Add GUIs to your programs and scripts easily with PySimpleGUI

PySimpleGUI now supports BOTH Python 2.7 and Python 3

Introduction

Few people run Python programs by double clicking the .py file as if it were a .exe file. When a typical user (non-programmer types) double clicks an exe file, they expect it to pop open with a window they can interact with. While GUIs, using tkinter, are possible using standard Python installations, it's unlikely many programs do this.

What if it were easy so to open a Python program into a GUI that complete beginners could do it? Would anyone care? Would anyone use it? It's difficult to answer because to date it's not been "easy" to build a custom GUI.

There seems to be a gap in the ability to add a GUI onto a Python program/script. Complete beginners are left using only the command line and many advanced programmers don't want to take the time required to code up a tkinter GUI.

v: latest

GUI Frameworks

There is no shortage of GUI frameworks for Python. tkinter, WxPython, Qt, Kivy are a few of the major packages. In addition, there are a good number of dumbed down GUI packages that wrap one of the major packages. These include EasyGUI, PyGUI, Pyforms, ...

The problem is that beginners (those with experience of less than 6 weeks) are not capable of learning even the simplest of the major packages. That leaves the wrapper-packages. Users will likely find it difficult or impossible to build a custom GUI layout using the smaller packages.

PySimpleGUI attempts to address these GUI challenges by providing a super-simple, easy to understand interface to GUIs that can be easily customized. Complex GUIs are often less than 20 lines of code when PySimpleGUI is used.

The Secret

What makes PySimpleGUI superior for newcomers is that the package contains the majority of the code that the user is normally expected to write. Button callbacks are handled by PySimpleGUI, not the user's code. Beginners struggle to grasp the concept of a function, expecting them to understand a call-back function in the first few weeks is a stretch.

With some GUIs arranging the GUI Widgets often requires several lines of code.... at least one or two lines per widget. PySimpleGUI uses an "auto-packer" that creates the layout for the user automatically. There is no concept of a pack nor a grid system needed to layout a GUI Window.

Finally, PySimpleGUI leverages the Python language constructs in clever ways that shortens the amount of code and returns the GUI data in a straightforward manner. When a Widget is created in a window layout, it is configured in-place, not several lines of code away. Results are returned as a simple list or a dictionary.

What is a GUI?

Most GUIs do one thing.... they collect information from the user and return it. From a programmer's viewpoint a GUI that collects information, like a window, could be summed up as a function call that looks like this:

```
button, values = GUI_Display(gui_layout)
```

What's expected from most GUIs is the button that was clicked (OK, cancel, save, yes, no, etc), and the values that were input by the user. The essence of a GUI can be boiled down into a single line of code.

This is exactly how PySimpleGUI works (for these simple kinds of GUIs). When the call is made to display the GUI, execution does no return until a button is clicked that closes the window.

There are more complex GUIs such as those that don't close after a button is clicked. These resemble a windows program and also be created with PySimpleGUI. A remote control interface for a robot and a chat window are a couple of examples where you want to keep the window open after a button is clicked.

The 5-Minute GUI

When is PySimpleGUI useful? ***Immediately***, anytime you've got a GUI need. It will take under 5 minutes for you to create and try your GUI. With those kinds of times, what do you have to lose trying it?

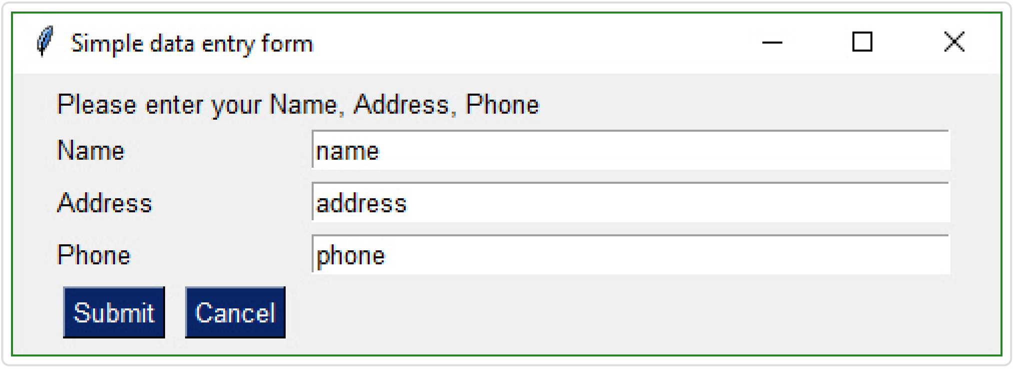
v: latest

The best way to go about making your GUI in under 5 minutes is to copy one of the GUIs from the PySimpleGUI Cookbook (https://pysimplegui.readthedocs.io/en/latest/cookbook/). Follow these steps:
*Install PySimpleGUI (see short section in readme on installation) * Find a GUI that looks similar to what you want to create*
Copy code from Cookbook
* Paste into your IDE and run

Let's look at the first recipe from the book

```
import PySimpleGUI as sg

# Very basic window.  Return values as a list

layout = [
          [sg.Text('Please enter your Name, Address, Phone')],
          [sg.Text('Name', size=(15, 1)), sg.InputText('name')],
          [sg.Text('Address', size=(15, 1)), sg.InputText('address')],
          [sg.Text('Phone', size=(15, 1)), sg.InputText('phone')],
          [sg.Submit(), sg.Cancel()]
         ]

window = sg.Window('Simple data entry window').Layout(layout)
button, values = window.Read()

print(button, values[0], values[1], values[2])
```

It's a reasonably sized window.



If you only need to collect a few values and they're all basically strings, then you would copy this recipe and modify it to suit your needs.

Python 2.7 Differences

The only noticeable difference between PySimpleGUI code running under Python 2.7 and one running on Python 3 is the import statement.

Python 3.x:

v: latest

```
import PySimpleGUI as sg
```

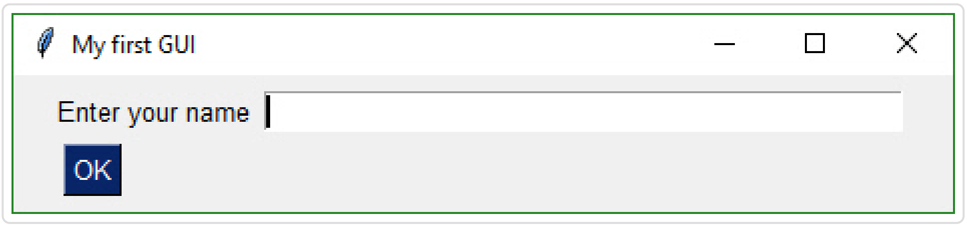
Python 2.7:

```
import PySimpleGUI27 as sg
```

The 5-line GUI

Not all GUIs take 5 minutes. Some take 5 lines of code. This is a GUI with a custom layout contained in 5 lines of code.

```
import PySimpleGUI as sg

layout = [ [sg.Text('Enter your name'), sg.InputText()],
           [sg.OK()] ]

window = sg.Window('My first GUI').Layout(layout)
button, (name,) = window.Read()
```



Making Your Custom GUI

If you find a Recipe similar to your project. You may be able to modify the code within 5 minutes in order to get to *your layout*, assuming you've got a straightforward layout.

Widgets are called Elements in PySimpleGUI. This list of Elements are spelled exactly as you would type it into your Python code.

Core Element list

v: latest

```
  Buttons including these types:
      File Browse
      Folder Browse
      Color chooser
      Date picker
      Read window
      Close window
      Realtime
  Checkbox
  Radio Button
  Listbox
  Slider
  Multi-line Text Input
  Scroll-able Output
  Progress Bar
  Option Menu
  Image
  Menu
  Frame
  Column
  Graph
  Table
  Tabbed windows
  Redirected Python Output/Errors to scrolling Window
```

You can also have short-cut Elements. There are 2 types of shortcuts. One is simply other names for the exact same element (e.g. T instead of Text). The second type configures an Element with particular setting, sparing the programmer from specifying all of the parameters (e.g. Submit is a button with the text "Submit" on it).

Shortcut list

```
T = Text
Txt = Text
In = InputText
Input = IntputText
Combo = DropDown = Drop = InputCombo
DropDown = InputCombo
Drop = InputCombo
OptionMenu = InputOptionMenu
CB - CBox = Check = Checkbox
RButton = ReadButton
Button = SimpleButton
```

A number of common buttons have been implemented as shortcuts. These include:

Button Shortcuts

v: latest

```
FolderBrowse
FileBrowse
FilesBrowse
FileSaveAs
Save
Open
Submit
OK
Ok
Cancel
Quit
Help
Exit
Yes
No
```

The more generic button functions, that are also shortcuts

Generic Buttons

```
Button
RButton (ReadButton)
RealtimeButton
```

These are all of the GUI Widgets you have to choose from. If it's not in this list, it doesn't go in your window layout. (Maybe... unless there's been an update with more features and this tutorial wasn't updated)

GUI Design Pattern

The stuff that tends not to change in GUIs are the calls that setup and show the Window. It's the layout of the Elements that changes from one program to another. This is the code from above with the layout removed:

```
import PySimpleGUI as sg
# Define your window here (it's a list of lists)
layout = [[ your layout ]]
window = sg.Window('Simple data entry window').Layout(layout)
button, values = window.Read()
```

The flow for most GUIs is:
* Create the window object
* Define GUI as a list of lists
* Show the GUI and get results

Some windows act more like Windows programs. These windows have an "Event Loop". Please see the readme for more info on these kinds of windows (Persistent windows)

These are line for line what you see in design pattern above.

GUI Layout

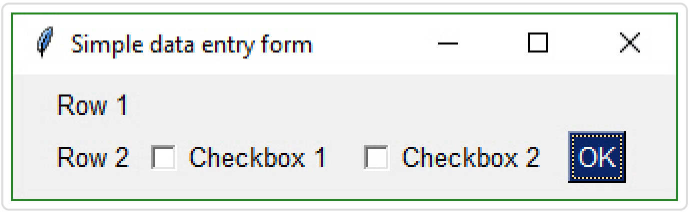
v: latest

To create your custom GUI, first break your window down into "rows". You'll be defining your window one row at a time. Then for each for, you'll be placing one Element after another, working from left to right.

The result is a "list of lists" that looks something like this:

```
layout = [  [Text('Row 1')],
            [Text('Row 2'), Checkbox('Checkbox 1', OK()), Checkbox('Checkbox 2'), OK()] ]
```

The layout produced this window:



Display GUI & Get Results

Once you have your layout complete and you've copied over the lines of code that setup and show the window, it's time to look at how to display the window and get the values from the user.

First get a window and display it.
`window = sg.Window('Simple data entry window').Layout(layout)`

Then read the window to get the button clicked and values.:

```
button, values = window.Read()
```

windows return 2 values, the text of the button that was clicked and a ***list of values*** the user entered into the window. More advanced windows return the values as a **dictionary of values**,

If the example window was displayed and the user did nothing other than click the OK button, then the results would have been:

```
button == 'OK'
values == [False, False]
```

Checkbox Elements return a value of True/False. Because these checkboxes defaulted to unchecked, the values returned were both False.

Displaying Results

Once you have the values from the GUI it would be nice to check what values are in the variables. Rather than print them out using a `print` statement, let's stick with the GUI idea and output to a window.

PySimpleGUI has a number of Popup Windows to choose from. The data passed to the Popup will be displayed in a window. The function takes any number of arguments, just like a print call would. Simply pass in all the variables you would like to see in the call.

v: latest

The most-commonly used display window is the `Popup`. To display the results of the previous example, one would write:

```
Popup('The GUI returned:', button, values)
```

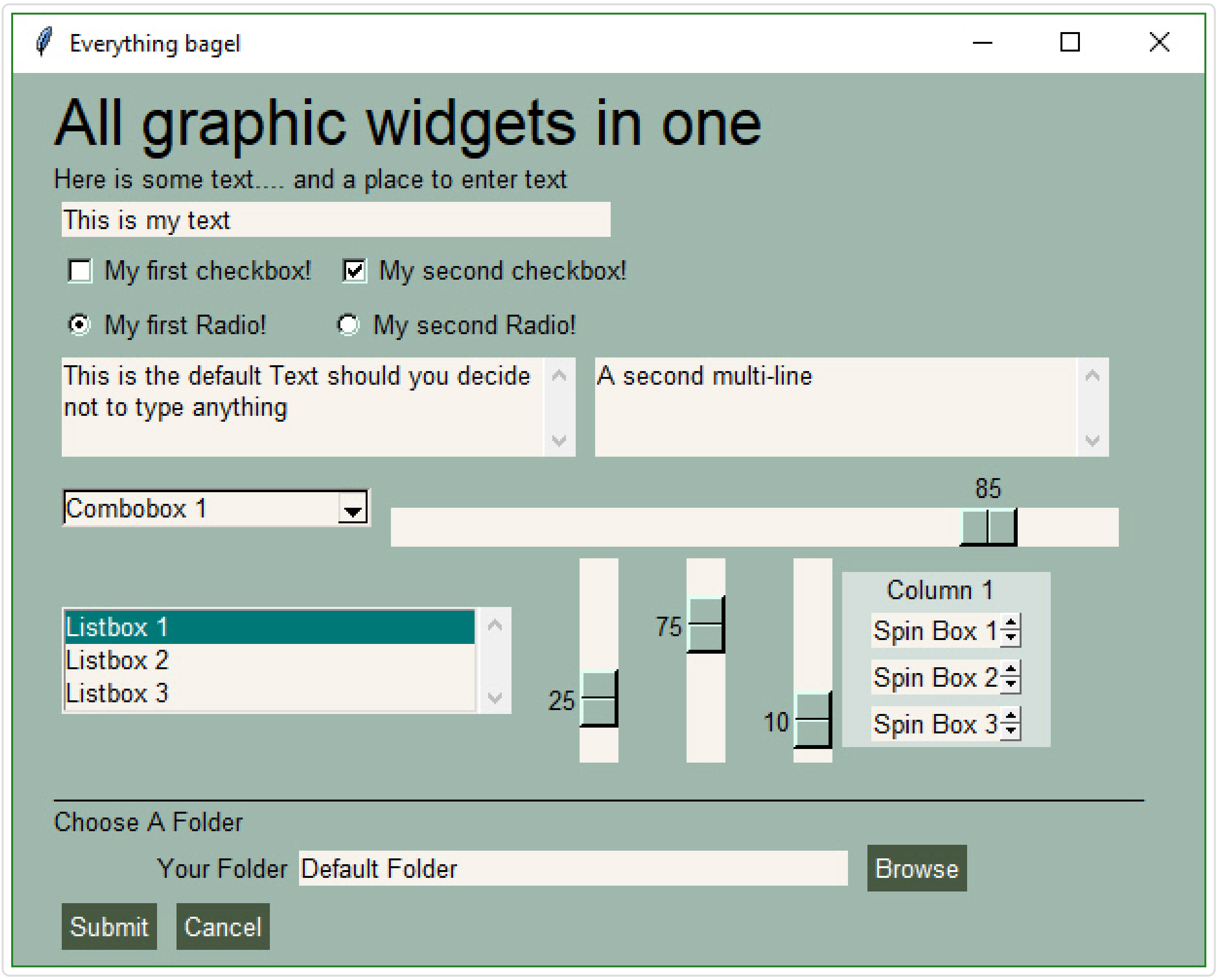
Putting It All Together

Now that you know the basics, let's put together a window that contains as many PySimpleGUI's elements as possible. Also, just to give it a nice look, we'll change the "look and feel" to a green and tan color scheme.

```
import PySimpleGUI as sg

sg.ChangeLookAndFeel('GreenTan')

column1 = [[sg.Text('Column 1', background_color='#d3dfda', justification='center', size=(10, 1))],
           [sg.Spin(values=('Spin Box 1', '2', '3'), initial_value='Spin Box 1')],
           [sg.Spin(values=('Spin Box 1', '2', '3'), initial_value='Spin Box 2')],
           [sg.Spin(values=('Spin Box 1', '2', '3'), initial_value='Spin Box 3')]]
layout = [
    [sg.Text('All graphic widgets in one window!', size=(30, 1), font=("Helvetica", 25))],
    [sg.Text('Here is some text.... and a place to enter text')],
    [sg.InputText('This is my text')],
    [sg.Checkbox('My first checkbox!'), sg.Checkbox('My second checkbox!', default=True)],
    [sg.Radio('My first Radio!     ', "RADIO1", default=True), sg.Radio('My second Radio!', "RADIO1")],
    [sg.Multiline(default_text='This is the default Text should you decide not to type anything', size=
     sg.Multiline(default_text='A second multi-line', size=(35, 3))],
    [sg.InputCombo(('Combobox 1', 'Combobox 2'), size=(20, 3)),
     sg.Slider(range=(1, 100), orientation='h', size=(34, 20), default_value=85)],
    [sg.Listbox(values=('Listbox 1', 'Listbox 2', 'Listbox 3'), size=(30, 3)),
     sg.Slider(range=(1, 100), orientation='v', size=(5, 20), default_value=25),
     sg.Slider(range=(1, 100), orientation='v', size=(5, 20), default_value=75),
     sg.Slider(range=(1, 100), orientation='v', size=(5, 20), default_value=10),
     sg.Column(column1, background_color='#d3dfda')],
    [sg.Text('_'  * 80)],
    [sg.Text('Choose A Folder', size=(35, 1))],
    [sg.Text('Your Folder', size=(15, 1), auto_size_text=False, justification='right'),
     sg.InputText('Default Folder'), sg.FolderBrowse()],
    [sg.Submit(), sg.Cancel()]
]

window = sg.Window('Everything bagel', default_element_size=(40, 1)).Layout(layout)
button, values = window.Read()
sg.Popup(button, values)
```

That may seem like a lot of code, but try coding this same GUI layout using any other GUI framework and it will be lengthier and what you see here.... by a WIDE margin. 10's of times longer.

v: latest



Everything bagel

All graphic widgets in one

Here is some text.... and a place to enter text

This is my text

☐ My first checkbox! ☑ My second checkbox!

● My first Radio! ○ My second Radio!

This is the default Text should you decide not to type anything

A second multi-line

Combobox 1

85

Listbox 1
Listbox 2
Listbox 3

25 75 10

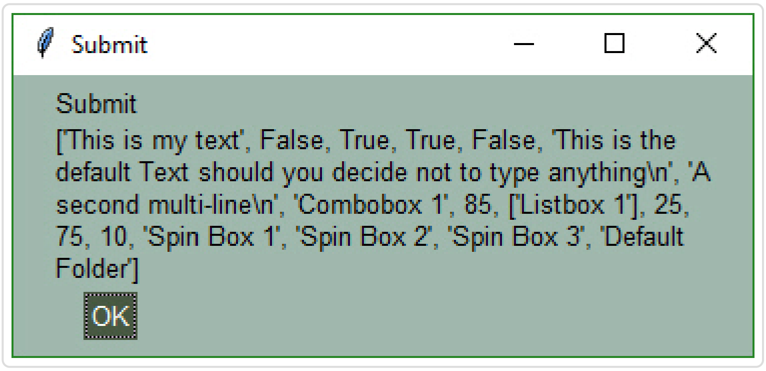
Column 1
Spin Box 1
Spin Box 2
Spin Box 3

Choose A Folder

Your Folder Default Folder Browse

Submit Cancel

The last line of code opens a message box. This is how it looks:



Submit

Submit
['This is my text', False, True, True, False, 'This is the default Text should you decide not to type anything\n', 'A second multi-line\n', 'Combobox 1', 85, ['Listbox 1'], 25, 75, 10, 'Spin Box 1', 'Spin Box 2', 'Spin Box 3', 'Default Folder']

OK

Each parameter to the message box call is displayed on a new line. There are actually 2 lines of text in the message box. The second line is very long and wrapped a number of times

Take a moment and pair up the results values with the GUI to get an understanding of how results are created and returned.

Adding a GUI to Your Program or Script

If you have a script that uses the command line, you don't have to abandon it in order to add a GUI. An easy solution is that if there are zero parameters given on the command line, then the GUI is run. Otherwise, execute the command line as you do today.

This kind of logic is all that's needed:

```
if len(sys.argv) == 1:
   # collect arguments from GUI
else:
    # collect arguements from sys.argv
```

Copy one of the Recipes from the Cookbook and run it. See if it resembles something you would like to build: PySimpleGUI Cookbook (https://pysimplegui.readthedocs.io/en/latest/cookbook/)

Have some fun! Spice up the scripts you're tired of running by hand. Spend 5 or 10 minutes playing with the demo scripts. You may find one already exists that does exactly what you need. If not, you will find it's 'simple' to create your own. If you really get lost, you've only invested 10 minutes.

Resources

Installation

Requires Python 3

```
pip install PySimpleGUI
```

If on a Raspberry Pi or Linux, may need to do this instead:

```
sudo pip3 install --upgrade pysimplegui
```

Works on all systems that run tkinter, including the Raspberry Pi

Documentation

Main manual (http://www.PySimpleGUI.org)

Cookbook (http://cookbook.PySimpleGUI.org)

Tutorial (http://tutorial.pysimplegui.org)

Home Page (GitHub)

www.PySimpleGUI.com (http://www.PySimpleGUI.com)

v: latest

Documentation built with MkDocs (http://www.mkdocs.org/).

v: latest