PySimpleGUI 2020
Clearly Seeing a GUI for You

"Simplicity is the ultimate sophistication" - Leonardo Da Vinci

The Basic Questions:

- Who
    - Is it for? Students, beginners, experienced developers, non-professional programmers, practical-programmers of *all ages* & *experience*
    - You - fellow developer, are the center of the PSG Universe. It really is all about you and your success.
    - Me - I'm Mike. Long history of commercial product development, no prior GUI history ("avoided like the plague"). Newcomer to Python. I don't "know it all" but know a few things and like to teach/share.
    - User community – The PySimpleGUI users are awesome, helpful.
- What is PySimpleGUI?
    - PySimpleGUI is a multi-OS, multi-GUI-framework Python package
    - Windows, Linux, Mac, Raspberry Pi, Android using PyDroid3
    - Open Source LGPL licensed, commercially managed project

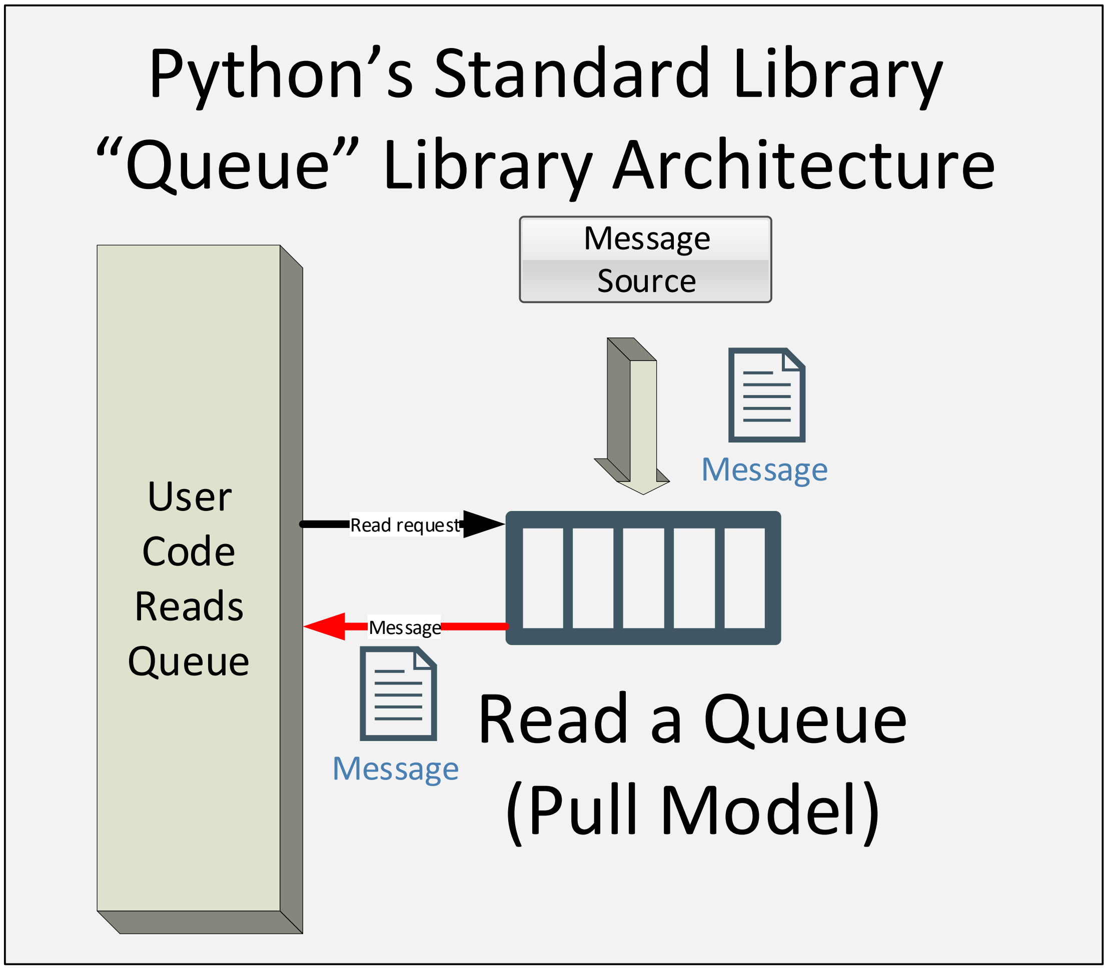
    - Intellectual Property from another (patent pending) product
    - Written in Python, for Python, utilizing Python features
    - If PySimpleGUI resonates with you great, it not, that's OK there are plenty of choices. There is no "right answer".
    - Simple describes the ease of working with PySimpleGUI, ***not*** the problem space it can solve.
    - Beautiful & magical
- When
    - Released July 2018. PSG is young, still growing, not complete...yet.
- Where
    - pip install from PyPI or copy single PySimpleGUI.py file
    - http://www.PySimpleGUI.com GitHub
    - http://www.PySimpleGUI.org The documentation
    - http://Cookbook.PySimpleGUI.org Cookbook
    - http://Trinket.PySimpleGUI.org PySimpleGUI in your browser
    - http://Demos.PySimpleGUI.org long list of demos
- Why
    - Had a practical need for a GUI
    - Saw the magic and wanted to share
- How
    - Uses a message passing architecture & converts to OOP arch
    - Converts your calls into underlying GUI's calls
    - You are using the exact same underlying GUI widgets
- How Much / How Many
    - 4 Primary ports – tkinter, Qt, WxPython Web (Remi)
    - 1 .py file
    - 845,000 pip installs
    - 3,600 GitHub stars
    - 1 developer + a few consultants & friends
    - $0 – Your cost to use
    - $0 – Your cost for support

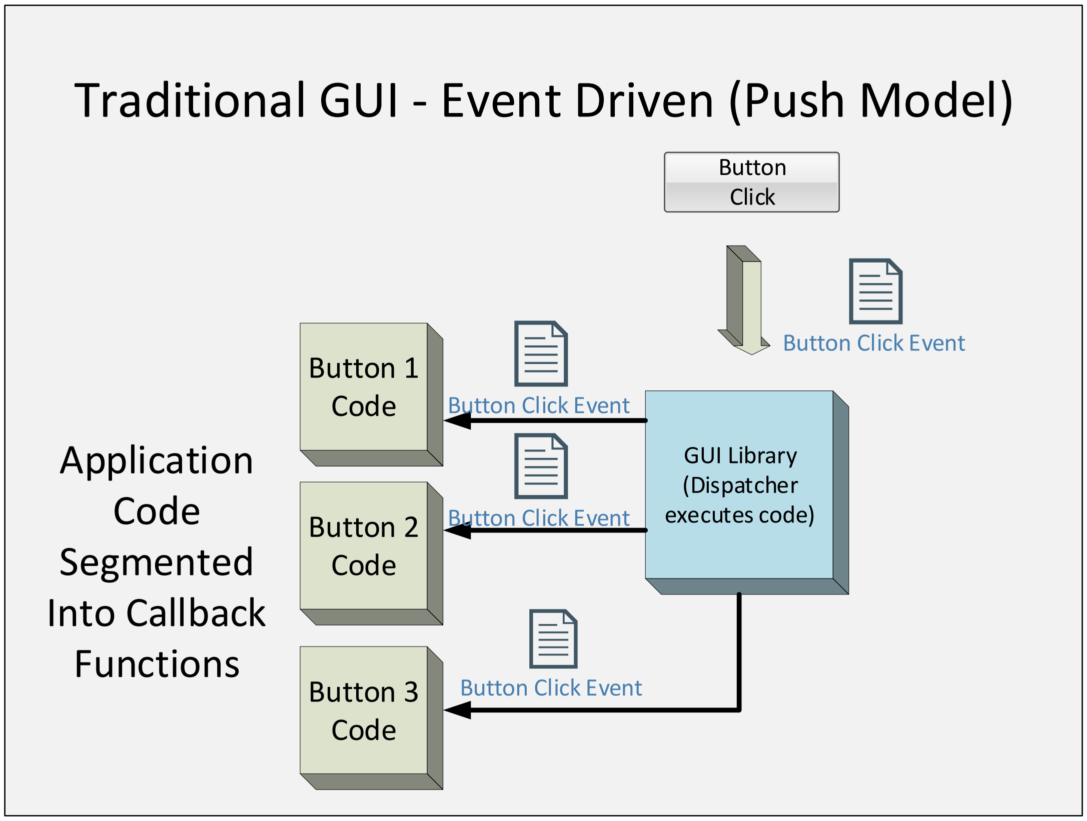
    - $0 – Your cost to say “thank you”
- Goals
    - Get the job done… get from point A to point B
    - Have fun
    - Be simple (shockingly simple when possible)
    - Democratize GUIs
    - Provide Python-like APIs
    - Up in 5 minutes
    - Copy, Paste, Run
    - Success after success
    - Execute the vision
    - Sustainability

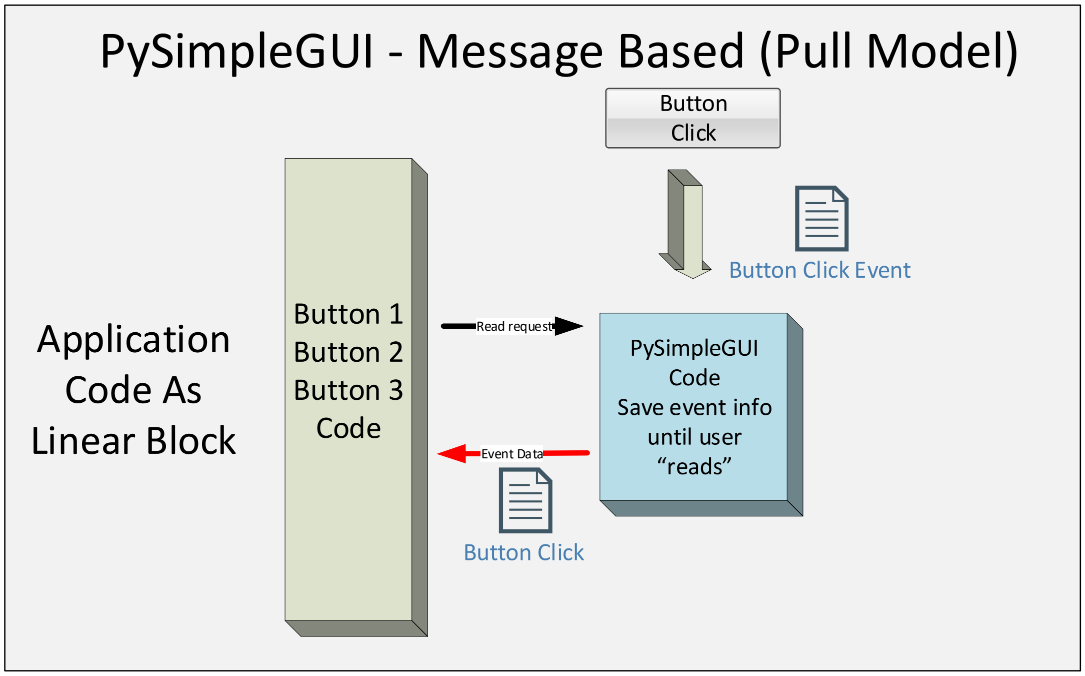
IDEs are important

- PyCharm is preferred
- Spend a weekend on your IDE
- Setup colors
- Learn commands
    - ^Q and ^P are your friends - docstrings
    - SINGLE key run
    - ^/ - Comment / uncomment line
    - CTRL+ALT+SHIFT+INSERT – New scratch
- Can Use IDE instead of documentation for call signatures (tkinter version)
- PyCharm and VS Code are the 2 officially supported IDEs. Others can struggle with PySimpleGUI if they are tkinter based (e.g. IDLE).

Event Driven Vs Message Passing Architectures

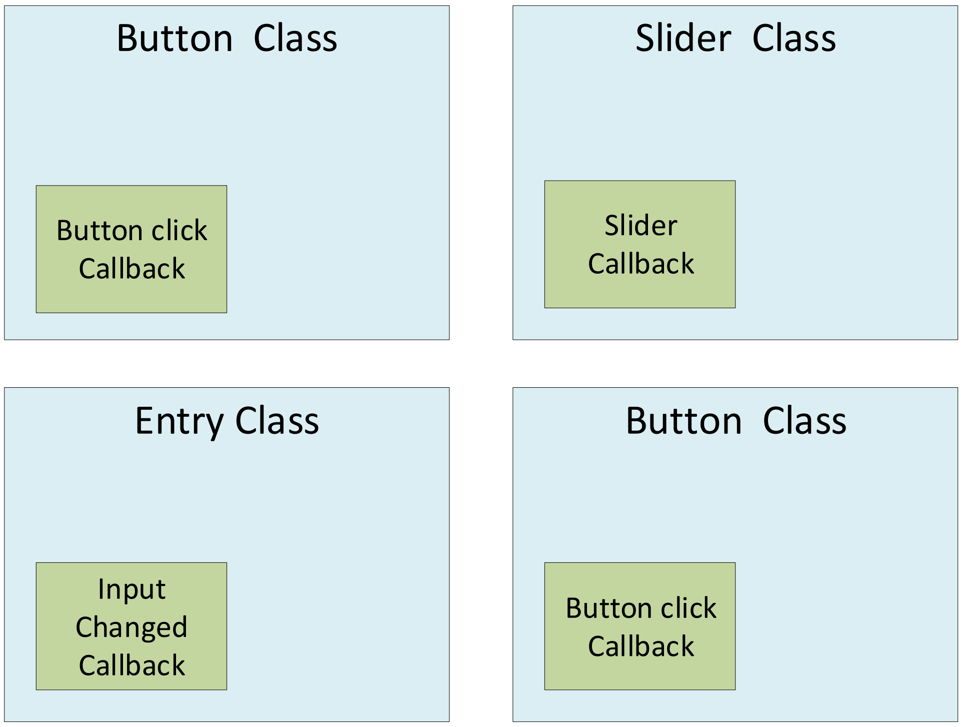

Python's Standard Library
"Queue" Library Architecture
Message Source
Message
User Code Reads Queue
Read request
Message
Message
Read a Queue (Pull Model)


Traditional GUI - Event Driven (Push Model)
Button Click
Button Click Event
Button 1 Code
Button Click Event
Button Click Event
GUI Library (Dispatcher executes code)
Button 2 Code
Application Code Segmented Into Callback Functions
Button Click Event
Button 3 Code

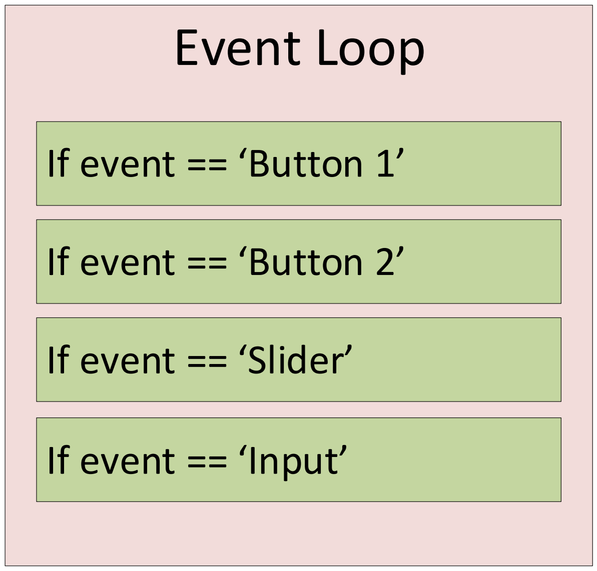

PySimpleGUI - Message Based (Pull Model)
Button Click
Button Click Event
Button 1 Button 2 Button 3 Code
Read request
PySimpleGUI Code Save event info until user "reads"
Application Code As Linear Block
Event Data
Button Click

Callbacks & Message Passing How they "Feel to me"

Callbacks feel disjointed



PySimpleGUI feels linear



Installing

Via pip

- pip install PySimpleGUI
  or
- pip3 install PySimpleGUI

GitHub

- Download PySimpleGUI.py and place in your application’s folder
- Use the new “upgrade from GitHub” tools

GitHub

- http://www.PySimpleGUI.com
- Latest code – Each port in a different folder with a readme
- Issues – Please use the Issue Form
- Demos – Universal folder and per-port folders

The PySimpleGUI “System”

Pieces designed to work together for a positive experience

Trinket – The 100,000 foot view. Get a “taste” without installing. http://Trinket.PySimpleGUI.org

Main Docs - http://www.PySimpleGUI.org

- Learn it, Love it, Live it
- Too much?
    - Sidebar Table of Contents
    - Use Control+F
    - Last sections have call signatures
- Mainly covers tkinter port. Separate readme for each port on GitHub

Cookbook – “Your Recipe for Success” http://Cookbook.PySimpleGUI.org

Demos - http://Demos.PySimpleGUI.org

- Teaches you coding conventions, style and techniques
- Per Element usage
- Integration to other packages
- Primary list sometimes runs on multiple ports

Coding Conventions

By following a coding convention, PySimpleGUI users can understand each other's code quickly. It gives us all a "common language" to speak.

The primary *suggested* conventions are:

- `import PySimpleGUI as sg`
- Name your Window `window`
- Read your window into variables `event` and `values`
- Name your layout `layout`
- Use `window[key]` to lookup elements
- For keys that are strings, follow the pattern `'-KEY-'`

All of these can be seen in this short program:

```python
import PySimpleGUI as sg

layout = [  [sg.Text('My Window')],
            [sg.Input(key='-IN-')],
            [sg.Text(size=(20,1), key='-OUT-')],
            [sg.Button('Go'), sg.Button('Exit')]  ]

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

while True:             # Event Loop
    event, values = window.read()
    print(event, values)
    if event in (None, 'Exit'):
        break
    if event == 'Go':
        window['-OUT-'].update(values['-IN-'])
window.close()
```

Coding Tips

- Stay ***simple*** at every opportunity
- Read or search the documentation (http://www.PySimpleGUI.org)
- Use the coding conventions
- Write compact layouts
- Use the PEP8 bindings
- Use "user defined elements" when you find yourself repeating parameters many times (functions that return elements)
- Use PySimpleGUI constructs rather than other GUI frameworks' constructs
- Use reasonable timeout values (as large of non-zero values as possible... be a good CPU citizen)
- Do not try to make any PySimpleGUI call from a thread
- Close your windows before exiting
- Make linear event loops
- Use the `values` dictionary rather than `Element.get()` methods
- Look through Demo Programs for more tips / techniques (http://Demos.PySimpleGUI.org)

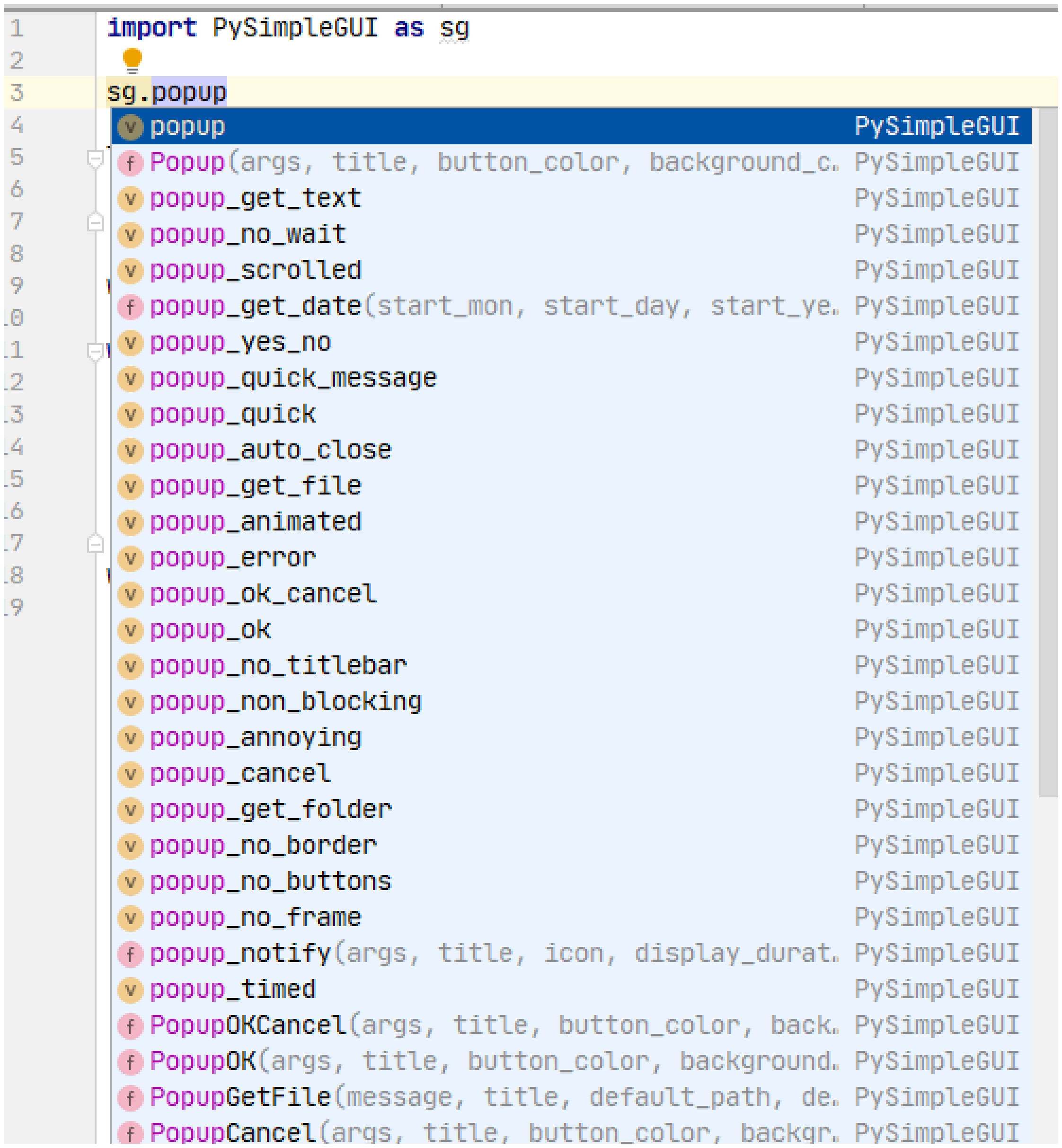
Popups

The high-level input/output mechanism.
In a GUI they can replace `input` and `print` (there often is no console)
There are a lot of them, and a ***lot*** of options.
If you're thinking of filing an enhancement request for popup:

- Don't
- Congratulations, it's time you learn how to make your own windows

Can be duplicated in 1 line of code.
Good for adding a frontend onto a command line program

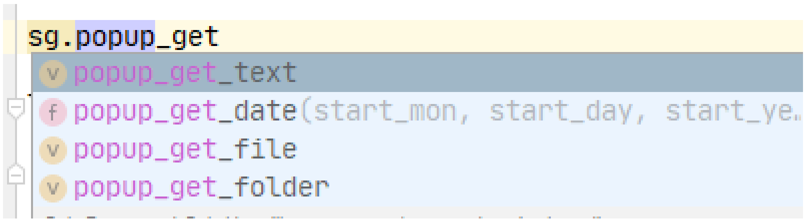
There are a lot of them



```
import PySimpleGUI as sg

sg.popup
  popup                                                  PySimpleGUI
  Popup(args, title, button_color, background_c…         PySimpleGUI
  popup_get_text                                         PySimpleGUI
  popup_no_wait                                          PySimpleGUI
  popup_scrolled                                         PySimpleGUI
  popup_get_date(start_mon, start_day, start_ye…         PySimpleGUI
  popup_yes_no                                           PySimpleGUI
  popup_quick_message                                    PySimpleGUI
  popup_quick                                            PySimpleGUI
  popup_auto_close                                       PySimpleGUI
  popup_get_file                                         PySimpleGUI
  popup_animated                                         PySimpleGUI
  popup_error                                            PySimpleGUI
  popup_ok_cancel                                        PySimpleGUI
  popup_ok                                               PySimpleGUI
  popup_no_titlebar                                      PySimpleGUI
  popup_non_blocking                                     PySimpleGUI
  popup_annoying                                         PySimpleGUI
  popup_cancel                                           PySimpleGUI
  popup_get_folder                                       PySimpleGUI
  popup_no_border                                        PySimpleGUI
  popup_no_buttons                                       PySimpleGUI
  popup_no_frame                                         PySimpleGUI
  popup_notify(args, title, icon, display_durat…         PySimpleGUI
  popup_timed                                            PySimpleGUI
  PopupOKCancel(args, title, button_color, back…         PySimpleGUI
  PopupOK(args, title, button_color, background…         PySimpleGUI
  PopupGetFile(message, title, default_path, de…         PySimpleGUI
  PopupCancel(args, title, button_color, backgr…         PySimpleGUI
```

Input popups



```
sg.popup_get
  popup_get_text
  popup_get_date(start_mon, start_day, start_ye…
  popup_get_file
  popup_get_folder
```

Particularly good to use on frontends that need a filename as a parameter.

One Line Progress Meter

Like popup, another single-line high level call.

Provides an window filled with progress stats by adding a single line to your loop.

Unlike tqdm and others, no “setup”, no changes to your loop itself, just add a line of code.

Also provides the ability to cancel the operation.

Design Patterns

“Design Patterns” provide a skeleton for you to build upon. They are not only for basic PySimpleGUI windows, but also specific topics like multi-threading, integrating with packages like OpenCV.

Two Basic PySimpleGUI Design Patterns

In main documentation, Cookbook,

1. “One Shot” – A window that is read one time and then closed
2. “Persistent” – A window that hangs around for a while. A user interacts with it on an ongoing basis

“One Shot” programs lack “Event Loops” and ***can be 1 line of code***.

PySimpleGUI Program Anatomy

5 sections of ***every*** PySimpleGUI program

1. Import
2. Layout
3. Window
4. Event Loop (optional)
5. Close

```python
# 1 - The import
import PySimpleGUI as sg

# 2 - Layout definition
layout = [[sg.Text('My layout')],
          [sg.Input(key='-INPUT-')],
          [sg.Button('OK'), sg.Button('Cancel')] ]

# 3 - Create window
window = sg.Window('Design Pattern 3 - Persistent Window', layout)

# 4 - Event Loop
while True:
    event, values = window.read()
    if event in (None, 'Cancel'):
        break

# 5 - Close window
window.close()
```

1 – Import

```
# 1 - The import
import PySimpleGUI as sg
import PySimpleGUIQt as sg
import PySimpleGUIWx as sg
import PySimpleGUIWeb as sg
```

- There are currently 4 ports of PySimpleGUI, each at a different level of completeness
  - PySimpleGUI – tkinter port. Most completed. Gets the most attention and new features first
  - PySimpleGUIQt – PySide2 port. Almost but not quite to feature complete. Supports System Tray
  - PySimpleGUIWeb – Remi port. Engineering release.
  - PySimpleGUIWx – WxPython port. Engineering release. Supports System Tray
- Moving between ports ***can be*** as simple as changing the import. Sometimes that's all that's required.
- Changes needed depends on features used.
  - Some features like right click menu aren't supported across all ports
  - Sizes are measured differently – crude conversion formula to go from (char, rows) to (pixels, pixels)
- Never, ever:
  - ~~`from PySimpleGUI import *`~~
  - Triggers an auto-delete of the .py file attempting the import (it's a joke… it doesn't delete your file…. Just don't do it please)

2- Layout

The “Layout” defines the look of your window

Layouts are lists of lists of Elements (as in “Atoms”). Piece them together into a molecule. Add control logic and you’ve got an organism, a standalone functioning creation.

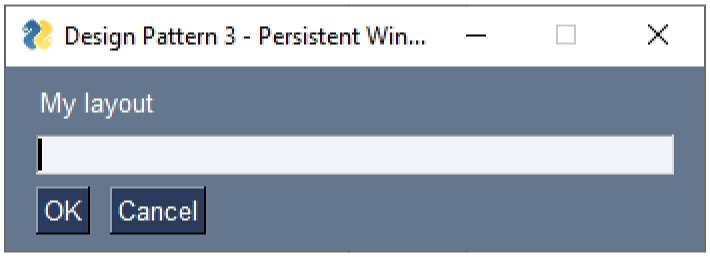
Each list in the layout represents a “row” in your window

Each row is a list of Elements

- “Elements” are PySimpleGUI’s “Widgets”.
- “Widget” refers to the underlying GUI’s implementation
- A Text Element is implemented using tkinter’s Label Widget
- A Spin Element is tkinter’s Spinbox, WxPython’s SpinButton, Qt’s QSpinBox, Remi’s SpinBox

```
# 2 - Layout definition
layout = [[sg.Text('My layout')],
          [sg.Input(key='-INPUT-')],
          [sg.Button('OK'), sg.Button('Cancel')] ]
```

Creates this window



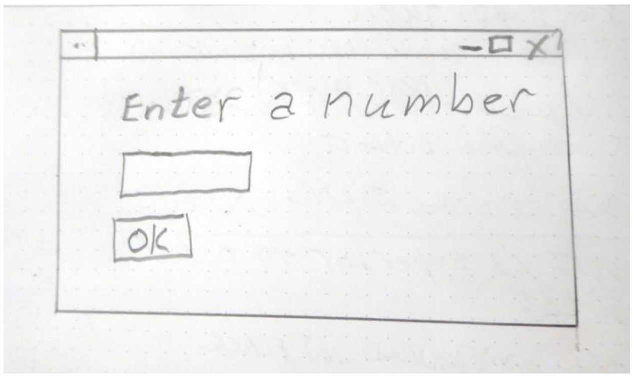
Layout Designer

Layouts are designed to be visual representations of your window, reducing the need for a “Designer”.

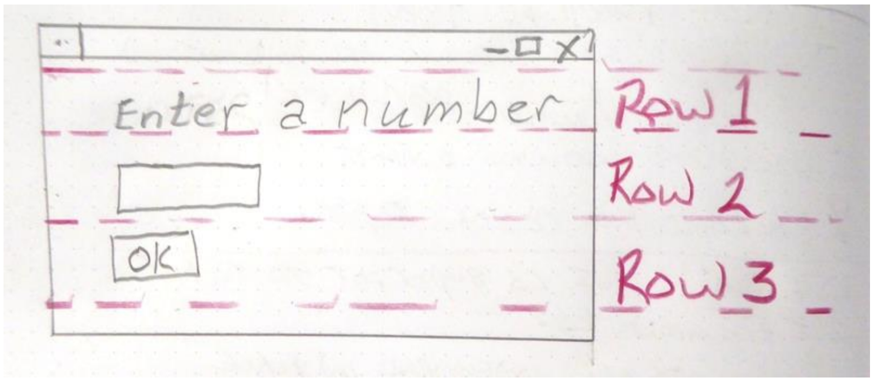
The “built-in” designer:

1. Launch the designer

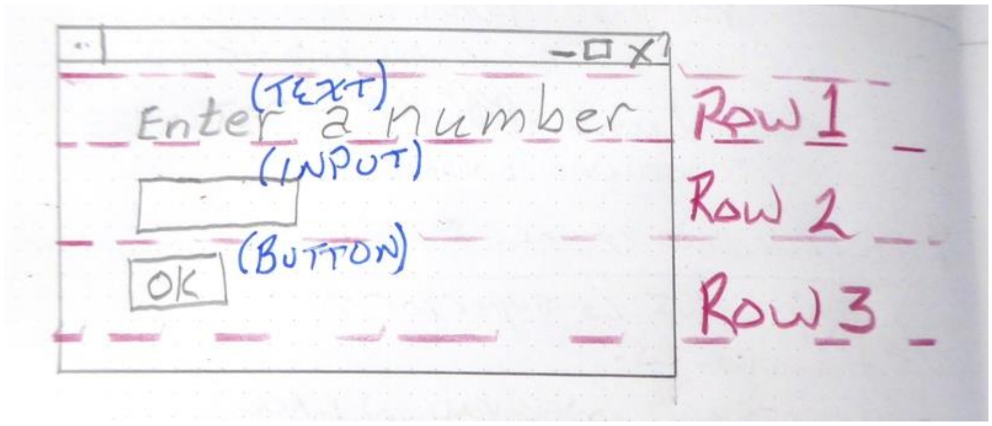
2. Sketch your window



3. Divide into “rows” and label the Elements


Enter a number
OK
Row 1
Row 2
Row 3

4. Label the elements



5. Write the layout code

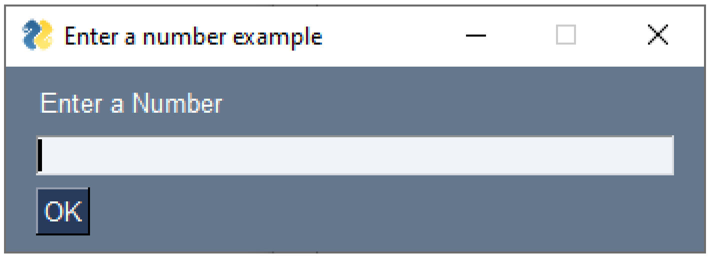
```
layout = [[sg.Text('Enter a Number')],
          [sg.Input()],
          [sg.OK()] ]
```

6. Add layout to your program and execute

```
import PySimpleGUI as sg

layout = [[sg.Text('Enter a Number')],
          [sg.Input()],
          [sg.OK()] ]

window = sg.Window('Enter a number example', layout)
event, values = window.read()
window.close()
```



Elements

- Text
- Single Line Input
- Buttons (multiple “types”)
- ButtonMenu
- Checkboxes
- Radio Buttons
- Listbox
- Slider
- Multi-line Text Input/Output
- Multi-line Text Output (not on tkinter version)
- Scroll-able Output
- Vertical Separator
- Progress Bar
- Option Menu
- Menu
- Graph
- Image
- Table
- Tree
- StatusBar
- Stretch (Qt only)
- Sizer (tkinter only)
- Containers
  - Column
  - Frame
  - Tab, TabGroup
  - Pane

Element Options

Elements are “configured in place” using long list of parameters.

Sometimes you don’t need to name each parameter in PySimpleGUI calls.

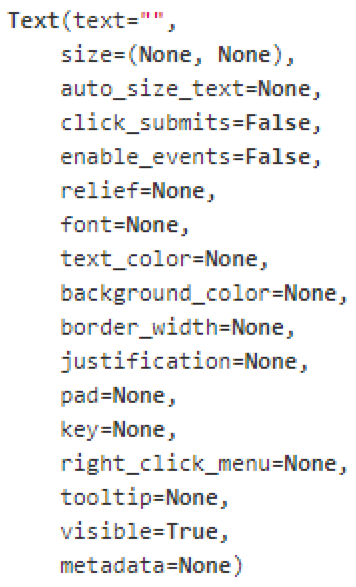
Common element parameters

- Size
- Key
- Font
- Pad
- Colors
- Enable Events
- Visibility
- Tooltip
- Metadata
- Right click menu (tkinter)

Use your IDE’s superpowers to get a list of available parameters.

Docstrings are awesome too.

Call signatures for all elements in docs http://www.PySimpleGUI.org



```
Text(text="",
    size=(None, None),
    auto_size_text=None,
    click_submits=False,
    enable_events=False,
    relief=None,
    font=None,
    text_color=None,
    background_color=None,
    border_width=None,
    justification=None,
    pad=None,
    key=None,
    right_click_menu=None,
    tooltip=None,
    visible=True,
    metadata=None)
```

3 – Window Creation

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

Like Elements, has a ***lot*** of options.

```
Window(title,
    layout=None,
    default_element_size=(45, 1),
    default_button_element_size=(None, None),
    auto_size_text=None,
    auto_size_buttons=None,
    location=(None, None),
    size=(None, None),
    element_padding=None,
    margins=(None, None),
    button_color=None,
    font=None,
    progress_bar_color=(None, None),
    background_color=None,
    border_depth=None,
    auto_close=False,
    auto_close_duration=3,
    icon=None,
    force_toplevel=False,
    alpha_channel=1,
    return_keyboard_events=False,
    use_default_focus=True,
    text_justification=None,
    no_titlebar=False,
    grab_anywhere=False,
    keep_on_top=False,
    resizable=False,
    disable_close=False,
    disable_minimize=False,
    right_click_menu=None,
    transparent_color=None,
    debugger_enabled=True,
    finalize=False,
    element_justification="left",
    ttk_theme=None,
    use_ttk_buttons=None,
    metadata=None)
```

Some parameters are used to configure elements (a window-level setting)

“Reading” Windows

`event,values = window.read()`

This one line of code summarizes how PySimpleGUI differs from the other GUI Frameworks.

Calling `Window.read()` displays the window on the screen if it’s not been displayed. “Finalizing” a window also displays it.

The call blocks / pends until an event happens.

As can be seen, 2 items are returned

1. The event that caused the read to return
2. The “values dictionary”

event

The most common event is a button click.

If events are enabled for an element, that element can generate an event.

An element’s “key” is returned as the event.

values

Dictionary containing values of all elements in the window.

You do not need to call “get” methods to get the latest values. They are all in the values variable.

If a simple window and no keys are specified, then will be a List.

“One Shot” Windows Have No Event Loop

“Get some information and move on”.

These windows you read one time and then close.

They can be a single line of code by using the read() parameter close=True.

Useful as frontends for converting a command line tool to GUI based.

4 - Event Loop

Used for "persistent windows".

In other frameworks the event loop is part of the framework.

In PySimpleGUI your code is the event loop.

A basic event loop:

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

Always check for None so that users can exit the program. Otherwise program will continue to run in the background forever or will crash after window is closed with "X"

Inside your event loop:

- Check for events and handle
- Output to window by "updating" elements
- Perform any other I/O or operations you need to do

Must call `window.read` or `window.refresh` frequently or Windows will complain that your application had become "unresponsive"

Lengthy operations need to be broken out into threads, but you cannot call PySimpleGUI from a thread. There are numerous demos for how to do this.

Can be a simple series of "if statements" checking for events. If the number of events being handled is large, then a simple dispatcher can be used.

```
if event == 'Go':
    start_motor()
elif event == 'Stop':
    stop_motor()
else:
    print('Unexpected event')
```

5 – Close Window

```
window.close()
```

When done with a window, close it

Exiting your program will close windows for ***some ports***

If you don’t close a PySimpleGUIWeb window, your program will not exit

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.)

Shortcuts & Code Compression

Elements have one of more shortcuts

- `Text, Txt, T`
- `InputText, Input, In, I`

`Window.FindElement` replaced with `Window[]`

Two little-known / rarely used shortcuts

- "Calling" a window object same as `Window.read`
- "Calling" an element object same as `Element.update`

Compressing code is often a multi-pass activity

"User Defined Elements"

- Functions that return Elements
- Use to avoid copying and pasting long element definitions
- Can create entire layouts also

Button Targets

The “chooser” Buttons have targets that are filled in with your choice.

Chooser buttons include:

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

The default is the element to the Button’s left.

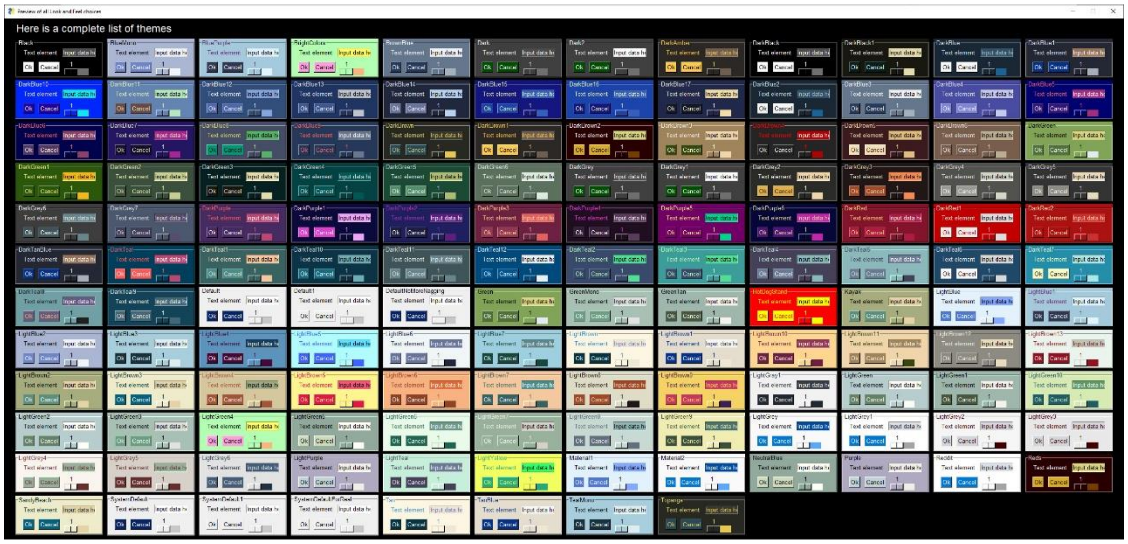
Window Beautification

Tkinter windows are not “ugly” by default.

“Beautiful” windows don’t just happen. There is a reason companies have graphic designers.

But, themes can give you a running start at a nice looking GUI.

There are over 140 “themes” available….. use them!



If you don’t like the buttons, make your own.

Experiment with no titlebar, alpha channel, padding, themes, custom graphics, etc.

Theme names can be guessed until you find one you like (e.g. “Dark Green 3”)

Rainmeter style widgets – no titlebar + grab anywhere + alpha channel

Embed your graphics in your source code

- Base64 graphics
- Demo programs available to make it easy (Demo_Base64_...)

Container Elements

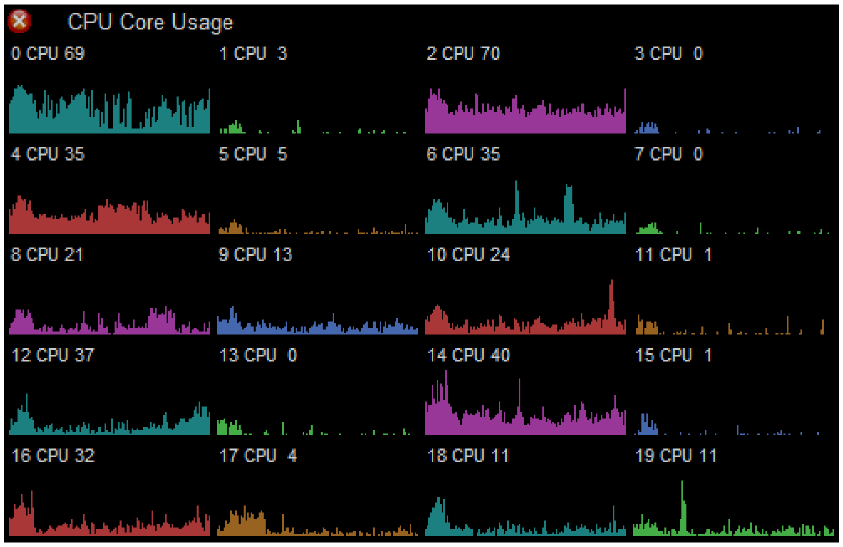
These elements hold other elements, usually in the format of a layout (a list of lists).

Container elements include:

- Column
- Frame
- Tab, TabGroup
- Pane

Use Columns:

- When horizontal alignment is needed
- To place multiple elements beside a large single element
- As a placeholder for "Invisible" elements
- To align elements center and right
- When you need multiple groups of element (e.g. many graphs)



Usually takes same list of lists as a "layout"

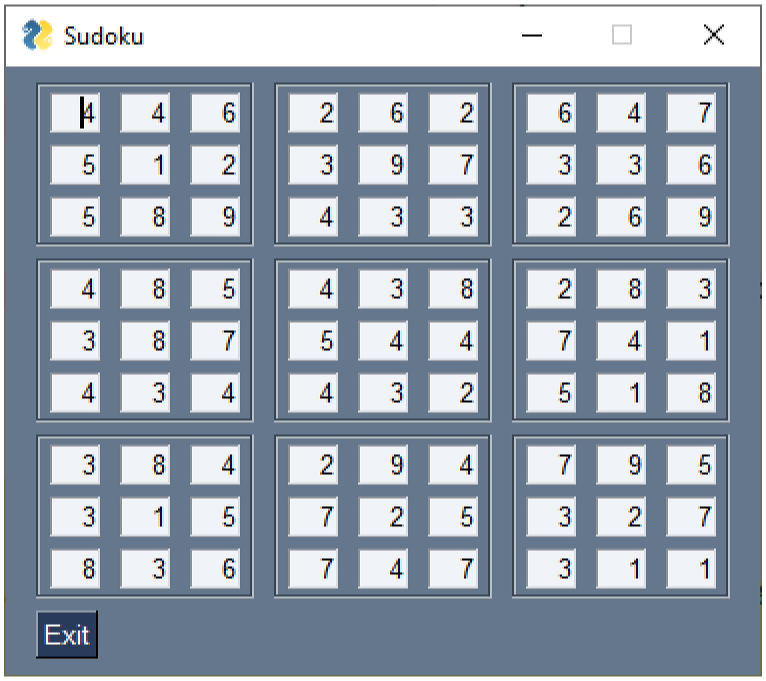
Building Layouts

Because Windows are defined using lists, it’s easy to make windows programmatically.

Use list comprehensions to create Layouts.

This is a “Single-Line-Sudoku”

```
sg.Window('Sudoku',[[sg.Frame('',[[sg.I(random.randint(1,9), justification='r',
size=(3,1),key=(frow*3+row,fcol*3+col)) for col in range(3)] for row in range(3)]) for
fcol in range(3)] for frow in range(3)]+ [[sg.B('Exit')]]).read()
```



```
4 4 6   2 6 2   6 4 7
5 1 2   3 9 7   3 3 6
5 8 9   4 3 3   2 6 9

4 8 5   4 3 8   2 8 3
3 8 7   5 4 4   7 4 1
4 3 4   4 3 2   5 1 8

3 8 4   2 9 4   7 9 5
3 1 5   7 2 5   3 2 7
8 3 6   7 4 7   3 1 1
```

“Add” together lists to create a layout

Async Windows

If you need to do something periodically, then run your window with a timeout.

Example uses:

- Stopwatch / timer
- Scraping web pages
- Polling hardware
- Servicing queues
- Multi-threaded
- Multi-window applications (Round-robin scheduling)
- Animations
- Built-in debugger

To run a window asynchronously, add a timeout to the `window.read()` call

Be a good corporate citizen

- ***Keep timeout values reasonable***
- Low timeout values will consume your CPU's core
- 100 ms is reasonable. 5 ms isn't.
- 0 can be used, but please make sure it's required

System Tray Icon

Run your program in the background and show as icon in the system tray

A System Tray Icon feature is available for Wx and Qt ports

For tkinter port the same SystemTray APIs exist, but instead of system tray, it creates a “super-icon” on the desktop.

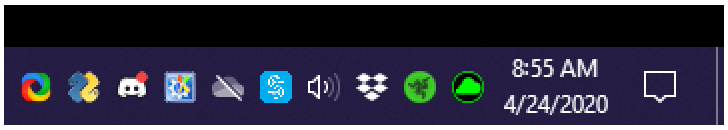
Uses same “read” call and architecture that windows have.

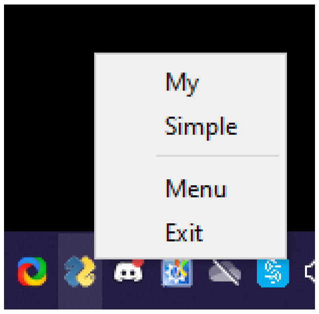
Features include:

- 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)

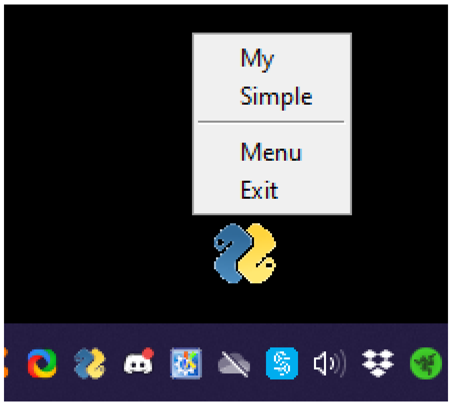
System Tray (cont)

Qt & Wx – Lives in the tray





Tk – Lives on the “desktop” (actually a floating window)



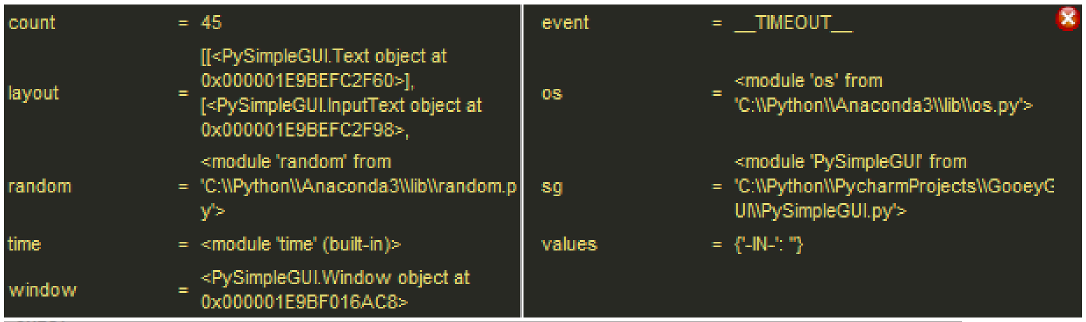
Debugger

PySimpleGUI (tkinter port) has the “Live REPL” (ImWatchingYou package) built-in.

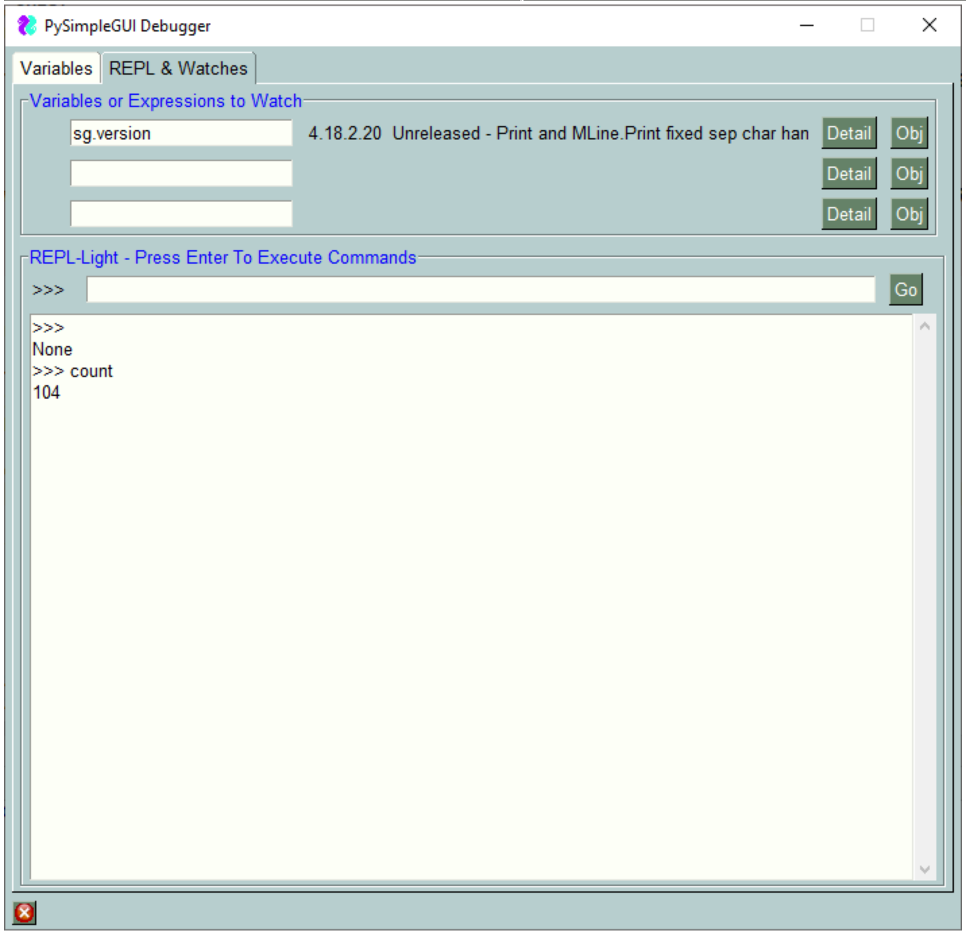
Can make explicit calls to show the windows

```
sg.show_debugger_window()
sg.show_debugger_popout_window()
```

Or use the Break / Control+Break keys   (pop-out, main window)



```
count    = 45                                     event   = __TIMEOUT__
layout   = [[<PySimpleGUI.Text object at          os      = <module 'os' from
           0x000001E9BEFC2F60>],                            'C:\\Python\\Anaconda3\\lib\\os.py'>
           [<PySimpleGUI.InputText object at
           0x000001E9BEFC2F98>,
random   = <module 'random' from                  sg      = <module 'PySimpleGUI' from
           'C:\\Python\\Anaconda3\\lib\\random.p             'C:\\Python\\PycharmProjects\\GooeyG
           y'>                                              UI\\PySimpleGUI.py'>
time     = <module 'time' (built-in)>             values  = {'-IN-': ''}
window   = <PySimpleGUI.Window object at
           0x000001E9BF016AC8>
```



PySimpleGUI Debugger
Variables
REPL & Watches
Variables or Expressions to Watch
Detail
Obj
REPL-Light - Press Enter To Execute Commands
Go

```
sg.version    4.18.2.20  Unreleased - Print and MLine.Print fixed sep char han
```

```
>>>
None
>>> count
104
```

Extending

It’s possible to “extend” PySimpleGUI’s capabilities by directly accessing the underlying GUI framework’s widget.

The member variable `Element.Widget` contains the widget that is used to implement the element.

If you want to access the widget for element with key `‘-KEY-’`:

```
Window[‘-KEY-‘].Widget
```

With tkinter this is usually done to access a configuration setting that’s not been exposed through PySimpleGUI.

```
Window[‘-KEY-‘].Widget.config(setting=1234)
```

To convert a normal progress bar into an indeterminate one:

```
window['bar'].Widget.config(mode='indeterminate')
```

Support

Only provided on GitHub

Don’t be afraid to ask questions.

Don’t suffer silently.

Fill out the Issue form

“Help me help you”

Today’s support cost is the same pricing as the introduction pricing of $0

It’s easy to find missing features. Generally speaking logging them isn’t helpful. Same with “it would be better if it was architected this way”

Enhancement Algorithm & Hurdles

- Is it impossible to do using existing APIs
- Is it an “80% feature”
- Does it fit the PySimpleGUI vision
- Some decisions may not feel right to you
- This is not a typical Open Source project
- There is a vision and a priority order for working on fixes / enhancements. It’s not possible to fully explain the process on paper

Please don’t submit code

- No pull requests accepted
- If you notice a 5 or 10 line fix, then OK to put in the Issue you file

Thank you!

A warm “Thank You!” to the users and supports of PySimpleGUI over the past 2 years. Your support is greatly appreciated. You’ve helped make PySimpleGUI what it is today. It couldn’t have come this far without your help and support.