CONTENTS:

1. Component Types and Explanation:	**Function/code:**
Sg.theme Sets Background colour Valid Themes here	**Syntax:** sg.theme(*'#Colour'*) Eg. **import** PySimpleGUI **as** sg
Layout Layout holds all the elements in the window that cause functions to be performed(Further explained below)	**Syntax:** layout = [*'#Elements go here'*]
2. Types of Windows: (Windows defined using lists)	
Pattern 1A - One Shot Window Used to read, then display the user's input and closed	***With an auto-generated key:*** import PySimpleGUI as sg layout = [[sg.Text('My one-shot

Explanation for components in the basic pattern:
Reading a window returns events and values. Events is stored as a tuple and values is stored as a dictionary.

Event is whatever causes the read to return. For example a button, text clicked or close window.

Values is a dictionary of values of all the input-style elements.
In the design pattern, Auto Generated keys(integers starting at 0) are provided for the elements without keys.
Design pattern can get the value of whatever was input by referencing values[0]
With Keys entered, highlighted in green is the difference

```
window.')],
            [sg.InputText()],
            [sg.Submit(), sg.Cancel()]]

window = sg.Window('Window Title',
layout)

event, values = window.read()
window.close()

text_input = values[0]
sg.popup('You entered', text_input)
```

With keys:

```
import PySimpleGUI as sg

layout = [[sg.Text('My one-shot
window.')],
            [sg.InputText(key='-IN-')],
            [sg.Submit(), sg.Cancel()]]

window = sg.Window('Window Title',
layout)

event, values = window.read()
window.close()

text_input = values['-IN-']
sg.popup('You entered', text_input)
```

Pattern 1B - One Shot Window (Self Closing, Single line)

close=True is written in the read() bracket for the window to close exactly after it is read
The difference is highlighted in green

```
import PySimpleGUI as sg

event, values = sg.Window('Login Window',
            [[sg.T('Enter your Login ID'),
sg.In(key='-ID-')],
            [sg.B('OK'), sg.B('Cancel')
```

between 1A and 1B	]]).read(close=True) login_id = values['-ID-']
Pattern 2A - Persistent Window with event loops(Stays open) Difference is highlighted in green	***Code:*** import PySimpleGUI as sg sg.theme('DarkAmber') # Keep things interesting for your users layout = [[sg.Text('Persistent window')], [sg.Input(key='-IN-')], [sg.Button('Read'), sg.Exit()]] window = sg.Window('Window that stays open', layout) while True: # The Event Loop event, values = window.read() print(event, values) if event == sg.WIN_CLOSED or event == 'Exit': break window.close() ***Output:*** Read {'-IN-': 'typed into input field'} Read {'-IN-': 'More typing'} Exit {'-IN-': 'clicking the exit button this time'}
Pattern 2B - 2A + update method The difference is highlighted in green	
3. ELEMENTS(Placed in Layout described later): **pysimplegui is imported as sg for all code below**	

Single Line Input The Input key takes in user input and the output key is used to display the text according to the customisation of Text as shown with the code	**Syntax:** [psg.Text("text to display", attribute-name=value, attribute-name=value…), Input(default text, attribute-name=value,…….)]
Multiline input For longer data like address, titles, remarks you may need to get the text from users in multiple lines so that users can view large text at the same time. ***Input*** displays only one line by default, so it is better to use the multiline element for such user input.	**Syntax:** [psg.Text("text to display", attribute-name=value, attribute-name=value…), Multiline ()]
Buttons (multiple "types") • **File Browse-** get the path of a file and point it to the code Eg. [sg.FileBrowse()] • **Files Browse-** [sg.Input(key='-FILENAME-', visible=False, enable_events=True), sg.FilesBrowse()] • **File Save As-** [sg.Input(key='-SAVEAS-FILENAME-', visible=False, enable_events=True), sg.FileSaveAs()] • **FolderBrowse-** • **ColorChooserButton-** sg.ColorChooserButton("", size=(1, 1), target='set_line_color', button_color=('#1f77b4', '#1f77b4'),border_width=1, key='set_line_color_chooser') • **CalendarButton-**	**Syntax:** [psg.Button("Text of the Button" attribute-name=value, attribute-name=value…),….]
Button Menu	**Syntax:**

button_text is mandatory and represents the text that appears on the button for a menu list. If you don't want text to appear on button set it to blank string menu_options_list is the list of lists of Strings, a mandatory attribute/parameter(Click to see list of attributes for buttons) that contains the list of lists of options to be given to the users to select from	ButtonMenu(button_text, menu_options_list,[attribute=value, attribute=value,…])
Option Menu dropdown linked to a button to display a set of options to the user. values is the list of Strings, a mandatory attribute/parameter(Click to see list of attributes for Option Menu) that contains the list of options visible to the users.	**Syntax:** OptionMenu(values=Options_list,[attribute=value, attribute=value,…])
Checkboxes(Select 1) Possible Checklist Boxes: - Red | Green | Blue | Black | White - Italic | Bold | Underline | Strike	**Syntax:** [sg.Checkbox('My Checkbox', default=True)]
Radio Buttons(select multiple) Possible Radio Buttons: - Big | Small - Encrypted | Not encrypted - Attachment | No attachment - Receipt | No Receipt - Private | Public - Am | Fm | Longwave | Shortwave	**Syntax:** [sg.Radio('Permission Granted', "RADIO1", default=False)], [sg.Radio('Permission not Granted', "RADIO1", default=True)]
Graph A graph can be drawn using:	Click here to see the Expansion of graph drawing:

<table>
<tr><td>1. A Graph Element(class used for drawing similar to tkinter's canvas)
2. Embedded Matplotlib and draw</td><td>Bar Chart:
Sine Wave:
Line Graph:</td></tr>
<tr><td>Slider
Click here to see list of attributes</td><td>Syntax:
Slider(range, [attributes])</td></tr>
<tr><td>Vertical Separator</td><td>Syntax:
import PySimpleGUIQt as psg

psg.SetOptions(border_width=0)
lines = []
lines.append([psg.HorizontalSeparator()])
for line in range(4):
  cols = []
  col = []
  for cell in range(4):
  col = []
  cols.append(psg.VerticalSeparator())
  for x in range(3):
  row = []
  for y in range(3):
  row.append(psg.InputText(default_text='',
do_not_clear=True, size=(3, 1), change_submits=True,
justification='center'))
  col.append(row)
  cols.append(psg.Column(col))
  cols.append(psg.VerticalSeparator())
  lines.append(cols)
  lines.append([psg.HorizontalSeparator()])
layout = [*lines,
  [psg.OK(), psg.Cancel()]]

window = psg.Window('Submit
Question',auto_size_text=True).Layout(layout)

prev_val = {}
while True:
  event, value = window.Read()</td></tr>
<tr><td>Progress Bar
to see how an activity or task is progressing to completion</td><td>Syntax:
ProgressBar(max_value,[other attributes])</td></tr>
</table>

Click here to see list of attributes for Progress Bar	
Menu textual navigation element usually placed at the top of an application window. A menu provides all the actions, tasks, processes or activities available in the application containing the menu. Click here to see list of attributes for Menu	**Syntax:** Menu(menu_list_def,[attribute=value, attribute=value,…])
Listbox The Listbox displays a fixed number of values depending on the height of the list. If the values exceed the height, a vertical scroll appears automatically to browse through all the values. User can choose one or more values from the Listbox depending upon the ***selection mode*** set	**Syntax:** Listbox(values List, [attributes])
Combo Python Combo element in pySimpleGUI allows displaying a list of values to choose from in an application. The dropdown list of combo is visible when the small down arrow is clicked at the right end of element. The name combo means that you get a DropDown List along with the option to type the value if it is not available in the dropdown List. User can choose ***only one value*** from the combo dropdown list or type a value if he is unable to find a value of her choice Click here to see list of attributes	**Syntax:** Combo(values List, [attributes])
Stretch (Qt only)	Code Subjective to each case
Sizer (tkinter only) specify a size in pixels	Code Subjective to each case

Column A column can be understood as a container element appearing as a vertical section of a window	**Syntax:** sg.Column(layout,[other attributes])
Frame A frame is a container element that contains other pySimpleGUI elements. The frame groups the elements such that they can be relocated as one unit	**Syntax:** sg.Column(Title, layout,[other attributes])
Other components subject to requirements case by case: Tree,Table, Status Bar, image, pane, Tab, TabGroup, StatusBar	
4. Imports(NOTE: Never, ever: from RySimpleGUI import *)	
import PySimpleGUI as sg	tkinter port.
import PysimpleGuigt as sy	PySide2 port, Supports System Tray
import PySimpleGUIVIX as sg	Remi port
import PySimpleGUIVleb as sy	WxPython port
5. Layout(lists of lists of Elements): Combine them with control logic specified below, and we have a standalone function creation	
Elements	Elements are PySimpleGUI's "Widgets".
Widgets	Widgets refers to the underlying GUI's implementation
Text Element	A Text Element is implemented using tkinter's Label Widget
Spin Element	A Spin Element is tkinter's Spinbox, WxPython's SpinButton, Qt's QSpinBox, Remi's SpinBox
6. Container Elements: (Contain other Elements)	

Right click menu
Single icon click event
Double icon click event
Display message in tray
Message clicked event
Change icon
Runs in blocking or async mode (just like windows)

Changing Window Contents:

Element.update()
To change your window's contents, call update method for the element
A window must first be read or finalized prior to changing any element
To change an element, first locate the element. Use the element's key:
Window[key]

The old format was
window.find element (key)

Call the update method to change something about the element
window[key].update ('New value')

Each element has a different set of things that can be changed/updated
See the documentation/IDE for details on each element
First parameter is almost always the “value” of the element (text, button text, checked/unchecked, etc.)

Attributes for buttons:

default_value – String, value that is stored by the Button Menu if the user does not select one of the options in the list.

disabled- Boolean, set True or False to disable or enable to control use of button menu for interaction.

tooltip – String, a small help text that appears with mouse hover over the element.

default – Boolean, to enable or disable this pySimpleGUI element to allow or disallow the interaction of user with this element.

image_filename – String, name of the image file to display on the Button for this menu.

image_size– **–** (integer, integer), Size of the image in width and height.

border_width– Integer, to set width of the border of button.

background_color- String, color name to define the background color of the Python menu of buttons.

text_color- String, color name to define the text color of the Button Menu.

button_color – String, color name to define the color of the button of the menu

disabled_text_color – String, color name to define the text color of the disabled button in the button menu.

Size – (integer, integer), Size of the button menu in width and height.

font – String, font to be applied to the text of the button menu.

key/K – String, integer, tuple or object, the identification of the button menu by the window read method.

visible – Boolean, defines if the button menu is visible or not.

Option Menu:

default_value – String, value that is stored by the option menu if the user does not select one of the options in the list.

default – Boolean, to enable or disable the pySimpleGUI element to allow or disallow the interaction of user with this element.

background_color- String, color name to define the background color of the menu.

text_color- String, color name to define the text color of the option menu.

Size – (integer, integer), Size of the option menu in width and height.

font – String, font to be applied to the text of the option menu.

key/K – String, integer, tuple or object, the identification of the option menu by the window read method.

visible – Boolean, defines if the option menu is visible or not.

tooltip – String, a small help text that appears with mouse hover over the element.

max_value- Integer, the maximum value that will be achieved when a Python Progress Bar increments and progresses to its end.

Progress Bar:

orientation String, a value that defines how the progress bar will be presented in the interface. Allowed values are ‘horizontal’ or ‘vertical’

size (Integer, Integer), integer values to define width and height of the Progress Bar.

bar_color (String,String) a tuple of two color names that defines the fore and background colors respectively of the Progress Bar. First color is the moving bar color and second color is the background of the Progress Bar.

style String, it is set to define the style of the progress bar. The options are 'default', 'winnative', 'clam', 'alt', 'classic', 'vista', 'xpnative'

border_width Integer, to set the border width of the progress bar.

Key/K String, set the key that can be used to refer in coding for processing in the application

visible Boolean, to set the visibility of the Python Progress Bar.

Menu

background_color- String, color name to set the background color of the menu.

text_color- String, color name to define the text color of the menu.

disabled_text_color – String, color name to define the text color of the disabled options in the menu.

Size – (integer, integer), size of the menu in width and height.

font – String, font to be applied to the text of the menu.

key/K – String,integer, tuple or object, the identification of the menu to be used by the window read method.

visible – Boolean, used to set whether the menu should be visible to the user or not.

Combo:

default_value– String, it is the value that you want the Python combo to store if a user fails to select one. It is the value which is displayed automatically in the text area of the combo and is returned by the read method of the window.

size –(integer,integer), the pair of integer values to define the width and height of the combo.

background_color– String, sets the color of the combo's background.

text_color-String, sets the color of the text visible in the combo and its dropdown list.

disabled -Boolean, enables or disables user interaction in the dropdown. By default the combo is enabled to let user type or select an option. If disabled the combo is visible but user will not be able to use the combo. Values allowed are True or False.

key/K -String, the key assigned to combo element by which you can identify this element in the values list returned by the window read() method.

tooltip -String, the small text that appears when the user rests mouse cursor on the element.

font -String, to set the font family for combo text.

Slider:

range, (Integer, Integer), two values that represent the two end points of the range of values to select from

default_value, Integer/Float, a value from range list that you wish to set in the first go/ or make a default value of the Slider.

resolution Integer/Float, a value by which the Slider will move.

tick_interval Integer/Float, an interval after by which the tick will happen for a slider increment.

orientation String, a value that defines how the Slider will be oriented in the interface. Allowed values are ‘horizontal’ or ‘vertical’

disable_number_display Boolean, set True so that the numbers are displayed with movement of the Slider.

border_width Integer, to set the border width of the Slider.

disabled Boolean, By default disabled is False, to make it enabled for user to interact.

enable_events Boolean, by default it is set to False so that the Slider events are not triggered when the spin value is changed.

size (Integer, Integer), integer values to define width and height of the Slider element.

font String, Set a font name to define the font of the text of the Slider.

background_color String, Set a color name to set the color of the background of the Slider.

text_color String, Set a color name to set the color of the text of the Slider.

Key/K String, Set the key that can be used to refer for processing in the application

tooltip String, the tooltip for the Slider element

visible Boolean, to set the visibility of the Slider.

Graph Drawing and types:

Methods to draw:
DrawCircle: Draw a circle
DrawLine: Draw a line
DrawPoint: Place points
DrawRectangle: Draw a rectangle
DrawOval: Draw an arc
DrawImage: Display the image
DrawText: Draw characters
DrawLine: Used to draw lines, auxiliary lines, and scales on line graphs
DrawPoint: Used to draw plotted points on a line chart or to draw an arc like a sine wave
DrawText: Used to display graph labels and values per plot
Erase: Clears the drawing area. It is used for non to delete the graph that was being drawn when switching graphs
Move: Moves what is drawn in the x and y directions. Used to move the graph in real time

Bar Chart:

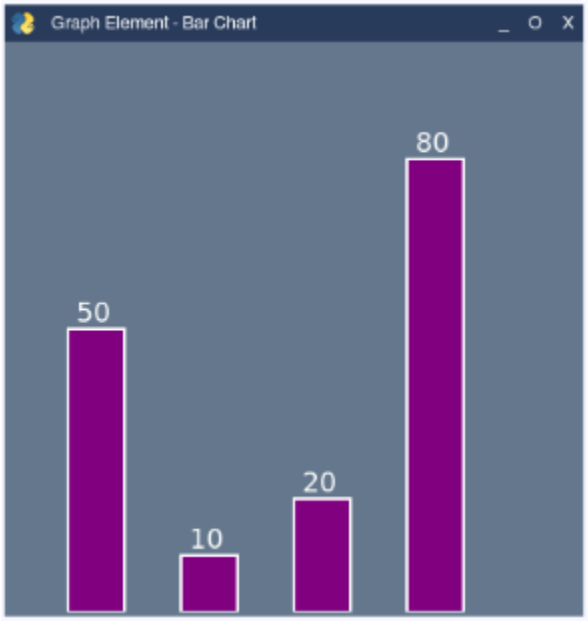
Steps:

1. Import PySimpleGUI
2. Make the Window
3. Loop through the data
4. Draw a Bar
5. Draw a label above it with the value of the data point
6. Wait for window to be closed by user

```
import PySimpleGUI as sg

window = sg.Window('Graph Element - Bar Chart', [[sg.Graph((400, 400), (0,0), (10, 100), k='-GRAPH-')]], finalize=True)

for i, data in enumerate([50, 10, 20,80]):
```

```
    window['-GRAPH-'].draw_rectangle((i*2+1, data), (i*2+2, 0), fill_color='purple',
line_color='white', line_width=2)
    window['-GRAPH-'].draw_text(f'       {data}', (i*2+1, data+3), color='white', font='_ 18')

window.read(close=True)
```

Output:



1. **The X-Axis labels for each bar**
2. **The color of each bar**
3. **The Y-Value for each bar**

```
import PySimpleGUI as sg

sg.theme_background_color('white')
sg.theme_element_background_color('white')

def main_bar_example():
    data_points = [.019, .026, .021, .051, .024, .058, .026]
    colors = ['#cb3ce3', '#b758e7', '#a374eb', '#8f90ef', '#73a4eb', '#57b8e7', '#3bcce3']
    dates  = ['2021-01-01','2021-02-01', '2021-03-01', '2021-04-01', '2021-05-01', '2021-06-01',
'2021-07-01']

    graph = sg.Graph((600, 600), (-1,-.01), (15, .07), k='-GRAPH-')

    layout = [[graph], [sg.Button('Exit')]]
```

```
    window = sg.Window('Graph Element - Example 1', layout, finalize=True)

    graph.draw_line((0,0), (14,0), width=3)
    graph.draw_line((0,0), (0,0.06), width=3)
    for i in range(7):
        graph.draw_text(f'{i*.01}', (-.5, i*.01), font='_ 10')
        graph.draw_line((-.1,i*.01), (0, i*.01), width=3)
    while True:

        for i, data in enumerate(data_points):
            color = colors[i]
            label = dates[i]
            graph.draw_rectangle((i*2+.1, data), (i*2+1.5, 0), fill_color=color, line_width=0)
            graph.draw_text(f'        {data}', (i*2+.2, data+.002), font='_ 15')
            graph.draw_text(f'          {label}', (i*2+.3, -.001), font='_ 10')

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

    window.close()

main_bar_example()
```

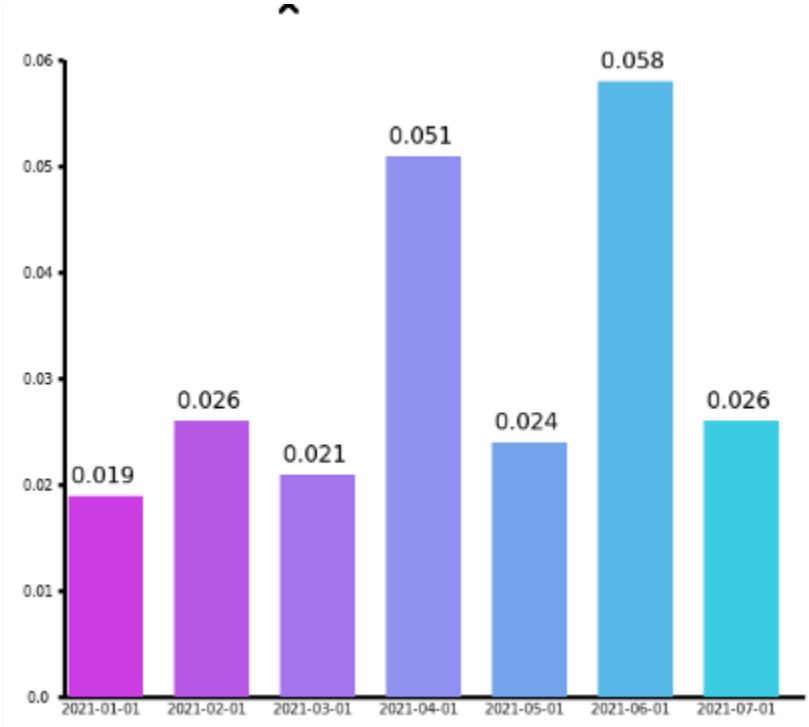
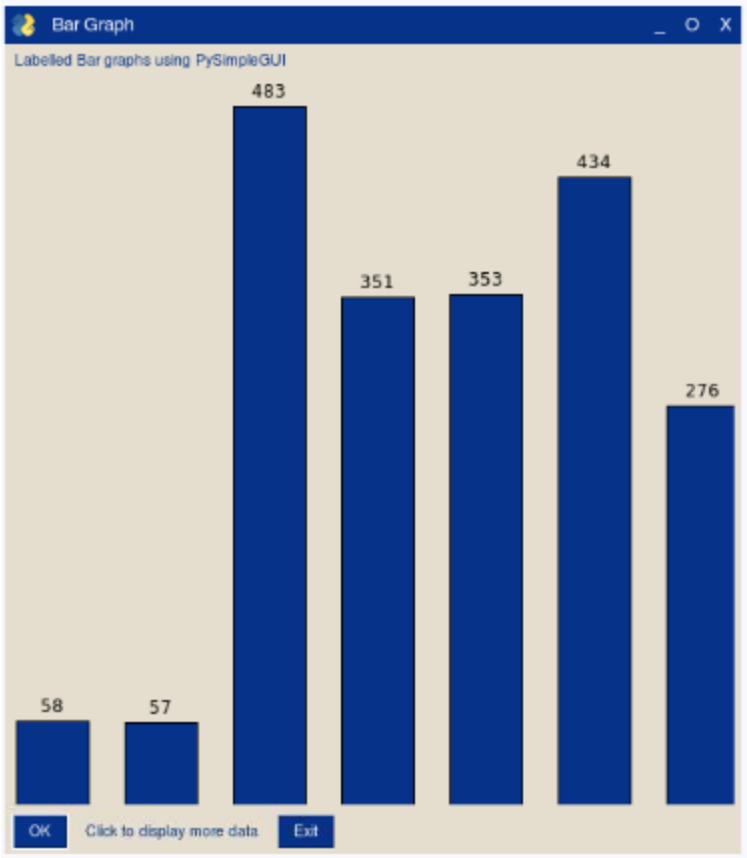


chart updates every time you click the button

```
import PySimpleGUI as sg
```

```
import random

BAR_WIDTH = 50      # width of each bar
BAR_SPACING = 75    # space between each bar
EDGE_OFFSET = 3     # offset from the left edge for first bar
GRAPH_SIZE= DATA_SIZE = (500,500)       # size in pixels

sg.theme('Light brown 1')

layout = [[sg.Text('Labelled Bar graphs using PySimpleGUI')],
          [sg.Graph(GRAPH_SIZE, (0,0), DATA_SIZE, k='-GRAPH-')],
          [sg.Button('OK'), sg.T('Click to display more data'), sg.Exit()]]

window = sg.Window('Bar Graph', layout, finalize=True)

graph = window['-GRAPH-']       # type: sg.Graph

while True:

    graph.erase()
    for i in range(7):
        graph_value = random.randint(0, GRAPH_SIZE[1])
        graph.draw_rectangle(top_left=(i * BAR_SPACING + EDGE_OFFSET, graph_value),
                             bottom_right=(i * BAR_SPACING + EDGE_OFFSET + BAR_WIDTH, 0),
                             fill_color=sg.theme_button_color()[1])

        graph.draw_text(text=graph_value, location=(i*BAR_SPACING+EDGE_OFFSET+25,
graph_value+10))

    # Normally at the top of the loop, but because we're drawing the graph first, making it at the
bottom
    event, values = window.read()

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


window.close()
```



Sine Wave graph:

```
# import PySimpleGUIWeb as sg
# import PySimpleGUIQt as sg
import PySimpleGUI as sg
import math

SIZE_X = 200
SIZE_Y = 100
NUMBER_MARKER_FREQUENCY = 25

def draw_axis():
    graph.draw_line((-SIZE_X,0), (SIZE_X, 0))               # axis lines
    graph.draw_line((0,-SIZE_Y), (0,SIZE_Y))

    for x in range(-SIZE_X, SIZE_X+1, NUMBER_MARKER_FREQUENCY):
        graph.draw_line((x,-3), (x,3))                      # tick marks
        if x != 0:
            graph.draw_text( str(x), (x,-10), color='green', font='Algerian 15')      # numeric labels

    for y in range(-SIZE_Y, SIZE_Y+1, NUMBER_MARKER_FREQUENCY):
        graph.draw_line((-3,y), (3,y))
```

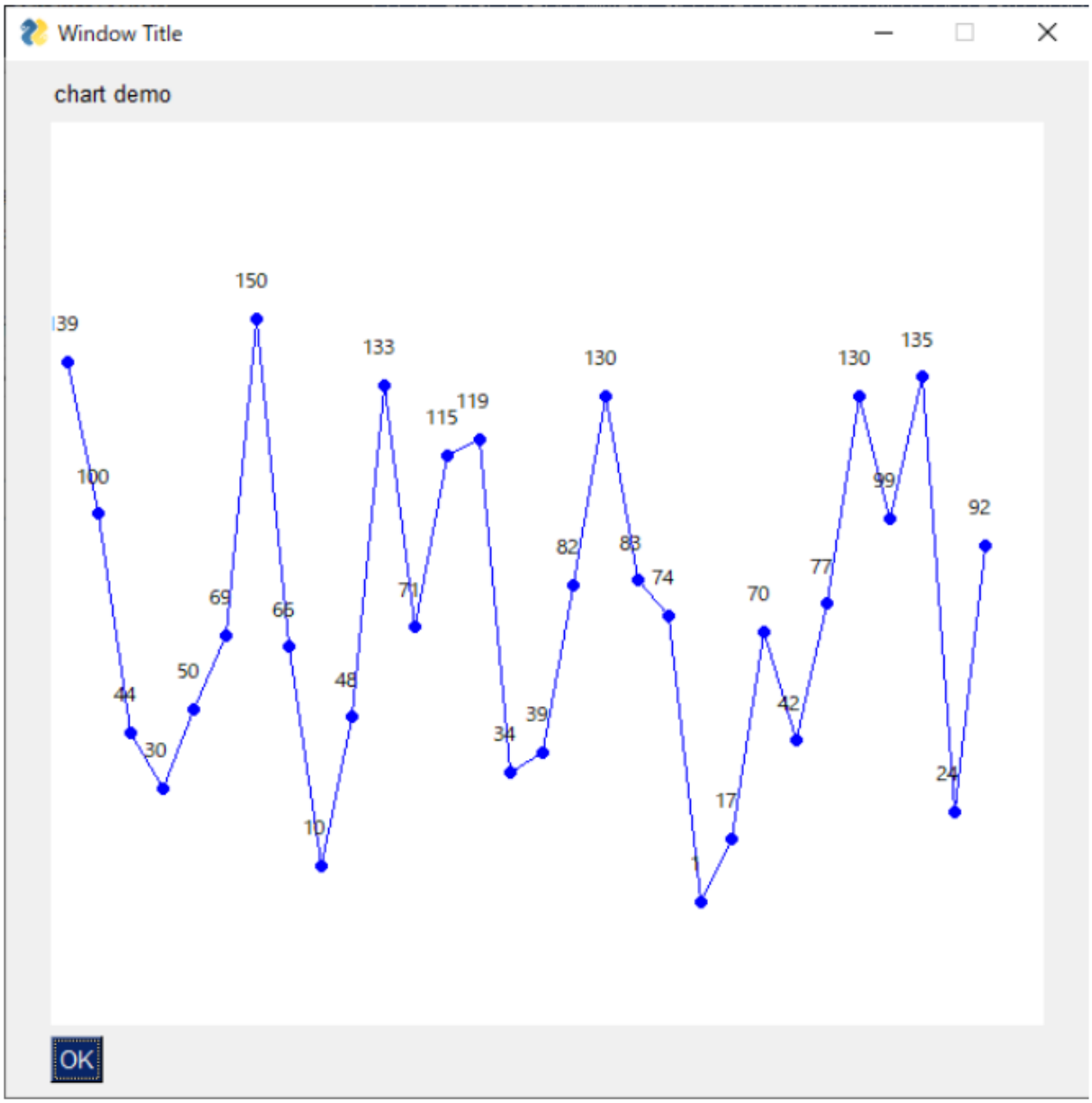
```
        if y != 0:
            graph.draw_text( str(y), (-10,y), color='blue')

sg.change_look_and_feel('DarkAmber')  # let's add a little color

# Create the graph that will be put into the window
graph = sg.Graph(canvas_size=(400, 400),
          graph_bottom_left=(-(SIZE_X+5), -(SIZE_Y+5)),
          graph_top_right=(SIZE_X+5, SIZE_Y+5),
          background_color='white',
          key='graph')
# Window layout
layout = [[sg.Text('Example of Using Math with a Graph', justification='center', size=(50,1),
relief=sg.RELIEF_SUNKEN)],
          [graph],
          [sg.Text('y = sin(x / x2 * x1)', font='Algerian 18')],
          [sg.Text('x1'),sg.Slider((0,200), orientation='h', enable_events=True,key='_SLIDER_')],
          [sg.Text('x2'),sg.Slider((1,200), orientation='h', enable_events=True,key='_SLIDER2_')]]

window = sg.Window('Graph of Sine Function', layout)

while True:
    event, values = window.read()
    if event is None:
        break
    graph.erase()
    draw_axis()
    prev_x = prev_y = None
    for x in range(-SIZE_X,SIZE_X):
        y = math.sin(x/int(values['_SLIDER2_']))*int(values['_SLIDER_'])
        if prev_x is not None:
            graph.draw_line((prev_x, prev_y), (x,y), color='red')
        prev_x, prev_y = x, y
```

Line Graph:

```
  import
  PySimpl
  eGUI as
  sg
                  import random
                  import json
                  import datetime
```

```
BAR_WIDTH = 10
# BAR_WIDTH = 5
BAR_SPACING = 16
# BAR_SPACING = 6
EDGE_OFFSET = 3
GRAPH_SIZE = (550,500)
DATA_SIZE = (500,200)
graph = sg.Graph(GRAPH_SIZE, (0,-30), DATA_SIZE,
background_color='white',)


layout = [[sg.Text('chart demo')],
        [graph],
        [sg.Button('OK')]]


window = sg.Window('Window Title', layout)


before_value = 0
while True:
  event, values = window.Read()
  graph.Erase()
  if event is None:
     break


  for i in range(30):
     graph_value = random.randint(0, 150)
     if i > 0:
        graph.DrawLine(((i-1) * BAR_SPACING + EDGE_OFFSET+
BAR_WIDTH/2 ,  before_value)  ,  (i * BAR_SPACING + EDGE_OFFSET+
BAR_WIDTH/2 , graph_value ),color='blue', width=1 )


     graph.DrawText(text=graph_value,
location=(i*BAR_SPACING+EDGE_OFFSET+2, graph_value+10))


     graph.DrawPoint((i * BAR_SPACING + EDGE_OFFSET+ BAR_WIDTH/2
,graph_value), size=3 ,color='blue',)
     before_value = graph_value


window.Close()
```

Line graph



Valid Themes:

['Black', 'BlueMono', 'BluePurple', 'BrightColors', 'BrownBlue', 'Dark', 'Dark2', 'DarkAmber', 'DarkBlack', 'DarkBlack1', 'DarkBlue', 'DarkBlue1', 'DarkBlue10', 'DarkBlue11', 'DarkBlue12', 'DarkBlue13', 'DarkBlue14', 'DarkBlue15', 'DarkBlue16', 'DarkBlue17', 'DarkBlue2', 'DarkBlue3', 'DarkBlue4', 'DarkBlue5', 'DarkBlue6', 'DarkBlue7', 'DarkBlue8', 'DarkBlue9', 'DarkBrown', 'DarkBrown1', 'DarkBrown2', 'DarkBrown3', 'DarkBrown4', 'DarkBrown5', 'DarkBrown6', 'DarkBrown7', 'DarkGreen', 'DarkGreen1', 'DarkGreen2', 'DarkGreen3', 'DarkGreen4', 'DarkGreen5', 'DarkGreen6', 'DarkGreen7', 'DarkGrey', 'DarkGrey1', 'DarkGrey10', 'DarkGrey11',

'DarkGrey12', 'DarkGrey13', 'DarkGrey14', 'DarkGrey2', 'DarkGrey3', 'DarkGrey4', 'DarkGrey5', 'DarkGrey6', 'DarkGrey7', 'DarkGrey8', 'DarkGrey9', 'DarkPurple', 'DarkPurple1', 'DarkPurple2', 'DarkPurple3', 'DarkPurple4', 'DarkPurple5', 'DarkPurple6', 'DarkPurple7', 'DarkRed', 'DarkRed1', 'DarkRed2', 'DarkTanBlue', 'DarkTeal', 'DarkTeal1', 'DarkTeal10', 'DarkTeal11', 'DarkTeal12', 'DarkTeal2', 'DarkTeal3', 'DarkTeal4', 'DarkTeal5', 'DarkTeal6', 'DarkTeal7', 'DarkTeal8', 'DarkTeal9', 'Default', 'Default1', 'DefaultNoMoreNagging', 'GrayGrayGray', 'Green', 'GreenMono', 'GreenTan', 'HotDogStand', 'Kayak', 'LightBlue', 'LightBlue1', 'LightBlue2', 'LightBlue3', 'LightBlue4', 'LightBlue5', 'LightBlue6', 'LightBlue7', 'LightBrown', 'LightBrown1', 'LightBrown10', 'LightBrown11', 'LightBrown12', 'LightBrown13', 'LightBrown2', 'LightBrown3', 'LightBrown4', 'LightBrown5', 'LightBrown6', 'LightBrown7', 'LightBrown8', 'LightBrown9', 'LightGray1', 'LightGreen', 'LightGreen1', 'LightGreen10', 'LightGreen2', 'LightGreen3', 'LightGreen4', 'LightGreen5', 'LightGreen6', 'LightGreen7', 'LightGreen8', 'LightGreen9', 'LightGrey', 'LightGrey1', 'LightGrey2', 'LightGrey3', 'LightGrey4', 'LightGrey5', 'LightGrey6', 'LightPurple', 'LightTeal', 'LightYellow', 'Material1', 'Material2', 'NeutralBlue', 'Purple', 'Python', 'Reddit', 'Reds', 'SandyBeach', 'SystemDefault', 'SystemDefault1', 'SystemDefaultForReal', 'Tan', 'TanBlue', 'TealMono', 'Topanga']