

PySimpleGUI入門

基本的な使用方法

第0.5.1版

Copyright © 2020-2022, Katsunori Nakamura

中村勝則

2022年6月10日

免責事項

本書の内容は参考資料であり、掲載したプログラムリストは全て試作品である。本書の使用に伴って発生した不利益、損害の一切の責任を筆者は負わない。

目次

1	はじめに	1
2	導入	1
2.1	使用方法の概略	1
2.1.1	ウィジェットの配置	1
2.1.2	イベントループ	2
2.1.3	GUIプログラムの終了	2
2.1.4	サンプルに沿った説明	2
2.1.4.1	ウィジェットのサイズ設定	3
2.1.4.2	デザインテーマの設定	3
2.1.4.3	ウィジェットのイベントハンドリングの有効化	4
2.1.4.4	ウィジェットへの名前の付与	4
2.1.4.5	既存のウィジェットへのアクセス	5
3	ウィジェット (Widgets)	6
3.1	文字列, 画像の表示	6
3.1.1	Text	6
3.1.1.1	使用できるフォント	6
3.1.1.2	余白と枠	7
3.1.2	Image	8
3.2	文字の入力	9
3.2.1	InputText	9
3.2.1.1	イベント, 値, ウィジェットの対応付け	10
3.2.1.2	InputText へのイベントハンドリングの登録	10
3.2.1.3	パスワード入力フィールド	11
3.2.1.4	InputText への値の設定	11
3.2.2	Multiline	12
3.3	ボタン類	13
3.3.1	Button	13
3.3.2	Checkbox	15
3.3.3	Radio	15
3.4	選択入力	16
3.4.1	Listbox	16
3.4.2	Spin	17
3.4.3	Slider	18
3.4.3.1	Slider への値の設定	18
3.5	枠, 区切り線	19
3.5.1	Frame	19
3.5.2	VerticalSeparator	21
3.6	進捗バー	21
3.7	表, ツリー	22
3.7.1	Table	22
3.7.2	Tree	23
3.8	様々な表示構造	26
3.8.1	Tab, TabGroup	26
3.8.2	Pane, Column	27

3.8.2.1	Column を応用したウィジェットの表示位置の調整	28
3.9	メニュー	31
3.9.1	OptionMenu	31
3.9.2	MenuBar	31
3.9.3	ButtonMenu	32
3.10	Canvas	33
4	ポップアップウィンドウ	35
4.1	入力	35
4.1.1	PopupGetText	35
4.2	報告, 確認のためのポップアップ表示	36
5	ウィンドウの扱い	38
5.1	リサイズ可能にする設定	38
5.2	ウィンドウのサイズと位置について	38
5.3	テーマの設定	40
5.3.1	デザインテーマの一覧表示	40
5.3.2	使用できるデザインテーマの調査	40
5.4	タイトルバーの有無	41
5.4.1	ウィンドウの透明度	41
A	付録	42
A.1	matplotlib で作成したグラフをレイアウトに埋め込む方法	42
A.1.1	matplotlib によって作成されるグラフ	42
A.1.2	Figure オブジェクトのグラフを Canvas に変換する方法	43
A.2	Tkinter のオブジェクトとの対応	45

1 はじめに

PySimpleGUI は Python 言語で GUI (Graphical User Interface) を構築するためのツールキット (ライブラリ) であり, The PySimpleGUI Organization が開発し管理している. このライブラリは LGPL 3 のライセンスで公開されており, 公式インターネットサイト <https://pysimplegui.readthedocs.io/> からソフトウェア本体と関連の情報を入手することができる.

PySimpleGUI を用いると, 他の GUI ツールキットに比べ, より少ないコードの記述によって GUI を実現することができる. またこのライブラリには, 多くのデザインテーマが予め提供されており, 多様なルック・アンド・フィール (look and feel) を GUI に与えることができる.

2 導入

PySimpleGUI は wheel 形式のライブラリ¹ として配布されており, 利用する計算機環境において pip コマンドなどでインストールすることができる.

例. PySimpleGUI のインストール (コマンドライン)

```
pip install pysimplegui
```

2.1 使用方法の概略

Python 処理系で PySimpleGUI を利用するには, 次のようにして処理系にライブラリを読み込む.

```
import PySimpleGUI
```

このようにすることで, ライブラリが提供する API を PySimpleGUI の接頭辞を付けて呼び出すことができるが, 実際には

```
import PySimpleGUI as sg
```

として別名を与え, 短い接頭辞 `sg` で API を呼び出すことが標準的であり, 本書でもそのような形での利用を前提とする.

PySimpleGUI では GUI の構成要素を **ウィジェット** と呼び², これらのインスタンスを必要に応じて生成し組み合わせることで GUI を構築する. また, 他のツールキットと同様に, PySimpleGUI でもイベント駆動型のプログラミングのスタイルで GUI アプリケーションを構築する.

2.1.1 ウィジェットの配置

GUI のウィンドウは Window オブジェクトであり, これを生成するとウィンドウが表示される. また使用する個々のウィジェットは Window オブジェクトに登録する. このとき, Window オブジェクトに登録するウィジェット群はリストの形 (下記参照) で用意する.

■ ウィンドウに登録するウィジェットのリスト

```
[ [1 行目のウィジェット 1, 1 行目のウィジェット 2, ...],  
  [2 行目のウィジェット 1, 2 行目のウィジェット 2, ...],  
  ... ]
```

このように入れ子のリスト, すなわち, 各行に配置するウィジェットをリストにして更にそれらを要素とするリストの形でウィジェット群を用意する.

■ Window オブジェクトの生成

```
Window( タイトル, ウィジェットのリスト )
```

これを実行することで GUI のウィンドウである Window オブジェクトが生成される.

¹Python のライブラリを配布する際に多く用いられる形式.

²ウィジェット: PySimpleGUI 以外の GUI ツールキットにおいてはコンポーネントあるいはコントロールと呼ばれることがある.

2.1.2 イベントループ

イベント駆動型プログラミングは次のような流れで実現される。

1. イベント発生のを待機

GUIからイベントが発生するのを待ち続ける。イベントが発生すれば次に進む。

2. イベントに応じた処理（コールバック）の実行

イベント発生源のウィジェットや、発生したイベントの種類から、行うべき処理（イベントハンドラ）を選択して実行する。処理が終われば1に戻る。

発生したイベントは、Window オブジェクトに対して read メソッドを実行することで受信する。read メソッドの戻り値は

(イベント名, ウィジェットからの戻り値)

の形のタプルである。実際にはこのような戻り値から実行すべき処理を判断して起動する。「ウィジェットからの戻り値」は辞書オブジェクトであり、実装されたウィジェット群の値が辞書オブジェクトの形で得られる。read メソッドの戻り値の扱いに関しては後の「3.2.1.1 イベント, 値, ウィジェットの対応付け」(p.10) で例を挙げて説明する。

ウィンドウのクローズボタンがクリックされるとそれがイベントとなり、イベント名として None が得られる。

2.1.3 GUI プログラムの終了

GUI の処理を終了するには Window オブジェクトに対して close メソッドを実行する。

2.1.4 サンプルに沿った説明

ここでは、図 1 に示すような極めて単純な動作をする GUI について考える。

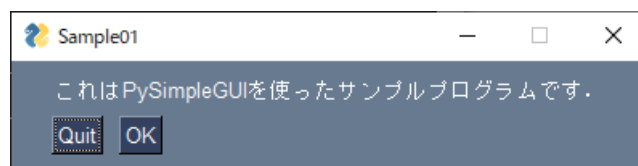


図 1: 「Quit」をクリックすると終了するプログラム

この GUI は文字列「これは PySimpleGUI を使ったサンプルプログラムです。」を表示し、その下に 2 つのボタン「Quit」「OK」を備え、「Quit」もしくはクローズボタンをクリックすると終了する。これを実装したものが次に示す psgui01.py である。

プログラム：psgui01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 layout = [[sg.Text('これはPySimpleGUIを使ったサンプルプログラムです。')],
5           [sg.Button('Quit'),sg.Button('OK')]]
6 # ウィンドウ作成
7 window = sg.Window('Sample01', layout)
8 # イベントループ
9 while True:
10     event, values = window.read() # イベントの読み取り（イベント待ち）
11     print('イベント:',event,', 値:',values) # 確認表示
12     if event in (None,'Quit'):      # 終了条件（None:クローズボタン）
13         print('終了します。')
14         break
15 # 終了処理
16 window.close()
```

解説：4～5行目でウィジェットのリストを作成して変数 `layout` に与えている。それを用いて、7行目で Window オブジェクトを生成し、変数 `window` に与えている。このプログラムでは、文字列のウィジェット `Text` と2つのボタンウィジェット `Button` をウィンドウに配置する。

9～14行目の `while` ループがイベントハンドリングのためのループであり、イベントの読み取り（10行目）とそれに基づく処理（12～14行目）を行う。

ボタンオブジェクト (`Button`) はクリックイベントを受け付け、その名前をイベント名とする。 `read` メソッドによりイベントを受信する際、変数 `event` にそのイベント名が与えられる。(変数 `values` の扱いについては後に説明する)

このプログラムは、「Quit」ボタンあるいはウィンドウのクローズボタンのクリックを受けて終了するだけの単純なものである。

2.1.4.1 ウィジェットのサイズ設定

ウィジェットを生成するメソッドにキーワード引数 `'size=(幅, 高さ)'` を与えることでそのサイズを設定することができる。「幅」と「高さ」の単位は1文字の幅と高さ³である。先のプログラム `psgui01.py` において、4～5行目を

```
layout = [[sg.Text('これは PySimpleGUI を使ったサンプルプログラムです。')],
          [sg.Button('Quit', size=(15,1)), sg.Button('OK', size=(15,1))]]
```

と変更し、ボタンのサイズを設定して実行した例を図2に示す。

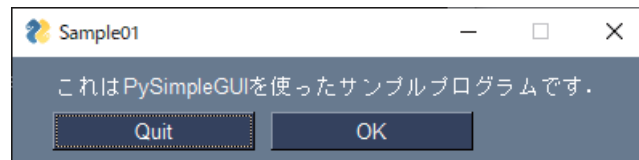


図 2: ボタンのサイズを 15 × 1 に設定

図1と比較してボタンのサイズが変わっていることがわかる。

2.1.4.2 デザインテーマの設定

PySimpleGUI では GUI のデザインテーマを変更することができる。例えば、ライブラリに標準で添付されているテーマ `'Dark Brown'` を GUI に適用するには、 `theme` メソッドを用いて、Window オブジェクト作成に先立って

```
sg.theme('Dark Brown')
```

と実行する。先のプログラム `psgui01.py` において、ライブラリ読み込みの直後にこの記述を挿入（3行目に挿入）して実行すると図3ようになる。

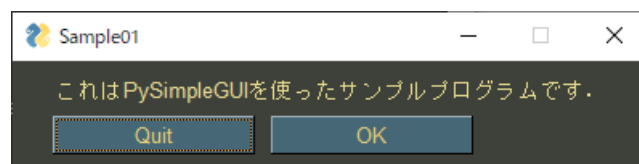


図 3: テーマを 'Dark Brown' に設定した例

この他にも多くのテーマが用意されている。ウィンドウのデザインテーマに関しては「5.3 テーマの設定」(p.40)で解説する。

³対象のウィジェットによってはサイズをピクセル単位で設定する場合もある。

2.1.4.3 ウィジェットのイベントハンドリングの有効化

ウィジェットにはイベントを受け付けるものと受け付けないものがある。例えば先の例 `psgui01.py` では `Text` ウィジェットと `Button` ウィジェットの使用を例示したが、`Text` ウィジェットの方はクリックなどのイベントを受け付けない。`psgui01.py` はイベント受信の様子を標準出力に出力するのでそれを確認することができる。

例. `psgui01.py` 実行時の標準出力（ターミナルウィンドウ）の表示

```
イベント: OK , 値: {}          ← 「OK」 ボタンのクリック
イベント: Quit , 値: {}       ← 「Quit」 ボタンのクリック
終了します.                  ← 終了メッセージ
```

`Text` オブジェクト「これは `PySimpleGUI` を使ったサンプルプログラムです。」をクリックしても何も出力されないことが確認できる。このようなウィジェットにおいては、作成時のメソッドにキーワード引数 `'enable_events=True'` を与えることで、イベントの受信が可能となる。例えばプログラム `psgui01.py` において、ウィジェットのリストを記述する部分を下記のように変更すると、`Text` オブジェクトに対するクリックが有効になる。

```
layout=[[sg.Text('これはPySimpleGUIを使ったサンプルプログラムです。',enable_events=True)],
         [sg.Button('Quit',size=(15,1)),sg.Button('OK',size=(15,1))]]
```

このような変更を施した後で実行した例を示す。

例. クリックを受け付ける `Text` オブジェクト

```
イベント: これはPySimpleGUIを使ったサンプルプログラムです。 , 値: {} ← 文字列部分のクリック
イベント: None , 値: None      ← クローズボタンのクリック
終了します.                  ← 終了メッセージ
```

`Text` オブジェクトをクリックすると、そのテキストの内容がイベント名となる。

2.1.4.4 ウィジェットへの名前の付与

ウィジェットの作成時にキーワード引数 `'key=名前'` で名前を与えることができる。また、このような形で名前を与えることで、それがイベントハンドリングにおける当該ウィジェットのイベント名となる。このことを次のサンプルプログラム `psgui01-renew.py` で示す。

プログラム： `psgui01-renew.py`

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 sg.theme('Dark Brown')       # テーマの設定
4 # レイアウト
5 layout = [[sg.Text('これはPySimpleGUIを使ったサンプルプログラムです。',
6                   enable_events=True, key='tx1')],
7           [sg.Button('Quit',size=(15,1),key='btn1'),
8            sg.Button('OK',size=(15,1),key='btn2')]]
9 # ウィンドウ作成
10 window = sg.Window('Sample01 (改)', layout)
11 # イベントループ
12 while True:
13     event, values = window.read() # イベントの読み取り（イベント待ち）
14     print('イベント:',event,', 値:',values) # 確認表示
15     if event in (None,'btn1'):      # 終了条件（None:クローズボタン）
16         print('終了します。')
17         break
18 # 終了処理
19 window.close()
```

このプログラムでは、`Text` ウィジェットに `'tx1'`、「Quit」ボタンに `'btn1'`、「OK」ボタンに `'btn2'` という名前を与えている。これを実行すると図3のようなウィンドウが表示され、各ウィジェットをクリックすると標準出力に次のように出力される。

例. 各ウィジェットをクリック

イベント: tx1 , 値: {}	← Text ウィジェットをクリック
イベント: btn2 , 値: {}	← 「OK」 ボタンをクリック
イベント: btn1 , 値: {}	← 「Quit」 ボタンをクリック

終了します.

重要) キーワード引数 'key=' を与えない場合は, ウィジェットには整数値の名前が自動的に与えられる.

■ ウィジェットから得られる値

先に示した例では, ウィジェットから得られる値は '{}' となっているが, 後に説明する各種の入力系ウィジェットからは辞書オブジェクトの形で各ウィジェットの値が得られる. これに関しては後の「3.2.1.1 イベント, 値, ウィジェットの対応付け」(p.10) で, 例を挙げて説明する.

2.1.4.5 既存のウィジェットへのアクセス

ウィンドウに実装された既存のウィジェットにアクセスするには, ウィンドウオブジェクトの後ろに '[ウィジェット名]' を付ける.

入力系のウィジェットに値を設定するには, 当該ウィジェットに対して Update メソッドを実行する. これに関しては後の「3.2.1.4 InputText への値の設定」(p.11) で具体的に例を挙げて説明する.

3 ウィジェット (Widgets)

PySimpleGUI で利用できるウィジェットの内、代表的なものを取り上げ、特に重要な機能に関して説明する。

3.1 文字列, 画像の表示

3.1.1 Text

Text オブジェクトは文字列を配置する際に使用する。

```
Text( 文字列, font=( フォント名, サイズ ), text_color=色, background_color=色,  
      pad=((左余白, 右余白),(上余白, 下余白)), border_width=太さ, relief=枠スタイル )
```

font に与える**フォント名**は文字列で、**サイズ**はポイント値を整数で与える。text_color には**文字の色**を、background_color には**背景色**を 16 進数表現 '#rrggbb' の文字列⁴ で与える。これらの値を指定して文字列を表示するサンプルプログラム psguiText01.py を示す。

プログラム：psguiText01.py

```
1 # coding: utf-8  
2 import PySimpleGUI as sg          # ライブラリの読み込み  
3 sg.theme('SystemDefault')        # テーマの設定  
4 # レイアウト  
5 layout = [[sg.Text('文字列の表示：HG行書体,24ポイント',  
6                   font=('HG行書体',24),  
7                   text_color='#ff0000', background_color='#0000ff')],  
8           [sg.Text('文字列の表示：HG正楷書体-PRO,24ポイント',  
9                   font=('HG正楷書体-PRO',24),  
10                  text_color='#00ff00', background_color='#ff00ff')],  
11          [sg.Text('文字列の表示：小塚ゴシック Pro B,24ポイント',  
12                  font=('小塚ゴシック Pro B',24),  
13                  text_color='#0000ff', background_color='#ffff00')]]  
14 # ウィンドウ作成  
15 window = sg.Window('psguiText01.py', layout)  
16 # イベントループ  
17 while True:  
18     event, values = window.read() # イベントの読み取り (イベント待ち)  
19     if event == None:             # 終了条件 (None:クローズボタン)  
20         break  
21 # 終了処理  
22 window.close()
```

このプログラムを実行した例を図に示す。



図 4: psguiText01.py の実行例

3.1.1.1 使用できるフォント

使用できるフォント名としては tkinter で使用するものが指定できるが、それらを調べるためのプログラムの例を tk_font01.py に示す。

⁴'rr' の部分には赤, 'gg' の部分には緑, 'bb' の部分には青の値を 16 進数で記述する。

プログラム：tk_font01.py

```
1 # coding: utf-8
2 # モジュールの読み込み
3 import tkinter # Tkinter
4 import tkinter.font as tkfont # フォント関連モジュール
5
6 root = tkinter.Tk() # アプリケーションのウィジェット
7
8 fntL = list(tkfont.families(root)) # フォントリストの取得
9 for f in fntL: # フォント名の出力
10     print(f)
```

これは tkinter の応用プログラムの例である。この例のように、tkinter.font モジュールの families 関数を使用するとフォント名の列が得られる。

tk_font01.py の実行例 (Windows の場合)

```
C:\Users\katsu > py tk_font01.py
System
@System
Terminal
⋮
(途中省略)
⋮
Highlight LET
Odessa LET
Scruff LET
```

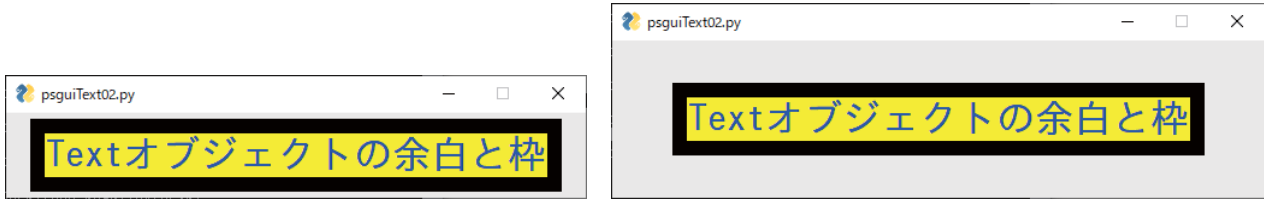
3.1.1.2 余白と枠

文字列全体に枠を付けることができ、枠の太さを border_width で指定する。また、relief に与える値で枠にスタイルを設定することができる。余白と枠の設定に関するサンプルプログラムを psguiText02.py に示す。

プログラム：psguiText02.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 sg.theme('SystemDefault') # テーマの設定
4 # レイアウト
5 layout = [[sg.Text('Textオブジェクトの余白と枠',
6 font=('IPAゴシック',24),
7 text_color='#0000ff',
8 relief=sg.RELIEF_SOLID,
9 #
10 # relief=sg.RELIEF_RAISED,
11 # relief=sg.RELIEF_SUNKEN,
12 # relief=sg.RELIEF_FLAT,
13 # relief=sg.RELIEF_RIDGE,
14 # relief=sg.RELIEF_GROOVE,
15 border_width=12,
16 background_color='#ffff00',
17 # pad=((30,30),(30,30))
18 # pad=((0,0),(0,0))
19 )]]
20 # ウィンドウ作成
21 window = sg.Window('psguiText02.py', layout)
22 # イベントループ
23 while True:
24     event, values = window.read() # イベントの読み取り (イベント待ち)
25     if event == None: # 終了条件 (None: クローズボタン)
26         break
27 # 終了処理
28 window.close()
```

8~17行目の各設定は、コメントを切り替えることで個々に確認することができる。pad に指定する値で左、右、上、下の余白の大きさを個別に設定できる。図5に余白の設定値の違いによる効果を示す。



(a) 余白なし : pad=((0,0),(0,0))

(b) 余白あり : pad=((30,30),(30,30))

図 5: 余白と枠の設定

relief に設定する値毎の効果を図 6 に示す.



(a) relief=sg.RELIEF_SOLID

(b) relief=sg.RELIEF_RAISED

(c) relief=sg.RELIEF_SUNKEN

(d) relief=sg.RELIEF_FLAT

(e) relief=sg.RELIEF_RIDGE

(f) relief=sg.RELIEF_GROOVE

図 6: 枠のスタイル

重要) ここで説明したキーワード引数 'font=', 'text_color=', 'background_color=', 'pad=', 'border_width=', 'relief=' は後に説明する他のウィジェットにおいても有効な場合があり, そのような場合においては同様の設定を行う.

3.1.2 Image

Image オブジェクトは画像を配置するウィジェットである.

Image(filename=ファイル名, background_color=背景色, pad=((左余白, 右余白),(上余白, 下余白)))

ファイル名で指定した画像ファイル⁵を読み込む. 透過型の画像の場合は背景色が指定できる.

画像ファイルを読み込んで表示するサンプルプログラムを に示す.

プログラム : psguiImage01.py

```

1  # coding: utf-8
2  import PySimpleGUI as sg          # ライブラリの読み込み
3  sg.theme('SystemDefault')       # テーマの設定
4  # レイアウト
5  layout = [[sg.Image(filename='trnsprnt.png',
6                background_color='#000000',
7                background_color='#ff0000',
8                background_color='#00ff00',
9                background_color='#0000ff',
10               background_color='#ffffff'
11               )]]

```

⁵本書執筆時点の PySimpleGUI の版では PNG, GIF といった, tkinter で扱える画像形式のみが使用可能である.

```

12 # ウィンドウ作成
13 window = sg.Window('psguiImage01.py', layout)
14 # イベントループ
15 while True:
16     event, values = window.read() # イベントの読み取り (イベント待ち)
17     if event == None:           # 終了条件 (None: クローズボタン)
18         break
19 # 終了処理
20 window.close()

```

プログラムの5行目で画像ファイル'trnsprnt.png'を読み込んでいる。このファイルの内容が図7のようなもの(背景は透明)である場合のプログラムの実行例を図8に示す。

背景が透明の画像

図 7: 透明の背景を持つ画像 'trnsprnt.png'



図 8: 様々な背景色の設定

プログラムの6~10行目のコメントを切り替えて背景色を確認することができる。

3.2 文字の入力

3.2.1 InputText

1行のテキスト(文字列)を入力するためのウィジェット⁶としてInputTextがある。

```

InputText( default_text=初期値, text_color=色, background_color=色,
           font=( フォント名, サイズ ), pad=((左余白, 右余白),(上余白, 下余白)) )

```

キーワード引数'default_text'に初期値を与えることができる。他の引数に関してはTextオブジェクトの場合に準じる。

InputTextオブジェクトで文字列を編集するサンプルプログラムをpsguiInputText01.pyに示す。

プログラム: psguiInputText01.py

```

1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # レイアウト
4 layout = [[sg.InputText(default_text='ここで文字列を編集',
5                         size=(35,1), key='tx1',
6                         text_color='#0000ff', background_color='#ffff00')],
7           [sg.Button('read', key='bt1')]]
8 # ウィンドウ作成
9 window = sg.Window('psguiInputText01.py', layout)

```

⁶いわゆるテキストフィールド

```

10 # イベントループ
11 while True:
12     event, values = window.read() # イベントの読み取り (イベント待ち)
13     print('イベント:', event, ', 値:', values) # 確認表示
14     if event == None: # 終了条件 (None: クローズボタン)
15         break
16 # 終了処理
17 window.close()

```

このプログラムを実行すると図9のようなウィンドウが表示される。「read」ボタンをクリックすると、テキストフィールドの内容を標準出力に出力する。

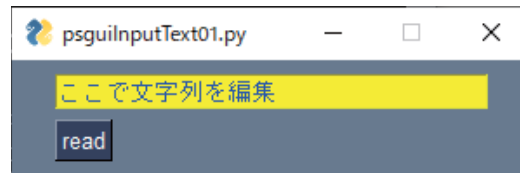


図 9: InputText (テキストフィールド)

このウィンドウにおいて「read」ボタンをクリックすると次のような出力が標準出力に得られる。

例. 「read」ボタンのクリック

イベント: bt1 , 値: {'tx1': 'ここで文字列を編集'}

3.2.1.1 イベント, 値, ウィジェットの対応付け

先のプログラム psguiInputText01.py では, InputText, Button 両オブジェクトの作成時にキーワード引数

key=名前

で一意名を与えている (5,7 行目). このような方法で名前を与えておくと, イベント処理の際のイベント名がそのウィジェットの名前となる.

■ read メソッドの戻り値

Window オブジェクトに対して read メソッドを実行することでイベントを受信するが, このメソッドの戻り値 (タプル) の先頭要素が「イベント名」であり, 次の要素が GUI 全体の値を表す辞書オブジェクトである. 先のプログラム psguiInputText01.py では, 12 行目で read メソッドを実行してイベントを受信しており, その戻り値が event, values という 2 つの変数に得られる. 後者の values は, GUI 全体の値を保持する辞書オブジェクトであり,

values[ウィジェットの名前]

とすることでウィジェットの値にアクセスすることができる. 実際の方法については後に説明する.

3.2.1.2 InputText へのイベントハンドリングの登録

InputText オブジェクト作成時にキーワード引数 'enable_events=True' を与えるとキー入力のイベントの受信が可能となる. これに関して次の例 psguiInputText02.py を用いて考える.

プログラム: psguiInputText02.py

```

1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # レイアウト
4 layout = [[sg.InputText(default_text='ここで文字列を編集', size=(35,1),
5                       enable_events=True, key='txt1',
6                       text_color='#0000ff', background_color='#ffff00')]]
7 # ウィンドウ作成
8 window = sg.Window('psguiInputText02.py', layout)
9 # イベントループ
10 while True:
11     event, values = window.read() # イベントの読み取り (イベント待ち)
12     if event == None: # 終了条件 (None: クローズボタン)
13         break
14     else:

```

```

15     print('ウィジェット:', event, ', テキスト:', values[event])    # 確認表示
16 # 終了処理
17 window.close()

```

このプログラムは InputText オブジェクトに対してイベント受信を可能とする設定（5行目）をしている。これにより、当該テキストフィールドでキー入力をする度にそれをイベントとして受信する。

このプログラムを起動して 'abc' とキー入力する様子を以下に示す。

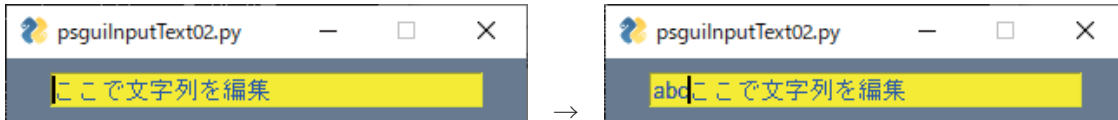


図 10: 'abc' とキー入力

テキストフィールドで 'a', 'b', 'c' とキー入力する過程で次のように標準出力に出力される。

例. キー入力に伴う表示

```

ウィジェット: txt1 , テキスト: a ここで文字列を編集    ←'a' のキーを押した
ウィジェット: txt1 , テキスト: ab ここで文字列を編集   ←'b' のキーを押した
ウィジェット: txt1 , テキスト: abc ここで文字列を編集 ←'c' のキーを押した

```

InputText オブジェクトがキー入力のイベントを受け付けていることがわかる。

このプログラムでは、イベント受信時に採取したウィジェットの値が変数 values に得られている。(11行目) values は辞書オブジェクトであり、

```
values[ウィジェット名]
```

としてアクセスすることができる。(15行目)

3.2.1.3 パスワード入力フィールド

InputText オブジェクト作成時にキーワード引数

```
password_char=記号
```

を与えることで、パスワード入力フィールドにすることができる。入力フィールドに入力すると、引数に与えた「記号」がダミーとして表示される。

3.2.1.4 InputText への値の設定

ウィンドウ内に実装された既存の InputText オブジェクトに値（文字列）を与える方法について説明する。

■ ウィジェットへのアクセス

Window オブジェクト内の特定のウィジェットにアクセスするには、Window オブジェクトの後ろに「[ウィジェット名]」を付ける。

例. Window オブジェクト window 内のウィジェット tx1 にアクセスする記述

```
window['tx1']
```

ウィジェットに値を設定するには Update メソッドを使用する。

既存のテキストフィールドに値（文字列）を設定するサンプルプログラムを psguiInputText03.py に示す。

プログラム：psguiInputText03.py

```

1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 layout = [[sg.InputText(default_text='',
5                          size=(35,1), key='tx1',
6                          text_color='#0000ff',background_color='#ffff00')],
7           [sg.Button('文1',key='bt1'),sg.Button('文2',key='bt2')],

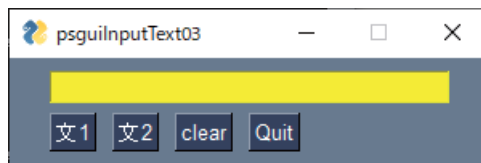
```

```

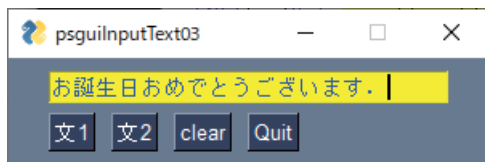
8         sg.Button('clear',key='bt3'),sg.Button('Quit',key='bt4')]]
9 # ウィンドウ作成
10 window = sg.Window('psguiInputText03', layout)
11 # イベントループ
12 while True:
13     event, values = window.read() # イベントの読み取り (イベント待ち)
14     if event in (None,'bt4'): # 終了条件 (None:クローズボタン)
15         break
16     elif event == 'bt1':
17         window['tx1'].Update('お誕生日おめでとうございます。')
18     elif event == 'bt2':
19         window['tx1'].Update('心よりお祝い申し上げます。')
20     elif event == 'bt3':
21         window['tx1'].Update('')
22 # 終了処理
23 window.close()

```

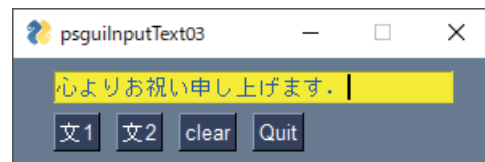
プログラムの17,19,21行目でテキストフィールド tx1 に文字列を設定している。このプログラムを実行した例を図11に示す。



(a) 起動したところ



(b) 「文1」をクリック



(c) 「文2」をクリック

図 11: テキストフィールドに値を与える例

テキストフィールドに値 (文字列) が設定できていることがわかる。

重要) ここでは InputText オブジェクトに値を設定する例を挙げたが、入力系ウィジェットには概ね同様の方法で値が設定できる。

3.2.2 Multiline

複数行に渡るテキスト編集エリアを作成するには Multiline オブジェクトを使用する。

```

Multiline( default_text=初期値, text_color=色, background_color=色, border_width=太さ,
            font=( フォント名, サイズ ), pad=((左余白, 右余白),(上余白, 下余白)) )

```

キーワード引数 'default_text' に初期値を与えることができる。他の引数に関しては Text オブジェクトの場合に準じる。

Multiline オブジェクトで文字列を編集するサンプルプログラムを psguiMultiline01.py に示す。

プログラム：psguiMultiline01.py

```

1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # レイアウト
4 layout = [[sg.Multiline(default_text='ここでテキストを編集',
5                          size=(35,5), border_width=2, key='ta1',
6                          text_color='#0000ff',background_color='#ffff00')],
7           [sg.Button('read',key='bt1')]]
8 # ウィンドウ作成

```



```

9 window = sg.Window('psguiMultiline01.py', layout)
10 # イベントループ
11 while True:
12     event, values = window.read() # イベントの読み取り (イベント待ち)
13     print('イベント:', event, ', 値:', values) # 確認表示
14     if event == None: # 終了条件 (None: クローズボタン)
15         break
16 # 終了処理
17 window.close()

```

このプログラムを実行すると図 12 のようなウィンドウが表示される。「read」ボタンをクリックすると、テキストフィールドの内容を標準出力に出力する。

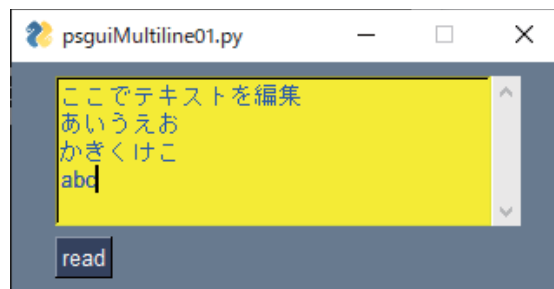


図 12: InputText (テキストフィールド)

このウィンドウにおいて「read」ボタンをクリックすると次のような出力が標準出力に得られる。

例. 「read」ボタンのクリック

```
イベント:bt1, 値:{'ta1': 'ここでテキストを編集\nあいうえお\nかきくけこ\nabc\n'}
```

Multiline オブジェクト作成時にキーワード引数 'enable_events=True' を与えることで、キー入力イベントが受信可能となる。

3.3 ボタン類

3.3.1 Button

ボタンを配置するには Button オブジェクトを使用する。

```
Button( ボタントップの文字列, border_width=枠の太さ, button_color=(文字の色, 地の色),
        font=( フォント名, サイズ ) )
```

様々な枠の太さや色を持つボタンを表示するサンプルプログラム psguiButtons01.py を示す。

プログラム: psguiButtons01.py

```

1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # レイアウト
4 layout = [ [sg.Button('枠太さ:8', size=(10,1), border_width=8),
5             sg.Button('枠太さ:4', size=(10,1), border_width=4),
6             sg.Button('デフォルト', size=(10,1))],
7           [sg.Button('白地に黒', size=(10,1), button_color=('000000', 'ffffff')),
8             sg.Button('黒地に白', size=(10,1), button_color=('ffffff', '000000')),
9             sg.Button('赤地に青', size=(10,1), button_color=('0000ff', 'ff0000'))]]
10 # ウィンドウ作成
11 window = sg.Window('psguiButtons01.py', layout)
12 # イベントループ
13 while True:
14     event, values = window.read() # イベントの読み取り (イベント待ち)
15     print('イベント:', event, ', 値:', values) # 確認表示
16     if event == None: # 終了条件 (None: クローズボタン)

```

```

17         break
18 # 終了処理
19 window.close()

```

これを実行した例を図 13 に示す。



図 13: 様々な枠の太さ、色を持つボタン

ボタントップに画像を与えるには、Button オブジェクト作成時にキーワード引数 'image_filename=画像ファイル名' を与える。これを次のサンプルプログラム psguiButtons02.py で示す。

プログラム：psguiButtons02.py

```

1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # レイアウト
4 layout = [[
5     sg.Button(image_filename='Earth.png', button_color=(None, '#000000'), key='b1'),
6     sg.Button(image_filename='Earth.png', button_color=(None, '#ffffff'), key='b2')]
7 # ウィンドウ作成
8 window = sg.Window('psguiButtons02.py', layout)
9 # イベントループ
10 while True:
11     event, values = window.read() # イベントの読み取り (イベント待ち)
12     print('イベント:', event, ', 値:', values) # 確認表示
13     if event == None: # 終了条件 (None: クローズボタン)
14         break
15 # 終了処理
16 window.close()

```

このプログラムでは、ボタンに画像 'Earth.png' (図 14) を与えている。



図 14: 透明の背景を持つ画像 'Earth.png'

プログラムを実行した例を図 15 に示す。

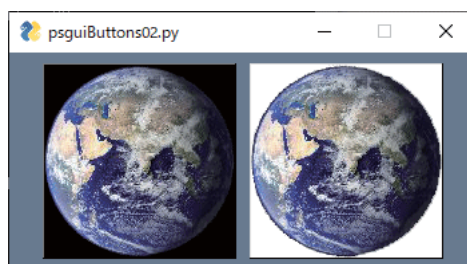


図 15: ボタントップに画像を与えた例

注意) PySimpleGUI の本書執筆時点の版 (4.16) では、ボタンに使用できる画像フォーマットは GIF, PNG など、tkinter で扱えるものに限られている。

3.3.2 Checkbox

チェックボックスは Checkbox オブジェクトで実現する。

```
Checkbox( 文字列, default=[False/True], background_color=色, text_color=色,  
          font=( フォント名, サイズ ), pad=((左余白, 右余白),(上余白, 下余白)) )
```

キーワード引数 default には、初期のチェック状態を真理値で与える。

3.3.3 Radio

ラジオボタンは Radio オブジェクトで実現する。

```
Radio( 文字列, default=[False/True], group_id=グループ, text_color=色,  
        background_color=色, font=( フォント名, サイズ ),  
        pad=((左余白, 右余白),(上余白, 下余白)) )
```

ラジオボタンはあるグループに属する。すなわち、複数のラジオボタンから成るグループ内で、1つだけを選択する (True にする) のものである。属するグループはキーワード引数 group_id に指定する。キーワード引数 default には、初期のチェック状態を真理値で与える。

■ サンプルプログラム

チェックボックス、ラジオボタンを配置するサンプルプログラム psguiButtons03.py を示す。

プログラム：psguiButtons03.py

```
1 # coding: utf-8  
2 import PySimpleGUI as sg      # ライブラリの読み込み  
3 # レイアウト  
4 layout = [[sg.Checkbox('項目11'), sg.Checkbox('項目12'), sg.Checkbox('項目13')],  
5           [sg.Radio('項目21', group_id='g1', default=True),  
6           sg.Radio('項目22', group_id='g1'), sg.Radio('項目23', group_id='g1')],  
7           [sg.Button('Check')]]  
8 # ウィンドウ作成  
9 window = sg.Window('psguiButtons03.py', layout)  
10 # イベントループ  
11 while True:  
12     event, values = window.read()    # イベントの読み取り (イベント待ち)  
13     print('イベント:', event, ', 値:', values)    # 確認表示  
14     if event == None:                # 終了条件 (None: クローズボタン)  
15         break  
16 # 終了処理  
17 window.close()
```

プログラムを実行した例を図 16 に示す。

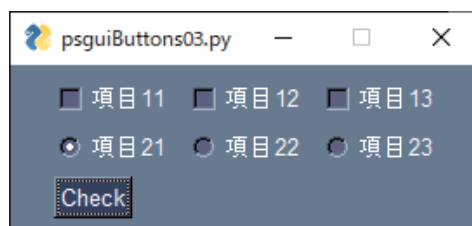


図 16: チェックボックスとラジオボタン

「Check」ボタンをクリックすると、各要素の値が標準出力に出力される。

3.4 選択入力

3.4.1 Listbox

Listbox オブジェクトは、複数の選択肢から項目を選ぶ、いわゆるリストボックスである。

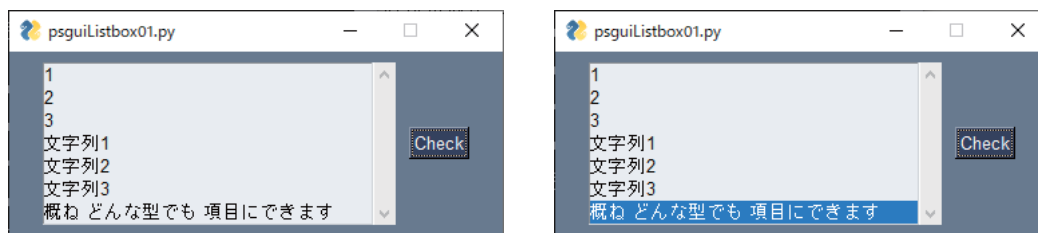
```
Listbox( 選択肢のリスト, default_values=初期値のリスト, background_color=色, text_color=色,  
        font=( フォント名, サイズ ), pad=((左余白, 右余白),(上余白, 下余白)) )
```

これを応用したサンプルを psguiListbox01.py に示す。

プログラム：psguiListbox01.py

```
1 # coding: utf-8  
2 import PySimpleGUI as sg      # ライブラリの読み込み  
3 # リスト項目  
4 itm = [1,2,3,'文字列1','文字列2','文字列3',  
5        ['概ね','どんな型でも','項目にできます']]  
6 # レイアウト  
7 layout = [[sg.Listbox(itm,size=(35,7),  
8                      default_values=[['概ね','どんな型でも','項目にできます']],  
9                      select_mode=sg.LISTBOX_SELECT_MODE_MULTIPLE  
10             ),  
11            sg.Button('Check')]]  
12 # ウィンドウ作成  
13 window = sg.Window('psguiListbox01.py', layout)  
14 # イベントループ  
15 while True:  
16     event, values = window.read() # イベントの読み取り (イベント待ち)  
17     print('イベント:',event,', 値:',values) # 確認表示  
18     if event == None: # 終了条件 (None:クローズボタン)  
19         break  
20 # 終了処理  
21 window.close()
```

プログラムを実行した例を図 16 に示す。



(a) 初期値設定なし

(b) 初期値設定あり

図 17: リストボックス

図 16 の (b) はプログラムの 8 行目のコメントを外して初期値 (最初に選択されている項目) の設定を有効にした例である。項目を選択して「Check」ボタンをクリックすると、選択された項目のリストが得られる。(次の例参照)

例. 「Check」ボタンのクリック

イベント:Check, 値:{0:[['概ね','どんな型でも','項目にできます']]}

プログラムの 9 行目のコメントを外し, Listbox にキーワード引数 'select_mode=sg.LISTBOX_SELECT_MODE_MULTIPLE' を与えると, 複数項目の選択が可能 (図 18) となる。

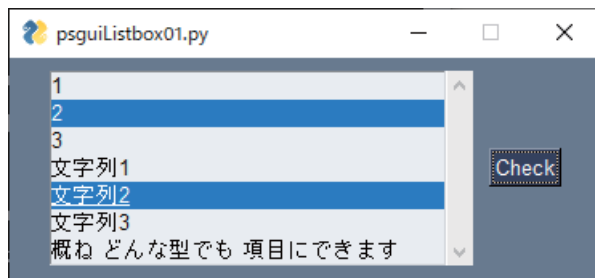


図 18: 複数選択可能な設定

例. 複数項目の選択結果

イベント:Check, 値:{0:[2,'文字列2']}

3.4.2 Spin

Spin オブジェクトは、複数の選択肢から1つの項目を選ぶ、いわゆるスピナーである。

Spin(選択肢のリスト, initial_value=初期値, background_color=色, text_color=色,
font=(フォント名, サイズ), pad=((左余白, 右余白),(上余白, 下余白)))

これを応用したサンプルを psguiSpin01.py に示す。

プログラム: psguiSpin01.py

```

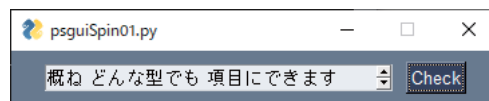
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # リスト項目
4 itm = [1,2,3,'文字列1','文字列2','文字列3',
5        ['概ね','どんな型でも','項目にできます']]
6 # レイアウト
7 layout = [[sg.Spin(itm,size=(35,7),
8                   #
9                   initial_value=['概ね','どんな型でも','項目にできます'],
10                  ),
11           sg.Button('Check')]]
12 # ウィンドウ作成
13 window = sg.Window('psguiSpin01.py', layout)
14 # イベントループ
15 while True:
16     event, values = window.read() # イベントの読み取り (イベント待ち)
17     print('イベント:',event,' 値:',values) # 確認表示
18     if event == None: # 終了条件 (None:クローズボタン)
19         break
20 # 終了処理
21 window.close()

```

プログラムを実行した例を図 19 に示す。



(a) 初期値設定なし



(b) 初期値設定あり

図 19: スピナー

図 19 の (b) はプログラムの 8 行目のコメントを外して初期値 (最初に選択されている項目) の設定を有効にした例である。項目を選択して「Check」ボタンをクリックすると、選択された項目の値が得られる。(次の例参照)

例. 「Check」ボタンのクリック

イベント:Check, 値:{0:'概ね どんな型でも 項目にできます'}

3.4.3 Slider

縦あるいは横方向の1次元的に移動するスライド式つまみによって値を入力する、いわゆるスライダを実現するには Slider オブジェクトを使用する。

```
Slider( range=(最小値, 最大値), default_value=初期値, resolution=刻み幅, orientation=向き,
        size=(移動方向の長さ, スライダの幅), background_color=色, text_color=色,
        font=( フォント名, サイズ ), pad=((左余白, 右余白),(上余白, 下余白)),
        border_width=太さ, relief=枠スタイル )
```

キーワード引数 orientation に 'h' を指定すると横向き, 'v' を指定すると縦向きのスライダとなる。暗黙値は 'v' (縦) である。値は range に指定した最小値から最大値までの範囲を取る。スライダつまみの刻み幅は resolution に指定する。

これを応用したサンプルを psguiSlider01.py に示す。

プログラム：psguiSlider01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 layout = [ [sg.Slider(range=(0.0,100.0), default_value=0.0, resolution=1.0,
5                     orientation='h', size=(35,None)),
6             sg.Slider(range=(0.0,100.0), default_value=0.0, resolution=1.0,
7                     orientation='v', size=(5,None))],
8            [sg.Button('Check')]]
9 # ウィンドウ作成
10 window = sg.Window('psguiSlider01.py', layout)
11 # イベントループ
12 while True:
13     event, values = window.read()    # イベントの読み取り (イベント待ち)
14     print('イベント:', event, ', 値:', values)    # 確認表示
15     if event == None:                # 終了条件 (None: クローズボタン)
16         break
17 # 終了処理
18 window.close()
```

これは、横方向のスライダと縦方向のスライダを配置するものである。このプログラムを実行した例を図 20 に示す。



図 20: 横, 縦のスライダ

「Check」ボタンをクリックするとスライダの値が出力される。

例. 「Check」をクリック

イベント: Check , 値: {0:19.0, 1:73.0}

3.4.3.1 Slider への値の設定

InputText の扱いの際に説明したのと同様に、Slider オブジェクトに対しても Update メソッドで値を設定することができる。このことをサンプルプログラム psguiSlider02.py で示す。

プログラム：psguiSlider02.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 L = [[sg.Slider(range=(0.0,100.0), default_value=0.0, resolution=1.0,
5             orientation='h', size=(35,None),key='s11',enable_events=True)],
6       [sg.InputText(default_text='0',size=(10,1),key='tx1'),
7       sg.Button('Set',key='bt1')]]
8 # ウィンドウ作成
9 window = sg.Window('psguiSlider02.py', L)
10 # イベントループ
11 while True:
12     event, values = window.read() # イベントの読み取り (イベント待ち)
13     print('イベント:',event,', 値:',values) # 確認表示
14     if event == None: # 終了条件 (None:クローズボタン)
15         break
16     elif event == 's11':
17         window['tx1'].Update(values['s11'])
18     elif event == 'bt1':
19         window['s11'].Update(values['tx1'])
20 # 終了処理
21 window.close()
```

このプログラムはスライダーとテキストフィールドの値を相互に設定し合うものである。スライダーを変化させるとその値がテキストフィールドに設定される。また、テキストフィールドに値を入力して「Set」ボタンをクリックすると、その値がスライダーに反映される。(図 21)

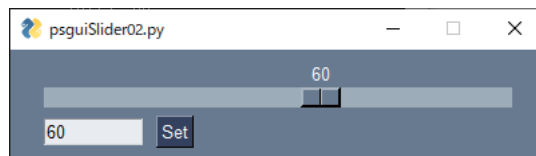


図 21: スライダー／テキストフィールド相互の値設定

3.5 枠, 区切り線

3.5.1 Frame

Frame オブジェクトを用いると、レイアウトのリストを1つのグループとしてまとめ、1つのレイアウト要素にすることができ。

```
Frame( タイトル, レイアウトのリスト, title_color=色, title_location=位置, background_color=色,
       font=( フォント名, サイズ ), pad=((左余白, 右余白),(上余白, 下余白)),
       border_width=太さ, relief=枠スタイル )
```

Frame でまとめられた GUI は1つの枠としてウィンドウに表示される。Frame に与えたタイトルは枠の上部に表示される。複数のボタンを Frame でまとめるサンプルを psguiFrame01.py に示す。

プログラム：psguiFrame01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 L1=[[sg.Button('A',size=(5,1)),sg.Button('B',size=(5,1))],
5     [sg.Button('C',size=(5,1)),sg.Button('D',size=(5,1))]]
6
7 L2=[[sg.Button('E',size=(5,1)),sg.Button('F',size=(5,1))],
8     [sg.Button('G',size=(5,1)),sg.Button('H',size=(5,1))]]
9
10 L=[[sg.Frame('Group 1',L1),sg.Frame('Group 2',L2)]]
11
```

```

12 # ウィンドウ作成
13 window = sg.Window('psguiFrame01.py', L)
14 # イベントループ
15 while True:
16     event, values = window.read() # イベントの読み取り (イベント待ち)
17     print('イベント:', event, ', 値:', values) # 確認表示
18     if event == None: # 終了条件 (None: クローズボタン)
19         break
20 # 終了処理
21 window.close()

```

これを実行した例を図 22 に示す。

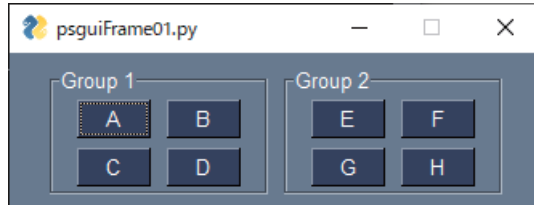


図 22: グループ化されたレイアウト

キーワード引数 `title_color` の値によって Frame のタイトルの色を設定できる。また、`title_location` に与える値 (表 1) によって、Frame のタイトルの位置を変更することができる。

表 1: タイトルの位置: `title_location` に与える値

設定値	位置	設定値	位置
TITLE_LOCATION_TOP	上部中央	TITLE_LOCATION_BOTTOM	下部中央
TITLE_LOCATION_LEFT	左辺	TITLE_LOCATION_RIGHT	右辺
TITLE_LOCATION_TOP_LEFT	上部左	TITLE_LOCATION_TOP_RIGHT	上部右
TITLE_LOCATION_BOTTOM_LEFT	下部左	TITLE_LOCATION_BOTTOM_RIGHT	下部右

これらのことをサンプルプログラム `psguiFrame02.py` で示す。

プログラム: `psguiFrame02.py`

```

1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # レイアウト
4 L1=[[sg.Button('A',size=(5,1)),sg.Button('B',size=(5,1)),
5     sg.Button('C',size=(5,1)),sg.Button('D',size=(5,1))]
6
7 L=[[sg.Frame('Group 1',L1, title_color='#ffff00',
8 #     title_location = sg.TITLE_LOCATION_TOP,
9 #     title_location = sg.TITLE_LOCATION_BOTTOM,
10 #     title_location = sg.TITLE_LOCATION_LEFT,
11 #     title_location = sg.TITLE_LOCATION_RIGHT,
12 #     title_location = sg.TITLE_LOCATION_TOP_LEFT,
13 #     title_location = sg.TITLE_LOCATION_TOP_RIGHT,
14 #     title_location = sg.TITLE_LOCATION_BOTTOM_LEFT,
15 #     title_location = sg.TITLE_LOCATION_BOTTOM_RIGHT,
16 )]]
17
18 # ウィンドウ作成
19 window = sg.Window('psguiFrame02.py', L)
20 # イベントループ
21 while True:
22     event, values = window.read() # イベントの読み取り (イベント待ち)
23     print('イベント:', event, ', 値:', values) # 確認表示
24     if event == None: # 終了条件 (None: クローズボタン)
25         break
26 # 終了処理
27 window.close()

```

プログラムの 8~15 行目のコメントを選択的に外して実行した例を図 23 に示す。

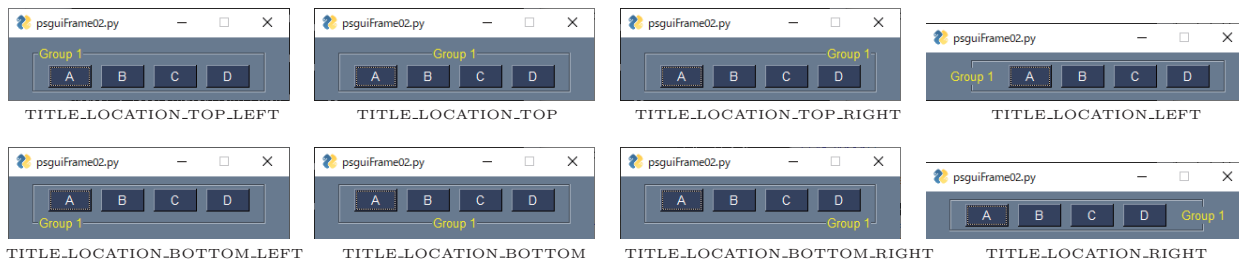


図 23: タイトルの位置

3.5.2 VerticalSeparator

VerticalSeparator オブジェクトは垂直の区切り線である。

VerticalSeparator(pad=((左余白, 右余白),(上余白, 下余白)))

先示したサンプルプログラム psguiFrame01.py の 10 行目を

```
L=[[sg.Frame('Group 1',L1),sg.VerticalSeparator(),sg.Frame('Group 2',L2)]]
```

と変更して実行すると図 24 のように垂直の区切り線が表示される。

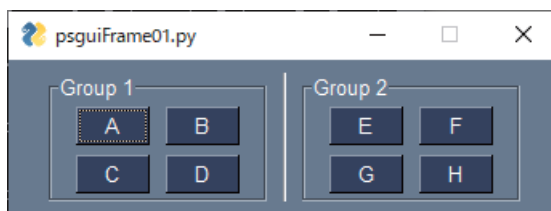


図 24: 垂直の区切り線

3.6 進捗バー

進捗バーは ProgressBar オブジェクトで実現する。

**ProgressBar(最大値, orientation=向き, bar_color=(背景色, バーの色),
pad=((左余白, 右余白),(上余白, 下余白)), border_width=太さ, relief=枠スタイル)**

これを応用したサンプルプログラム psguiProgressBar01.py を示す。

プログラム：psguiProgressBar01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # レイアウト
4 L=[
5     [sg.Slider(range=(0,100),orientation='h',size=(25,None),resolution=1,
6         enable_events=True,key='sld1'),
7     sg.ProgressBar(100,orientation='h',size=(25,15),key='pbar1',
8         bar_color=(' #aaa', '#fff'))],
9     [sg.ProgressBar(100,orientation='h',size=(25,15),key='pbar2',
10        bar_color=(' #000', '#aaa')),
11     sg.ProgressBar(100,orientation='h',size=(25,15),key='pbar3',
12        bar_color=(' #f00', '#aaa'))],
13     [sg.ProgressBar(100,orientation='h',size=(25,15),key='pbar4',
14        bar_color=(' #0f0', '#aaa')),
15     sg.ProgressBar(100,orientation='h',size=(25,15),key='pbar5',
16        bar_color=(' #00f', '#aaa'))],
17     [sg.ProgressBar(100,orientation='h',size=(25,15),key='pbar6',
18        bar_color=(' #ff0', '#aaa')),
19     sg.ProgressBar(100,orientation='h',size=(25,15),key='pbar7')]
20 ]
```

```

21 # ウィンドウ作成
22 window = sg.Window('psguiProgressBar01.py', L)
23 # 進捗バーを取得
24 p1 = window['pbar1']; p2 = window['pbar2']; p3 = window['pbar3']
25 p4 = window['pbar4']; p5 = window['pbar5']; p6 = window['pbar6']
26 p7 = window['pbar7']
27 # イベントループ
28 while True:
29     event, values = window.read() # イベントの読み取り (イベント待ち)
30     print('イベント:', event, ', 値:', values) # 確認表示
31     if event in (None, 'Quit'): # 終了条件 (None: クローズボタン)
32         break
33     # 進捗バーに値を設定
34     p1.UpdateBar(values['sld1']); p2.UpdateBar(values['sld1'])
35     p3.UpdateBar(values['sld1']); p4.UpdateBar(values['sld1'])
36     p5.UpdateBar(values['sld1']); p6.UpdateBar(values['sld1'])
37     p7.UpdateBar(values['sld1'])
38 # 終了処理
39 window.close()

```

これは、進捗バーの背景色とバーの色を様々に設定する例である。実行例を図 25 に示す。

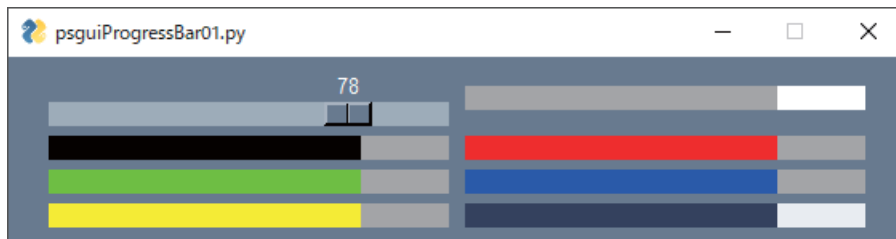


図 25: スライダに連動する進捗バー

スライダの動きに合わせて進捗バーが変化する。

3.7 表, ツリー

3.7.1 Table

行・列の形式の表は Table オブジェクトで実現する。

```

Table( 表データ, headings=列見出しのリスト, def_col_width=列の幅, num_rows=表示行数,
       display_row_numbers=[True/False], font=( フォント名, サイズ ),
       text_color=表の文字の色,
       background_color=表の背景色 1, alternating_row_color=表の背景色 2,
       header_font=( フォント名, サイズ ),
       header_text_color=列見出しの文字色, header_background_color=列見出しの背景色 )

```

表データには次のような形式のリストを与える。

```

[ [1 行目第 1 列, 1 行目第 2 列, 1 行目第 3 列, ...],
  [2 行目第 1 列, 2 行目第 2 列, 2 行目第 3 列, ...],
  :

```

キーワード引数 `display_row_numbers` には行番号の表示の有無を真偽値で与える。キーワード引数 `alternating_row_color` には偶数インデックスの行の背景色を与えることができ、これにより表の背景を 2 色の縞模様を設定することができる。表の見出しの体裁はキーワード引数 `header_font`, `header_text_color`, `header_background_color` に与える値で設定する。

Table オブジェクトを用いた例をサンプルプログラム `psguiTable01.py` に示す。

プログラム：psguiTable01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # 表
4 T = [[x+20*y for x in range(20)] for y in range(50)]
5 H = [chr(65+x) for x in range(20)]
6 # レイアウト
7 L=[[sg.Table(T,headings=H,auto_size_columns=False,vertical_scroll_only=False,
8             def_col_width=5,
9             num_rows=10,
10            display_row_numbers=True,
11            font=('小塚ゴシック Pro R',10),
12            text_color='#000000',
13            background_color='#cccccc',
14            alternating_row_color='#ffffff',
15            header_font=('小塚ゴシック Pro B',10),
16            header_text_color='#0000ff',
17            header_background_color='#cccccc'
18            )]]
19 # ウィンドウ作成
20 window = sg.Window('psguiTable01.py', L, resizable=True, size=(400,240) )
21 # イベントループ
22 while True:
23     event, values = window.read() # イベントの読み取り (イベント待ち)
24     if event == None: # 終了条件 (None:クローズボタン)
25         break
26 # 終了処理
27 window.close()
```

このプログラムでは、表を縦・横にスクロール可能にするためにキーワード引数 'vertical_scroll_only=False' を与えている。また、列の幅の自動調整を抑止するためにキーワード引数 'auto_size_columns=False' を与えている。このプログラムの実行例を図 26 の (a) に示す。

Row	A	B	C	D	E	F	G
0	0	1	2	3	4	5	6
1	20	21	22	23	24	25	26
2	40	41	42	43	44	45	46
3	60	61	62	63	64	65	66
4	80	81	82	83	84	85	86
5	100	101	102	103	104	105	106
6	120	121	122	123	124	125	126
7	140	141	142	143	144	145	146
8	160	161	162	163	164	165	166
9	180	181	182	183	184	185	186

(a) 12,13,14 行目をコメントにした場合

Row	A	B	C	D	E	F	G
0	0	1	2	3	4	5	6
1	20	21	22	23	24	25	26
2	40	41	42	43	44	45	46
3	60	61	62	63	64	65	66
4	80	81	82	83	84	85	86
5	100	101	102	103	104	105	106
6	120	121	122	123	124	125	126
7	140	141	142	143	144	145	146
8	160	161	162	163	164	165	166
9	180	181	182	183	184	185	186

(b) 12~15 行目の '#' を外した場合

図 26: 表の表示

プログラムの 12~15 行目先頭の '#' を外して実行すると図 26 の (b) のようになる。

3.7.2 Tree

Tree オブジェクトは階層構造の図を作成するためのものである。Tree が扱う階層構造データは TreeData オブジェクトである。

Tree(TreeData オブジェクト, headings=列見出しのリスト,
def_col_width=列の幅, num_rows=表示行数, col0_width=階層表示部の幅,
font=(フォント名, サイズ), text_color=文字の色,
background_color=背景色, header_font=(フォント名, サイズ),
header_text_color=列見出しの文字色, header_background_color=列見出しの背景色)

見出しの体裁はキーワード引数 `header_font`, `header_text_color`, `header_background_color` に与える値で設定する。

■ TreeData オブジェクト

TreeData オブジェクトには `Insert` メソッドで要素を追加する。これら要素は階層の上下関係（親子関係）を保持するものである。実際の扱いについてサンプルプログラム `psguiTree01.py` を示して説明する。

プログラム：psguiTree01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # ツリー
4 T = sg.TreeData()
5 T.Insert('', 'node1', 'The root', values=['folder', 1], icon='icon_folder_tiny.png')
6 T.Insert('node1', 'node11', 'file1', values=['file', 2], icon='icon_file_tiny.png')
7 T.Insert('node1', 'node12', 'file2', values=['file', 2], icon='icon_file_tiny.png')
8 T.Insert('node1', 'node2', 'folder1', values=['folder', 2], icon='icon_folder_tiny.png')
9 T.Insert('node2', 'node21', 'file3', values=['file', 3], icon='icon_file_tiny.png')
10 T.Insert('node2', 'node22', 'file4', values=['file', 3], icon='icon_file_tiny.png')
11 # 見出し
12 H = ['種別', '深さ']
13 # レイアウト
14 L=[[sg.Tree(T, headings=H, num_rows=5, auto_size_columns=False,
15           col0_width=12, def_col_width=4,
16           font=('小塚ゴシック Pro R', 10),
17           #
18           text_color='#000000',
19           #
20           background_color='#ddddcc',
21           #
22           header_font=('小塚ゴシック Pro B', 10),
23           header_text_color='#000000',
24           header_background_color='#bbbbbb'
25          )]]
26 # ウィンドウ作成
27 window = sg.Window('psguiTree01.py', L)
28 # イベントループ
29 while True:
30     event, values = window.read() # イベントの読み取り（イベント待ち）
31     if event == None:             # 終了条件（None:クローズボタン）
32         break
33 # 終了処理
34 window.close()
```

プログラムの4~10行目で階層構造を構築している。4行目で TreeData オブジェクトを作成して変数 `T` に与えており、以後これに対して `Insert` メソッドで要素を追加している。

■ Insert メソッド

`Insert(親要素名, 当該要素名, 表示名, values=値のリスト, icon=アイコンファイル名)`

各要素は名前を持ち、これによって上下関係（親子関係）が定義される。すなわち「当該要素名」の要素が「親要素名」の要素の子要素となる。各要素は値のリストを持ち、実際の表示において「表」のような体裁となる。「アイコンファイル名」には当該要素を表示する際のアイコンのファイル名を与える。

サンプルプログラムに用いているアイコンファイルが図 27 のようなものである場合の実行例を図 28 に示す。

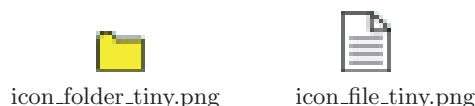


図 27: アイコン用の画像ファイル

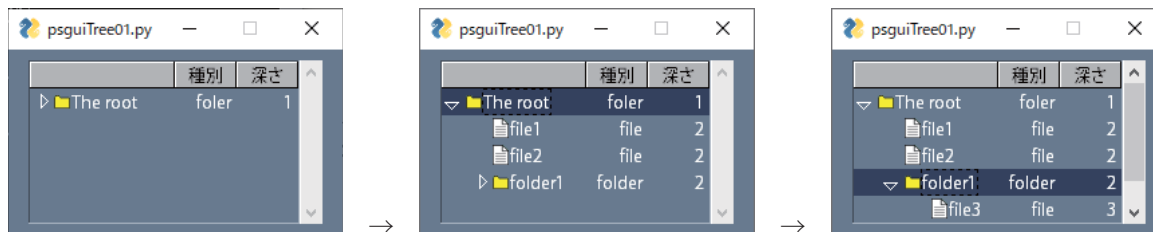


図 28: ファイルブラウザのような階層図

左端の列に階層構造が表示され、各要素に与えた値のリストはそれより右に表示される。子要素を持つ要素は展開あるいは縮小することができる。

左端の列の幅は Tree 作成時のキーワード引数 'col0_width=幅' で設定する。他のキーワード引数は概ね Table オブジェクトの場合と同じである。

プログラム 17~19 行目先頭の '#' を外して実行した様子を図 29 に示す。

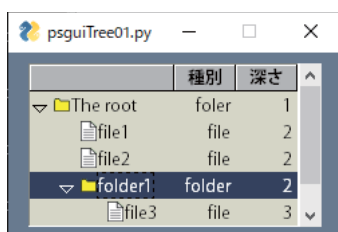


図 29: 体裁の変更

3.8 様々な表示構造

3.8.1 Tab, TabGroup

ウィジェットのレイアウトを保持する**タブ**を切り替え表示するには、Tab, TabGroup オブジェクトを使用する。TabGroup は複数の Tab をまとめるものである。

```
Tab( タイトル, レイアウト, title_color=色, font=( フォント名, サイズ ), background_color=色,
     pad=((左余白, 右余白),(上余白, 下余白)), border_width=太さ )
```

Tab オブジェクトは個々のタブを実現するもので、タブ内に表示するレイアウトを与える。タブはタイトルを持つ。

```
TabGroup( レイアウト, tab_location=位置, tab_background_color=色,
          selected_title_color=色, selected_background_color=色,
          pad=((左余白, 右余白),(上余白, 下余白)), border_width=太さ )
```

レイアウトには Tab オブジェクト のリストを与える。タブを実装するサンプルを psguiTab01.py に示す。

プログラム：psguiTab01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # タブ
4 T1 = sg.Tab('水星', [[sg.Image(filename='ImageMercury.png')]])
5 T2 = sg.Tab('金星', [[sg.Image(filename='ImageVenus.png')]])
6 T3 = sg.Tab('地球', [[sg.Image(filename='ImageEarth.png')]])
7 T4 = sg.Tab('火星', [[sg.Image(filename='ImageMars.png')]])
8 T5 = sg.Tab('木星', [[sg.Image(filename='ImageJupiter.png')]])
9 T6 = sg.Tab('土星', [[sg.Image(filename='ImageSaturn.png')]])
10 T7 = sg.Tab('天王星', [[sg.Image(filename='ImageUranus.png')]])
11 T8 = sg.Tab('海王星', [[sg.Image(filename='ImageNeptune.png')]])
12 T9 = sg.Tab('冥王星', [[sg.Image(filename='ImagePluto.png')]])
13 # レイアウト
14 L=[[sg.TabGroup([[T1,T2,T3,T4,T5,T6,T7,T8,T9]], tab_background_color='#ccc',
15              selected_title_color='#ff0', selected_background_color='#000',
16              #
17              tab_location='top'
18              #
19              tab_location='topleft'
20              #
21              tab_location='topright'
22              #
23              tab_location='bottom'
24              #
25              tab_location='bottomleft'
26              #
27              tab_location='bottomright'
28              #
29              tab_location='left'
30              #
31              tab_location='lefttop'
32              #
33              tab_location='leftbottom'
34              #
35              tab_location='right'
36              #
37              tab_location='righttop'
38              #
39              tab_location='rightbottom'
40              )]]
41 # ウィンドウ作成
42 window = sg.Window('psguiTab01.py', L )
43 # イベントループ
44 while True:
45     event, values = window.read() # イベントの読み取り (イベント待ち)
46     if event == None:           # 終了条件 (None:クローズボタン)
47         break
48 # 終了処理
49 window.close()
```

このプログラムで使用する画像データが図 30 のようなものである場合の実行例を図 31 に示す。

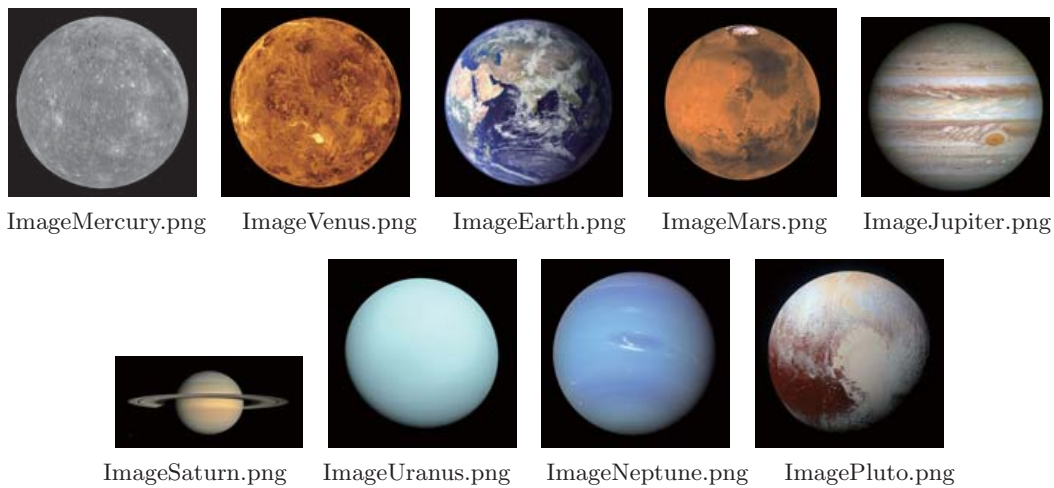


図 30: タブに表示する画像

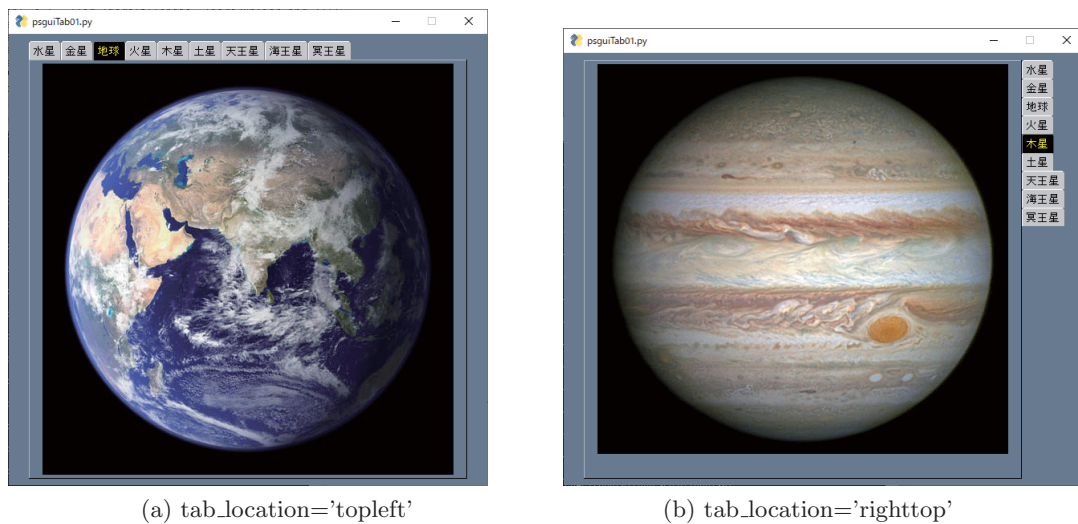


図 31: タブの位置の変更

3.8.2 Pane, Column

Column オブジェクトに構築したレイアウトをスライド表示⁷するために Pane オブジェクトが使用できる。具体的には、Column オブジェクトに構築したレイアウトを Pane オブジェクトで切り替えて表示する。

Column(レイアウト, background_color=色, pad=((左余白, 右余白),(上余白, 下余白)))

Column に表示する「レイアウト」を引数に与える。複数の Column オブジェクトをリストにしたものから、次の Pane オブジェクトを作成する。

Pane(Column のリスト, orientation=向き, pad=((左余白, 右余白),(上余白, 下余白)), border_width=太さ, relief=枠スタイル)

これらを応用したサンプルプログラムを `psguiColPane01.py` に示す。

⁷いわゆるアコーディオン形式のインターフェース。

プログラム：psguiColPane01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # タブ
4 T1 = sg.Column([[sg.Image(filename='ImageMercury.png')]])
5 T2 = sg.Column([[sg.Image(filename='ImageVenus.png')]])
6 T3 = sg.Column([[sg.Image(filename='ImageEarth.png')]])
7 T4 = sg.Column([[sg.Image(filename='ImageMars.png')]])
8 T5 = sg.Column([[sg.Image(filename='ImageJupiter.png')]])
9 T6 = sg.Column([[sg.Image(filename='ImageSaturn.png')]])
10 T7 = sg.Column([[sg.Image(filename='ImageUranus.png')]])
11 T8 = sg.Column([[sg.Image(filename='ImageNeptune.png')]])
12 T9 = sg.Column([[sg.Image(filename='ImagePluto.png')]])
13 # レイアウト
14 L=[[sg.Pane([T1,T2,T3,T4,T5,T6,T7,T8,T9], orientation='h')]]
15 # ウィンドウ作成
16 window = sg.Window('psguiColPane01.py', L, resizable=True, size=(800,550) )
17 # イベントループ
18 while True:
19     event, values = window.read() # イベントの読み取り (イベント待ち)
20     if event == None:           # 終了条件 (None:クローズボタン)
21         break
22 # 終了処理
23 window.close()
```

このプログラムの実行例を図 32 に示す。

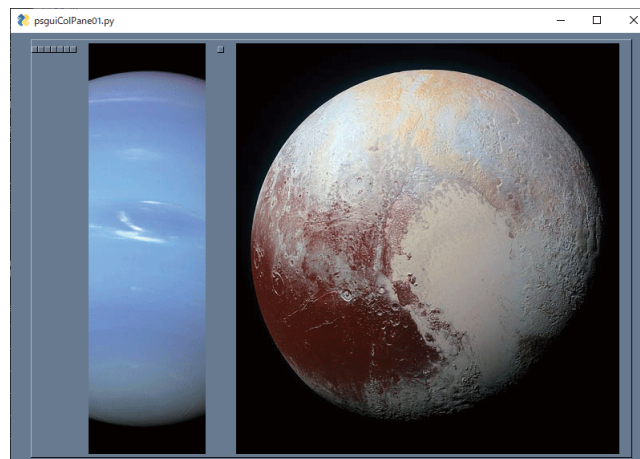


図 32: 表示位置とサイズを調整できる

Pane のハンドル  で各 Column をスライドして表示位置とサイズを調整する。

3.8.2.1 Column を応用したウィジェットの表示位置の調整

Column ウィジェットはレイアウト用のコンテナとして利用することができ、ウィジェットのレイアウトにおける表示の位置と大きさの微調整に应用することができる。例えば、サンプルプログラム psguiColumn01-1.py について考える。

プログラム：psguiColumn01-1.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 L=[[sg.Text('12', font=('',12), border_width=4, relief=sg.RELIEF_SOLID),
5     sg.Text('36', font=('',36), border_width=4, relief=sg.RELIEF_SOLID),
6     sg.Text('60', font=('',60), border_width=4, relief=sg.RELIEF_SOLID),
7     sg.Text('96', font=('',96), border_width=4, relief=sg.RELIEF_SOLID)]]
8
9 # ウィンドウ作成
10 window = sg.Window('psguiColumn01-1.py', L)
11 # イベントループ
```



```

12 while True:
13     event, values = window.read() # イベントの読み取り (イベント待ち)
14     print('イベント:', event, ', 値:', values) # 確認表示
15     if event == None: # 終了条件 (None: クローズボタン)
16         break
17 # 終了処理
18 window.close()

```

このプログラムは、サイズの異なる Text ウィジェットを1行に複数表示するもので、実行すると図 33 のようなウィンドウが表示される。

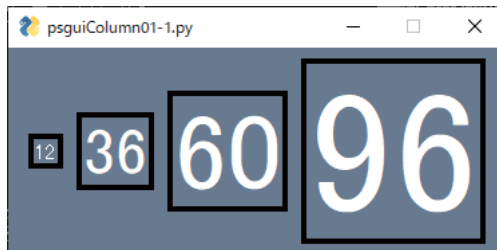


図 33: サイズの異なるウィジェットの表示

大きさの違う複数の Text ウィジェットが、縦方向 (上下方向) の中央に配置されていることがわかる。次に、Column ウィジェットを利用して、これらのウィジェットを「上付き」や「下付き」などの位置に配置する方法について考える。

Column ウィジェットにキーワード引数「vertical_alignment=縦位置」を与えることで、その Column ウィジェットのレイアウト中における縦位置を指定することができる⁸。この引数に与える「縦位置」には

```

'top'      →   上付き
'center'   →   中央配置
'bottom'   →   下付き

```

といった文字列を与える。

Column ウィジェットにキーワード引数「size=(横幅, 高さ)」を与えることで、その大きさを設定できる。(単位はピクセル)

以上のことを応用して、大きさの異なる複数の Text ウィジェットを配置するサンプルプログラム psguiColumn01-2.py を示す。

プログラム: psguiColumn01-2.py

```

1  # coding: utf-8
2  import PySimpleGUI as sg # ライブラリの読み込み
3  # レイアウト
4  C1=sg.Column(
5      [[sg.Text('12', font=('', 12), border_width=4, relief=sg.RELIEF_SOLID)]],
6      vertical_alignment='top', size=(34,34), pad=((0,0),(0,0)))
7  C2=sg.Column(
8      [[sg.Text('36', font=('', 36), border_width=4, relief=sg.RELIEF_SOLID)]],
9      vertical_alignment='top', size=(66,66), pad=((0,0),(0,0)))
10 C3=sg.Column(
11     [[sg.Text('60', font=('', 60), border_width=4, relief=sg.RELIEF_SOLID)]],
12     vertical_alignment='top', size=(100,100), pad=((0,0),(0,0)))
13 C4=sg.Column(
14     [[sg.Text('96', font=('', 96), border_width=4, relief=sg.RELIEF_SOLID)]],
15     vertical_alignment='top', size=(150,150), pad=((0,0),(0,0)))
16
17 L=[[C1,C2,C3,C4]]
18
19 # ウィンドウ作成
20 window = sg.Window('psguiColumn01-2.py', L)
21 # イベントループ
22 while True:

```

⁸キーワード引数「vertical_alignment=」は古い版の PySimpleGUI では使用できないので注意すること。

```

23     event, values = window.read() # イベントの読み取り (イベント待ち)
24     print('イベント:', event, ', 値:', values) # 確認表示
25     if event == None: # 終了条件 (None: クローズボタン)
26         break
27 # 終了処理
28 window.close()

```

このプログラムでは、C1～C4 の 4 つの Column ウィジェットにキーワード引数「vertical_alignment='top'」を与えて、Column ウィジェットを上付き配置のレイアウトにしている。このプログラムを実行すると図 34 の (a) のような表示となる。

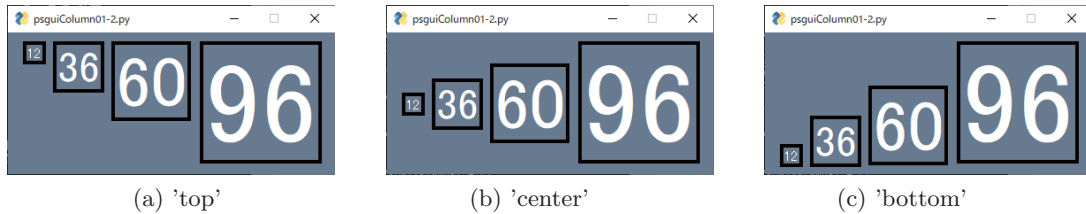


図 34: 大きさの異なるウィジェットの表示

vertical_alignment を 'center' にした場合の表示が (b), 'bottom' にした場合の表示が (c) である。

3.9 メニュー

3.9.1 OptionMenu

スピナー (Spin) に似た入力用ウィジェットに OptionMenu がある。

```
OptionMenu( 値のリスト, default_value=初期値, text_color=色, background_color=色,  
            pad=((左余白, 右余白),(上余白, 下余白)) )
```

「値のリスト」の要素を選択するウィジェットである。これを応用したサンプルプログラムを psguiOptionMenu01.py に示す。

プログラム：psguiOptionMenu01.py

```
1 # coding: utf-8  
2 import PySimpleGUI as sg # ライブラリの読み込み  
3 # レイアウト  
4 L=[[sg.OptionMenu([1,2,3,'項目1','項目2','項目3'],size=(7,1)),  
5     [sg.Button('Check')]]  
6 # ウィンドウ作成  
7 window = sg.Window('psguiOptionMenu01.py', L, size=(300,70))  
8 # イベントループ  
9 while True:  
10     event, values = window.read() # イベントの読み取り (イベント待ち)  
11     print('イベント:',event,', 値:',values) # 確認表示  
12     if event == None: # 終了条件 (None:クローズボタン)  
13         break  
14 # 終了処理  
15 window.close()
```

このプログラムの実行例を図 35 に示す。

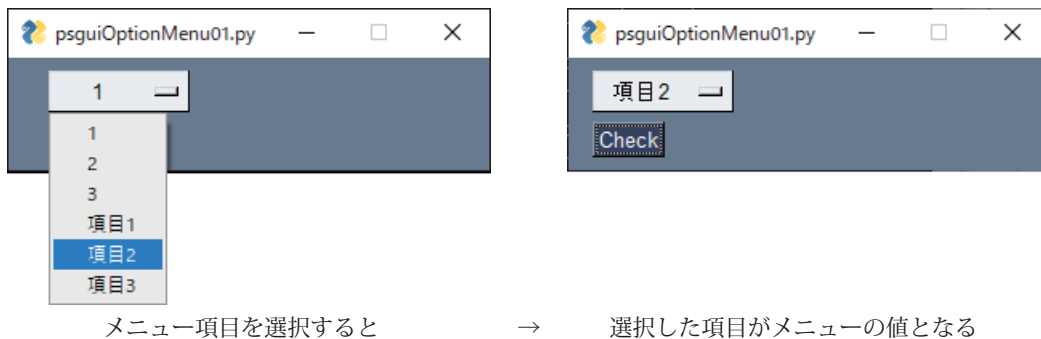


図 35: OptionMenu による値の選択

図 35 のウィンドウで「Check」ボタンをクリックすると、OptionMenu の値が標準出力に表示される。

3.9.2 MenuBar

プルダウン形式のメニューを構築するには MenuBar を使用する。

```
MenuBar( メニュー定義リスト )
```

メニュー定義を要素とする「メニュー定義リスト」に基づいてプルダウンメニューを構築する。

■ メニュー定義

[メニュー見出し, [項目 1, 項目 2, ...]]

このようなリストを要素とするリストを MenuBar の引数に与える。

メニューバーを構築するサンプルプログラムを psguiMenuBar01.py に示す。

プログラム：psguiMenuBar01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 L=[[sg.MenuBar([
5     ['&File', ['New', 'Open', '&Save', ['as Text', 'as HTML'], 'Close', '---', '&Quit']],
6     ['&Edit', ['Copy', 'Paste', 'Cut', '---', 'All']]], key='mb1'))]
7 # ウィンドウ作成
8 window = sg.Window('psguiMenuBar01.py', L, size=(300,40))
9 # イベントループ
10 while True:
11     event, values = window.read()    # イベントの読み取り (イベント待ち)
12     print('イベント:', event, ' 値:', values)    # 確認表示
13     if event==None or values['mb1']=='Quit':    # 終了条件 (None:クローズボタン)
14         break
15 # 終了処理
16 window.close()
```

このプログラムの実行例を図 36 に示す。

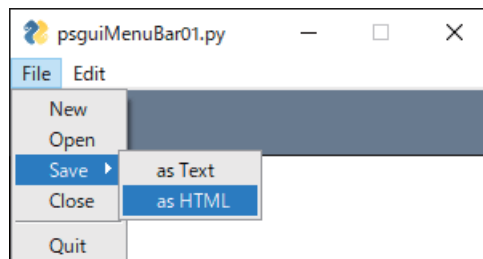


図 36: プルダウンメニュー

メニューの中の区切り線は '---', サブメニューは項目のリストを入れ子にして実現する。

このプログラムは、メニュー項目を選択すると、その値を標準出力に出力する。

■ メニュー選択のショートカットの設定

メニュー項目の選択には **Alt** キーによるショートカットが設定できる。先のプログラムでは 5,6 行目でメニュー項目を記述しているが、所々に「&」がある。これがショートカットの設定であり、「&」直後の文字がショートカットのためのキーとなる。例えば「File」メニューは **Alt+F**, 「Quit」メニューは **Alt+Q** がショートカットとなる。

3.9.3 ButtonMenu

ButtonMenu はボタン形式のメニューである。

```
ButtonMenu( メニュー見出し, メニュー定義, font=( フォント名, サイズ ), button_color=色,
            border_width=太さ, pad=((左余白, 右余白),(上余白, 下余白)) )
```

メニュー定義は MenuBar のところで説明したものと概ね同じであるが、ボタントップのメニュー見出しは ButtonMenu の第 1 引数に与えたものが採用される。

ボタンメニューを実装するサンプルプログラムを psguiButtonMenu01.py に示す。

プログラム：psguiButtonMenu01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 L=[[sg.ButtonMenu('File',
5     ['dummy', ['New', 'Save', ['as Text', 'as HTML'], 'Open', 'Close', '---', 'Quit']],
```

```

6         size=(7,1),key='btm1')]]
7 # ウィンドウ作成
8 window = sg.Window('psguiButtonMenu01.py', L, size=(300,70))
9 # イベントループ
10 while True:
11     event, values = window.read() # イベントの読み取り (イベント待ち)
12     print('イベント:',event,', 値:',values) # 確認表示
13     if event==None or values['btm1']=='Quit': # 終了条件 (None:クローズボタン)
14         break
15 # 終了処理
16 window.close()

```

このプログラムの実行例を図 37 に示す。

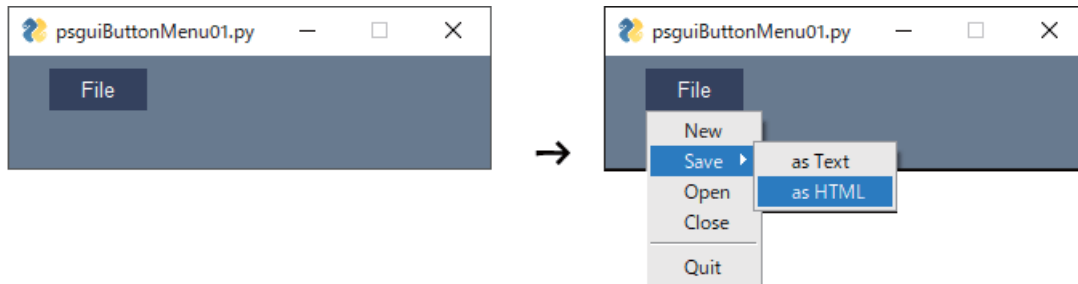


図 37: ボタンメニュー

このプログラムは、メニュー項目を選択すると、その値を標準出力に出力する。

3.10 Canvas

自由な描画を行うための描画領域のウィジェットとして Canvas がある。

```

Canvas( size=(横幅, 高さ), background_color=色,
        pad=((左余白, 右余白),(上余白, 下余白)), border_width=太さ )

```

「横幅」「高さ」には描画領域のサイズ (単位はピクセル) を与える。このウィジェットはプロパティ TKCanvas に Tkinter の Canvas ウィジェットを持つ。すなわち、その Tkinter の Canvas ウィジェットに対して各種の描画メソッドを用いて描画を行う。(ここでは、異なる 2 種類の Canvas オブジェクトを取り上げるので注意されたい)

実際の描画処理について、サンプルプログラム psguiCanvas01.py を例に挙げて説明する。

プログラム：psguiCanvas01.py

```

1 # coding: utf-8
2 import PySimpleGUI as sg # PySimpleGUIの読み込み
3
4 # レイアウト
5 cv = sg.Canvas(size=(400,140)) # PySimpleGUIのCanvasオブジェクト
6 L=[[cv]]
7
8 # ウィンドウ作成
9 window = sg.Window('psguiCanvas01.py', L,
10                 finalize=True # TkinterのCanvasを生成するための設定
11                 )
12
13 # Canvasへの描画
14 tcv = cv.TKCanvas # PySimpleGUIのCanvasからTkinterのCanvasを取得
15 tcv.create_oval(10,10,390,130, # 楕円の描画
16               outline='#ff0000',width=10,fill='#00ff00')
17
18 # イベントループ
19 while True:
20     event, values = window.read() # イベントの読み取り (イベント待ち)
21     print('イベント:',event,', 値:',values) # 確認表示

```

```

22     if event == None:                # 終了条件 (None: クローズボタン)
23         break
24
25 # 終了処理
26 window.close()

```

このプログラムはウィンドウ内に Canvas オブジェクトを1つだけ配置し、それに対して楕円を描画するものである。このプログラムを実行すると図 38 のようなウィンドウが表示される。



図 38: 実行例

解説.

5行目で PySimpleGUI の Canvas オブジェクト `cv` を作成し、それを6行目でレイアウト `L` に配置している。この段階ではまだ `cv.TKCanvas` には Tkinter の Canvas は与えられておらず、描画領域としての機能は持っていない。Window オブジェクトを作成する (9~11 行目) 際にキーワード引数「`finalize=True`」を指定することにより、実際に `cv.TKCanvas` に Tkinter の Canvas が与えられる。(14 行目ではそれを `tcv` に取得している)

各種の図形描画は Tkinter の Canvas に対して行うものであり、15~16 行目では `create_oval` メソッドを用いて楕円を描画している。このメソッド以外にも Tkinter の Canvas に対する描画メソッドが多数あり、詳しくは Python の公式インターネットサイトを始めとする他の資料⁹ を参照のこと。

⁹拙書「Python3 入門」でも Tkinter に関して基礎的な解説をしています。

4 ポップアップウィンドウ

主たるウィンドウとは別に、入力や確認を促すためのポップアップウィンドウを表示することができる。

4.1 入力

4.1.1 PopupGetText

ポップアップを表示して入力を促すには `PopupGetText` を使用する。

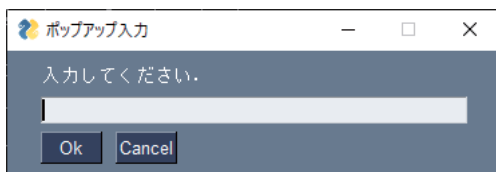
```
PopupGetText( メッセージ, title=ウィンドウタイトル, text_color=色, background_color=色,  
              font=( フォント名, サイズ ), button_color=(文字の色, 背景色) )
```

`PopupGetText` は入力用ウィンドウを表示する。この場合、ウィンドウ内に入力用のテキストフィールドと、「OK」、
「Cancel」2つのボタンを備える。「OK」をクリックするとテキストフィールドに入力した文字列を返し、「Cancel」を
クリックすると `None` を返す。

例. `PopupGetText` による入力

```
import PySimpleGUI as sg  
r = sg.PopupGetText('入力してください。', title='ポップアップ入力')
```

この結果、次のようなポップアップウィンドウが表示されて入力待ちとなる。



ボタンをクリックすると戻り値が `r` に得られる。

`PopupGetText` にキーワード引数 `'no_titlebar=True'` を与えるとタイトルバーが非表示となる。この場合、キー
ワード引数 `'grab_anywhere=True'` を与えておくと、ウィンドウの任意の位置をドラッグして移動ができる。

また、キーワード引数 `'password_char=文字'` を与えるとパスワード入力フィールドとなり、入力の際に「文字」が
ダミーとして表示される。

`PopupGetText` を応用したサンプルプログラムを `psguiPupGetTxt01.py` に示す。

プログラム： `psguiPupGetTxt01.py`

```
1 # coding: utf-8  
2 import PySimpleGUI as sg          # ライブラリの読み込み  
3 # レイアウト  
4 layout = [[sg.Multiline(size=(35,5), border_width=2, key='ta1')],  
5           [sg.Button('入力', key='bt1')]]  
6 # ウィンドウ作成  
7 window = sg.Window('psguiPupGetTxt01.py', layout)  
8 # テキストデータ  
9 txtdat = ''  
10 # イベントループ  
11 while True:  
12     event, values = window.read()    # イベントの読み取り (イベント待ち)  
13     if event == None:                # 終了条件 (None: クローズボタン)  
14         break  
15     elif event == 'bt1':  
16         r = sg.PopupGetText('入力してください。', title='ポップアップ入力')  
17         if r:                        # 戻り値が None や空文字列でない場合の処理  
18             txtdat += r + '\n'  
19             window['ta1'].Update( txtdat )  
20 # 終了処理  
21 window.close()
```

このプログラムの実行例を図 39 に示す。

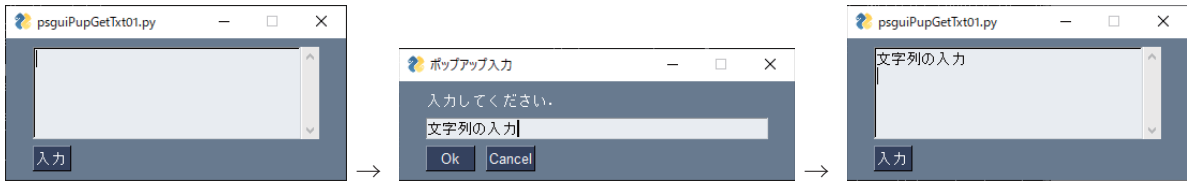


図 39: 「入力」をクリックするとポップアップが表示され、入力結果が反映される

4.2 報告、確認のためのポップアップ表示

報告、確認のメッセージを表示するためのポップアップとして表 2 に挙げるものが使用できる。

表 2: 報告、確認のためのポップアップ

API	解説	API	解説
Popup	「OK」ボタンの促し	PopupYesNo	Yes / No の選択
PopupCancel	キャンセルボタンの促し	PopupOKCancel	OK / Cancel の選択
PopupError	エラーメッセージ	PopupTimed	一定時間メッセージを表示

これらの実行例を次に示す。

実行例

<code>sg.Popup('よろしいですか?')</code> 	<code>sg.PopupYesNo('よろしいですか?')</code>
<code>sg.PopupCancel('許可されません')</code> 	<code>sg.PopupOKCancel('よろしいですか?')</code>
<code>sg.PopupError('!エラー発生!')</code> 	<code>sg.PopupTimed('残念でした')</code>

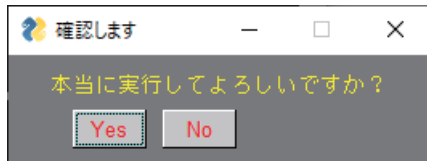
これら API はクリックしたボタンの名前が戻り値となる。またクローズボタンで閉じた場合は戻り値は None となる。これら API に共通する重要なキーワード引数を次に示す。

キーワード引数	解説
title=文字列	ウィンドウタイトル
font=(フォント名, サイズ)	表示するメッセージのフォント
text_color=色	表示するメッセージの色
background_color=色	ウィンドウの背景色
button_color=(文字色, 背景色)	ボタンの色
no_titlebar=[True/False]	True の場合はタイトル非表示

例. PopupYesNo のカスタマイズ

```
sg.PopupYesNo(' 本当に実行してよろしいですか?',title=' 確認します',  
text_color='#ff0',background_color='#777',button_color=('f00','#ccc'))
```

これを実行すると次のようなポップアップが表示される.



5 ウィンドウの扱い

5.1 リサイズ可能にする設定

Window オブジェクトの表示サイズは自動的に調整されるが、作成時にキーワード引数 'resizable=True' を与えると変更可能なものとなる。またこの場合、キーワード引数 'size=(横幅, 高さ)' を与えると初期のサイズが設定できる。

これに関するサンプルプログラムを psguiWinResize01.py に示す。

プログラム：psguiWinResize01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 L=[[sg.Image(filename='ImageEarth.png')]]
5 # ウィンドウ作成
6 window = sg.Window('psguiWinResize01.py', L, resizable=True, size=(300,300) )
7 # イベントループ
8 while True:
9     event, values = window.read() # イベントの読み取り (イベント待ち)
10    if event==None: # 終了条件 (None:クローズボタン)
11        break
12 # 終了処理
13 window.close()
```

このプログラムの実行例を図 40 に示す。

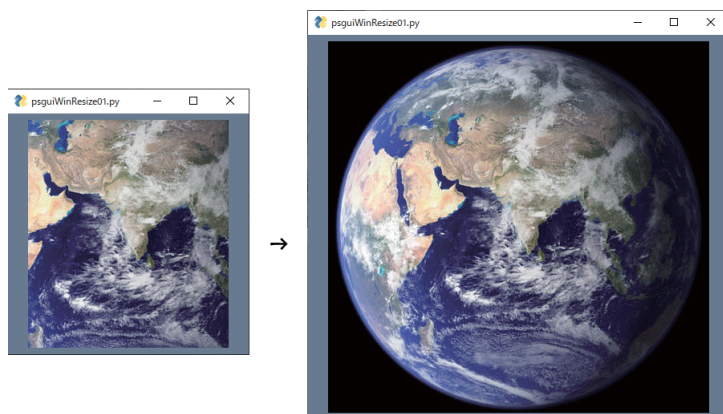


図 40: サイズ変更が可能なウィンドウ

ウィンドウのサイズ調整ができることが確認できる。

5.2 ウィンドウのサイズと位置について

ウィンドウのサイズは Window オブジェクトの size プロパティから (幅, 高さ) のタプルの形で得られる。またウィンドウの現在位置は Window オブジェクトに対して CurrentLocation メソッドを実行することで (水平位置, 垂直位置) のタプルの形で得られる。これらを応用したサンプルプログラム psguiWinResize02-1.py を示す。

プログラム：psguiWinResize02-1.py

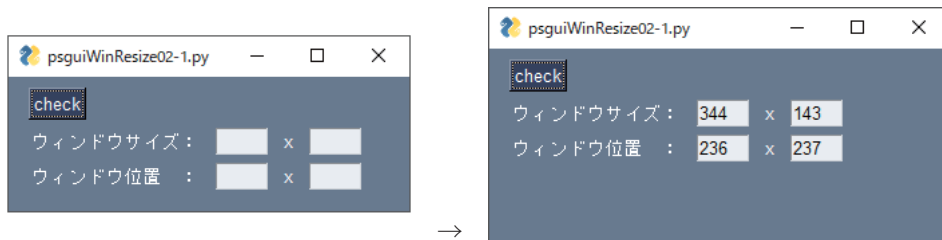
```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 L=[ [sg.Button('check',key='btn1')],
5     [sg.Text('ウィンドウサイズ:'), sg.InputText(size=(5,1),key='w'),
6      sg.Text('x'), sg.InputText(size=(5,1),key='h')],
7     [sg.Text('ウィンドウ位置 :'), sg.InputText(size=(5,1),key='px'),
8      sg.Text('x'), sg.InputText(size=(5,1),key='py')]]
9 # ウィンドウ作成
10 window = sg.Window('psguiWinResize02-1.py', L, resizable=True, size=(300,100) )
```

```

11 # イベントループ
12 while True:
13     event, values = window.read() # イベントの読み取り (イベント待ち)
14     if event==None: # 終了条件 (None:クローズボタン)
15         break
16     elif event=='btn1':
17         print( 'size:', window.size )
18         window['w'].Update( window.size[0] )
19         window['h'].Update( window.size[1] )
20         pos = window.CurrentLocation() # ウィンドウの現在位置の取得
21         print( 'position:', pos )
22         window['px'].Update( pos[0] )
23         window['py'].Update( pos[1] )
24 # 終了処理
25 window.close()

```

このプログラムの実行例を図 41 に示す。



起動後 (左) ウィンドウのサイズと位置を変更して「check」をクリックすると右の表示となる

図 41: ウィンドウのサイズと位置の取得

Window オブジェクトの size プロパティに値 (タプル) を設定することで、そのウィンドウのサイズを変更することができる。このことをプログラム psguiWinResize02-2.py で示す。

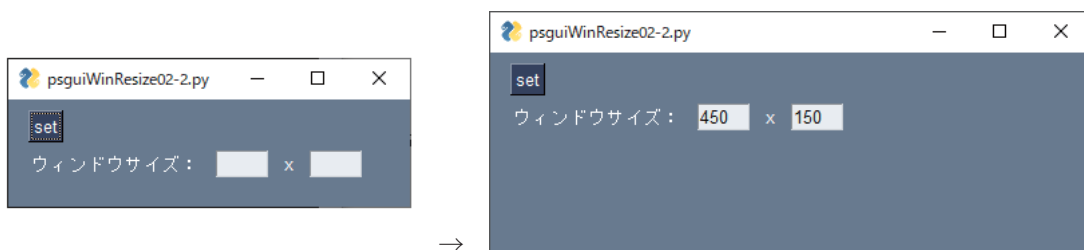
プログラム：psguiWinResize02-2.py

```

1 # coding: utf-8
2 import PySimpleGUI as sg # ライブラリの読み込み
3 # レイアウト
4 L=[ [sg.Button('set',key='btn1')],
5     [sg.Text('ウィンドウサイズ: '), sg.InputText(size=(5,1),key='w'),
6     sg.Text('x'), sg.InputText(size=(5,1),key='h')]]
7 # ウィンドウ作成
8 window = sg.Window('psguiWinResize02-2.py', L, resizable=True, size=(300,80) )
9 # イベントループ
10 while True:
11     event, values = window.read() # イベントの読み取り (イベント待ち)
12     if event==None: # 終了条件 (None:クローズボタン)
13         break
14     elif event=='btn1':
15         window.size = ( values['w'], values['h'] )
16         print( 'size:', window.size )
17 # 終了処理
18 window.close()

```

このプログラムの実行例を図 42 に示す。



起動後のウィンドウ (左) にサイズを入力して「set」をクリックするとウィンドウのサイズが変わる (右)

図 42: ウィンドウのサイズの変更

5.3 テーマの設定

5.3.1 デザインテーマの一覧表示

PySimpleGUI で利用できるウィンドウのデザインテーマの一覧を表示するには

```
preview_all_look_and_feel_themes
```

を実行する。

例. ウィンドウのデザインテーマの一覧表示

```
import PySimpleGUI as sg
sg.preview_all_look_and_feel_themes()
```

この結果, 図 43 のような一覧が表示される。

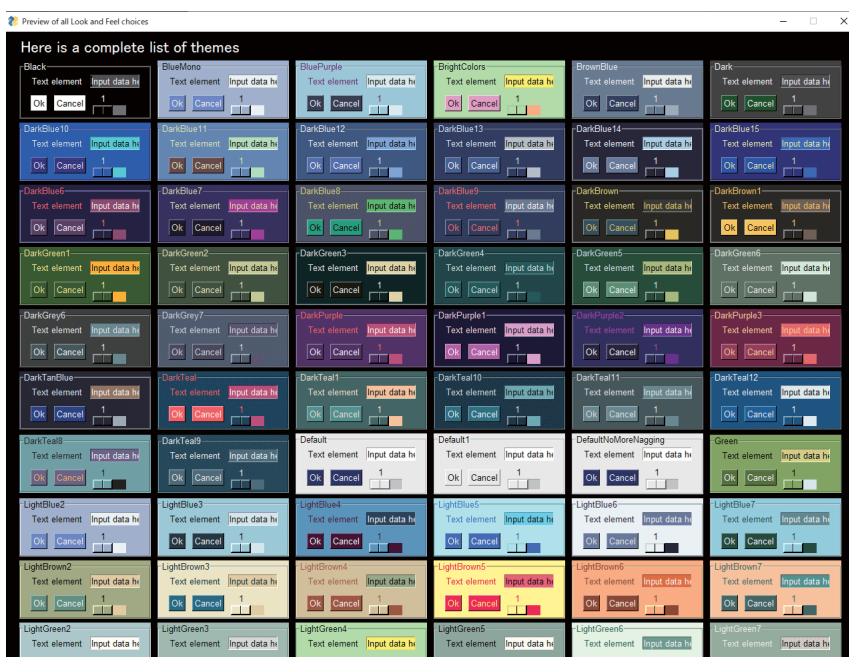


図 43: デザインテーマの一覧表示

5.3.2 使用できるデザインテーマの調査

PySimpleGUI で利用できるウィンドウのデザインテーマを調べるには

```
theme_list
```

を実行する。戻り値としてテーマ名のリストが得られる。

例. デザインテーマの調査

```
import PySimpleGUI as sg
TL = sg.theme_list()
```

この結果, 変数 TL にテーマ名のリストが得られる。

例. テーマ名のリスト

```
['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', ... ]
```

5.4 タイトルバーの有無

Window オブジェクト作成時にキーワード引数 'no_titlebar=True' を与えるとタイトルバーを非表示にできる。この場合、キーワード引数 'grab_anywhere=True' を与えておくと、ウィンドウの任意の位置をドラッグして移動ができる。このことを次のサンプルプログラム psguiNoTitlebar01.py で確認できる。

プログラム：psguiNoTitlebar01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 layout = [[sg.Checkbox('項目11'), sg.Checkbox('項目12'), sg.Checkbox('項目13')],
5           [sg.Radio('項目21', group_id='g1', default=True),
6            sg.Radio('項目22', group_id='g1'), sg.Radio('項目23', group_id='g1')],
7           [sg.InputText(size=(35,1))],
8           [sg.Button('Check'), sg.Button('Quit')]]
9 # ウィンドウ作成
10 window = sg.Window('psguiNoTitlebar01.py', layout,
11                   no_titlebar=True, grab_anywhere=True, alpha_channel=0.6 )
12 # イベントループ
13 while True:
14     event, values = window.read() # イベントの読み取り (イベント待ち)
15     print('イベント:', event, ', 値:', values) # 確認表示
16     if event in (None, 'Quit'): # 終了条件 (None: クローズボタン)
17         break
18 # 終了処理
19 window.close()
```

このプログラムの実行例を図 44 に示す。

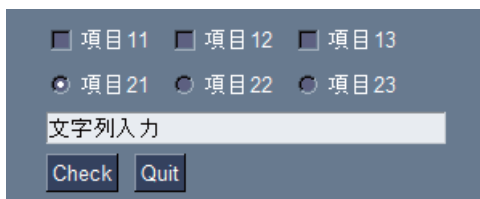


図 44: タイトルバーを持たないウィンドウ

移動のために、ウィンドウの任意の位置をドラッグできる。

5.4.1 ウィンドウの透明度

Window オブジェクト作成時にキーワード引数 'alpha_channel=不透明度' を与えるとウィンドウの不透明度 (0.0~1.0) を調整できる。(図 45)



図 45: 背景が透けるウィンドウ

A 付録

A.1 matplotlib で作成したグラフをレイアウトに埋め込む方法

matplotlib はグラフをプロットするための Python 用のライブラリとして広く普及している。matplotlib は作成したグラフを Tkinter の Canvas ウィジェットに変換する機能を提供しており、これを応用すると作成したグラフを PySimpleGUI のレイアウトに埋め込むことができる。ここでは、そのための考え方と手順について簡単な形で説明する。Tkinter と matplotlib に関する詳しい事柄は、それらの公式インターネットサイトを始めとする他の資料¹⁰ を参照のこと。

A.1.1 matplotlib によって作成されるグラフ

matplotlib が作成するグラフは Figure オブジェクトとして作成される。matplotlib を用いて正弦関数をプロットするサンプルプログラム psguiMatplotlib01.py を示す。

プログラム：psguiMatplotlib01.py

```
1 # coding: utf-8
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # 正弦関数の計算
6 x = np.linspace(0,4*np.pi,200) # 定義域
7 y = np.sin(x)                  # 値域
8
9 # グラフのプロット (matplotlib)
10 fig = plt.figure(figsize=(6,4)) # 描画用 Figure オブジェクト
11 plt.plot(x,y)                  # グラフのプロット
12 plt.xlabel('x')
13 plt.ylabel('sin(x)')
14 plt.show()                     # グラフの表示
```

解説.

2行目で数値演算のためのライブラリ NumPy を読み込んでいる。このライブラリを用いて関数の値を算出する。3行目でグラフを描画するライブラリ matplotlib を読み込んでいる。6行目で用意した定義域の配列 x ($0 \leq x \leq 4\pi$) に対する正弦関数の値を7行目で算出して配列 y に得ている。

10行目でグラフ描画のための Figure オブジェクトを fig に作成し、11行目でプロット（作図）を行っている。12,13行目はグラフの軸のラベルを描く処理である。グラフを作成する処理はここまでであり、14行目でそれを実際に表示する。

このプログラムを実行すると図 46 のようなグラフが表示される。

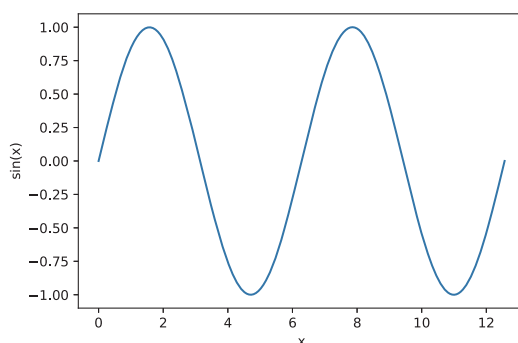


図 46: matplotlib で作成した正弦関数のグラフ

次に、このような手順で作成されるグラフ（Figure オブジェクト）を Tkinter の Canvas ウィジェットに変換し、それを PySimpleGUI のレイアウト内に埋め込む手順について説明する。

¹⁰ 拙書「Python3 入門」で Tkinter に関して基礎的な解説をしています。また、拙書「Python3 ライブラリブック」で matplotlib に関する基本的な解説をしています。

A.1.2 Figure オブジェクトのグラフを Canvas に変換する方法

matplotlib が提供する FigureCanvasTkAgg クラスの機能を使用することで、Figure オブジェクトのグラフを Tkinter の Canvas に変換することができる。FigureCanvasTkAgg クラスは次のようにして matplotlib ライブラリから読み込む。

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

先のサンプルプログラム psguiMatplotlib01.py で作成したのと同じグラフを PySimpleGUI のレイアウト中に埋め込む方法について例を挙げて説明する。(PySimpleGUI の Canvas ウィジェットへ描画に関しては、p.33 「3.10 Canvas」を参照のこと)

次のようなサンプルプログラム psguiMatplotlib02.py について考える。

プログラム：psguiMatplotlib02.py

```
1 # coding: utf-8
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
5 import PySimpleGUI as sg
6
7 # レイアウト
8 cv = sg.Canvas() # PySimpleGUIのCanvasオブジェクト
9 L=[[cv]]
10
11 # ウィンドウ作成
12 window = sg.Window('psguiMatplotlib02.py', L,
13                    finalize=True # TkinterのCanvasを生成するための設定
14                    )
15
16 # 正弦関数のプロット (matplotlib)
17 x = np.linspace(0,4*np.pi,200) # 定義域
18 y = np.sin(x) # 値域
19 fig = plt.figure(figsize=(6,4)) # 描画用Figureオブジェクト
20 plt.plot(x,y) # グラフのプロット
21 plt.xlabel('x')
22 plt.ylabel('sin(x)')
23
24 # Canvasへの描画
25 tcv = cv.TKCanvas # PySimpleGUIのCanvasからTkinterのCanvasを取得
26 fag = FigureCanvasTkAgg(fig, tcv) # figのグラフをTkinterのCanvasに関連付け
27 fag.draw() # TkinterのCanvasに描画
28 fag.get_tk_widget().pack() # TkinterのCanvasをレイアウトに反映
29
30 # イベントループ
31 while True:
32     event, values = window.read() # イベントの読み取り (イベント待ち)
33     print('イベント:',event,', 値:',values) # 確認表示
34     if event== None: # 終了条件 (None:クローズボタン)
35         break
36
37 # 終了処理
38 window.close()
```

解説.

8,9 行目で PySimpleGUI のレイアウトを作成している。cv はグラフを埋め込む対象の Canvas ウィジェットである。この cv は PySimpleGUI の Canvas ウィジェットである。実際の描画対象は cv が保持する Tkinter の Canvas ウィジェットであり、25 行目ではそれを tcv に取得している。

26 行目では FigureCanvasTkAgg クラスのオブジェクト fag を作成している。この際、FigureCanvasTkAgg コンストラクタの引数に Figure オブジェクト fig と、描画対象の tcv を与えており、これらが実際に関連付けられる。得られた fag は描画処理に必要な各種のメソッドやプロパティを持っている。fag に対して draw メソッドを実行 (27 行目) すると fig が保持するグラフが Tkinter の Canvas ウィジェットに変換され、それに対して pack メソッドを実行 (28 行目) するとグラフが Tkinter のレイアウトに反映される。

psguiMatplotlib02.py を実行すると図 47 のようなウィンドウが表示される。

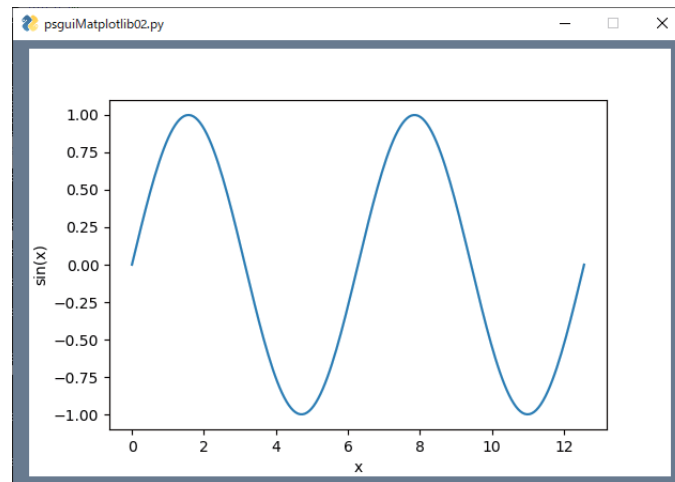


図 47: matplotlib で作成したグラフを PySimpleGUI のレイアウトに埋め込んだ例

参考.

FigureCanvasTkAgg コンストラクタで当該クラスのオブジェクトを作成する際、第 1 引数に与えた Figure オブジェクトから得られた Canvas (Tkinter 側のもの) を、第 2 引数に与えたウィジェット (Tkinter 側のもの) に子要素として接続する処理を行う。従って、上の例では tev に対して直接描画したのではなく、その子要素としてグラフの Canvas を接続した形になっている。この処理に関する詳しいことは matplotlib の公式インターネットサイトなどの資料を参照のこと。

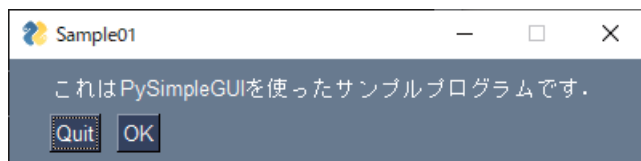
A.2 Tkinter のオブジェクトとの対応

Tkinter を基盤にして作られた PySimpleGUI では、各種のウィジェットはシステム内部では Tkinter のウィジェットとして実現されている。ここでは PySimpleGUI のウィジェットと Tkinter のウィジェットとの対応に関する最も基本的な事柄について説明する。

PySimpleGUI ではアプリケーションのウィンドウは Window オブジェクトとして実装されるが、それに対応する Tkinter 側のウィンドウオブジェクト (tkinter.Toplevel オブジェクト) は当該 Window オブジェクトの TKroot プロパティから得られる。このことに関して、本書の冒頭で示したサンプルプログラム psgui01.py を例に挙げて説明する。

プログラム：psgui01.py

```
1 # coding: utf-8
2 import PySimpleGUI as sg      # ライブラリの読み込み
3 # レイアウト
4 layout = [[sg.Text('これは PySimpleGUIを使ったサンプルプログラムです. '),
5            [sg.Button('Quit'),sg.Button('OK')]]
6 # ウィンドウ作成
7 window = sg.Window('Sample01', layout)
8 # イベントループ
9 while True:
10     event, values = window.read() # イベントの読み取り (イベント待ち)
11     print('イベント:',event,', 値:',values) # 確認表示
12     if event in (None,'Quit'):      # 終了条件 (None:クローズボタン)
13         print('終了します. ')
14         break
15 # 終了処理
16 window.close()
```



psgui01.py を実行した際のウィンドウ

このプログラムでは、アプリケーションのウィンドウは変数 window が保持している。これに対応する tkinter.Toplevel オブジェクトは window.TKroot を参照することで得られる。

例. tkinter.Toplevel オブジェクトの取得

```
root = window.TKroot
```

この例では変数 root に tkinter.Toplevel オブジェクトを得ており、これが、Tkinter のアプリケーションのウィンドウにおける最上位のオブジェクトである。

Tkinter では、tkinter.Toplevel オブジェクトを頂点として、各種のウィジェットを階層的に配置する形でウィンドウを構築する。従って、ウィンドウ内の個々のウィジェットにはその親となるウィジェットがある (最上位は例外)。

Tkinter において、あるウィジェットの子ウィジェットを調べるには、当該ウィジェットに対して winfo_children メソッドを実行する。

例. ウィジェット root の子ウィジェットを調べる

```
cld = root.winfo_children()
```

この例では変数 root が保持するウィジェットの子ウィジェットをリストの形で変数 cld に取得している。子ウィジェットが存在しない場合は winfo_children メソッドは空リスト [] を返す。

Tkinter において、あるウィジェットの親ウィジェットを調べるには、当該ウィジェットの master プロパティを参照する。

例. ウィジェット root の親ウィジェットを調べる

```
mst = root.master
```

この例では変数 root が保持するウィジェットの親ウィジェットを変数 mst に取得している。またこの例では、root は tkinter.Toplevel オブジェクトであるので、その親は Tkinter の tkinter.Tk オブジェクト（GUI 全体を統括するオブジェクト）となる。tkinter.Tk より上には親はなく、その master プロパティを参照すると None（ヌルオブジェクト）となる。

課題.

サンプルプログラム psgui01.py のウィジェットの構成を、Tkinter のウィジェットの階層関係として全て調べよ。

索引

- alpha_channel, 41
- background_color, 6
- border_width, 6, 7
- Button, 3, 13
- ButtonMenu, 32
- Canvas, 33
- Checkbox, 15
- close, 2
- Column, 27
- CurrentLocation, 38
- families, 7
- Figure, 42
- FigureCanvasTkAgg, 43
- font, 6
- Frame, 19
- grab_anywhere, 35, 41
- Image, 8
- InputText, 9
- Insert, 24
- key, 4
- Listbox, 16
- master, 45
- matplotlib, 42
- MenuBar, 31
- Multiline, 12
- no_titlebar, 41
- NumPy, 42
- OptionMenu, 31
- pad, 6, 7
- Pane, 27
- password_char, 11
- Popup, 36
- PopupCancel, 36
- PopupError, 36
- PopupGetText, 35
- PopupOKCancel, 36
- PopupTimed, 36
- PopupYesNo, 36
- preview_all_look_and_feel_themes, 40
- ProgressBar, 21
- Radio, 15
- read, 2
- relief, 6, 7
- size, 38, 39
- Slider, 18
- Spin, 17
- Tab, 26
- TabGroup, 26
- Table, 22
- Text, 3, 6
- text_color, 6
- theme, 3
- theme_list, 40
- Tkinter, 45
- TKroot, 45
- Tree, 23
- TreeData, 23
- Update, 5, 11, 18
- VerticalSeparator, 21
- Window, 1, 45
- winfo_children, 45
- アコーディオン形式, 27
- ウィジェット, 1
- ウィジェットへの名前の付与, 4
- 既存のウィジェットへのアクセス, 5
- 区切り線, 21
- ショートカット, 32
- スピナー, 17
- スライダ, 18
- タイトルバーの有無, 41
- タブ, 26
- チェックボックス, 15
- テキストフィールド, 9
- テキスト編集エリア, 12
- デザインテーマの一覧表示, 40
- 背景色, 6
- パスワード入力フィールド, 11, 35
- フォント名, 6

文字の色, 6
文字の入力, 9
余白, 7
ラジオボタン, 15
リサイズ, 38
リストボックス, 16
枠, 7

PySimpleGUI 入門 基本的な使用方法

著者：中村勝則

発行：2022 年 6 月 10 日

テキストの最新版と更新情報

本書の最新版と更新情報を，プログラミングに関する情報コミュニティ Qiita で配信しています。

→ <https://qiita.com/KatsunoriNakamura/items/376da645e52f7ef7f9ef>



上記 URL の QR コード

本書はフリーソフトウェアです，著作権は保持していますが，印刷と再配布は自由にさせていただいて結構です。（内容を改変せずをお願いします） 内容に関して不備な点がありましたら，是非ご連絡ください。ご意見，ご要望も受け付けています。

● 連絡先

nkatsu2012@gmail.com

中村勝則