

はじめに

本書は、Python で GUI 付きデスクトップアプリ開発の基礎と実践力を身に着けるための世界一やさしい入門書です。Python の入門書を読み終えたばかりの方でも、コード解説を読み、実際に手を動かして作りながら GUI アプリ開発を覚えられるように出来る限り多くの画像とコード解説を加えています。

本書を読み終えた後に実務で活かすことを目的としていますので、コーディング量は一般的な入門書よりも多めです。作りながら実務で活かせる GUI 付きアプリ制作をおぼられるようにしています。入門者の方は、とにかくコードを書いて、動かして、エラーが発生した場合にはデバッグ機能を使い自力で解決する事でコードへの理解が深まり、実践力が養われます。

実践編では、実務でニーズが高い Excel と連携させた実践的な GUI 付きデスクトップアプリを開発します。各種機能の関数化やファイルのモジュール化などコードの再利用性も考慮した実践的なアプリ制作の一連の流れを学びます。実践編を終える頃には、明日から実務で活かせる GUI 付きアプリ開発の実践力が身に付いている事でしょう。

本書が皆様の GUI 付きデスクトップアプリ開発のきっかけになれば、著者として望外の喜びです。

本書について

本書は、以下のような読者を対象としています。Python の入門書を読み終えた方、もしくは Python を全く学んだ事がない方でも、他のプログラミング言語を使った経験があれば、Python の入門書を片手に読み進められる内容となっています。本書を読み進めて、わからない部分だけを入門書で確認すればよいでしょう。実践編を終える頃には、入門書の基本的な文法が実践を通じて身についている事でしょう。

- ・ Python の入門書を読み終えて、次は GUI 付きデスクトップアプリを作ってみたい方
- ・ デスクトップアプリで業務の自動化を実現したいが、何から初めて良いか分からない方
- ・ GUI ライブラリ選びに迷っている方、難しそうというイメージで一步踏み出せない方

本書の概要

GUI プログラミング初心者の学習コストを最小にするため、Python の GUI ライブラリで最も簡単な PySimpleGUI を使用します。PySimpleGUI は Python 標準の GUI ライブラリの Tkinter がベースとなっており、特別に凝った GUI を出なければほとんどの GUI 機能を実装出来ます。また、実際の現場で使えるようにファイル操作や Excel との連携についても取り上げています。

プログラミングを習得するには、実際にコーディングしてプログラムを動かしながら学ぶことが重要です。Visual Studio Code 上で実際にプログラムを動かしながら学べるように、サンプルコードを多数掲載しています。実際にコーディングしながら学ぶ事をお勧めします。サンプルコードは、[こちらのサイト](#)からダウンロード出来るようにしています。ご自身の PC 上でプログラムを実際に動かしながら使い方の理解を深めて下さい。サンプルコードは、書籍を購入いただいた方への特典となりますのでダウンロードにはパスワードが必要です。パスワードは、本書の一番後ろに記載しております。

本書の構成

1 章

GUI 付きデスクトップアプリを開発するための環境構築をします。プログラミングを始める際の最初の障壁が開発環境の構築です。Python のインストールから、その他のライブラリを簡単にインストールするためのツール、そしてアプリ開発の効率を上げてくれるツールの使い方などを解説します。

2 章

プログラムの基本となるファイル操作の基本を解説します。GUI アプリに限らず、シンプルなプログラムを除いてファイル操作が必ず必要となります。プログラム初学者の方がしばしば混乱してしまう相対パス、絶対パスの違いからファイルの作成、更新、削除などを実際に Visual Studio Code 上で手を動かしながら学びます。

3 章

Python で Excel を操作する方法について解説します。Excel は、業務上のあらゆる場面で使われており、Excel 形式のデータと連携するニーズはかなり高いと思います。お使いの PC に Excel がインストールされていなくても

構いません。本書では、Excel のインストールが不要な `openpyxl` というライブラリを使用して、Excel 形式のデータを操作します。

4 章

Python の GUI ライブラリである `PySimpleGUI` の使い方について詳しく解説します。最初に `PySimpleGUI` の基本的な使い方や GUI アプリ開発の流れを解説します。次に、各種ウィジェットの使い方を実際に簡単な GUI アプリを作成しながら学びます。本書で取り扱うウィジェットは、一般的によく使われるウィジェットに厳選しています。各ウィジェットの主要プロパティも載せておりますので、必要な時に再度確認し易くなっています。

5 章

本書の仕上げとして、実践的な GUI 付きデスクトップアプリの開発を詳しく解説します。2 章、3 章、4 章で学んだ内容を組み合わせて実践的なアプリを開発していきます。それぞれの知識をバラバラに習得しても、組み合わせられなければ実際に現場で使えるアプリを開発する事は出来ません。この章を終える頃には、現場で使える GUI 付きデスクトップアプリを自身で開発するための基礎が身に付きます。

対象 OS

著者が動作を確認している環境は次の通りです。

- ・ Windows10 Home, Pro

macOS Monterey では、GUI ウィンドウのメニューバーが動作しないため Windows10 を前提として解説します。

4.GUI アプリの基本

この章では、最初に Python の GUI ライブラリについて紹介します。その後、本書で使用する PySimpleGUI という Tkinter ベースの GUI ライブラリの使い方を簡単な GUI アプリを作りながら説明を加えていきます。

PySimpleGUI を使えば、とても簡単に少ないコードで GUI アプリを作成出来ますので是非活用下さい。本書では、PySimpleGUI の全てを紹介する事は出来ませんが[公式ページ](#)が非常に充実していますので、詳細についてはそちらをご確認下さい。

4.1 Python の GUI ライブラリー

Python には様々な GUI ライブラリが提供されています。GUI とは、Graphical User Interface の略称でありユーザーが視覚的に操作できるインタフェースのことです。一般的なアプリは、全て GUI で操作するのが基本となっています。

表 4.1 に Python の主要な GUI ライブラリとその特徴をまとめました。

まず、Tkinter ですが Python 標準搭載 GUI ライブラリですので、新たにインストールする必要がないという特徴があります。また、機能はシンプルであり使い易い部分もあるのですが、機能が物足りない人は別のライブラリを使うことになります。

次は、PyQt です。こちらは、デザインがモダンであり Android や iOS にも対応しています。ただし、GPL ライセンスですので商用利用に制限がありロイヤリティーも発生します。

次に Kivy ですが、こちらはゲームに強みがあり、Android や iOS にも対応しています。ライセンスは、MIT ライセンスですので著作権とライセンスの表示をすれば商用利用可能です。

最後は、WxPython です。こちらは、機能が充実し安定しているため手の込んだ GUI を作成する際に使用されます。ライセンスは LGPL で、配布に制限がありますが気をつければ商業利用もできます。

表 4.1 Python の GUI ライブラリー一覧

ライブラリ名	特徴
Tkinter	Pythonに標準搭載。機能はシンプル。
PyQt	デザインがモダン。Android、iOS対応 [GPLライセンス]。
Kivy	ゲームに強みがあり Android、iOS対応 [MITライセンス]。
wxPython	機能が充実し安定。手の込んだGUIに対応[LGPLライセンス]。

図 4.1.は、Google トレンドで過去 3 年間（2019 年 12 月から）の各ライブラリの人気動向を調べた結果になります。

Tkinter が Python 標準搭載ということもあり、他の 3 つに比べて 3 倍以上の人気があります。人気があるという事は、それだけ多くの人に支持され、関連する参考書やネット上の情報も豊富にあるということです。本書で使用する PySimpleGUI も Tkinter をベースとした GUI ライブラリーです。



図 4.1. Python の GUI ライブラリー人気動向

4.2 PySimpleGUI の基本

4.2.1 PySimpleGUI について

本書で使用する PySimple GUI について紹介します。先ほどの説明で、PySimpleGUI が紹介されなかった事に違和感を感じた方もいらっしゃるかもしれません。それは、PySimpleGUI というのは Tkinter や Qt、WxPython のラッパーライブラリーのためです。ラッパーライブラリーとは、簡単に言いますと Tkinter や Qt などの GUI ライブラリーの中身は同じで、インタフェースの部分だけを変えて扱いやすくしたライブラリーのことです。LGPL ライセンスになります。

PySimpleGUI は、あらゆるレベルの Python プログラマーが GUI を作成出来るように作成されています。画像は、PySimpleGUI 公式ページの readme 冒頭に記載されているライブラリの説明です。PySimpleGUI の[公式サイト](https://pysimplegui.readthedocs.io/en/latest/)は英語となりますが、日本人の有志の方が[日本語版の readme](#)を作成されています。

Tkinter よりも更にシンプルかつ短い記述で GUI を作成出来るようになっていきますので、GUI 付きデスクトップアプリ開発の入門者には最適のライブラリーとなります。

機能についても、Tkinter がベースとなっておりますので非常に凝った GUI でなければ、大部分の方が求める GUI 付きデスクトップアプリを PySimpleGUI で作成可能です。



図 4.2.1. PySimpleGUI Readme 冒頭

PySimpleGUI 公式 URL : <https://pysimplegui.readthedocs.io/en/latest/>

PySimpleGUI は OSS として開発されており、[PySimpleGUI の Github](#) ではスポンサーを募集しています。OSS 開発を続けるためにも資金が必要です。\$ 2 /月から支援する事が出来ますので、もし PySimpleGUI が気に入った場合は支援を検討してみても如何でしょうか？ ちなみに、筆者達もスポンサーとして PySimpleGUI を応援しております。

4.2.2 PySimpleGUI の基本構成

PySimpleGUI の基本として、GUI プログラムの構成と各要素について図 4.2.2 を用いて説明します。

図 4.2.2 左側に示す、非常にシンプルな GUI ウィンドウで説明します。こちらに示す GUI のウィンドウすべてをウィンドウと呼びます。このウィンドウの上段をタイトルと呼びます。そして頭中では緑で囲んであるウィンドウ内のメインとなるエリアを呼びます。レイアウト内のそれぞれの部品、例えばボタンや入力などをウィジェットもしくは要素と呼びます。本講座では一般的な名称であるウィジェットと呼んでいくこととします。

図 4.2.2 右側は、このウィンドウを表示させるためのソースコードコードです。それぞれを詳しく説明していきます。

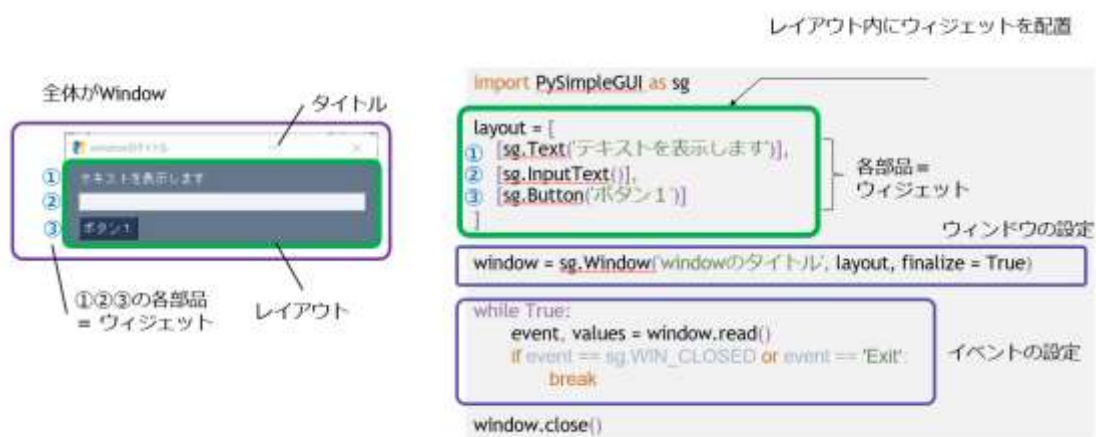


図 4.2.2. GUI プログラムの構成と各要素の説明

PySimpleGUI のインポート

まずは、プログラムに PySimpleGUI をインポートします。PySimpleGUI とそのまま記述するのは長いので、`sg` として別名インポートします。

```
import PySimpleGUI as sg
```

レイアウトの設定

レイアウトは、`layout = [[ボタンウィジェット], [テキストウィジェット]]`として、角カッコ内にボタンやテキストなどのウィジェットを角カッコで配置して設定します。`layout` は任意の名前を付けることが出来ます。`main_layout` や `sub_layout` などわかり易いレイアウト名を付けると良いでしょう。

```
layout = [  
    [sg.Text('テキストを表示します')],  
    [sg.InputText()],  
    [sg.Button('ボタン1')]  
]
```

ウィジェットの設定

ウィジェットは、`sg.ウィジェット名('ウィジェットの詳細設定')`としてカッコ内にウィジェットの詳細（プロパティ）を設定します。ここでは、テキスト、インプットそしてボタンの3つのウィジェットを設置しています。PySimpleGUI で使用可能なウィジェットについては、後ほど詳しく紹介していきます。

ウィンドウの設定

GUI ウィンドウは、`window = sg.Window('window タイトル', layout, finalize=True)`としてウィンドウのタイトルとレイアウトを渡して設定します。もし、レイアウトを `main_layout` として設定した場合は、`sg.Window('window のタイトル', main_layout, finalize=True)`としてウィンドウを作成します。最後の `finalize=True` ですが、5 章で開発する複数のウィンドウを関数化して使用するような場合に必要となります。これを忘れてしまうと、ウィンドウがクラッシュして正しく表示されません。クラッシュを避けるために、単一のウィンドウ設定の時にも省略せずに設定しておいた方が無難です。

```
window = sg.Window('window のタイトル', layout, finalize=True)
```

イベントの設定

イベントは、`while True:`のブロック内に設定していきます。まずは、`event, values = window.read()`で作成した GUI ウィンドウを読み込み、各種のイベント（ボタンクリックなど）の入力待ち状態にします。変数 `event` で、発生したイベントのキーを受け取ります。また、変数 `values` には、ウィンドウ内の各ウィジェッ

トの値を受け取ります。例えば、インプットエリアに入力されている値などです。if 文を使って、ボタン A が押された場合の処理を設定していきます。詳細については、次のチャプター以降で説明していきます。**if event == sg.WIN_CLOSED or event == 'Exit':**は、ウィンドウが閉じられた時にプログラムを終了するためのコードです。定型文として使用します。

```
while True:
    event, values = window.read() # イベントの入力を待つ

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

ウィンドウのクローズ

最後にウィンドウを **window.close()** としてクローズすれば、GUI ウィンドウを作成できます。

4.2.3 PySimpleGUI を用いたアプリ開発の流れ

PySimpleGUI を用いた GUI 付きデスクトップアプリ開発の流れを GUI プログラムを作成しながら説明します。まずは、GUI アプリを保存するフォルダを作成します。ご自分の PC に GUIPractice フォルダを作成して下さい。本書では、**C:\¥Users¥ns-systems¥PythonProjects¥GUIPractice** にフォルダを作成しました。この中に Python のプログラムファイルを作成していきます。

GUI アプリ開発の流れ

開発の流れは、①レイアウトの設定→②ウィンドウの設定→③イベントの設定→④ウィンドウのクローズとなります。実際には、ウィジェットやイベント追加する度に①と③を交互に追加・変更していく事になります。また、ウィンドウを複数作成してボタンで切り替える事も可能です。

最初の GUI アプリ作成

まずは、仮想環境 **GUIenv** で GUI アプリ開発のための環境が正しく環境が構築されているのかを確認するため、画面に **Popup** を表示するだけの簡単な GUI アプリを作成します。GUIPractice フォルダ内に **FirstGUI.py** を作成し、以下のようにコーディングして下さい。

```
import PySimpleGUI as sg

sg.popup('最初の GUI アプリ')
```

ファイルが作成出来ましたら、**Visual Studio Code** の実行ボタンで **FirstGUI.py** を実行します。画像のようなポップアップが表示されたら、仮想環境 **GUIenv** が正しく構築され、**PySimpleGUI** で **GUI** 付きデスクトップアプリを開発する準備が整っています。

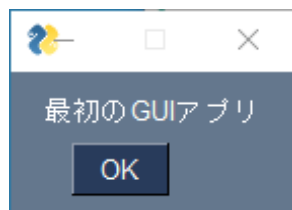


図 4.2.3.1 最初のポップアップ

先ほど紹介した GUI アプリの開発の流れの順に、これから GUI 付きデスクトップアプリを作成していきます。まずは、出来上がりのイメージを図 4.2.3.2 で掴んで下さい。FirstGUI.py にコードを追記していきます。最終的なコードは以下の通りです。



図 4.2.3.2. GUI アプリの開発の流れ

```

1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Text(text="表示させたい文字列", text_color='ff0000')],
5     [sg.Input(key='-Input-')],
6     [sg.Button('ボタン', key='-Btn-')]
7 ]
8
9 window = sg.Window('FirstGUI', layout, size=(300, 150), finalize=True)
10
11 while True:
12     event, values = window.read() # イベントの入力を待つ
13
14     if event == '-Btn-':
15         message = values['-Input-']
16         sg.popup(message)
17         break
18     if event == sg.WIN_CLOSED or event == 'Exit':
19         break
20
21 window.close()

```

①レイアウトの設定

まずは、レイアウトを設定します。図 4.2.3.2 のように、テキストウィジェット、インプットウィジェット、ボタンウィジェットの3つを設置します。

各ウィジェットの説明

テキストウィジェットは、`sg.Text(text="表示させたい文字列", text_color='16進数カラーコード')`のように設定します。プロパティ`text_color`に16進数カラーコードを指定して色を変更します。16進数カラーコードは、#と6桁の数字で色を表します。HTMLやCSSでも色を指定するときに使用しますので、馴染み深い方も多いと思います。テキストのフォント、サイズ、位置などの指定方法については、後ほど説明します。

ユーザーの入力を受け付けるインプットウィジェットは、`sg.Input(key='-Input-')`のように設定します。こちらでも、プロパティを指定してフォント、サイズ、デフォルトで表示させておく文字列などを指定出来ます。

ボタンウィジェットは、`sg.Button('表示させたい文字列', key='-Btn-')`として設定します。ボタンに表示する文字列はシングルクォーテーションもしくはダブルクォーテーションで括ります。今回は、インプットウィジェットに入力した値が表示されるようにします。ボタンの色変更や画像追加などについては、各種ウィジェットの章で説明します。

キーの説明

各ウィジェットを識別するためのキーを割り当てる事が出来ます。全てのウィジェットに割り当てる必要はありませんが、イベントの起点もしくはイベント後に変更を加えたいウィジェットにはキーを割り当てましょう。キーを割り当てないと、イベント処理の設定が難しくなってしまいます。また、本書ではウィジェットのキーは **'-Input-'** や **'-Btn-'** のようにキー名をハイフンで括り、最初の文字列のみを大文字で命名していきます。

```
layout = [
    [sg.Text(text="表示させたい文字列", text_color='#ff0000')],
    [sg.Input(key='-Input-')],
    [sg.Button('ボタン', key='-Btn-')]
]
```

②ウィンドウの設定

ウィンドウは、**window = sg.Window('FirstGUI', layout, size=(300, 150), finalize=True)** として設定します。これでタイトルには **FirstGUI** と表示され、①で設定したレイアウトが渡されます。また、**size=(幅 , 高さ)** としてウィンドウサイズをピクセルで指定出来ます。サイズを指定しない場合は、自動でサイズが決定されます。

```
window = sg.Window('FirstGUI', layout, size=(300, 150), finalize=True)
```

③イベントの設定

イベントは、**while True:** のイベントブロック内に設定していきます。繰り返しになりますが、**event, values = window.read()** で作成した GUI ウィンドウを読み込み、各種イベントの入力待ち状態にします。今回は、ボタンが押された場合にポップアップが表示されるイベントを設定します。今回のイベント処理コードは以下となります。

```
while True:
    event, values = window.read() # イベントの入力を待つ

    if event == '-Btn-':
        message = values['-Input-']
        sg.popup(message)
        break
    if event == sg.WIN_CLOSED or event == 'Exit':
        break
```

まずは、イベントが入力された時（ボタンがクリックされた時）の処理をコーディングします。ボタンにはキー **'-Btn-'** を割り当てていましたので、**if event == '-Btn-':** として、ボタンがクリックされた時のイベントを設定します。

次の `message = values['-Input-']` ですが、変数 `message` にテキストウィジェットの値を格納しています。 `event, values = window.read()` でウィンドウ内の各ウィジェットの値を `values` に格納していますので、テキストウィジェットのキー `'-Input-'` を指定すれば値を参照することが出来ます。

ポップアップは、 `sg.popup(message)` として表示させる事が出来ます。表示させたい文字列は、直接文字列を指定する事も出来ますし、今回のように変数を指定する事も出来ます。その後、 `break` で `while` ループを中断して、イベント入力を待っている状態を解除します（つまりアプリを終了します）。

`if event == sg.WIN_CLOSED or event == 'Exit':` についてですが、これはウィンドウを `x` ボタンで閉じた際の処理をコーディングしています。イベントとしてウィンドウが閉じられた場合 `break` (アプリを終了) 処理を追加しています。この処理を入れずにウィンドウを `x` ボタンで閉じた場合は、図 4.2.3.3 のようなエラーメッセージが表示されます。PySimpleGUI で GUI アプリを作成するときの定型文として使用していきます。

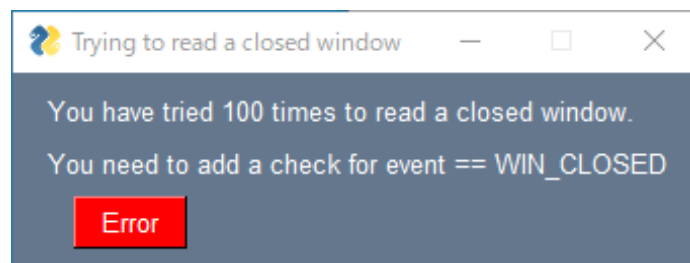


図 4.2.3.3. エラーメッセージ

④ ウィンドウのクローズ

最後は、 `window.close()` としてアプリを終了します。プログラムは上から下に逐次実行されますので、イベントループを `break` で中断した後に `window.close()` へ到達し、GUI ウィンドウを閉じます（アプリを終了します）。④については、お決まりの部分ですので、これ以降のコードでは特に説明を加えない事とします。

GUI 付きデスクトップアプリ開発のおおまかな流れは掴めましたでしょうか。実際のアプリ開発では、特に①と③が重要となります。機能を追加・更新する場合は①と③を行ったり来たりしながらコーディングしていく事になります。

4.2.4 PySimpleGUI のイベント処理

PySimpleGUI のイベント処理について、もう少し詳しく見ていきましょう。イベントのトリガーは大きく2つあります。1つ目は、GUI アプリで最も一般的なボタンをクリックした際のイベント処理です。2つ目は、ウィジェットに変更が加えられた際のイベント処理です。また、イベント処理内でウィジェットのプロパティを更新する事も出来ますので、そちらについても解説します。

ボタンクリックイベント

ボタンクリックは、最も一般的なイベントのトリガーです。ボタンがクリックされた時、イベントブロック内で **event, values = window.read()** の **event** へボタンもしくはそれに準ずるウィジェットのキーが渡されます。キーを設定していない場合は、**button_text** の値が渡されます。先ほど作成した **FirstGUI.py** の場合を考えます。キーを **key='-Btn-'** と設定した場合は、そのボタンをクリックすると **event** に **-Btn-** が格納されます。イベントループ内の if 文で **if event == '-Btn-':** として、イベント処理を開始する事が出来ます。先ほどは、この if 文内にポップアップが表示される処理をコーディングしました。このようにして、ボタンクリックをトリガーとして各種のイベントを作成可能です。

ウィジェット変更時のイベント

もう1つのイベントトリガーを紹介します。ウィンドウ内に配置しているウィジェットに何らかの変更が加えられた場合、イベントを発生させたいとします。例えば、インプットに文字を入力した時に図 4.2.1.のようにポップアップが表示されるイベントを作成してみましょう。**widgetchange.py** のコードは以下の通りです。

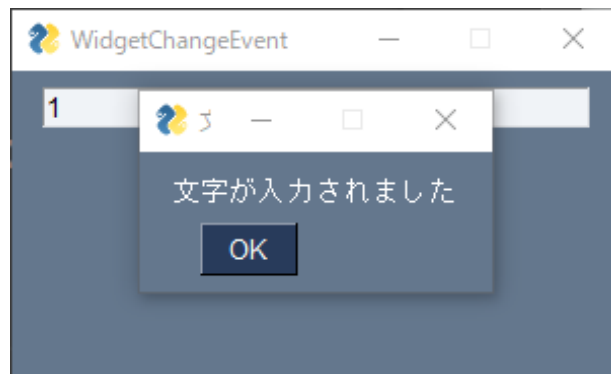


図 4.2.1.インプットウィジェットのイベント

```

1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Input(key='-Input-',enable_events=True)],
5 ]
6
7 window = sg.Window("WidgetChangeEvent", layout, size=(300, 150), finalize=True)
8
9 while True:
10     event, values = window.read() # イベントの入力を待つ
11
12     if event == '-Input-':
13         sg.popup('文字が入力されました')
14         break
15     if event == sg.WIN_CLOSED or event == 'Exit':
16         break
17
18 window.close()

```

レイアウトの設定

レイアウトに配置しているウィジェットはインプットウィジェットのみです。インプットウィジェットには、2つのプロパティを指定しています。1つ目は、キーで **key='-Input-'** として設定しています。2つ目は、**enable_events** プロパティです。このプロパティを **True** に設定すると、ウィジェットに変更が加えられるたびにイベントを発生させる事が出来るようになります。

イベントの設定

イベントは **while True:** で括られたイベントブロック内にコーディングします。先ほど、インプットウィジェットに **enable_events = True** を設定しましたので、インプットの値に何か変更があった場合（入力された場合）にイベントが発生します。インプットに入力をする、**event** にキー **-Input-** が格納されます。それを、if 文で捕捉してイベント処理をコーディングします。ここでは、ポップアップが表示されるようにしています。

以上のようにウィジェットのプロパティに **enable_events = True** を設定すると、そのウィジェットの値が変更される度にイベントを発生させる事が出来ます。次に紹介するウィジェットの更新と組み合わせると、インプットに文字が入力された場合にボタンを押すことが出来るようにすることも出来ます。例えば、申し込みフォームなどで必須事項が入力されないと次へボタンが押せないようにされているフォームをよく見かけます。PySimpleGUI でも、応用次第でそのような処理を作成可能です。

ウィジェットのプロパティ更新

ウィジェットのプロパティは、イベント処理内でも更新可能です。更新の方法は、ウィンドウ内で特定のウィジェットをキーで特定し、そのウィジェットのプロパティを新しい値で更新します。言葉だけでは分かり難いので、簡単な GUI アプリを作成して解説します。アプリは、図 4.2.4.2 のようなボタンを押すことでテキストの色を変更するという非常にシンプルなものです。コードは以下の通りです。

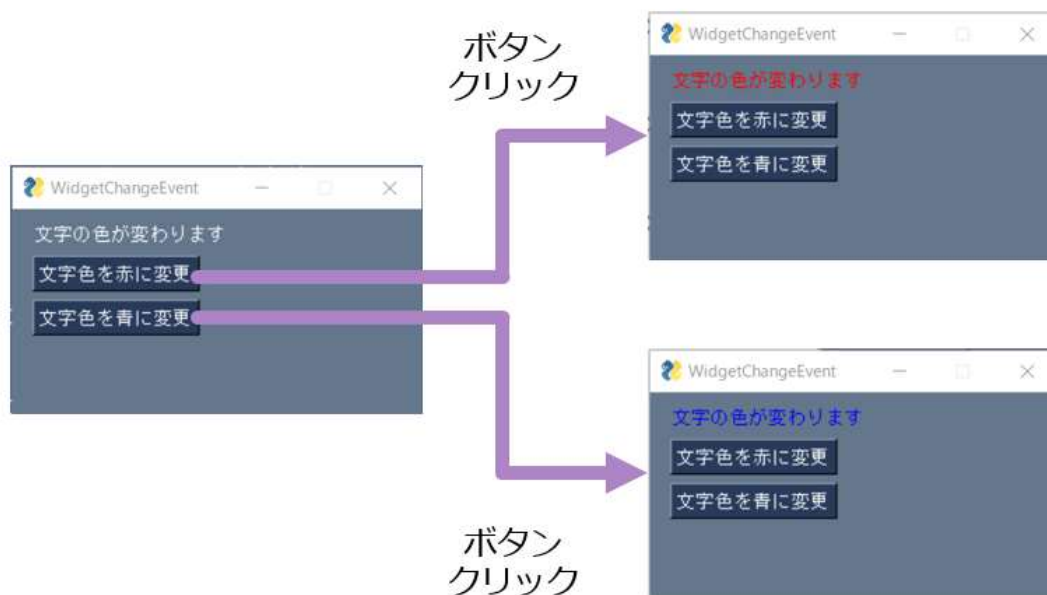


図 4.2.4.2 ウィジェットのプロパティ更新

```

1  import PySimpleGUI as sg
2
3  layout = [
4      [sg.Text('文字の色が変わります', key='-Txt-')],
5      [sg.Btn('文字色を赤に変更', key='-Btn_Red-')],
6      [sg.Btn('文字色を青に変更', key='-Btn_Blue-')],
7  ]
8
9  window = sg.Window('WidgetChangeEvent', layout, size=(300, 150), finalize=True)
10
11  while True:
12      event, values = window.read() # イベントの入力を待つ
13
14      if event == '-Btn_Red-':
15          window['-Txt-'].Update(text_color='Red')
16
17      if event == '-Btn_Blue-':
18          window['-Txt-'].Update(text_color='Blue')
19
20      if event == sg.WIN_CLOSED or event == 'Exit':
21          break
22
23  window.close()

```

レイアウトの設定

レイアウトには、テキストウィジェットと2つのボタンウィジェットを配置しています。テキストウィジェットには、テキストエリアに表示させる文字列とキー`key='-Txt-'`を指定しています。2つのボタンウィジェットには、ボタンに表示させたい文字列とそれぞれを識別するキーを `key='-Btn_Red-'`、`key='-Btn_Blue-'`としてそれぞれ設定しています。

イベントの設定

ここでは、ボタンクリックをイベントトリガーとしてボタンがクリックされた際の処理をそれぞれコーディングしています。14行目は、ボタン`-Btn_Red-`がクリックされた際のイベントです。ここでは、テキストウィジェットの色を赤に変更させます。まずは、`window['-Txt-']`としてウィンドウ内で対象のウィジェットをキーで特定します。そして、`Update()`メソッドでプロパティの値を更新します。更新するプロパティは `text_color` で、その値（色）を赤にしますので、`Update(text_color='Red')`としてテキストウィジェットの色を赤色に更新します。ボタン`-Btn_Blue-`がクリックされた際のイベントも基本は全く同じですので、説明は割愛いたします。

以上のようにウィジェットのプロパティを更新する事が出来ます。5章では、ウィジェットのプロパティ更新の実例を紹介していますので5章も参照しながら使い方をマスターしましょう。

4.2.5 PySimpleGUI のウィジェット共通設定

各種ウィジェットの詳細な使い方説明に移る前に、複数のウィジェットで共通する設定を説明していきます。GUIPractice フォルダーに **CommonProperties1.py** と **CommonProperties2.py** を作成し、その中にコーディングしていきます。

色の指定

Tkinter ベースの PySimpleGUI では、色の指定方法が複数あります。選択肢が多過ぎても混乱してしまいますので、本書では以下の標準的な色名もしくは 16 進数カラーコードで色を指定します。[こちら](#)に Tkinter で使用出来る色名が掲載されておりますが、環境にも依存する場合がありますので、標準色以外は 16 進数カラーコードの使用を推奨します。

標準的な色名 : 'white', 'black', 'red', 'green', 'blue', 'cyan', 'yellow', 'magenta'

テキストカラーの設定

すでに説明しておりますが、テキストの色はプロパティ **text_color** に 16 進数カラーコードもしくは標準色を指定して設定します。テキストの背景色を緑（16 進数カラーコード : #008000）に指定する場合は、**sg.Text('表示したい文字列', text_color = '#008000')**として設定します。標準色で指定する場合は、**sg.Text('表示したい文字列', text_color = 'Green')**とします。

GUI ウィンドウに 4 つのテキストエリアを設置し、それぞれ異なるカラーコードを設定してみましょう。**CommonProperties1.py** に以下のようにコーディングします。表示されるウィンドウは、図 4.2.5.1 となります。

```

1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Text('オリジナルテキスト')],
5     [sg.Text('テキスト 1', text_color='#008000')],
6     [sg.Text('テキスト 2', text_color='white')],
7     [sg.Text('テキスト 3', text_color='#ff0000')],
8 ]
9
10 window = sg.Window('テキストカラーの設定', layout, finalize=True)
11
12 while True:
13     event, values = window.read() # イベントの入力を待つ
14
15     if event == sg.WIN_CLOSED or event == 'Exit':
16         break
17
18 window.close()

```

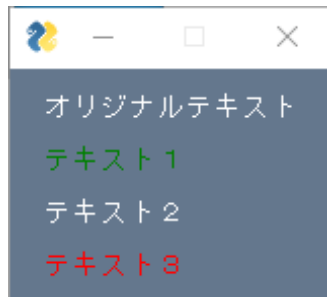


図 4.2.5.1. テキストの表示

フォントの設定

テキストのフォントは、`sg.Text(text = "", font = ('フォント名', フォントサイズ))` で指定できます。フォントは、Tkinter のフォントファミリーを使用出来ます。使用可能なフォントを調べるには、以下のコードを実行して下さい。

Code4.2.5.2. tkinterfont.py

```
1 import tkinter
2 import tkinter.font
3
4 tkinter.Tk()
5 print(tkinter.font.families())
```

コマンドプロンプトにフォント一覧が表示されますので、好みのフォントを指定します。Windows10 では、以下のようなフォントが使用可能です。

System, @system, terminal, @terminal, fixedsys, @fixedsys, modern, roman, script, courier, @ms serif, @ms sans serif, 'Small Fonts', 'Marlett', 'Arial', 'Arabic Transparent', 'Arial Baltic', 'Arial CE', 'Arial CYR', 'Arial Greek', 'Arial TUR', 'Arial Black', 'Bahnschrift Light', 'Bahnschrift SemiLight', 'Bahnschrift', 'Bahnschrift SemiBold', 'Bahnschrift Light SemiCondensed', 'Bahnschrift SemiLight SemiConde', 'Bahnschrift SemiCondensed', 'Bahnschrift SemiBold SemiCondensed', 'Bahnschrift Light Condensed', 'Bahnschrift SemiLight Condensed', 'Bahnschrift Condensed', 'Bahnschrift SemiBold Condensed', 'Calibri', 'Calibri Light', 'Cambria', 'Cambria Math', 'Candara', 'Candara Light', 'Comic Sans MS', 'Consolas', 'Constantia', 'Corbel', 'Corbel Light', 'Courier New', 'Courier New Baltic', 'Courier New CE', 'Courier New CYR', 'Courier New Greek', 'Courier New TUR', 'Ebrima', 'Franklin Gothic Medium', 'Gabriola', 'Gadugi', 'Georgia', 'Impact', 'Ink Free', 'Javanese Text', 'Leelawadee UI', 'Leelawadee UI Semilight', 'Lucida Console', 'Lucida Sans Unicode', 'Malgun Gothic', '@Malgun Gothic', 'Malgun Gothic Semilight', '@Malgun Gothic Semilight', 'Microsoft Himalaya', 'Microsoft JhengHei', '@Microsoft JhengHei', 'Microsoft JhengHei UI', '@Microsoft JhengHei UI', 'Microsoft JhengHei UI Light', '@Microsoft JhengHei UI Light', 'Microsoft New Tai Lue', 'Microsoft PhagsPa', 'Microsoft Sans Serif', 'Microsoft Tai Le', 'Microsoft YaHei', '@Microsoft YaHei', 'Microsoft YaHei UI', '@Microsoft YaHei UI', 'Microsoft YaHei Light', '@Microsoft YaHei Light', 'Microsoft YaHei UI Light', '@Microsoft YaHei UI Light', 'Microsoft Yi Baiti', 'MingLiU-ExtB', '@MingLiU-ExtB', 'PMingLiU-ExtB', '@PMingLiU-ExtB', 'MingLiU_HKSCS-ExtB', '@MingLiU_HKSCS-ExtB', 'Mongolian Baiti', 'MS Gothic', '@MS Gothic', 'MS UI Gothic', '@MS UI Gothic', 'MS P Gothic', '@MS P Gothic', 'MV Boli', 'Myanmar Text', 'Nirmala UI', 'Nirmala UI Semilight', 'Palatino Linotype', 'Segoe MDL2 Assets', 'Segoe Print', 'Segoe Script', 'Segoe UI', 'Segoe UI Black', 'Segoe UI Emoji', 'Segoe UI Historic', 'Segoe UI Light', 'Segoe UI Semibold', 'Segoe UI Semilight', 'Segoe UI Symbol', 'SimSun', '@SimSun', 'NSimSun', '@NSimSun', 'SimSun-ExtB', '@SimSun-ExtB', 'Sitka Small', 'Sitka Text', 'Sitka Subheading', 'Sitka Heading', 'Sitka Display', 'Sitka Banner', 'Sylfaen', 'Symbol', 'Tahoma', 'Times New Roman', 'Times New Roman Baltic', 'Times New Roman CE', 'Times New Roman CYR', 'Times New Roman Greek', 'Times New Roman TUR', 'Trebuchet MS', 'Verdana', 'Webdings', 'Wingdings', '游ゴシック', '@游ゴシック', 'Yu Gothic UI', '@Yu Gothic UI', 'Yu Gothic UI Semibold', '@Yu Gothic UI Semibold', '游ゴシック Light', '@游ゴシック Light', 'Yu Gothic UI Light', '@Yu Gothic UI Light', '游ゴシック Medium', '@游ゴシック Medium', 'Yu Gothic UI Semilight', '@Yu Gothic UI Semilight', 'HoloLens MDL2 Assets', 'BIZ UDゴシック', '@BIZ UDゴシック', 'BIZ UDPゴシック', '@BIZ UDPゴシック', 'BIZ UD明朝 Medium', '@BIZ UD明朝 Medium', 'BIZ UDP明朝 Medium', '@BIZ UDP明朝 Medium', 'メイリオ', '@メイリオ', 'Meiryo UI', '@Meiryo UI', 'MS 明朝', '@MS 明朝', 'MS P 明朝', '@MS P 明朝', 'UD デジタル 教科書体 N-B', '@UD デジタル 教科書体 N-B', 'UD デジタル 教科書体 NP-B', '@UD デジタル 教科書体 NP-B', 'UD デジタル 教科書体 NK-B', '@UD デジタル 教科書体 NK-B', 'UD デジタル 教科書体 N-R', '@UD デジタル 教科書体 N-R', 'UD デジタル 教科書体 NP-R', '@UD デジタル 教科書体 NP-R', 'UD デジタル 教科書体 NK-R', '@UD デジタル 教科書体 NK-R', '游明朝', '@游明朝', '游明朝 DemiBold', '@游明朝 DemiBold', '游明朝 Light', '@游明朝 Light', 'Agency FB', 'Algerian', 'Book Antiqua', 'Arial Narrow', 'Arial Rounded MT Bold', 'Baskerville Old Face', 'Bauhaus 93', 'Bell MT', 'Bernard MT Condensed', 'Bodoni MT', 'Bodoni MT Black', 'Bodoni MT Condensed', 'Bodoni MT Poster Compressed', 'Bookman Old Style', 'Bradley Hand ITC', 'Britannic Bold', 'Berlin Sans FB', 'Berlin Sans FB Demi', 'Broadway', 'Brush Script MT', 'Bookshelf Symbol 7', 'Californian FB', 'Calisto MT', 'Castellar', 'Century Schoolbook', 'Centaur', 'Century', 'Chiller', 'Colonna MT', 'Cooper Black', 'Copperplate Gothic Bold', 'Copperplate Gothic Light', 'Curlz MT', 'Dubai', 'Dubai Light', 'Dubai Medium', 'Elephant', 'Engravers MT', 'Eras Bold ITC', 'Eras Demi ITC', 'Eras Light ITC', 'Eras Medium ITC', 'Felix Titling', 'Forte', 'Franklin Gothic Book', 'Franklin Gothic Demi', 'Franklin Gothic Demi Cond', 'Franklin Gothic Heavy', 'Franklin Gothic Medium Cond', 'Freestyle Script', 'French Script MT', 'Footlight MT Light', 'Garamond', 'Gigi', 'Gill Sans MT', 'Gill Sans MT Condensed', 'Gill Sans Ultra Bold Condensed', 'Gill Sans Ultra Bold', 'Gloucester MT Extra Condensed', 'Gill Sans MT Ext Condensed Bold', 'Century Gothic', 'Goudy Old Style', 'Goudy Stout', 'Harlow Solid Italic', 'Harrington', 'Haettenschweiler', 'HG正楷書体-PRO', '@HG正楷書体-PRO', 'HG丸ゴシックM-PRO', '@HG丸ゴシックM-PRO', 'High Tower Text', 'Imprint MT Shadow', 'Informal Roman', 'Blackadder ITC', 'Edwardian Script ITC', 'Kristen ITC', 'Jokerman', 'Juice ITC', 'Kunstler Script', 'Wide Latin', 'Lucida Bright', 'Lucida Calligraphy', 'Leelawadee', 'Lucida Fax', 'Lucida Handwriting', 'Lucida Sans', 'Lucida Sans Typewriter', 'Magneto', 'Maiandra GD', 'Matura MT Script Capitals', 'Mistral', 'Modern No. 20', 'Microsoft Uighur', 'Monotype Corsiva', 'MT Extra', 'Niagara Engraved', 'Niagara Solid', 'OCR A Extended', 'OCRB', 'Old English Text MT', 'Onyx', 'MS Outlook', 'Palace Script MT', 'Papyrus', 'Parchment', 'Perpetua', 'Perpetua Titling MT', 'Playbill', 'Poor Richard', 'Pristina', 'Rage Italic', 'Ravie', 'MS Reference Sans Serif', 'MS Reference Specialty', 'Rockwell Condensed', 'Rockwell', 'Rockwell Extra Bold', 'Script MT Bold', 'Showcard Gothic', 'Snap ITC', 'Stencil', 'Tw Cen MT', 'Tw Cen MT Condensed', 'Tw Cen MT Condensed Extra Bold', 'Tempus Sans ITC', 'Viner Hand ITC', 'Vivaldi', 'Vladimir Script', 'Wingdings 2', 'Wingdings 3', 'HGゴシックE', '@HGゴシックE', 'HGPゴシックE', '@HGPゴシックE', 'HGSゴシックE', '@HGSゴシックE', 'HG行書体', '@HG行書体', 'HGP行書体', '@HGP行書体', 'HGS行書体', '@HGS行書体', 'HG教科書体', '@HG教科書体', 'HGP教科書体', '@HGP教科書体', 'HGS教科書体', '@HGS教科書体', 'HG明朝B', '@HG明朝B', 'HGP明朝B', '@HGP明朝B', 'HGS明朝B', '@HGS明朝B', 'HG明朝E', '@HG明朝E', 'HGP明朝E', '@HGP明朝E', 'HGS明朝E', '@HGS明朝E', 'HG創英角ゴシック', '@HG創英角ゴシック', 'HGP創英角ゴシック', '@HGP創英角ゴシック', 'HGS創英角ゴシック', '@HGS創英角ゴシック', 'HG創英角ゴシックUB', '@HG創英角ゴシックUB', 'HGP創英角ゴシックUB', '@HGP創英角ゴシックUB', 'HGS創英角ゴシックUB', '@HGS創英角ゴシックUB', 'HG創英角ゴシックUB', '@HG創英角ゴシックUB'

図 4.2.5.2. Tkinter のフォント一覧

先ほどのコードに、フォントとフォントサイズを追加すると以下のようになります。コードについては、レイアウトエリアのみを掲載します。

```
layout = [
    [sg.Text('オリジナルテキスト')],
    [sg.Text('テキスト 1', text_color='#008000', font=('游ゴシック', 20))],
    [sg.Text('テキスト 2', text_color='#000000', font=('游ゴシック', 30))],
    [sg.Text('テキスト 3', text_color='#ff0000', font=('游ゴシック', 40))],
]
```

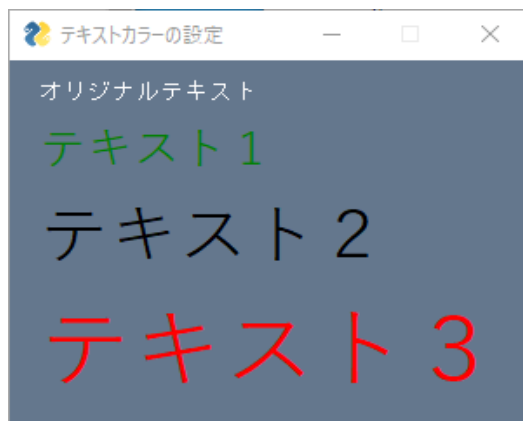


図 4.2.5.3. テキストのフォントとサイズ指定

テキストエリアのサイズ設定

テキストエリアのサイズは、**`sg.Text(size = (文字数, 行数))`** で指定できます。文字数は、半角での文字数となります。全角は2文字扱いとなります。先ほどのコードに、テキストエリアのサイズを追加すると以下のようになります。コードについては、レイアウトエリアのみを掲載します。

```
layout = [
    [sg.Text('オリジナルテキスト')],
    [sg.Text('テキスト 1', text_color='#008000', font=('游ゴシック', 20), size=(10, 2))],
    [sg.Text('テキスト 2', text_color='#000000', font=('游ゴシック', 30), size=(4, 2))],
    [sg.Text('テキスト 3', text_color='#ff0000', font=('游ゴシック', 40))],
]
```

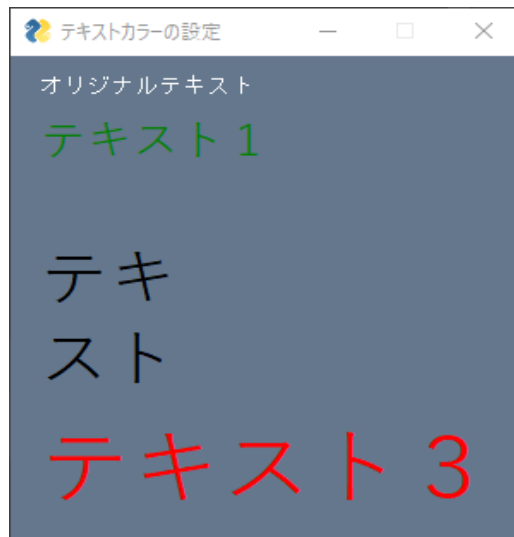


図 4.2.5.4 テキストエリアのサイズ指定

背景色の設定

ウィジェットの背景色は、プロパティー**background_color** に **16 進数カラーコード** もしくは標準色を指定して設定します。テキストウィジェットの背景色を白（16 進数カラーコード：**#ffffff**）指定する場合は、**sg.Text('表示したい文字列', background_color='#ffffff')**として設定します。

位置の設定

ウィジェット位置は、ウィジェットのプロパティーに **pad=(left/right, top/bottom)** を指定する事で設定出来ます。例えば、ボタンの位置を設定する場合は、**sg.Button(text = "", pad=(left/right, top/bottom))** で左右、上下の余白を指定できます。単位はピクセルとなります。4 方向全てを個別設定したい場合は、**pad= ((left, right), (top, bottom))** で指定する事が出来ます。テキストの場合も、同様に **sg.Button(text = "", pad=(left/right, top/bottom))** で設定出来ます。これ以外のウィジェットも同様に設定が出来ますので、位置を調整したい場合は **pad** で指定してみてください。

GUI ウィンドウに複数のテキストとボタンを設置して、それぞれのウィジェット位置を変更させた図 4.2.5.5 とコードは以下の通りです。それぞれのウィジェット位置が **pad** でどのように変化するかわかりやすくするために、各テキストの間に区切り文字を **[sg.Text('-' * 100)]** で入れています。次は、**CommonProperties2.py** にコーディングしていきます。


```
1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Text("オリジナルテキスト")],
5     [sg.Text('-' * 100)],
6     [sg.Text('テキスト1', pad=(50, 0))],
7     [sg.Text('-' * 100)],
8     [sg.Text('テキスト 2', pad=(150, 100))],
9     [sg.Text('-' * 100)],
10    [sg.Text('テキスト 3', pad=((50, 200), (0, 100)))],
11    [sg.Text('-' * 100)],
12 ]
13
14 window = sg.Window("ウィジェット位置の設定", layout, finalize=True)
15
16 while True:
17     event, values = window.read() # イベントの入力を待つ
18
19     if event == sg.WIN_CLOSED or event == 'Exit':
20         break
21
22 window.close()
```

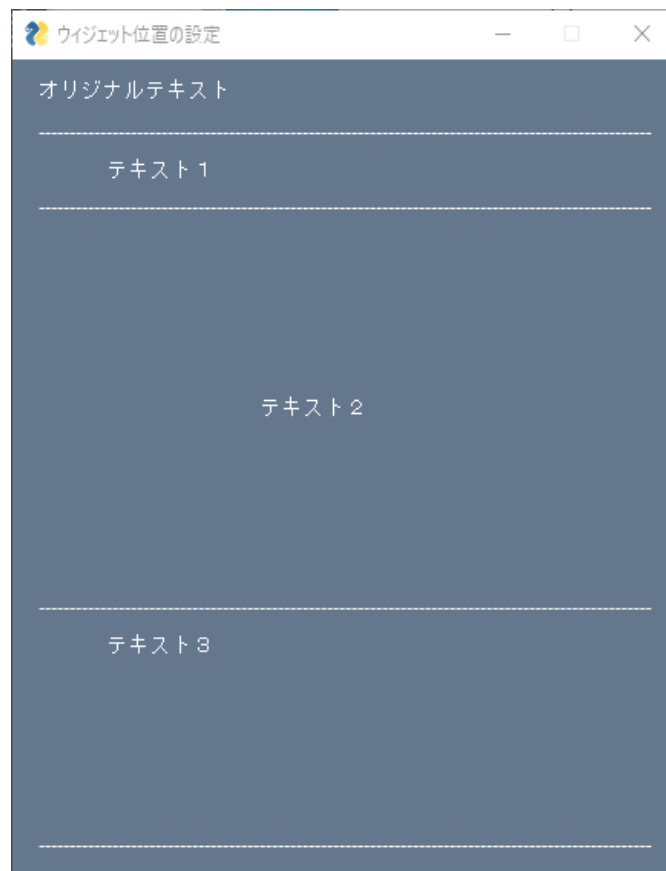


図 4.2.5.5. テキスト位置の設定

表示、非表示設定

ウィジェット表示・非表示は、ウィジェットのプロパティに **visible=(True/False)** を指定する事で設定出来ます。例えば、ボタンの表示・非表示を設定する場合は、**sg.Button(text = "", visible=(True/False))** で設定出来ます。デフォルトでは **visible=True** となっていますので、指定しなければボタンは表示されます。テキストの場合も、同様に **sg.Text(text = "", visible=(True/False))** で設定出来ます。これ以外のウィジェットも同様に設定が出来ます。このプロパティ設定を使えば、でボタンを押した場合だけ、別のボタンを表示させるというイベント処理を作成する事も出来ます。

ここからは、**CommonProperties3.py** コーディングしていきます。テキストウィジェットのプロパティに **visible** を設定してみます。テキスト 1 とテキスト 2 には **visuble = False** を設定していますので、図 4.2.5.6 のように、テキスト 1 とテキスト 2 は非表示となります。コードは以下の通りです。

```
1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Text('オリジナルテキスト')],
5     [sg.Text('-' * 100)],
6     [sg.Text('テキスト1', pad=(50, 0), visible=False)],
7     [sg.Text('-' * 100)],
8     [sg.Text('テキスト 2', pad=(150, 100), visible=False)],
9     [sg.Text('-' * 100)],
10    [sg.Text('テキスト 3', pad=((50, 200), (0, 100)), visible=True)],
11    [sg.Text('-' * 100)],
12 ]
13
14 window = sg.Window('ウィジェットの表示・非表示設定', layout, finalize=True)
15
16 while True:
17     event, values = window.read() # イベントの入力を待つ
18
19     if event == sg.WIN_CLOSED or event == 'Exit':
20         break
21
22 window.close()
```

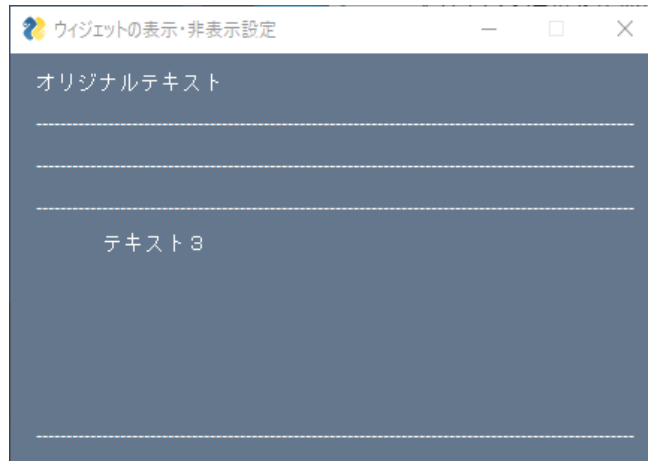


図 4.2.5.6 テキストの表示、非表示設定

無効化（disabled）の設定

ウィジェットの無効化（disabled）を設定出来ます。ウィジェットのプロパティに **disabled = (False/True)** を指定する事で設定出来ます。例えば、ボタンを押せない状態にしたい場合は、プロパティに **disabled = True** を設定する事で無効化する事が出来ます。デフォルトでは **disabled = False** となっていますので、指定しなければボタンは押せる状態になっています。ラジオボタン、チェックボックス、インプットなども同様に無効化設定が出来ます。ボタンが無効になっている事が判るように、**disabled_button_color** プロパティで無効状態のボタンテキスト色を変更しても良いでしょう。

先ほどのコードを少し変更し、テキスト 2 とテキスト 3 のウィジェットをボタン 1、ボタン 2 のウィジェットに変更します。ボタン 2 には無効化 **disabled = True** を設定しています。ボタン 1 が押された場合に、ボタン 2 の無効化が解除されるイベントを追加した図 4.2.5.7 のようなアプリを作成します。コードは以下の通りとなります。

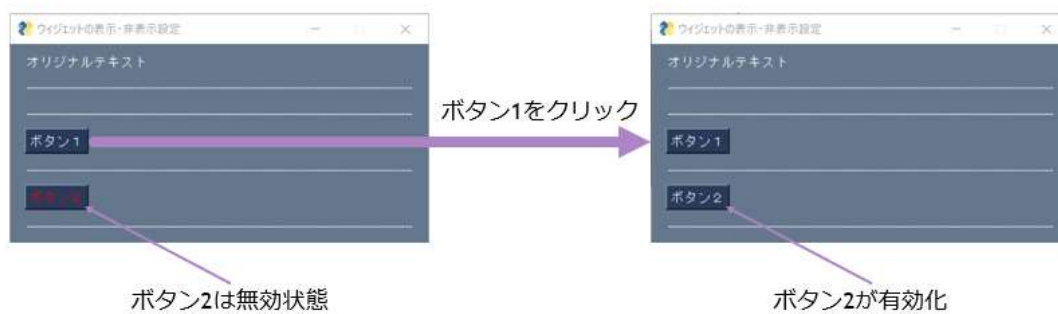


図 4.2.5.7 ボタンの無効化と有効化の切り替え設定

```

1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Text('オリジナルテキスト')],
5     [sg.Text('-' * 100)],
6     [sg.Text('テキスト1', pad=(50, 0), visible=False)],
7     [sg.Text('-' * 100)],
8     [sg.Button('ボタン1', key='-Btn1-')],
9     [sg.Text('-' * 100)],
10    [sg.Button('ボタン2', key='-Btn2-',
11              disabled=True, disabled_button_color=('red', ''))],
12    [sg.Text('-' * 100)],
13 ]
14
15 window = sg.Window('ウィジェットの無効化', layout, finalize=True)
16
17 while True:
18     event, values = window.read() # イベントの入力を待つ
19
20     if event == '-Btn1-':
21         window['-Btn2-'].Update(disabled=False)
22
23     if event == sg.WIN_CLOSED or event == 'Exit':
24         break
25
26 window.close()

```

レイアウトの設定

1 行目～4 行目、6 行目、8 行目の説明は割愛します。

5 行目でボタン 1 に **key='-Btn1-'** として、キーを割り当てています。

7 行目でボタン 2 に **key='-Btn2-'** としてキーを割り当て、**disabled=True** でボタンを無効状態に設定しています。また、**disabled_button_color=('red', '')** で、ボタンの無効状態が判るようにボタンテキスト色を赤色に指定しています。

```

layout = [
    [sg.Text('オリジナルテキスト')],
    [sg.Text('-' * 100)],
    [sg.Text('テキスト1', pad=(50, 0), visible=False)],
    [sg.Text('-' * 100)],
    [sg.Button('ボタン1', key='-Btn1-')],
    [sg.Text('-' * 100)],
    [sg.Button('ボタン2', key='-Btn2-',
              disabled=True, disabled_button_color=('red', ''))],
    [sg.Text('-' * 100)],
]

```

```
[sg.Button('ボタン 2', key='-Btn2-',  
          disabled=True, disabled_button_color=('red','')),  
 [sg.Text('-' * 100)],  
]
```

イベントの設定

ボタン 1 **'-Btn1-'** がクリックされた時のイベントを設定します。この時、**window['-Btn2-']** でボタン 2 を選択し、**Update(disabled=False)** でボタン 2 の無効状態を **disabled=False** として解除します。

```
if event == '-Btn1-':  
    window['-Btn2-'].Update(disabled=False)
```

以上のようにウィジェットの位置、表示・非表示そして **disabled** を設定する事が出来ます。イベントと組み合わせると、例えばフォームへ必要事項の記入が完了した場合に、初めて次へのボタンが押せるというようなことも出来るようになります。

4.3 各種ウィジェットのプロパティと使い方

ここからは、各種ウィジェットのプロパティ並びに使い方を解説していきます。PySimpleGUI の全てのウィジェットとプロパティについては、[公式ページ](#)を参照ください。本書では、よく使われるウィジェットとプロパティについて説明をしていきます。各ウィジェットの項では、ウィジェットの説明と主要プロパティを紹介します。その後、ウィジェットの使い方を簡単なアプリを作成して通して説明します。

4.3.1 Text (テキスト)

ウィジェットの説明：

テキストウィジェット：**Text** は、文字列や数値をウィンドウに表示します。すでにいくつかのプロパティを使用しておりますが、テキストウィジェットの主要プロパティは以下の通りです。プロパティ **text** は、省略して直接シングルクォーテーションもしくはダブルクォーテーションで表示させる文字列を指定出来ます。

ウィジェットの使い方：

4.2 で説明を加えておりますので、そちらを参照ください。

表 4.3.1. Text ウィジェットの主要プロパティ一覧

プロパティ名	説明
text	ウィンドウに表示させるテキスト。/nで改行し、複数の行を表示させる事が可能。
key	ウィジェットを一意に識別するためのキー。'-Key-'のように' 'もしくは" " で括って指定。
size	テキストエリアの幅（半角文字列の個数）、行数をタプルで（幅、行数）として整数で指定。
font	フォントやサイズ等をタプルで（フォント名、サイズ）として指定。フォント名は文字列、サイズは整数で指定。
text_color	テキストの色を標準色もしくは16進数のカラーコードで'#ffffff'のように指定。
background_color	テキストエリアの背景色を標準色もしくは16進数のカラーコードで'#ffffff'のように指定。
enable_events	ウィジェット固有のイベントをオンに設定。テキストがクリックされたときにイベントが発生。True/Falseで指定し、デフォルトはFalse。
justification	文字列をテキストエリア内でどのように整列させるかをcenter, left,rightで指定
pad	ウィジェット周囲のパディングの量（左/右、上/下）または（（左、右）、（上、下））またはint型で指定。単位はピクセル。
tooltip	ツールチップ。ウィジェットの上にマウスが乗ったときに表示させる文字列を指定。
visible	ウィジェットの表示・非表示を指定。True/Falseで指定し、デフォルトはTrue。

4.3.2 Input (インプット)

ウィジェットの説明：

インプットウィジェット：**Input** は、テキスト入力フィールドを表示します。ユーザーからの入力以外にもファイルブラウザで選択したファイル名をインプットに表示させる等の使い方もあります。また、パスワード入力をマスクするプロパティ**password_char** も指定出来ます。

ウィジェットの使い方：

図 4.3.2 のようなユーザーにパスワードを入力させる簡単なアプリを作成して使い方を学びます。インプットウィジェットにカーソルを合わせた際には、入力欄の説明（ツールチップ）を表示させるようにもします。コードは、以下の通りです。インプットウィジェットにプロパティとして、**password_char='¥'**と **tooltip='ログインに必要です'**を指定しています。マスクする文字は、1 文字であれば'¥'でも'秘'でも何でも構いません。また、今回はイベントを設定しておりません。

表 4.3.2. Input ウィジェットの主要プロパティ一覧

プロパティ名	説明
default_text	初期に表示させるテキスト
disabled	disabled状態をTrue/Falseで指定。デフォルトはFalse。
password_char	パスワード入力をマスク（デフォルトは '*' ）。
do_not_clear	Falseの場合は、任意のイベント（ボタン等）の後にフィールドがブランクに設定されます（デフォルト= True）。
key	ウィジェットを一意に識別するためのキー。'-Key-'のように' ' もしくは" " で括って指定。
size	テキストエリアの幅（半角文字列の個数）、行数をタプルで（幅、行数）として整数で指定。
font	フォントやサイズ等をタプルで（フォント名、サイズ）として指定。フォント名は文字列、サイズは整数で指定。
text_color	テキストの色を標準色もしくは16進数のカラーコードで'#ffffff'のように指定。
background_color	テキストエリアの背景色を標準色もしくは16進数のカラーコードで'#ffffff'のように指定。
enable_events	Trueの場合、このウィジェットへの変更で直ちにイベントが発生。デフォルトはFalse。
justification	文字列をテキストエリア内でどのように整列させるかをcenter, left,rightで指定
pad	ウィジェット周囲のパディングの量（左/右、上/下）または（（左、右）、（上、下））またはint型で指定。単位はピクセル。
tooltip	ツールチップ。ウィジェットの上にマウスが乗ったときに表示させる文字列を指定。
visible	ウィジェットの表示・非表示を指定。True/Falseで指定し、デフォルトはTrue。

Code4.3.2.Input.py

```
1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Input(text_color='blue', password_char='¥', tooltip='ログインが必要です')],
5     [sg.Text('-' * 100)],
6     [sg.Input(text_color='Red', password_char='秘密', tooltip='ログインが必要です')],
7 ]
8
9 window = sg.Window('Input', layout, size=(300, 150), finalize=True)
10
11 while True:
12     event, values = window.read() # イベントの入力を待つ
13
14     if event == sg.WIN_CLOSED or event == 'Exit':
15         break
16
17 window.close()
```

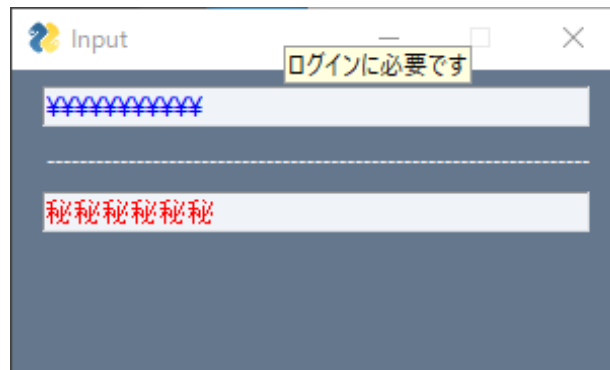


図 4.3.2. インプットウィジェット

4.3.3 Multiline (マルチライン)

ウィジェットの説明：

マルチラインウィジェット：**Multiline** は、複数行のテキストを表示または読み取ります。マルチラインウィジェットの主要プロパティは表 4.2.3 の通りです。テキストウィジェットとかなりの部分が重複します。

ウィジェットの使い方：

図 4.3.3 のような複数行の入力エリアであるマルチラインウィジェットを表示させるコードは以下の通りです。マルチラインウィジェットのサイズは、**size** プロパティで指定出来ます。入力行が指定したサイズで足りない場合は、スクロールバーが表示されます。このため、プログラムの実行とともに出力される行が増えていくような場合の出力エリアとしても使えます。

表 4.3.3. Multiline ウィジェットの主要プロパティ一覧

プロパティ名	説明
default_text	初期に表示させるテキスト
disabled	disabled状態をTrue/Falseで指定。デフォルトはFalse。
key	ウィジェットを一意に識別するためのキー。'-Key-'のように' ' もしくは" " で括って指定。
size	テキストエリアの幅（半角文字列の個数）、行数をタプルで（幅、行数）として整数で指定。
font	フォントやサイズ等をタプルで（フォント名、サイズ）として指定。フォント名は文字列、サイズは整数で指定。
text_color	テキストの色を標準色もしくは16進数のカラーコードで'#ffffff'のように指定。
background_color	テキストエリアの背景色を標準色もしくは16進数のカラーコードで'#ffffff'のように指定。
enable_events	ウィジェット固有のイベントをオンに設定。テキストがクリックされたときにイベントが発生。True/Falseで指定し、デフォルトはFalse。
justification	文字列をテキストエリア内でどのように整列させるかをcenter, left,rightで指定
pad	ウィジェット周囲のパディングの量（左/右、上/下）または（（左、右）、（上、下））またはint型で指定。単位はピクセル。
tooltip	ツールチップ。ウィジェットの上にマウスが乗ったときに表示させる文字列を指定。
visible	ウィジェットの表示・非表示を指定。True/Falseで指定し、デフォルトはTrue。

Code4.3.3.Multiline.py

```
1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Multiline(default_text='あらかじめ表示させたい文字列', size=(30, 6),
5         text_color='ff0000')],
6 ]
7
8 window = sg.Window('Multiline', layout, size=(300, 150), finalize=True)
9
10 while True:
11     event, values = window.read() # イベントの入力を待つ
12
13     if event == sg.WIN_CLOSED or event == 'Exit':
14         break
15
16 window.close()
```

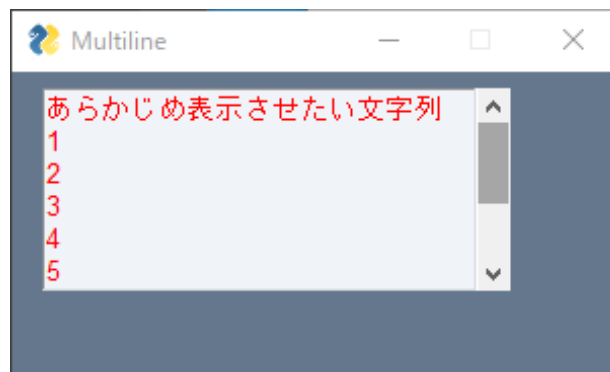


図 4.3.3. マルチラインウィジェット

4.3.4 Button (ボタン)

ウィジェットの説明：

ボタンウィジェット：**Button** は、イベントを発生させるためのボタンを表示します。イベントのトリガー（起点）となり、ファイル、フォルダーブラウズボタンやカレンダーボタンなど様々なボタンがあります。ここでは、基本となるボタンウィジェットについて解説します。また、ボタンには、テキスト以外にも画像を指定する事が出来ます。

ウィジェットの使い方：

図 4.3.4 に示すようなアプリを作成します。このアプリでは、「ON ボタンの Disable 解除」ボタンを押すことで、画像の ON ボタンが押せるようになります。ON ボタンを押すと、ポップアップが表示されるようにします。コードは以下の通りとなります。

表 4.3.4 Button ウィジェットの主要プロパティ一覧

プロパティ名	説明
button_text	ボタンに表示させるテキスト
disabled	True の場合、ボタンは無効状態で設定（ボタンが押せない状態）。
key	ウィジェットを一意に識別するためのキー。'-Key-'のように' 'もしくは" " で括って指定。
size	ボタンの幅（半角文字列の個数）、行数をタプルで（幅、行数）として整数で指定。
font	フォントやサイズ等をタプルで（フォント名、サイズ）として指定。フォント名は文字列、サイズは整数で指定。
border_width	ボタンの枠線幅を指定。単位はピクセル。
button_color	ボタンの背景色とテキストカラーを指定。デフォルトはテーマやウィンドウの色。通常は、タプルで（'テキストカラー', '背景色'）を16進数カラーコードもしくは色名で指定。背景色だけを（'背景色'）として指定する可能。
disabled_button_color	ボタンが無効化されている時のボタンカラー（'テキストカラー', '背景色'）を指定。Win10では、テキストカラーのみ変更可能。この場合は、('red', '')のように背景色はブランク" "を指定。
image_filename	ボタンに表示する画像のファイル名（パス）を指定。GIFとPNGをサポート。
image_data	ボタンに表示するRAWまたはBase64 画像をファイル名またはデータで指定。
image_size	画像のサイズを（幅、高さ）で指定。単位はピクセル。
enable_events	ウィジェット固有のイベントをオンに設定。このボタンがターゲットである場合、イベントを生成する。デフォルトは False
justification	文字列をテキストエリア内でどのように整列させるかをcenter, left, rightで指定
pad	ウィジェット周囲のパディングの量（左/右、上/下）または（（左、右）、（上、下））またはint型で指定。単位はピクセル。
tooltip	ツールチップ。ウィジェットの上にマウスが乗ったときに表示させる文字列を指定。
visible	ウィジェットの表示・非表示を指定。True/Falseで指定し、デフォルトはTrue。

```

1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Button('ONボタンのDisable解除', key='-Btn1-', pad=(50, 10))],
5     [sg.Button(image_filename='button.png', key='-Btn2-', pad=(80, 0),
6         disabled=True)],
7 ]
8
9 window = sg.Window('Button', layout, size=(300, 150), finalize=True)
10
11 while True:
12     event, values = window.read() # イベントの入力を待つ
13
14     if event == '-Btn1-':
15         window['-Btn2-'].Update(disabled=False)
16
17     if event == '-Btn2-':
18         sg.popup('ONボタンが押されました！')
19
20     if event == sg.WIN_CLOSED or event == 'Exit':
21         break
22
23 window.close()

```



図 4.3.4. ボタンウィジェットとイベント

レイアウトの設定：

配置するウィジェットは、2つのボタンウィジェットです。各ボタンには、それぞれキーを**-Btn1-**、**-Btn2-**と割り当てています。1つ目のボタンウィジェットのプロパティは、**button_text**、**key**、**pad**の3つです。**button_text**は省略可能ですので、今回は省略して**'ON ボタンの Disable 解除'**として設定しています。ボタンの位置は**pad=(50,10)**で調整しています。

2 つ目のボタンウィジェットのプロパティは、**image_filename**、**key**、**pad**、**disabled** の 4 つです。ボタンに画像を割り当てるため、**image_filename= 'button.png'** として、表示させたい画像のパスを設定しています。今回は、画像がプログラムと同一ディレクトリにある事を想定していますので、相対パスで画像ファイル名を指定しています。同一フォルダに表示させたい画像がない場合は、正しくパスを設定ください（2.1 相対パスと絶対パス参照）。また、**pad** でボタンの位置を調整しています。最後にデフォルト状態では、ボタンを無効化させておきたいため **disabled=True** を設定します。

```
layout = [  
    [sg.Button('ON ボタンの Disable 解除', key='-Btn1-', pad=(50,10))],  
    [sg.Button(image_filename='button.png', key='-Btn2-', pad=(80,0), disabled=True)],  
]
```

イベントの設定：

ボタン **-Btn1-** とボタン **-Btn2-** がクリックされた時のイベントを設定します。

まずは、ボタン **-Btn1-** が押された場合に、無効化されている ON ボタンを有効化させるイベントを設定します。if 文内で、**window['-Btn2-']** として、ウィンドウ内からキーを指定して ON ボタンを選択します。そして、ウィジェットのプロパティをアップデートしますので、**Update(disabled=False)** として **disabled** プロパティを無効にします。ボタンの無効化を無効化していますので、ボタンが有効化されて、押せる状態になります。画像でも分かりますが、ボタンが無効化されている場合は影がかかっているように見えます。ボタンが有効化されると、その影が取れてボタンが押せるようになっていることが視覚的に分かります。

次は、ボタン **-Btn2-** が押された場合に、ポップアップが表示されるようにイベントを設定します。ポップアップの表示は、すでに何度か説明していますので詳細は割愛します。

```
if event == '-Btn1-':  
    window['-Btn2-'].Update(disabled=False)  
if event == '-Btn2-':  
    sg.popup('ON ボタンが押されました！')
```

4.3.5 FileBrowse/FilesBrowse (ファイルブラウザ)

ウィジェットの説明：

ファイルブラウザウィジェット：**FileBrowse**、**FilesBrowse** は、ボタンウィジェットの 1 つであり、システム側で予め定義されているウィジェットです。ファイルダイアログという方が馴染みあるかもしれません。クリックする事で、ファイル選択のための別のウィンドウが表示され、ユーザーが GUI 上でファイルを選択する事が出来ます。ファイルは、1 つだけではなく複数選択可能です。1 つのファイルを選択する場合は、**FileBrowse**、複数のファイルを選択する場合は **FilesBrowse** を使用します。選択したファイルのパスを取得しますので、インプットウィジェットと組み合わせで選択したファイルのパスをインプットウィジェットに表示させる事も出来ます。

ウィジェットの使い方：

図 4.3.5 に示すようなアプリを作成します。このアプリでは、1 つのファイル選択と複数のファイル選択を試すことが出来ます。フィルターをかけて、選択出来るファイルを CSV ファイルに指定する方法も確認が出来ます。コードは以下の通りです。

表 4.3.5 FieleBrowse/FilesBrowse ウィジェットの主要プロパティ一覧

プロパティ名	説明
button_text	ボタンに表示させるテキスト（デフォルトは、Browse）。
disabled	ボタンの無効状態を設定（デフォルトはFalse）。
key	ウィジェットを一意に識別するためのキー。'-Key-'のように ' ' もしくは" " で括って指定。
target	選択したファイルのパスを渡すウィジェットのキーを指定。デフォルトでは、同一ラインにあるウィジェットをターゲットに指定。
file_types	ファイル形式のフィルターを設定(デフォルト値は全てのファイル形式)。csvでフィルターをかけたい場合は、(('CSVファイル', '*.csv'),)と指定。
files_delimiter	複数のファイルを選択したい場合に必要で、ファイル間に配置する文字列を指定。通常は; で設定される。
size	ボタンの幅（半角文字列の個数）、行数をダブルで（幅、行数）として整数で指定。
font	フォントやサイズ等をダブルで（フォント名、サイズ）として指定。フォント名は文字列、サイズは整数で指定。
button_color	ボタンの背景色とテキストカラーを指定。デフォルトはテーマやウィンドウの色。通常は、ダブルで（'テキストカラー', '背景色'）を16進数カラーコードもしくは色名で指定。背景色だけを（'背景色'）として指定することも出来る。
enable_events	ウィジェット固有のイベントをオンに設定。デフォルトはFalse
pad	ウィジェット周囲のパディングの量（左/右、上/下）または（（左、右）、（上、下））またはint型で指定。単位はピクセル。
tooltip	ツールチップ。ウィジェットの上にマウスが乗ったときに表示させる文字列を指定。
visible	ウィジェットの表示・非表示を指定。True/Falseで指定し、デフォルトはTrue。

Code4.3.5.FileBrowse.py

```

1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Input(), sg.FileBrowse('ファイルを1つ選択', file_types=(('CSVファイル', '*.csv'),)), ],
5     [sg.Input(key='-Output-')],
6     [sg.FilesBrowse('ファイルを複数選択', target='-Output-')],
7 ]
8
9 window = sg.Window('FileBrowse', layout, finalize=True)
10
11 while True:
12     event, values = window.read() # イベントの入力を待つ
13
14     if event == sg.WIN_CLOSED or event == 'Exit':
15         break
16
17 window.close()

```

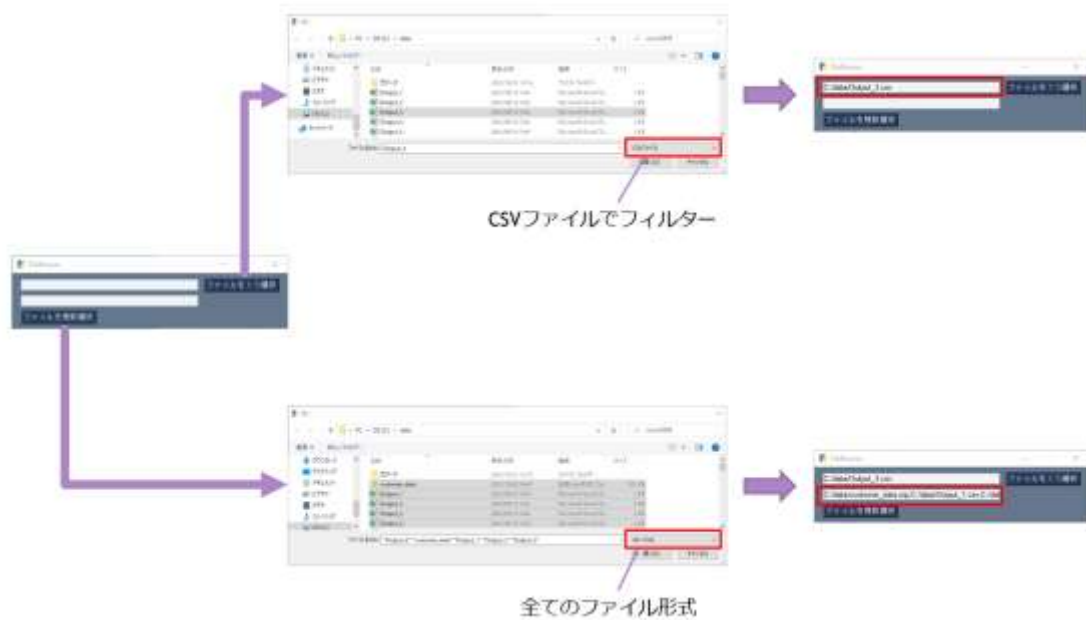


図 4.3.5. ファイルブラウズウィジェット

レイアウトの設定：

配置するウィジェットは、3 種類、計 4 つのウィジェットです。

レイアウトの1行目には、**Input** ウィジェットと **FileBrowse** ウィジェットを同一ラインに配置しています。**Input** ウィジェットには、デフォルトで同一ラインにある **FileBrowse** の出力を表示出来ますので、プロパティの指定は特に不要です。**FileBrowse** ウィジェットでは、**FilesBrowse** ボタンに表示させたい文字列と **file_types** プロパティを指定します。**file_types** プロパティは、タプルで(('CSV ファイル', '*.csv')),)のように指定します。タプルの要素1にファイルダイアログウィンドウで表示させたい文字列、要素2にファイル形式を*.csvのように指定します。もし、PDF ファイルでフィルターをかけたい場合は、(('PDF ファイル', '*.pdf')),)と指定する事が出来ます。開くボタンをクリックした後は、選択したファイルパスが同一ラインの **Input** ウィジェットに表示されます。

レイアウトの2行目は、**Input** ウィジェットを配置しています。先ほどとは異なり、同一ラインに **FileBrowse** ウィジェットを配置していません。プロパティには、キーを **key='-Output-'**として指定しています。

レイアウトの3行目は、**FilesBrowse** ウィジェットを配置しています。こちらは、複数のファイルを選択する場合に使用します。**FilesBrowse** ウィジェットでは、**FilesBrowse** ボタンに表示させたい文字列と **target** プロパティを指定します。**target** プロパティには、選択したファイルのパスを出力するウィジェットをキーで **target='-Output-'**のように指定します。出力先を **target** で指定出来ますので、同一ライン上に無いウィジェットにも出力する事が出来ます。

```
layout = [  
    [sg.Input(), sg.FileBrowse('ファイルを1つ選択', file_types=(('CSV ファイル', '*.csv')),)],  
    [sg.Input(key='-Output-')],  
    [sg.FilesBrowse('ファイルを複数選択', target='-Output-')],  
]
```

イベントの設定：

今回は、イベントを設定しません。

4.3.6 FolderBrowse (フォルダーブラウズ)

ウィジェットの説明：

フォルダーブラウズウィジェット：**FolderBrowse** は、ボタンウィジェットの 1 つであり、システム側で予め定義されているウィジェットです。ボタンをクリックする事で、フォルダ選択のための別のウィンドウが表示され、ユーザーが GUI 上でフォルダを選択する事が出来ます。**FileBrowse** と同様に選択したフォルダーのパスを取得します。

ウィジェットの使い方：

基本的な使い方は、**FileBrowse** と同じですので、使い方のコード例と画像のみを掲載し詳細説明は割愛します。プログラムで出力するファイルの保存先やプログラムの入力に必要となるフォルダーをユーザーに選択させる等の使い方があります。

表 4.3.6. FolderBrowse ウィジェットの主要プロパティ一覧

プロパティ名	説明
button_text	ボタンに表示させるテキスト（デフォルトは、Browse）。
disabled	ボタンの無効状態を設定（デフォルトはFalse）。
key	ウィジェットを一意に識別するためのキー。'-Key-'のように' 'もしくは" " で括って指定。
target	選択したフォルダのパスを渡すウィジェットのキーを指定。デフォルトでは、同一ラインにあるウィジェットをターゲットに指定。
size	ボタンの幅（半角文字列の個数）、行数をタプルで（幅、行数）として整数で指定。
font	フォントやサイズ等をタプルで（フォント名、サイズ）として指定。フォント名は文字列、サイズは整数で指定。
button_color	ボタンの背景色とテキストカラーを指定。デフォルトはテーマやウィンドウの色。通常は、タプルで（'テキストカラー','背景色'）を16進数カラーコードもしくは色名で指定。背景色だけを（'背景色'）として指定可能。
enable_events	ウィジェット固有のイベントをオンに設定。デフォルトはFalse
pad	ウィジェット周囲のパディングの量（左/右、上/下）または（（左、右）、（上、下））またはint型で指定。単位はピクセル。
tooltip	ツールチップ。ウィジェットの上にマウスが乗ったときに表示させる文字列を指定。
visible	ウィジェットの表示・非表示を指定。True/Falseで指定し、デフォルトはTrue。

Code4.3.6.FolderBrowse.py

```
1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Input(), sg.FolderBrowse('フォルダーを選択',key = '-Folder-')],
5 ]
6
7 window = sg.Window('FolderBrowse', layout, finalize=True)
8
9 while True:
10     event, values = window.read() # イベントの入力を待つ
11
12     if event == sg.WIN_CLOSED or event == 'Exit':
13         break
14
15 window.close()
```



図 4.3.6. フォルダーブラウズウィジェット

4.3.7 CalendarButton (カレンダーボタン)

ウィジェットの説明：

カレンダーボタンウィジェット：**CalendarButton** は、ボタンウィジェットの 1 つであり、システム側で予め定義されているウィジェットです。ボタンをクリックする事で、カレンダーが表示されて GUI 上で日付を選択出来るようになります。日付を入力する際には、このウィジェットがあると非常に便利です。**Input** ウィジェットと組み合わせて、選択した日付を表示させる事が出来ます。

ウィジェットの使い方：

図 4.3.7 に示すようなアプリを作成します。このアプリでは、カレンダー入力と出力結果の違いを確認する事が出来ます。出力結果は、フォーマットする事で変更する事が出来ます。コードは以下の通りです。

4.3.7 CalendarButton ウィジェットの主要プロパティ一覧

プロパティ名	説明
button_text	ボタンに表示させるテキスト
target	カレンダーの選択結果を反映するウィジェットのキーを指定。
month_names	表示させる月名をリスト形式で指定。
day_abbreviations	表示させる曜日名をリスト形式で指定。
location	カレンダーポップアップを表示する画面上の位置(x,y)を指定。
format	カレンダーで取得した値のフォーマットを指定。例えば、'%m-%d'と指定すれば1月13日は、01-13へフォーマットされる
key	ウィジェットを一意に識別するためのキー。'-Key-'のように ' ' もしくは" " で括って指定。
size	ボタンの幅（半角文字列の個数）、行数をタプルで（幅、行数）として整数で指定。
font	フォントやサイズ等をタプルで（フォント名、サイズ）として指定。フォント名は文字列、サイズは整数で指定。
border_width	ボタンの枠線幅を指定。単位はピクセル。
button_color	ボタンの前面色と背景色を指定。デフォルトはテーマやウィンドウの色。タプルで（'前面色','背景色'）を16進数カラーコードもしくは色名で指定。
enable_events	ウィジェット固有のイベントをオンに設定。このボタンがターゲットである場合、イベントを生成する。デフォルトはFalse。
pad	ウィジェット周囲のパディングの量（左/右、上/下）または（（左、右）、（上、下））またはint型で指定。単位はピクセル。
tooltip	ツールチップ。ウィジェットの上にマウスが乗ったときに表示させる文字列を指定。
visible	ウィジェットの表示・非表示を指定。True/Falseで指定し、デフォルトはTrue。

```

1 import PySimpleGUI as sg
2
3 layout = [
4     [sg.Input(), sg.CalendarButton('Date')],
5     [sg.Input(key='-Output-'),
6      sg.CalendarButton('日付', target='-Output-', format="%Y年%m月%d日")]
7 ]
8
9 window = sg.Window('カレンダーからの入力', layout, finalize=True)
10
11 while True:
12     event, values = window.read() # イベントの入力を待つ
13
14     if event == sg.WIN_CLOSED or event == 'Exit':
15         break
16
17 window.close()

```



図 4.3.7.カレンダーボタンウィジェット

レイアウトの設定：

配置するウィジェットは、2 種類、計 4 つのウィジェットです。

レイアウトの 1 行目には、**Input** ウィジェットと **CalendarButton** ウィジェットを同一ラインに配置しています。**Input** ウィジェットには、同一ラインにある **CalendarButton** の出力を表示させますので、プロパティの指定は特に不要です。**CalendarButton** ウィジェットでは、**CalendarButton** ボタンに表示させたい文字列を指定します。今回は、カレンダーボタンに **Date** と表示させています。カレンダー上で日付を選択した後は、選択した日付が同一ラインの **Input** ウィジェットに表示されます。

レイアウトの2行目は、**Input** ウィジェットを配置しています。先ほどとは異なり、同一ラインに **CalendarButton** ウィジェットを配置していません。プロパティには、キーを **key='-Output-'** として指定しています。

レイアウトの3行目は、**CalendarButton** ウィジェットを配置しています **CalendarButton** ウィジェットでは、**CalendarButton** ボタンに表示させたい文字列と **target** プロパティ、そして **format** プロパティを指定します。**target** プロパティには、選択した日付を出力するウィジェットをキーで **target='-Output-'** のように指定します。**format** プロパティでは、日付のフォーマットを指定します。プロパティを指定しないと、年月日に加えて時刻まで表示されます。今回は、年月日のみをハイフン区切りで表示させたいので **format='%Y-%m-%d'** としてフォーマットを指定します。もし、2022 年 1 月 11 日と表示させたい場合は、**format='%Y 年 %m 月 %d 日'** と指定します。

```
layout = [  
    [sg.Input(), sg.CalendarButton('Date')],  
    [sg.Input(key='-Output-')],  
    [sg.CalendarButton('日付', target='-Output-', format='%Y-%m-%d')]  
]
```

イベントの設定：

今回は、イベントを設定しません。

5.実践的デスクトップアプリの開発

これまでは、PySimpleGUI を使った GUI 付アプリの基本的な作成方法を紹介してきました。GUI のレイアウト作成は出来るようになって、実際どのように使えば良いかのイメージが湧かない方もいると思います。本書の仕上げとして、実践的な GUI 付きデスクトップアプリを開発し、ご自身のオリジナルデスクトップアプリを開発するための基礎を身につけます。

5.1 備品管理アプリの概要

まずは、今回開発する備品管理アプリのレイアウトを図 5.1.1 を用いて説明します。備品管理アプリは、全部で 4 つの画面（ウィンドウ）で構成されています。

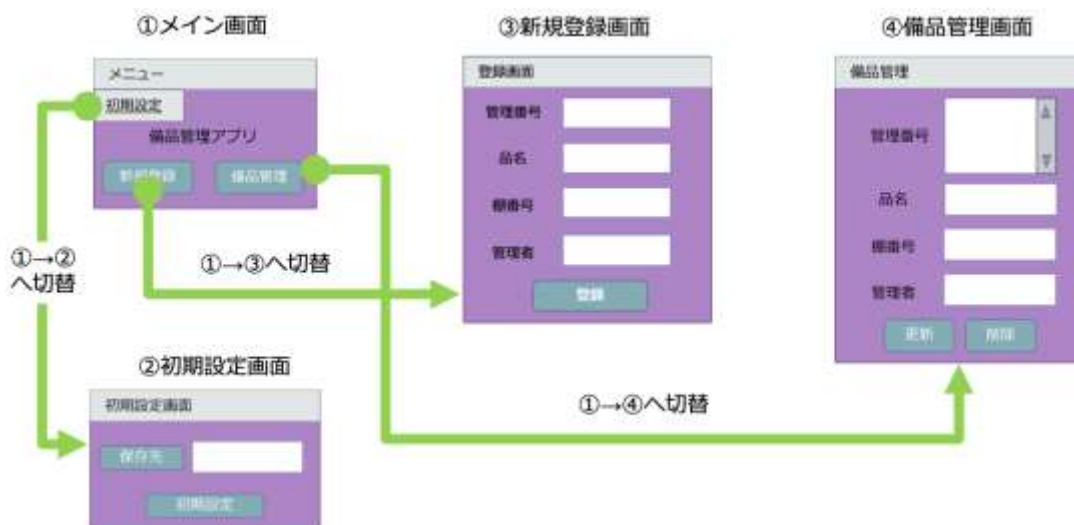


図 5.1.1.備品管理アプリの概要

①メイン画面は、アプリ起動時の最初に表示される画面です。この画面を中心として、②初期設定画面、③新規登録画面、④備品管理画面へ遷移します。メニューバー内に初期設定画面への遷移ボタン、メイン画面内に③新規登録画面と④備品管理画面への遷移ボタンを配置します。初期設定はアプリを最初に使い始める時にだけ使用します。常に見える位置に初期設定画面への遷移ボタンを配置する必要はありません。このため、初期設定画面への遷移ボタンは①メイン画面のメニューバー内に設定します。

②初期設定画面では、アプリデータの保存先をユーザーが指定出来るようにフォルダダイアログを設置します。初期設定ボタンも配置し、初期設定を出来るようにします。メイン画面へ戻るボタンは配置しない代わりに、初期設定が完了したらメイン画面に戻るように設定します。

③新規登録画面では、新規に備品を登録出来るようにします。それぞれの備品情報を入力するインプットウィジェットを配置します。登録ボタンを配置し、登録処理を実行するようにします。

④備品管理画面では、既存の備品情報を更新及び削除が更新や削除が行えるようにします。管理番号は、リストボックスとして既存の管理番号を1つ選択出来るようにします。また、その他の備品情報を入力するインプットウィジェットも配置します。変更ボタンと削除ボタンを配置し、変更と削除処理を実行するようにします。

図 5.1.2 で①メイン画面と②初期設定画面のファイル入出力操作について説明します。

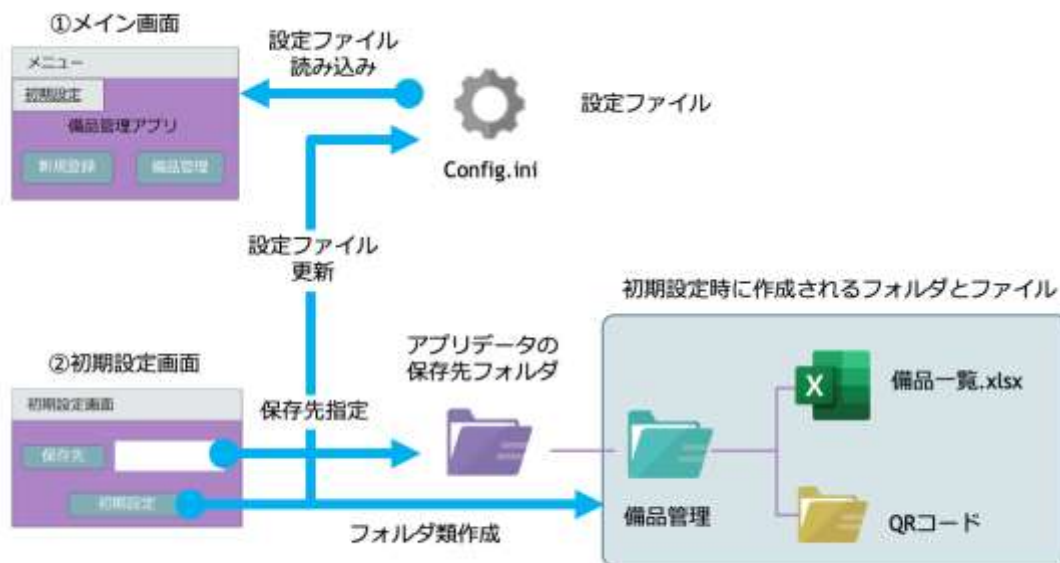


図 5.1.2.メイン画面と初期設定画面の動作説明

①メイン画面は、設定ファイル(config.ini)の内容を読み込み、アプリに必要な情報や変数を取得します。また、初期設定が完了しているかも判断します。設定ファイルについてですが、今回はブランクの Config.ini というファイルをメインプログラムと同じ階層に作成します。

②初期設定画面では、ユーザーが指定したアプリデータの保存先フォルダの直下に、必要なフォルダ類を作成します。初期設定ボタンから図に示すように備品管理フォルダを作成し、その中に備品管理一覧.xlsx と QR コードフォルダを作成します。また、初期設定内容（初期設定で作成したフォルダ類のパス）を設定ファイルに追記します。

図 5.1.3 で③新規登録画面、④備品管理画面と初期設定で作成されたファイル、フォルダとの入出力操作を説明します。

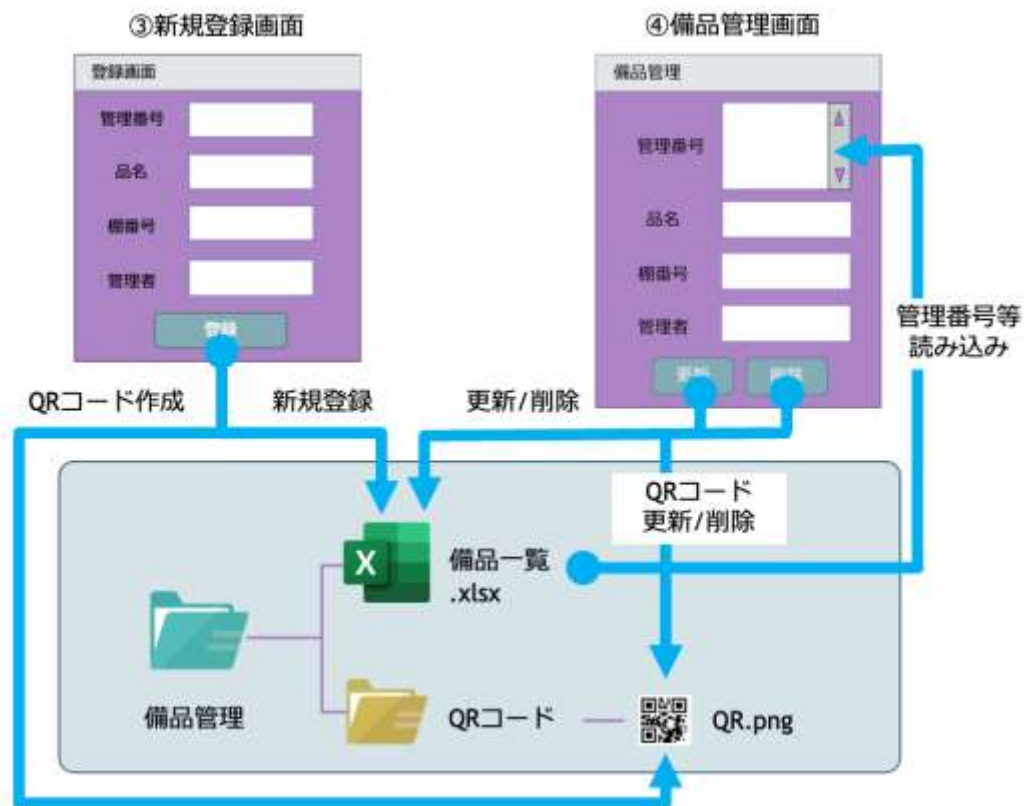


図 5.1.3.新規登録画面と備品管理画面の動作説明

③新規登録画面では、登録ボタンを押すと備品情報を備品一覧.xlsx に保存します。同時に、備品情報が記録された QR コードを png 形式で作成して QR コードフォルダに保存します。管理番号は、全ての備品内で固有の番号としますので重複を認めない制約も加えます。

④備品管理画面では、備品一覧.xlsx から管理番号等の情報を読み込みます。管理番号のリストボックスでユーザーが更新もしくは削除したい管理番号を選択すると対応する品名、棚番号、管理者の情報をウィジェットへ反映させるようにします。更新/削除ボタンを押すと備品一覧.xlsx の備品情報を更新/削除します。同時に、備品情報が記録された QR コードも更新/削除します。

以上が今回作成するアプリの概要となります。実践的なアプリを想定していますのでウィンドウを複数使用します。実際のデスクトップアプリでも、1つのウィンドウで完結するアプリは限られていると思います。今回のアプリ開発を通じて、複数ウィンドウの作成、切り替えを学ぶ事が出来ます。また、ファイルやフォルダの作成、Excel ファイルへの入出力、設定ファイルの使い方、QR コードの生成など実務で使いそうな内容を一通り網羅しています。今回のアプリ作成を理解できれば、ご自身のオリジナルデスクトップアプリを開発する基礎がしっかり身につきますので、少し複雑になりますが丁寧に解説していきますので頑張って作成しましょう。

さらに本格的なアプリを作成する場合は、データベース設定、Web API 導入、マウスの自動操作（RPA 機能）などを検討しても良いかも知れません。店舗の日々の売上や原価情報などを管理するのであれば Excel で管理するのではなく、SQLite や MySQL などのリレーショナルデータベースを使用する事で堅牢なデータベースを構築出来ます。また、Google、Microsoft、Twitter、Line、楽天などあらゆるサービスがさまざまな Web API を提供しています。Web API を使えば、外部サービスとの連携も容易になりますのでアプリケーションの幅が広がります。さらに、Python では PyAutoGUI という RPA パッケージも無償で提供されていますので、高額な RPA + 高額な年間費用を支払わなくても自分で GUI 付きの RPA を作る事も可能です。それらは本書の範疇を超えますので、詳細は触れませんが、ネット上にさまざまな情報がありますので、本書で学ぶ GUI プログラムと組み合わせてみて皆様のオリジナルアプリを作成してみても如何でしょうか。