PySimpleGUI

S'initier à un GUI (graphical user interface)

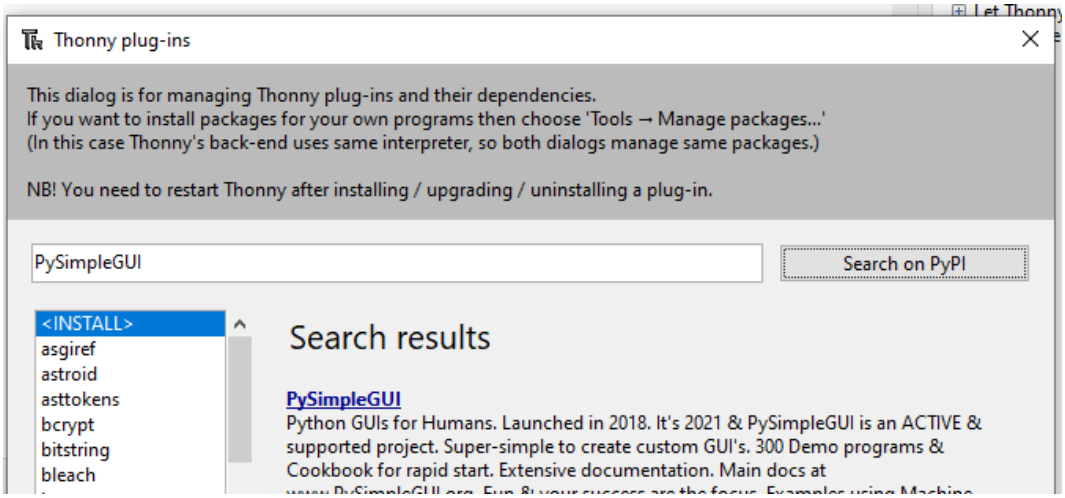
1. Installation

Le module PySimpleGUI est-il installé dans votre IDE (environnement de développement intégré) ? Pour cela tenter d'importer le module.

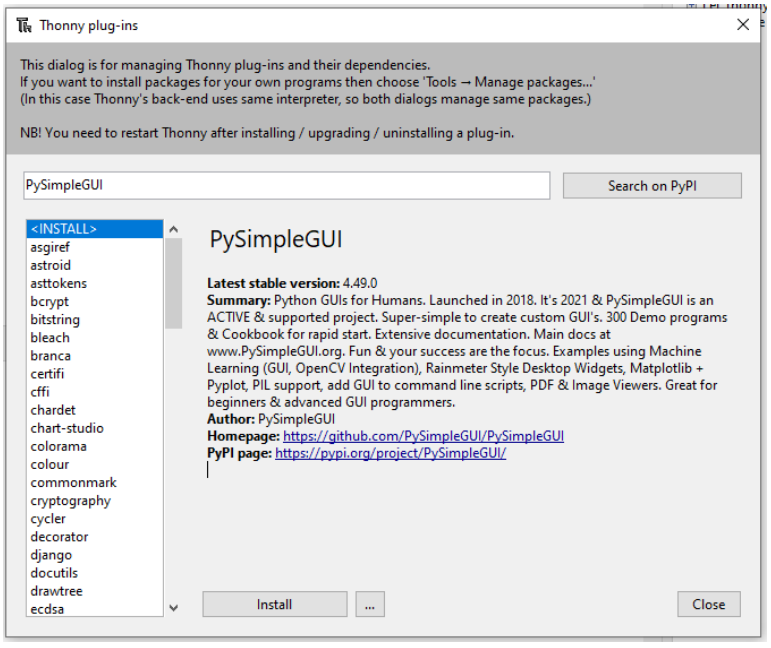
```
>>> import PySimpleGUI
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ModuleNotFoundError: No module named 'PySimpleGUI'
>>>
```

☞ Le module doit être installé

Dans Thonny, Menu Tools/ManagePlug-ins, faire une recherche



Cliquer sur le premier module PySimpleGUI, puis cliquer sur le bouton Install



Tester à nouveau l'import

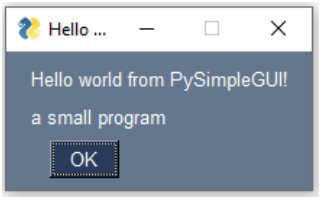
```
>>> import PySimpleGUI
>>>
```

Vérifier la version avant de commencer

```
>>> PySimpleGUI.version
'4.49.0 Released 30-Sept-2021'
>>>
```

2. Quelques exemples

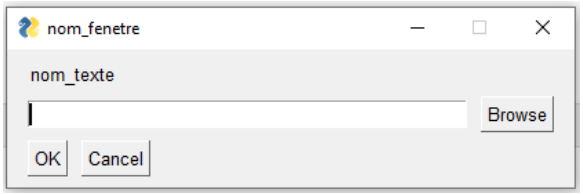
Commençons par un traditionnel Hello World !




```
import PySimpleGUI as sg
sg.popup('Hello world from PySimpleGUI!', 'a small program')
```

10_3_0_01.py

Il existe 2 niveaux de prise en charge du fenêtrage dans PySimpleGUI : Haut niveau ou Personnalisé. Dans ce premier exemple nous avons utilisé le haut niveau et bénéficié de la simplicité d'écriture. Que diriez-vous d'une interface graphique personnalisée ?




```
import PySimpleGUI as sg

# Choisir le thème
sg.theme('GrayGrayGray')

# Lister les éléments utilisés
layout = [[sg.Text('nom_texte')],
          [sg.Input(), sg.FileBrowse()],
          [sg.OK(), sg.Cancel()]]

# Créer la fenêtre
window = sg.Window('nom_fenetre', layout, finalize=True)

# Ecouter la fenêtre
event, values = window.read()

# Fermer par bouton
window.close()
```

10_3_0_02.py

3. Appels API de haut niveau - Popup's

Les "appels de haut niveau" sont ceux qui commencent par "popup". Ils sont nommés d'après le type de fenêtre qu'ils créent, une fenêtre contextuelle. Ces fenêtres sont censées être de courte durée lorsqu'elles fournissent des informations ou les collectent, puis disparaissent rapidement.

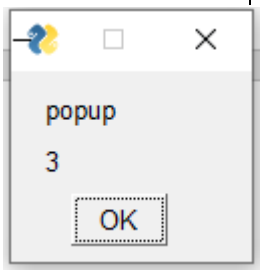
Voici la liste des appels contextuels disponibles :

```
popup_animated popup_annoying popup_auto_close popup_cancel popup_error
popup_get_file popup_get_folder popup_get_text popup_no_border
popup_no_buttons popup_no_frame popup_no_titlebar popup_no_wait
popup_notify popup_non_blocking popup_non_blocking popup_ok popup popup
```

3.1. Sortie contextuelle

On peut considérer les popup comme l'équivalent de la fonction print() dans le mode console. On peut tester ces quelques exemples.

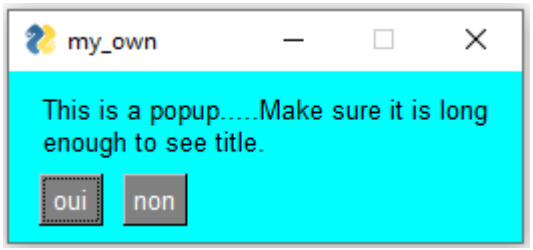
```
import PySimpleGUI as sg
sg.theme('GrayGrayGray')

sg.popup_ok('popup_ok')  # Shows OK button
sg.popup_yes_no('popup_yes_no')  # Shows Yes and No buttons
sg.popup_cancel('popup_cancel')  # Shows Cancelled button
sg.popup_ok_cancel('popup_ok_cancel')  # Shows OK and Cancel buttons
sg.popup_error('popup_error')  # Shows red error button
sg.popup_timed('popup_timed')  # Automatically closes
a = 3
sg.popup('popup', a)  # Shows OK button
```



10_3_0_03.py

Si cette fenêtre contextuelle n'a pas les fonctionnalités que vous souhaitez, vous pouvez facilement créer la vôtre.

```
import PySimpleGUI as sg
sg.popup('This is a popup.....Make sure it is long enough to see title.',
    title="my_own",
    button_color =  'grey',
    background_color = 'cyan',
    text_color = 'black',
    custom_text = ('oui', 'non'),
    line_width = 40)
```



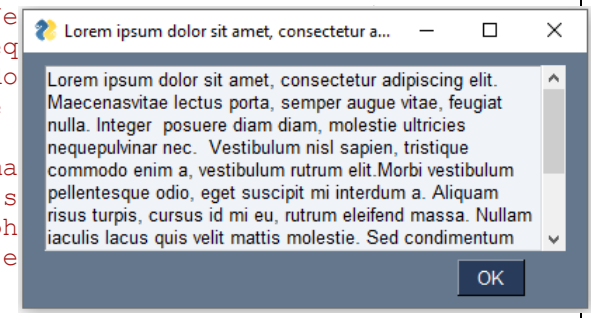
10_3_0_05.py

Voir annexe Popup()

3.2. Sortie avec défilement

Il existe une version déroulante de Popups si vous avez beaucoup d'informations à afficher.

```
import PySimpleGUI as sg
my_text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas"\
          "vitae lectus porta, semper augue vitae, fe
          " posuere diam diam, molestie ultricies neq
          " Vestibulum nisl sapien, tristique commodo
          "rutrum elit.Morbi vestibulum pellentesque
          " interdum a. Aliquam risus turpis, cursus
          " massa. Nullam iaculis lacus quis velit ma
          "condimentum pulvinar malesuada. Suspendiss
          "t amet tempus purus tempor sed. Vivamus ph
          " sodales. Donec in lacus auctor, dapibus e
          " mi. Aliquam erat volutpat."
sg.popup_scrolled(my_text, size=(50, 8))
```



10_3_0_06.py

Voir annexe popup_scrolled()

3.3. Entrée contextuelle

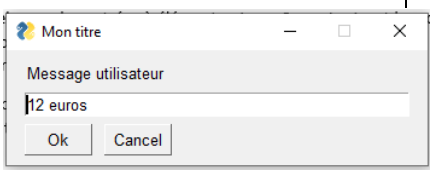
Il existe des appels contextuels pour les entrées à élément unique. Ceux-ci suivent le modèle de **popup_get** suivi du type d'élément à obtenir. Vous pouvez choisir parmi 3 de ces fenêtres contextuelles d'entrée, chacune avec des paramètres permettant la personnalisation.

- popup_get_text - obtenir une seule ligne de texte
- popup_get_file - obtenir un nom de fichier
- popup_get_folder - obtenir un nom de dossier

3.3.1. popup_get_text

Utilisez cette fenêtre contextuelle pour obtenir une ligne de texte de l'utilisateur. C'est l'équivalent de input() en mode console. Voici un exemple pour afficher popup_get_text qui renvoie le texte saisi ou Aucun si fermé/annulé

```
import PySimpleGUI as sg
sg.theme('GrayGrayGray')
text = sg.popup_get_text('Message utilisateur','Mon titre', \
                         text_color='black', \
                         default_text='12 euros')
```



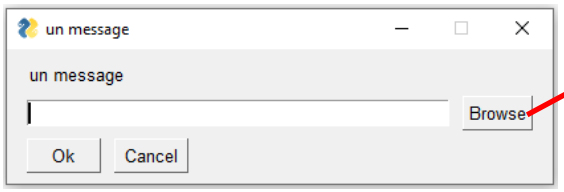
10_3_0_08.py

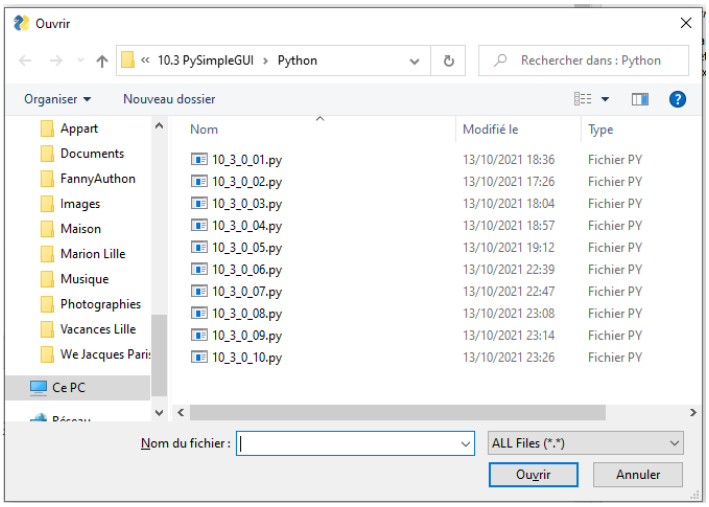
Voir annexe popup_get_text()

3.3.2. popup_get_file

Utilisez cette fenêtre contextuelle pour obtenir un ou plusieurs noms de fichiers de l'utilisateur. Il existe des options pour configurer le type de boîte de dialogue à afficher. Normalement, une boîte de dialogue "Ouvrir un fichier" s'affiche.

Voici un exemple pour afficher une fenêtre contextuelle avec un champ de saisie de texte et un bouton « Browse » afin qu'un fichier puisse être choisi par l'utilisateur.






```
import PySimpleGUI as sg
sg.theme('GrayGrayGray')
text = sg.popup_get_file("un message")
```

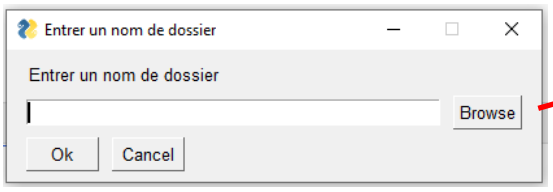
10_3_0_10.py

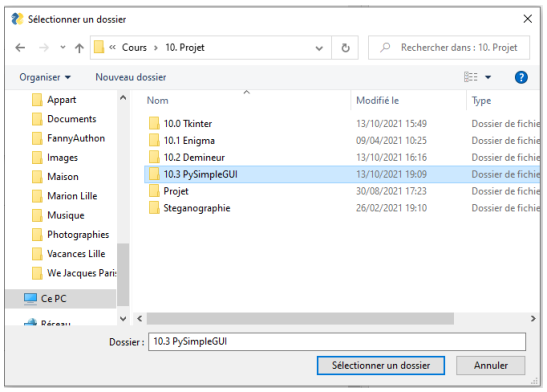
Voir annexe popup_get_file()

3.3.3. popup_get_folder

La fenêtre créée pour obtenir un nom de dossier ressemble à celle pour obtenir un nom de fichier. La différence réside uniquement dans ce que fait le bouton « Browse ». Popup_get_file affichait une boîte de dialogue Ouvrir un fichier Popup_get_folder affiche maintenant une boîte de dialogue Ouvrir un dossier.

Voici un exemple pour afficher une fenêtre contextuelle avec un champ de saisie de texte et un bouton de navigation afin de pouvoir choisir un dossier.






```
import PySimpleGUI as sg
sg.theme('GrayGrayGray')
text = sg.popup_get_file("un message")
```

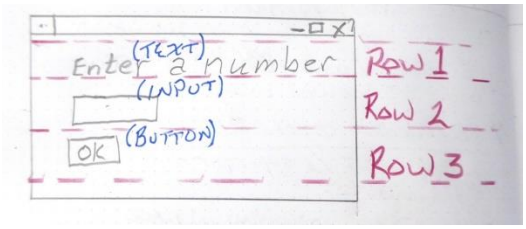
10_3_0_12.py

Voir annexe popup_get_folder()

4. Concevoir son interface

Pour concevoir son API (Application Programming Interface) la démarche est la suivante :

- Esquissez votre interface graphique sur papier
- Divisez votre interface graphique en rangées
- Étiquetez chaque élément avec le nom de l'élément
- Écrivez votre code Python en utilisant les étiquettes comme pseudo-code




```
import PySimpleGUI as sg
sg.theme('GrayGrayGray')
# 3 listes sont placées dans une liste qui représente la fenêtre entière.
layout = [[sg.Text('Enter a Number')],
          [sg.Input()],
          [sg.OK()] ]

window = sg.Window('Entrer un nombre', layout)
event, values = window.read()
window.close()
sg.Popup(event, values[0])
```

10_3_0_14.py

5. Des modèles de conception

Tous vos programmes PySimpleGUI utiliseront l'un de ces 2 modèles de conception en fonction du type de fenêtre que vous implémentez :

- Fenêtre unique : Lire une fenêtre une fois puis la fermer
- Fenêtre persistante : Lectures multiples à l'aide d'une boucle d'événement ou bien lectures multiples à l'aide d'une boucle d'événement et mises à jour des données dans la fenêtre.

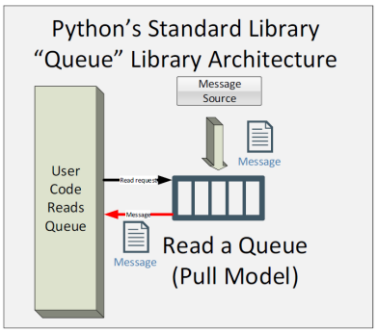
5.1. Fenêtre unique


```
>>> source_filename
'D:/Donnees/Cours_NSI_T/Cours/10.
Projet/10.3
PySimpleGUI/Python/10_3_0_11.py'
>>>
```

```
import PySimpleGUI as sg
sg.theme('GrayGrayGray')
layout = [[sg.Text('Chemin du fichier')],
          [sg.InputText(), sg.FileBrowse(button_text='Chercher')],
          [sg.Submit(button_text='Valider'), sg.Cancel(button_text='Annuler')]]
window = sg.Window('Sélecteur fichier', layout)
event, values = window.read()
window.close()
source_filename = values[0]
```

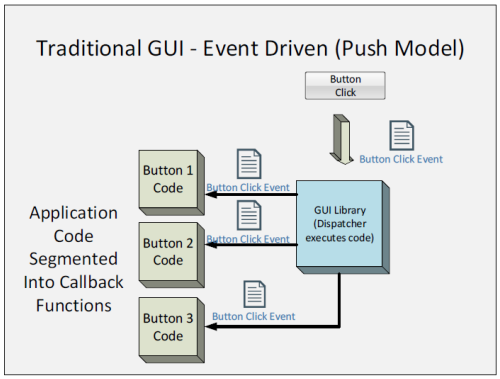
10_3_0_15.py

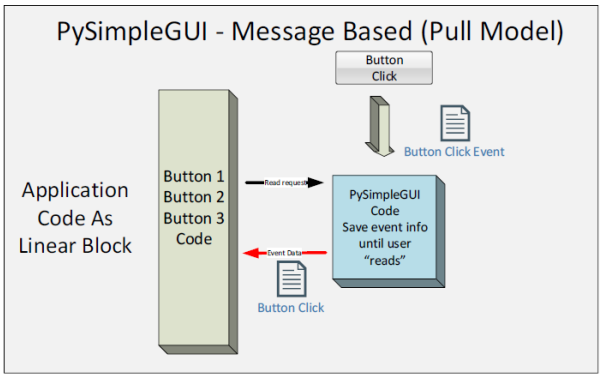
On peut associer cette situation au schéma suivant :



5.2. Fenêtre persistante

En programmation évènementielle, deux approches sont possibles :



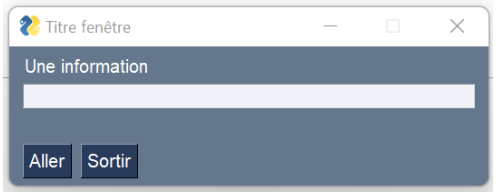


5.2.1. Le Pull Model

PySimpleGUI sauvegarde l'évènement. C'est l'utilisateur qui vient lire les événements dans le programme principal qui est une boucle de scrutation.

Exemple : 10_3_0_16.py

```
# On importe le module
import PySimpleGUI as sg
# On prépare la dsiposition (layout) des éléments
layout = [ [sg.Text('Une information')],
           [sg.Input(key='-IN-')],
           [sg.Text(size=(20,1), key='-OUT-')],
           [sg.Button('Aller', key='-GO-'), sg.Button('Sortir', key='-END-')] ]
# On définit l'objet fenêtre
window = sg.Window('Titre fenêtre', layout, finalize=True)
# On scrute les évenements
while True:
    # Observer l'évenement et sa valeur
    event, values = window.read()
    print(event, values)
    if event in (None, 'END'):
        # On peut sortir par la croix ou le bouton
        break
    if event == '-GO-':
        # On traite l'évenement
        window['-OUT-'].update(values['-IN-'])
# On ferme le programme
window.close()
```



5.3. Concepts de base

Les concepts de base de PySimpleGUI sont les suivants :

On déclare les différentes parties de son interface graphique sous forme de listes imbriquées d'éléments graphiques. La liste extérieure représente les lignes d'éléments graphiques à afficher à l'écran de haut en bas. Au sein de chaque ligne, les éléments sont également rangés de gauche à droite au sein d'une liste.

Ainsi, pour un écran avec

- Une ligne de texte,
- Puis une autre ligne juste en dessous avec une invite et un bouton

On peut déclarer la disposition comme suit :

```
ma_disposition = [[sg.T("Mon texte")], [sg.In("ce texte est modifiable"), sg.B("Cliquez sur ce bouton")]]
```

On attribue à chaque élément avec lequel on souhaite interagir un identifiant (key) sous forme de chaîne de caractère (par exemple "-MON-ELEMENT-"). Cet identifiant deviendra une clé utilisée dans les situations :

- Lorsque l'élément en question déclenche un événement, l'événement porte le même identifiant que l'élément qui l'a émis
- Les valeurs associées à l'événement, ainsi que celles associés à tous les autres éléments actifs à cet instant à l'écran sont transmises en même temps que chaque événement dans un dictionnaire nommé "values"

Ainsi dans notre exemple, la ligne 14 affiche dans la console

`-GO- {'-IN-': ''}`	Appui sur le bouton « Aller »
`-END- {'-IN-': ''}`	Appui sur le bouton « Sortir »
`None None`	Appui sur la croix de la fenêtre

- La mise à jour des données associées à un élément se fait via la syntaxe suivante : window["-MON-ELEMENT-"].update("ma nouvelle valeur")

Dans notre exemple ligne 20

```
window['-OUT-'].update(values['-IN-'])
```

6.1. Popup()

```
import PySimpleGUI as sg
sg.popup('demo'
    title = None,
    button_color = None,
    background_color = None,
    text_color = None,
    button_type = 0,
    auto_close = False,
    auto_close_duration = None,
    custom_text = (None, None),
    non_blocking = False,
    icon = None,
    line_width = None,
    font = None,
    no_titlebar = False,
    grab_anywhere = False,
    keep_on_top = None,
    location = (None, None),
    any_key_closes = False,
    image = None,
    modal = True)
```

10_3_0_04.py

Type	Name	Meaning
Any	*args	Variable number of your arguments. Load up the call with stuff to see!
str	title	Optional title for the window. If none provided, the first arg will be used instead.
(str, str) or None	button_color	Color of the buttons shown (text color, button color)
str	background_color	Window's background color
str	text_color	text color
int	button_type	NOT USER SET! Determines which pre-defined buttons will be shown (Default value = POPUP_BUTTONS_OK). There are many Popup functions and they call Popup, changing this parameter to get the desired effect.
bool	auto_close	If True the window will automatically close
int	auto_close_duration	time in seconds to keep window open before closing it automatically
(str, str) or str	custom_text	A string or pair of strings that contain the text to display on the buttons
bool	non_blocking	If True then will immediately return from the function without waiting for the user's input.

Type	Name	Meaning
str or bytes	icon	icon to display on the window. Same format as a Window call
int	line_width	Width of lines in characters. Defaults to MESSAGE_BOX_LINE_WIDTH
str or Tuple[font_name, size, modifiers]	font	specifies the font family, size, etc. Tuple or Single string format 'name size styles'. Styles: italic * roman bold normal underline overstrike
bool	no_titlebar	If True will not show the frame around the window and the titlebar across the top
bool	grab_anywhere	If True can grab anywhere to move the window. If no_titlebar is True, grab_anywhere should likely be enabled too
(int, int)	location	Location on screen to display the top left corner of window. Defaults to window centered on screen
bool	keep_on_top	If True the window will remain above all current windows
bool	any_key_closes	If True then will turn on return_keyboard_events for the window which will cause window to close as soon as any key is pressed. Normally the return key only will close the window. Default is false.
str or bytes	image	Image to include at the top of the popup window
bool	modal	If True then makes the popup will behave like a Modal window... all other windows are non-operational until this one is closed. Default = True
str or None	**RETURN**	Returns text of the button that was pressed. None will be returned if user closed window with X

```
popup_scrolled(args=*<1 or N object>,
    title = None,
    button_color = None,
    background_color = None,
    text_color = None,
    yes_no = False,
    auto_close = False,
    auto_close_duration = None,
    size = (None, None),
    location = (None, None),
    non_blocking = False,
    no_titlebar = False,
    grab_anywhere = False,
    keep_on_top = None,
    font = None,
    image = None,
    icon = None,
    modal = True,
    no_sizegrip = False)
```

10_3_0_07.py

Description des paramètres :

Type	Name	Meaning
Any	*args	Variable number of items to display
str	title	Title to display in the window.
(str, str) or str	button_color	button color (foreground, background)
bool	yes_no	If True, displays Yes and No buttons instead of Ok
bool	auto_close	if True window will close itself
int or float	auto_close_duration	Older versions only accept int. Time in seconds until window will close
(int, int)	size	(w,h) w=characters-wide, h=rows-high
(int, int)	location	Location on the screen to place the upper left corner of the window
bool	non_blocking	if True the call will immediately return rather than waiting on user input
str	background_color	color of background
str	text_color	color of the text

Type	Name	Meaning
bool	no_titlebar	If True no titlebar will be shown
bool	grab_anywhere	If True, than can grab anywhere to move the window (Default = False)
bool	keep_on_top	If True the window will remain above all current windows
(str or (str, int[, str]) or None)	font	specifies the font family, size, etc. Tuple or Single string format 'name size styles'. Styles: italic * roman bold normal underline overstrike
str or bytes	image	Image to include at the top of the popup window
bytes or str	icon	filename or base64 string to be used for the window's icon
bool	modal	If True then makes the popup will behave like a Modal window... all other windows are non-operational until this one is closed. Default = True
bool	no_sizegrip	If True no Sizegrip will be shown when there is no titlebar. It's only shown if there is no titlebar
str or None or TIMEOUT_KEY	**RETURN**	Returns text of the button that was pressed. None will be returned if user closed window with X

```
popup_get_text(message,
    title = None,
    default_text = "",
    password_char = "",
    size = (None, None),
    button_color = None,
    background_color = None,
    text_color = None,
    icon = None,
    font = None,
    no_titlebar = False,
    grab_anywhere = False,
    keep_on_top = None,
    location = (None, None),
    image = None,
    modal = True)
```

10_3_0_09.py

Description des paramètres :

Type	Name	Meaning
str	message	message displayed to user
str	title	Window title
str	default_text	default value to put into input area
str	password_char	character to be shown instead of actually typed characters
(int, int)	size	(width, height) of the InputText Element
(str, str) or str	button_color	Color of the button (text, background)
str	background_color	background color of the entire window
str	text_color	color of the message text
bytes or str	icon	filename or base64 string to be used for the window's icon
(str or (str, int[, str]) or None)	font	specifies the font family, size, etc. Tuple or Single string format 'name size styles'. Styles: italic * roman bold normal underline overstrike
bool	no_titlebar	If True no titlebar will be shown

Type	Name	Meaning
bool	grab_anywhere	If True can click and drag anywhere in the window to move the window
bool	keep_on_top	If True the window will remain above all current windows
(int, int)	location	(x,y) Location on screen to display the upper left corner of window
str or bytes	image	Image to include at the top of the popup window
bool	modal	If True then makes the popup will behave like a Modal window... all other windows are non-operational until this one is closed. Default = True
str or None	**RETURN**	Text entered or None if window was closed or cancel button clicked

```
popup_get_file(message,
    title = None,
    default_path = "",
    default_extension = "",
    save_as = False,
    multiple_files = False,
    file_types = (('ALL Files', '*.*'),),
    no_window = False,
    size = (None, None),
    button_color = None,
    background_color = None,
    text_color = None,
    icon = None,
    font = None,
    no_titlebar = False,
    grab_anywhere = False,
    keep_on_top = None,
    location = (None, None),
    initial_folder = None,
    image = None,
    files_delimiter = ";",
    modal = True,
    history = False,
    show_hidden = True,
    history_setting_filename = None)
```

10_3_0_11.py

Description des paramètres :

Type	Name	Meaning
str	message	message displayed to user
str	title	Window title
str	default_path	path to display to user as starting point (filled into the input field)
str	default_extension	If no extension entered by user, add this to filename (only used in saveas dialogs)
bool	save_as	if True, the "save as" dialog is shown which will verify before overwriting
bool	multiple_files	if True, then allows multiple files to be selected that are returned with ';' between each filename
Tuple[Tuple[str,str]]	file_types	List of extensions to show using wildcards. All files (the default) = (("ALL Files", "."),)

Type	Name	Meaning
bool	no_window	if True, no PySimpleGUI window will be shown. Instead just the tkinter dialog is shown
(int, int)	size	(width, height) of the InputText Element or Combo element if using history feature
(str, str) or str	button_color	Color of the button (text, background)
str	background_color	background color of the entire window
str	text_color	color of the text
bytes or str	icon	filename or base64 string to be used for the window's icon
(str or (str, int[, str]) or None)	font	specifies the font family, size, etc. Tuple or Single string format 'name size styles'. Styles: italic * roman bold normal underline overstrike
bool	no_titlebar	If True no titlebar will be shown
bool	grab_anywhere	If True: can grab anywhere to move the window (Default = False)
bool	keep_on_top	If True the window will remain above all current windows
(int, int)	location	Location of upper left corner of the window
str	initial_folder	location in filesystem to begin browsing
str or bytes	image	Image to include at the top of the popup window
str	files_delimiter	String to place between files when multiple files are selected. Normally a ;
bool	modal	If True then makes the popup will behave like a Modal window... all other windows are non-operational until this one is closed. Default = True
bool	history	If True then enable a "history" feature that will display previous entries used. Uses settings filename provided or default if none provided

Type	Name	Meaning
bool	show_hidden	If True then enables the checkbox in the system dialog to select hidden files to be shown
str	history_setting_filename	Filename to use for the User Settings. Will store list of previous entries in this settings file
str or None	**RETURN**	string representing the file(s) chosen, None if cancelled or window closed with X

```
popup_get_folder(message,
    title = None,
    default_path = "",
    no_window = False,
    size = (None, None),
    button_color = None,
    background_color = None,
    text_color = None,
    icon = None,
    font = None,
    no_titlebar = False,
    grab_anywhere = False,
    keep_on_top = None,
    location = (None, None),
    initial_folder = None,
    image = None,
    modal = True,
    history = False,
    history_setting_filename = None)
```

Description des paramètres :

Type	Name	Meaning
str	message	message displayed to user
str	title	Window title
str	default_path	path to display to user as starting point (filled into the input field)
bool	no_window	if True, no PySimpleGUI window will be shown. Instead just the tkinter dialog is shown
(int, int)	size	(width, height) of the InputText Element
(str, str) or str	button_color	button color (foreground, background)
str	background_color	color of background
str	text_color	color of the text
bytes or str	icon	filename or base64 string to be used for the window's icon

Type	Name	Meaning
(str or (str, int[, str]) or None)	font	specifies the font family, size, etc. Tuple or Single string format 'name size styles'. Styles: italic * roman bold normal underline overstrike
bool	no_titlebar	If True no titlebar will be shown
bool	grab_anywhere	If True: can grab anywhere to move the window (Default = False)
bool	keep_on_top	If True the window will remain above all current windows
(int, int)	location	Location of upper left corner of the window
str	initial_folder	location in filesystem to begin browsing
str or bytes	image	Image to include at the top of the popup window
bool	modal	If True then makes the popup will behave like a Modal window... all other windows are non-operational until this one is closed. Default = True
bool	history	If True then enable a "history" feature that will display previous entries used. Uses settings filename provided or default if none provided
str	history_setting_filename	Filename to use for the User Settings. Will store list of previous entries in this settings file
str or None	**RETURN**	string representing the path chosen, None if cancelled or window closed with X

7. Bibliographie

https://github.com/PySimpleGUI/PySimpleGUI/blob/master/docs/PySimpleGUI%202020%20Lesson.pdf

https://pysimplegui.readthedocs.io/en/latest/