoint
SIMPLY EASY LEARNING

21. PySimpleGUI – Canvas Element

The Canvas Element provides a drawable panel on the surface of the PySimpleGUI application window. It is equivalent to the Canvas widget in the original TKinter package.

First, declare an object of the Canvas class with following parameters:

```
can = PySimpleGUI.Canvas(canvas, background_color, size)
```

Where,

- **canvas**: Leave blank to create a Canvas
- **background_color**: color of background
- **size**: (width in char, height in rows) size in pixels to make canvas

Place this object in the layout for our application window.

We can use the various drawing methods of tkinter's Canvas by first obtaining the underlying canvas object with TkCanvas property.

```
tkc = can.TkCanvas
```

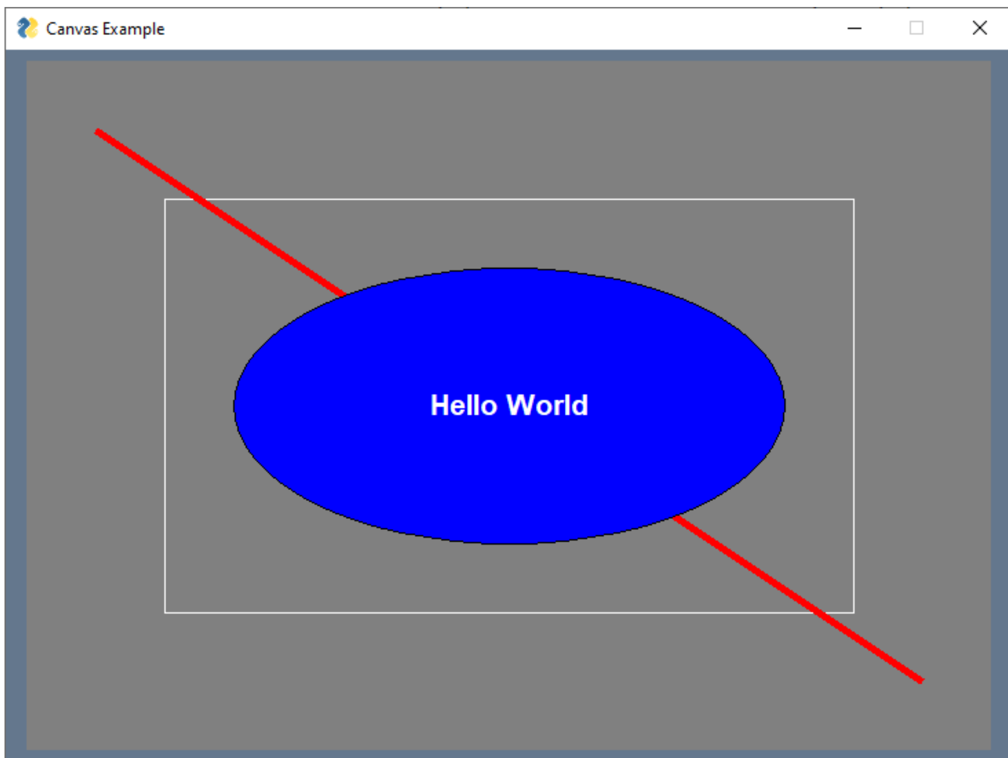
Now we can call the various draw methods as follows:

create_line	Draws a straight line from (x1,y1) to (x2,y2). Color is specified with fill parameter and thickness by width parameter.
create_rectangle	Draws rectangle shape where (x1,y1) denote the coordinates of top left corner and (x2,y2) are coordinates of right bottom corner. The fill parameter is used to display solid rectangle with specified colour.
create_oval	Displays an ellipse. (x1,y1) represents the coordinates of center. r1 and r2 stand for "x" radius and "y" radius. If r1 and r2 same, circle is drawn.
create_text	Displays a string value of text parameter at x1,y1 coordinates. Font parameter decides font name and size and fill parameter is given to apply font colour.

Given below is a simple implementation of Canvas element:

```
import PySimpleGUI as sg

can=sg.Canvas(size=(700,500),
              background_color='grey',
              key='canvas')

layout = [[can]]

window = sg.Window('Canvas Example', layout, finalize=True)

tkc=can.TKCanvas

fig = [tkc.create_rectangle(100, 100, 600, 400, outline='white'),
       tkc.create_line(50, 50, 650, 450, fill='red', width=5),
       tkc.create_oval(150,150,550,350, fill='blue'),
       tkc.create_text(350, 250, text="Hello World",
                       fill='white', font=('Arial Bold', 16)),
       ]

while True:
    event, values = window.read()
    if event == sg.WIN_CLOSED:
        break
```

When the above code is run, you get the following result:


Canvas Example
Hello World

22. PySimpleGUI – Graph Element

The Graph element is similar to Canvas, but very powerful. You can define your own coordinate system, working in your own units, and then displaying them in an area defined in pixels.

You should provide the following values to the Graph object:

- Size of the canvas in pixels
- The lower left (x,y) coordinate of your coordinate system
- The upper right (x,y) coordinate of your coordinate system

Graph Figures are created, and a Figure ID is obtained by calling following methods which are similar to the Tkinter Canvas:

```
draw_arc(self, top_left, bottom_right, extent, start_angle,
         style=None, arc_color='black',
         line_width=1, fill_color=None)

draw_circle(self, center_location, radius,
            fill_color=None, line_color='black',
            line_width=1)

draw_image(self, filename=None, data=None,
           location=(None, None))

draw_line(self, point_from, point_to,
          color='black', width=1)

draw_lines(self, points,
           color='black', width=1)

draw_oval(self, top_left, bottom_right,
          fill_color=None, line_color=None,
          line_width=1)
```


tutorialspoint
SIMPLYEASYLEARNING

```
draw_point(self, point,
           size=2, color='black')

draw_polygon(self, points,
             fill_color=None, line_color=None,
             line_width=None)

draw_rectangle(self, top_left, bottom_right,
               fill_color=None, line_color=None,
               line_width=None)

draw_text(self, text, location,
          color='black', font=None, angle=0,
          text_location='center')
```

Apart from the above draw methods, the Graph class also defines the **move_figure()** method by which the image identified by its ID is moved to its new position by giving new coordinates relative to its previous coordinates.

```
move_figure(self, figure, x_direction, y_direction)
```

The mouse event inside the graph area can be captured if you set drag_submits property to True. When you click anywhere in the graph area, the event generated is: Graph_key + '+UP'.

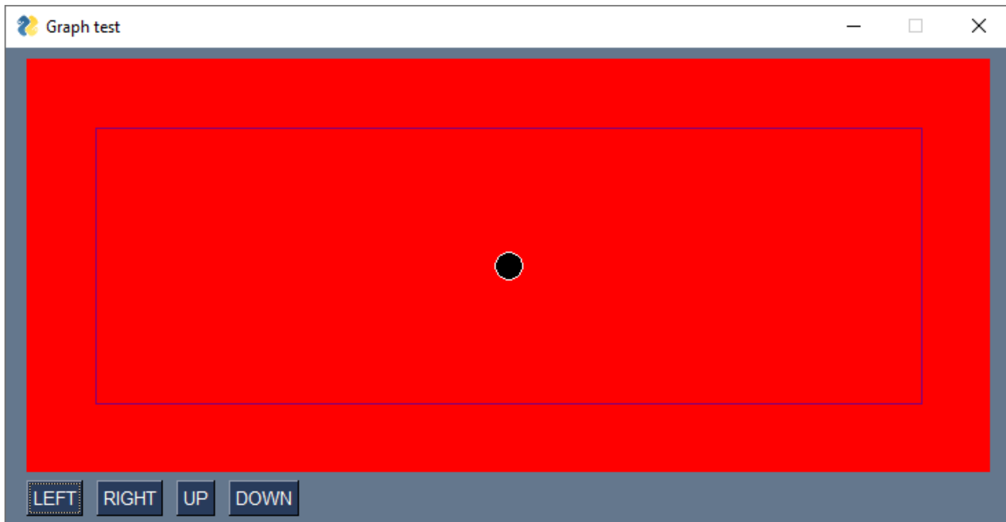
In the following example, we draw a small circle at the center of the graph element. Below the graph object, there are buttons for moving the circle in left, right, up and down direction. When clicked, the **mov_figure()** method is called.

```
import PySimpleGUI as psg

graph=psg.Graph(canvas_size=(700,300),
                graph_bottom_left=(0, 0),
                graph_top_right=(700,300),
                background_color='red',
```

```
                enable_events=True,
                drag_submits=True, key='graph')

layout = [[graph], [psg.Button('LEFT'), psg.Button('RIGHT'),
                    psg.Button('UP'), psg.Button('DOWN')]]

window = psg.Window('Graph test', layout, finalize=True)

x1,y1 = 350,150

circle = graph.draw_circle((x1,y1), 10,
                           fill_color='black',
                           line_color='white')

rectangle = graph.draw_rectangle((50,50), (650,250),
                                 line_color='purple')

while True:
    event, values = window.read()

    if event == psg.WIN_CLOSED:
        break
    if event == 'RIGHT':
        graph.MoveFigure(circle, 10, 0)
    if event == 'LEFT':
        graph.MoveFigure(circle, -10,0)
    if event == 'UP':
        graph.MoveFigure(circle, 0, 10)
    if event == 'DOWN':
        graph.MoveFigure(circle, 0,-10)
    if event=="graph+UP":
        x2,y2= values['graph']
```

```
        graph.MoveFigure(circle, x2-x1, y2-y1)
        x1,y1=x2,y2

window.close()
```

Run the above program. Use the buttons to move the circle.



23. PySimpleGUI – Menubar

Most of the desktop applications have a menu system to trigger different operations based on user's choice of options in the menu. In a typical application window, the menu bar is placed just below the title bar and above the client area of the window.

A menubar is a horizontal bar consisting of clickable buttons. When any of these buttons is clicked it generates a pull down list of option buttons. Such an option button triggers a click event which can be processed inside an event loop.

The menu system is designed just as the window layout is specified. It is also a list of lists. Each list has one or more strings. The starting string of the list at the first level is the caption for the button appearing in the horizontal menu bar. It is followed by a list of caption strings for the option buttons in the drop down menu. These option captions are in a list inside the first level list.

You may have a sub-menu under an option button, in which case the captions are put in a third level list. Likewise, the captions can be nested up to any level.

The general format of a menu definition is as follows:

```
menu_def = [

    ['Memu1', ['btn1', 'btn2', 'btn3', 'btn4',]],

    ['menu2', ['btn5', 'btn6','btn7', 'btn8'],],

]
```

To attach the menu system to the main layout of PysimpleGUI window, place the Menu object in the first row of the layout.

The Menu constructor is given the **menu_def** list as the argument. Other rows of the main layout may be given after the row having Menu object.

```
layout= [[psg.Menu(menu_def),[..], [..]]
```

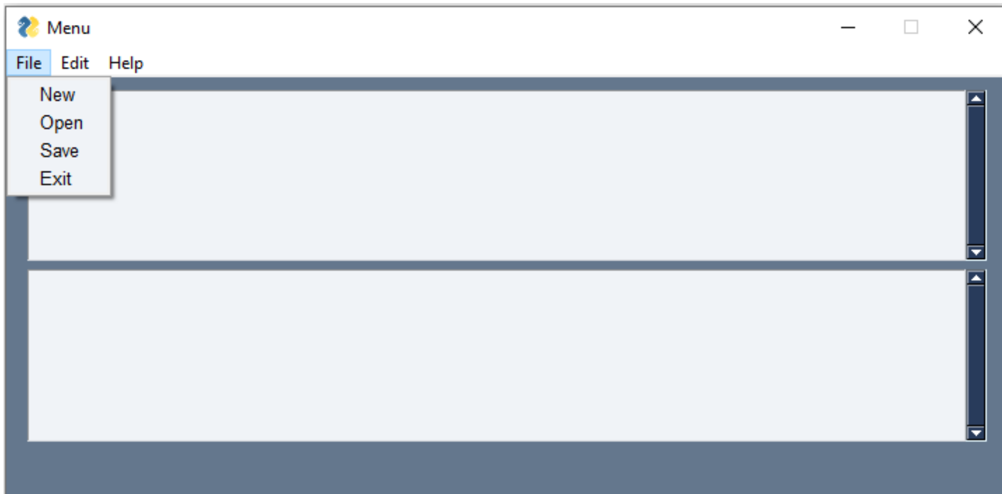
In the code given below, we have a menu bar with File, Edit and Help menus, each having a few menu buttons in respective menu bar.

tutorialspoint
SIMPLYEASYLEARNING

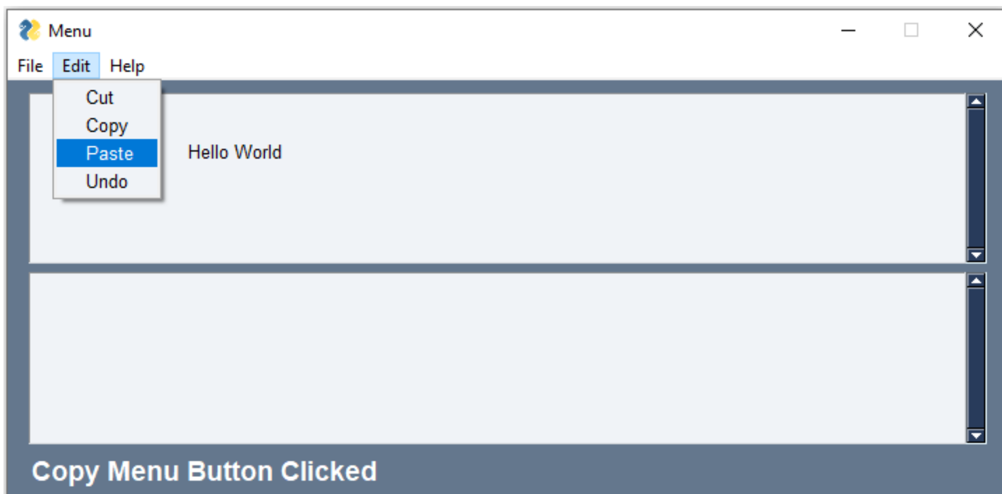
```
import PySimpleGUI as psg

menu_def = [['File', ['New', 'Open', 'Save', 'Exit', ]],
            ['Edit', ['Cut', 'Copy', 'Paste', 'Undo'], ],
            ['Help', 'About...'], ]

layout = [[psg.Menu(menu_def)],
          [psg.Multiline("", key='-IN-',
                         expand_x=True, expand_y=True)],
          [psg.Multiline("", key='-OUT-',
                         expand_x=True, expand_y=True)],
          [psg.Text("", key='-TXT-',
                    expand_x=True, font=("Arial Bold", 14))]
          ]
window = psg.Window("Menu", layout, size=(715, 300))

while True:
    event, values = window.read()
    print(event, values)

    if event != psg.WIN_CLOSED:
        window['-TXT-'].update(values[0] + "Menu Button Clicked")
    if event == 'Copy':
        txt = window['-IN-'].get()
    if event == 'Paste':
        window['-OUT-'].update(value=txt)
    if event == psg.WIN_CLOSED:
        break

window.close()
```

Below the Menubar, two Multiline elements are placed. The last row has a Text element.

When any menu option button is clicked, the event so generated is the caption of the button. This caption is displayed on the Text label in the last row. Refer to the following figure:



When the Copy event occurs, the text in the upper multiline box with -IN- key is stored in a txt variable. Afterwards, when Paste button is pressed, the -OUT- box is updated with the value of txt.



Menu button with Hot Key

To map a menu button with a key on the keyboard, put an ampersand & character before the desired character. For example, put & before File so that the string is '&File'. By doing so, the File menu can be accessed by pressing "Alf+F" key. Here "F" key is said to be a hot key.

tutorialspoint
SIMPLYEASYLEARNING

Add hot keys to the menu buttons in our menu definition.

```
menu_def = [
    ['&File', ['&New', '&Open', '&Save', 'E&xit',]],
    ['&Edit', ['C&ut', '&Copy','&Paste', '&Undo'],],
    ['&Help', '&About...'],
]
```

When the code is run, the hot keys in the menu are shown as underlined.

Right-click Menu

This menu is detached from the menubar which is at the top of the application window. Whenever the user presses the right click button of the mouse, this menu pops up at the same position where the click takes place.

In the menubar defined above, each list is a definition of a single menu. Such single menu definition can be attached to any element by the right_click_menu parameter in the constructor. This parameter can also be passed while constructing the main Window object.

Let us use rightclick as a variable for the list corresponding to the Edit menu.

```
rightclick=['&Edit', ['C&ut', '&Copy','&Paste', '&Undo']]

menu_def = [
```

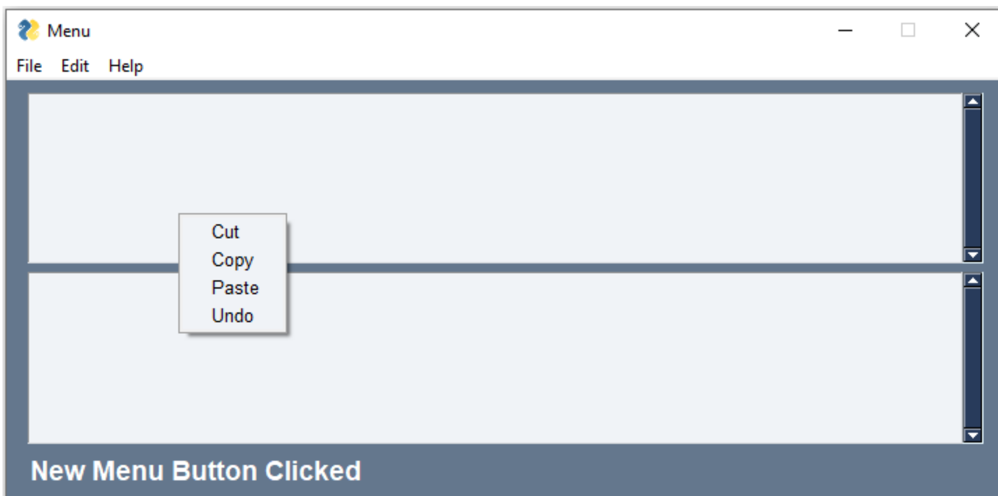

tutorialspoint
SIMPLYEASYLEARNING

```
    ['&File', ['&New', '&Open', '&Save', 'E&xit',]],
    rightclick,
    ['&Help', '&About...'],
]
```

Use it as the value of right_click_menu parameter in the Window constructor. See the following snippet:

```
window=psg.Window("Menu", layout, size=(715, 300),
right_click_menu=rightclick)
```

Make these changes and run the code. Click anywhere in the window. The menu pops up as shown:



ButtonMenu

This menu is similar to the right click menu, except that it is attached to a button and pops up when the button is clicked.

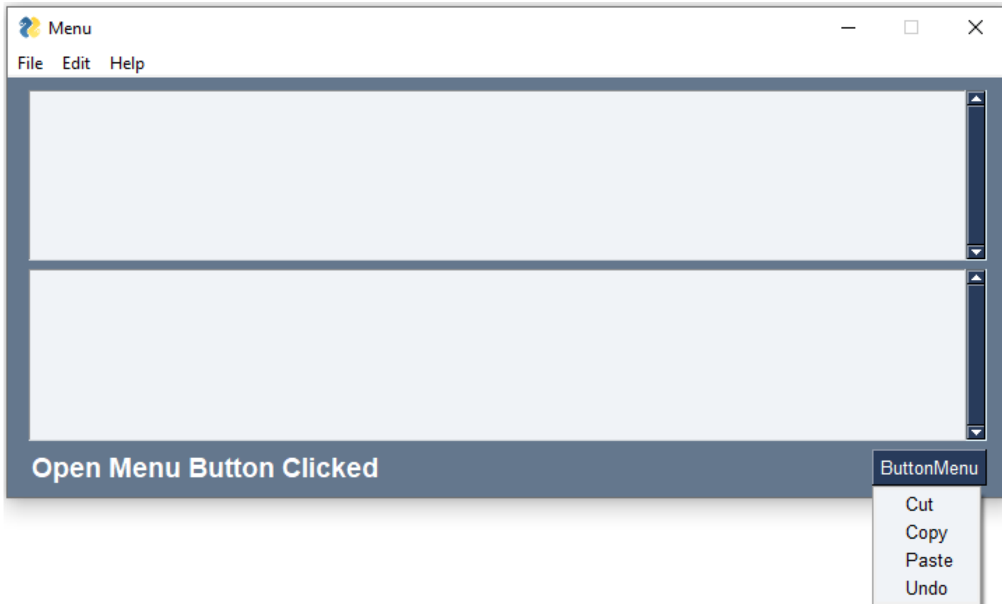
In the last row of the main layout, we add a ButtonMenu element and use the rightclick list as its layout.

```
layout= [
    [psg.Menu(menu_def)],
    [psg.Multiline("", key='-IN-',  expand_x=True, expand_y=True)],
    [psg.Multiline("", key='-OUT-',  expand_x=True, expand_y=True)],
```


tutorialspoint
SIMPLY EASY LEARNING

```
    [psg.Text("", key='-TXT-', expand_x=True, font=("Arial Bold", 14)),
     psg.ButtonMenu('ButtonMenu',  rightclick, key='-BMENU-')]
]
```

When the button at the lower right is clicked, the menu comes up as can be seen in the following figure:



24. PySimpleGUI – Table Element

The Table object is a useful GUI widget in any GUI library. Its purpose is to display a two-dimensional data structure of numbers and strings in a tabular form having rows and columns.

The important parameters to be passed to the Table class constructor are:

```
PySimpleGUI.Table(values, headings, col_widths,
                  auto_size_columns, select_mode,
                  display_row_numbers, num_rows,
                  alternating_row_color,
                  selected_row_colors,
                  header_text_color)
```

The following table explains the role of each of these parameters:

Values	Table data represented as a 2-dimensions table
Headings	The headings to show on the top line
col_widths	Number of characters that each column will occupy
auto_size_columns	If True columns will be sized automatically
select_mode	Select Mode. Valid values: • TABLE_SELECT_MODE_NONE • TABLE_SELECT_MODE_BROWSE • TABLE_SELECT_MODE_EXTENDED
display_row_numbers	If True, the first column of the table will be the row
num_rows	The number of rows of the table to display at a time
alternating_row_color	If True every other row will have this color in the background.
selected_row_colors	Sets the text color and background color for a selected row.
header_text_color	sets the text color for the header

tutorialspoint
SIMPLYEASYLEARNING

When any cell in the table is clicked, PySimpleGUI generates a tuple of CLICKED event having the table key, and the (row,col) of the clicked cell.

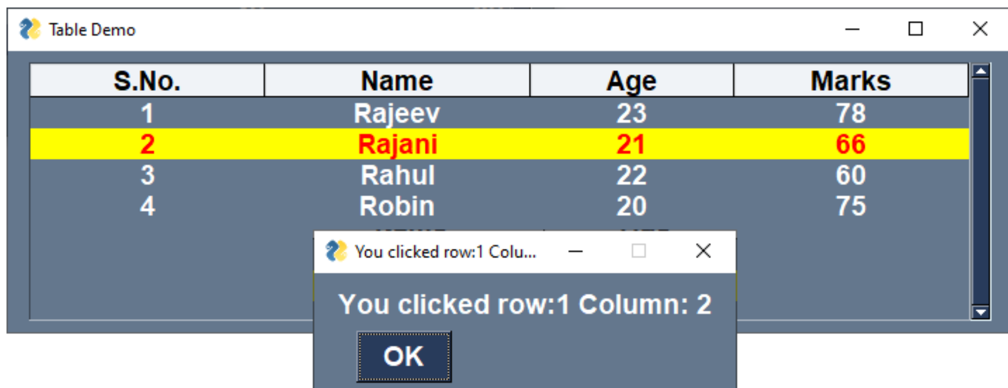
```
event: ('-TABLE-', '+CLICKED+', (0, 1))
```

Following code displays a list of students in a Table object on the PySimpleGUI window. A popup window appears when you click in any cell. The cell coordinates are displayed on the popup.

```
import PySimpleGUI as psg

psg.set_options(font=("Arial Bold", 14))

toprow = ['S.No.', 'Name', 'Age', 'Marks']

rows = [[1, 'Rajeev', 23, 78],
        [2, 'Rajani', 21, 66],
        [3, 'Rahul', 22, 60],
        [4, 'Robin', 20, 75]]

tbl1 = psg.Table(values=rows, headings=toprow,
                 auto_size_columns=True,
                 display_row_numbers=False,
                 justification='center', key='-TABLE-',
                 selected_row_colors='red on yellow',
                 enable_events=True,
                 expand_x=True,
                 expand_y=True,
                 enable_click_events=True)

layout = [[tbl1]]

window = psg.Window("Table Demo", layout,
                    size=(715, 200), resizable=True)
```

```
while True:
    event, values = window.read()
    print("event:", event, "values:", values)

    if event == psg.WIN_CLOSED:
        break
    if '+CLICKED+' in event:
        psg.popup("You clicked row:{} Column:
{}".format(event[2][0], event[2][1]))

window.close()
```

It will produce the following **output** window:



The Table object also has an **update()** method to dynamically update the table properties such as values, num_rows, and row_color.

25. PySimpleGUI – Tree Element

A Tree is a hierarchical data structure consisting of one or more nodes, and each node may have one or more children nodes. This arrangement of nodes is done in an object of TreeData object, which is used as a data parameter for creating a Table.

First of all, declare a TreeData object.

```
treedata = psg.TreeData()
```

Use the **insert()** method of the TreeData class to construct the hierarchy of nodes.

```
TreeData.insert(parent_key, key, display_text, values)
```

To insert a node at the first level of the tree, use the parant_key as "". So, every top-level node in the tree will have a parent node = "". To insert a child node, give the key of the node at the upper level as its parent_key.

For example,

```
insert("","MH", "Maharashtra", (175, 150, 200))
```

will insert a node at the root level with MH as the key.

On the other hand, the following command

```
insert("MH", "MUM", "Mumbai", (100, 100,100))
```

will insert a child node with its key as MUM.

The TreeData object is used to construct a Tree object with following parameters:

data	The data represented using TreeData class
headings	List of individual headings for each column
col_widths	List of column widths for individual columns
col0_width	Size of Column 0
col0_heading	Text to be shown in the header for the left-most column
def_col_width	Default column width

auto_size_columns	If True, the size of a column is determined by contents of the column
select_mode	Same as Table Element
show_expanded	If True, the tree will be initially shown with all nodes completely expanded

In the following example, we display a statewise list of cities in a tree structure

```
import PySimpleGUI as psg

psg.set_options(font=("Arial Bold",14))

treedata = psg.TreeData()

rootnodes=[
    ["","MH", "Maharashtra", 175, 150, 200],
    ["MH", "MUM", "Mumbai", 100, 100,100],
    ["MH", "PUN", "Pune", 30, 20, 40],
    ["MH", "NGP", "Nagpur", 45, 30, 60],
    ["","TEL", "Telangana", 120, 80, 125],
    ["TEL", "HYD", "Hyderabad", 75, 55, 80],
    ["TEL", "SEC", "Secunderabad", 25, 15, 30],
    ["TEL", "NZB", "Nizamabad", 20, 10, 15]
]

for row in rootnodes:
    treedata.Insert( row[0], row[1], row[2], row[3:])

tree=psg.Tree(data=treedata,
                   headings=['Product A','Product B','Product C' ],
                   auto_size_columns=True,
                   select_mode=psg.TABLE_SELECT_MODE_EXTENDED,
                   num_rows=10,
```


tutorialspoint
SIMPLYEASYLEARNING

```
                    col0_width=5,
                    key='-TREE-',
                    show_expanded=False,
                    enable_events=True,
                    expand_x=True,
                    expand_y=True,
                    )
layout=[[tree]]
window=psg.Window("Tree Demo", layout,
                   size=(715, 200), resizable=True)

while True:
    event, values = window.read()
    print ("event:",event, "values:",values)
    if event == psg.WIN_CLOSED:
        break
```

It will produce the following **output** window:

Tree Demo

	Product A	Product B	Product C
Maharashtra	175	150	200
Mumbai	100	100	100
Pune	30	20	40
Nagpur	45	30	60
Telangana	120	80	125
Hyderabad	75	55	80
Secunderabad	25	15	30

26. PySimpleGUI – Image Element

The PySimpleGUI library contains an Image element, which has the ability to display images of PNG, GIF, PPM/PGM format. The **Image()** function needs one mandatory argument which is the path to the image file.

The following code displays the PySimpleGUI logo on the application window.

```
import PySimpleGUI as psg

layout = [[psg.Text(text='Python GUIs for Humans',
                    font=('Arial Bold', 16),
                    size=20, expand_x=True,
                    justification='center')],
          [psg.Image('PySimpleGUI_Logo.png',
                     expand_x=True, expand_y=True )]
          ]

window = psg.Window('HelloWorld', layout, size=(715,350),
                    keep_on_top=True)

while True:
    event, values = window.read()
    print(event, values)

    if event in (None, 'Exit'):
        break

window.close()
```

It will produce the following **output** window:

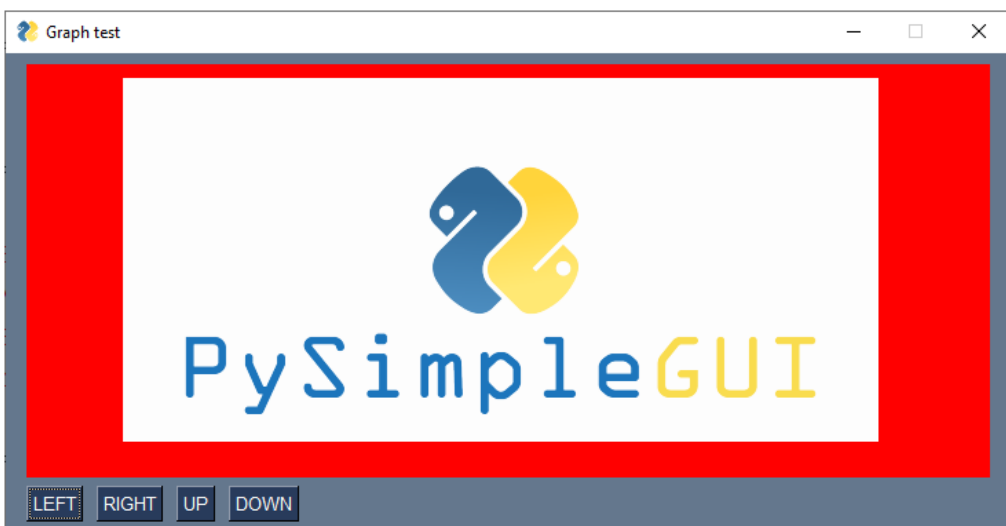
Using Graph Element

You can also display image on a Graph container element with its **draw_image()** method as the following code shows:

```
import PySimpleGUI as psg

graph = psg.Graph(canvas_size=(700, 300),
                  graph_bottom_left=(0, 0),
                  graph_top_right=(700, 300),
                  background_color='red',
                  enable_events=True,
                  drag_submits=True, key='graph')

layout = [
    [graph],
    [psg.Button('LEFT'), psg.Button('RIGHT'),
     psg.Button('UP'), psg.Button('DOWN')]
]

window = psg.Window('Graph test', layout, finalize=True)
x1, y1 = 350, 150
```

```
id = graph.draw_image(filename="PySimpleGUI_Logo.png",
                      location=(0, 300))
while True:
    event, values = window.read()
    if event == psg.WIN_CLOSED:
        break
    if event == 'RIGHT':
        graph.MoveFigure(id, 10, 0)
    if event == 'LEFT':
        graph.MoveFigure(id, -10, 0)
    if event == 'UP':
        graph.MoveFigure(id, 0, 10)
    if event == 'DOWN':
        graph.MoveFigure(id, 0, -10)
    if event == "graph+UP":
        x2, y2 = values['graph']
        graph.MoveFigure(id, x2 - x1, y2 - y1)
        x1, y1 = x2, y2
window.close()
```

It will produce the following **output** window:



27. PySimpleGUI – Matplotlib Integration

When Matplotlib is used from Python shell, the plots are displayed in a default window. The **backend_tkagg** module is useful for embedding plots in Tkinter.

The Canvas element in PySimpleGUI has TKCanvas method that returns original TKinter's Canvas object. It is given to the **FigureCanvasTkAgg()** function in the backend_tkagg module to draw the figure.

First, we need to create the figure object using the **Figure()** class and a plot to it. We shall draw a simple plot showing sine wave.

```
fig = matplotlib.figure.Figure(figsize=(5, 4), dpi=100)
t = np.arange(0, 3, .01)
fig.add_subplot(111).plot(t, 2 * np.sin(2 * np.pi * t))
```

Define a function to draw the matplotlib figure object on the canvas

```
def draw_figure(canvas, figure):
    figure_canvas_agg = FigureCanvasTkAgg(figure, canvas)
    figure_canvas_agg.draw()
    figure_canvas_agg.get_tk_widget().pack(side='top',
                                           fill='both',
                                           expand=1)
    return figure_canvas_agg
```

Obtain the Canvas from PySimpleGUI.Canvas object by calling its TkCanvas property.

```
layout = [
    [psg.Text('Plot test')],
    [psg.Canvas(key='-CANVAS-')],
    [psg.Button('Ok')]
]
```

Draw the figure by calling the above function. Pass the Canvas object and fifure object to it.

```
fig_canvas_agg = draw_figure(window['-CANVAS-'].TKCanvas, fig)
```

Example: Draw a Sinewave Line graph

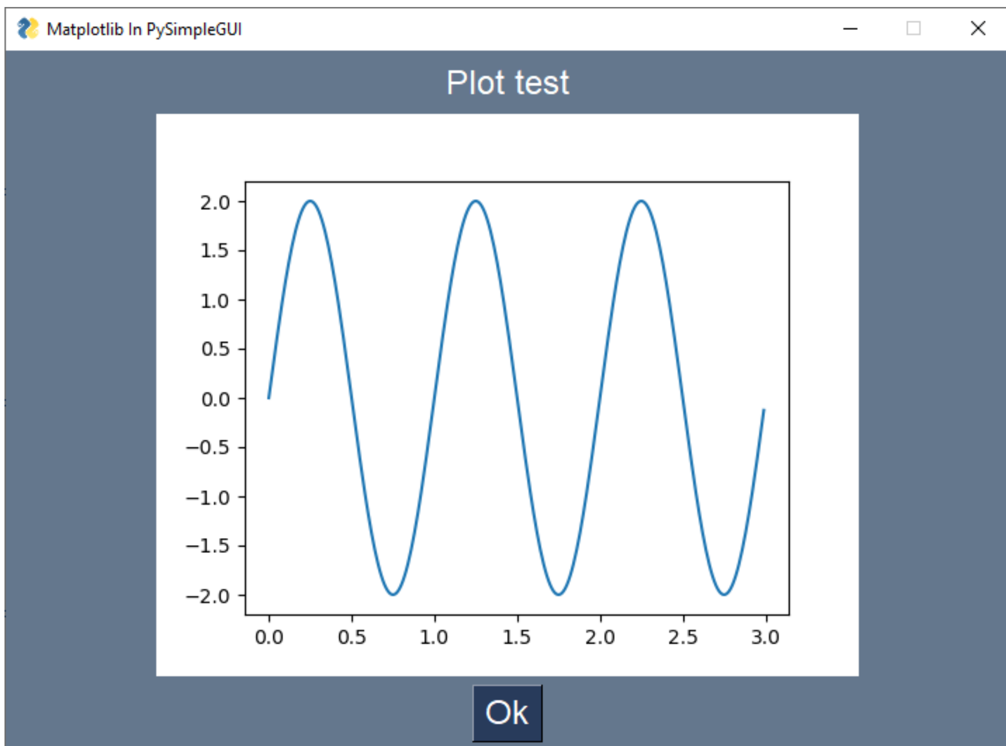
The complete code is given below:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg
import PySimpleGUI as sg
import matplotlib

matplotlib.use('TkAgg')

fig = matplotlib.figure.Figure(figsize=(5, 4), dpi=100)
t = np.arange(0, 3, .01)
fig.add_subplot(111).plot(t, 2 * np.sin(2 * np.pi * t))

def draw_figure(canvas, figure):
    tkcanvas = FigureCanvasTkAgg(figure, canvas)
    tkcanvas.draw()
    tkcanvas.get_tk_widget().pack(side='top', fill='both', expand=1)
    return tkcanvas

layout = [[sg.Text('Plot test')],
          [sg.Canvas(key='-CANVAS-')],
          [sg.Button('Ok')]]
```


tutorialspoint
SIMPLYEASYLEARNING

```
window = sg.Window('Matplotlib In PySimpleGUI', layout,
                   size=(715, 500), finalize=True,
                   element_justification='center',
                   font='Helvetica 18')

# add the plot to the window
tkcanvas = draw_figure(window['-CANVAS-'].TKCanvas, fig)

event, values = window.read()

window.close()
```

The generated graph is as follows:



28. PySimpleGUI – Working with PIL

Python Imaging Library is a free, cross-platform and open-source library for the Python programming language that has the functionality for opening, manipulating, and saving many different image file formats.

To install it, use the PIP command as follows:

```
pip3 install pillow
```

In the following example, we obtain the byte value of the PNG image with PIL function and display the same in Image element on a PySimpleGUI window.

```
import PySimpleGUI as sg
import PIL.Image
import io
import base64

def convert_to_bytes(file_or_bytes, resize=None):
    img = PIL.Image.open(file_or_bytes)
    with io.BytesIO() as bio:
        img.save(bio, format="PNG")
        del img
        return bio.getvalue()

imgdata = convert_to_bytes("PySimpleGUI_logo.png")
layout = [[sg.Image(key='-IMAGE-', data=imgdata)]]
window = sg.Window('PIL based Image Viewer', layout,resizable=True)

while True:
    event, values = window.read()
    if event == sg.WIN_CLOSED:
        break
window.close()
```


tutorialspoint
SIMPLYEASYLEARNING

It will produce the following **output** window:

tutorialspoint
SIMPLYEASYLEARNING

29. PySimpleGUI – Debugger

In addition to the built-in debugger that most IDEs such as PyCharm or VS Code have, PySimpleGUI offers its own debugger. This debugger provides you the ability to "see" and interact with your code, while it is running.

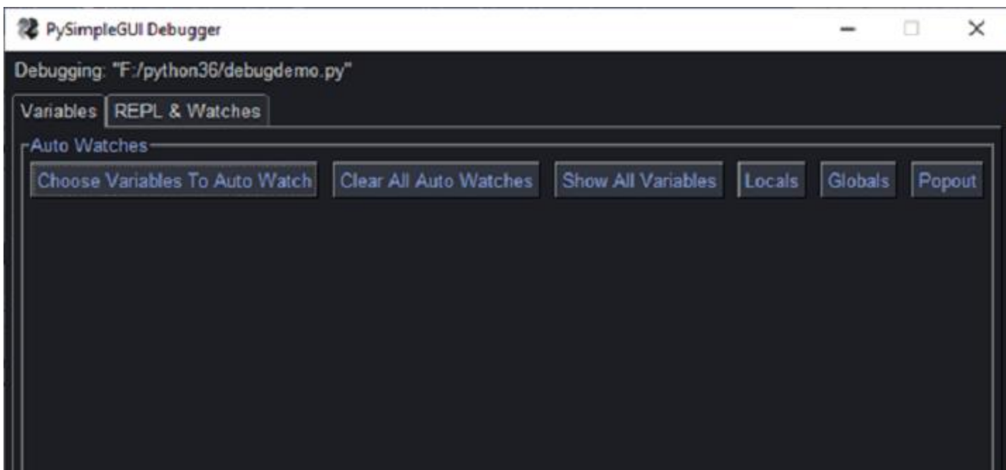
To use the debugger service effectively, the window should be red asynchronously, i.e., you should provide a timeout to the **read()** function.

The debugger window is invoked by calling **show_debugger_window()** function anywhere inside the program as shown below:

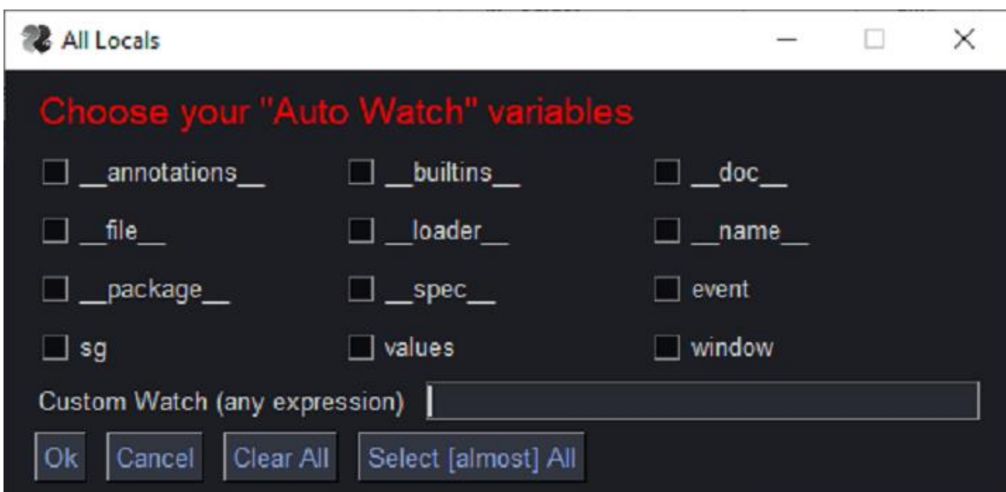
```
import PySimpleGUI as sg

sg.show_debugger_window(location=(10,10))

window = sg.Window('Debugger Demo',
                   [[sg.Text('Debugger'),
                     sg.Input('Input here'),
                     sg.Button('Push Me')]]
                   )

while True:
    event, values = window.read(timeout=500)
    if event == sg.TIMEOUT_KEY:
        continue
    if event == sg.WIN_CLOSED:
        break
    print(event, values)

window.close()
```

The PySimpleGUI debugger window appears at the specified screen location.

tutorialspoint
SIMPLYEASYLEARNING

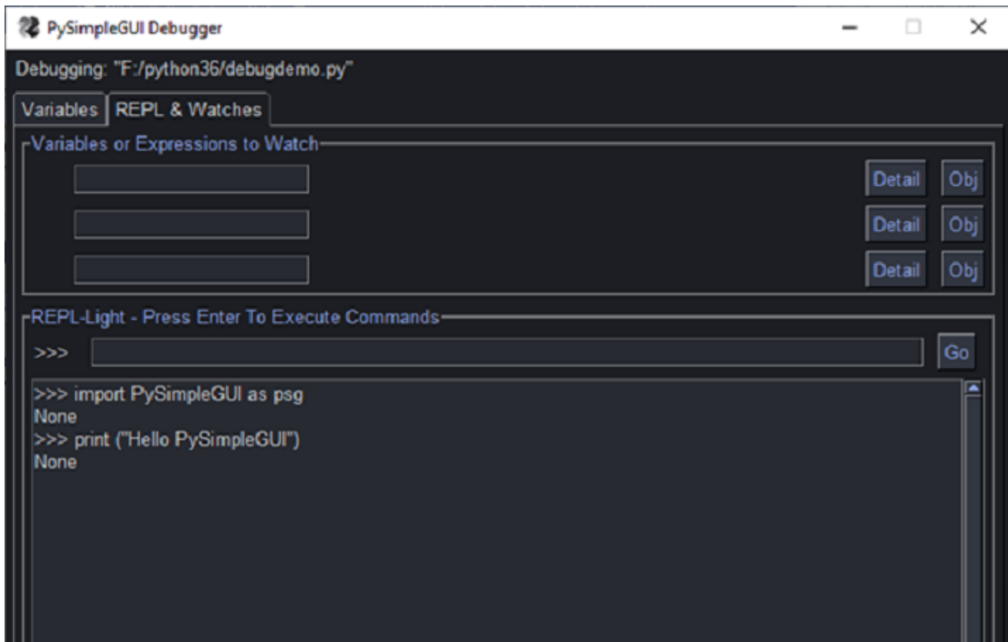


The window shows two tabs Variables and REPL. Click on the Variables tab. A list of variables to auto-watch is shown Check the ones that you want to watch during the execution of the program.



The second tab about REPL gives a Python interactive console to be executed around your program's environment so that you can inspect the values of desired variables in the code.

tutorialspoint
SIMPLYEASYLEARNING



tutorialspoint
SIMPLYEASYLEARNING

30. PySimpleGUI – Settings

Global Settings

Global settings are the application settings available application wide. These settings control the various properties of the Element class to be applied to all the Elements in the application.

These settings work in hierarchical manner. The global settings are overridden if those settings are given different value for a window. In turn the settings defined in Window object are given different value for a specific element.

For example, if the font size is set to 16 globally, the text of all elements is displayed accordingly. However, if a specific Text or Input element with Font property with size other than 16 is defined in the layout, it will change the appearance accordingly.

The function **set_options** is used to change settings that will apply globally. If it's a setting that applies to Windows, then that setting will apply not only to Windows that you create, but also to popup Windows.

```
import PySimpleGUI as sg
sg.set_options(font=('Arial Bold', 16))
```

User Settings

"User settings" is a dictionary that is automatically written to your hard drive. User settings are stored in a Python dictionary which is saved to and loaded from the disk. Individual settings are thus keys into a dictionary.

List of user setting functions:

Function	Description
user_settings	Returns settings as a dictionary
user_settings_delete_entry	Deletes a setting
user_settings_delete_filename	Deletes the settings file
user_settings_file_exists	Returns True if settings file specified exists

user_settings_filename	Returns full path and filename of settings file
user_settings_get_entry	Returns value for a setting. If no setting found, then specified default value is returned
user_settings_load	Loads dictionary from the settings file.
user_settings_save	Saves settings to current or newly specified file.
user_settings_set_entry	Sets an entry to a particular value
user_settings_write_new_dictionary	Writes a specified dictionary to settings file

Create the User Settings object.

```
settings = sg.UserSettings()
```

Use the dictionary-style [ ] syntax to read a setting. If the item's name is '-item-', then reading the value is achieved by writing

```
item_value = settings['-item-']
```

Following sttement is used to Write the setting.

```
settings['-item-'] = new_value
```

To delete an item, again the dictionary style syntax is used.

```
del settings['-item-']
```

You can also call the delete_entry method to delete the entry.

```
settings.delete_entry('-item-')
```

The following simple program demonstrates load/saving of user settings

```
import PySimpleGUI as sg
import json

sg.set_options(font=('Arial Bold', 16))
```


tutorialspoint
SIMPLY EASY LEARNING

```
layout = [
    [sg.Text('Settings', justification='left')],

    [sg.Text('User name', size=(10, 1), expand_x=True),
     sg.Input(key='-USER-')],

    [sg.Text('email ID', size=(10, 1), expand_x=True),
     sg.Input(key='-ID-')],

    [sg.Text('Role', size=(10, 1), expand_x=True),
     sg.Input(key='-ROLE-')],

    [sg.Button("LOAD"), sg.Button('SAVE'), sg.Button('Exit')]
]

window = sg.Window('User Settings Demo',
                   layout, size=(715, 200))

# Event Loop
while True:
    event, values = window.read()

    if event in (sg.WIN_CLOSED, 'Exit'):
        break

    if event == 'LOAD':
        f = open("settings.txt", 'r')
        settings = json.load(f)
        window['-USER-'].update(value=settings['-USER-'])
        window['-ID-'].update(value=settings['-ID-'])
        window['-ROLE-'].update(value=settings['-ROLE-'])
```


tutorialspoint
SIMPLYEASYLEARNING

```
    if event == 'SAVE':
        settings = {'-USER-': values['-USER-'],
                    '-ID-': values['-ID-'],
                    '-ROLE-': values['-ROLE-']}
        f = open("settings.txt", 'w')
        json.dump(settings, f)
        f.close()

window.close()
```

Enter the data in the input boxes and click the "Save" button.

A JSON file will be saved. To load the previously saved settings, click the "Load" button.