Langage Python

Interfaces graphiques - Tkinter

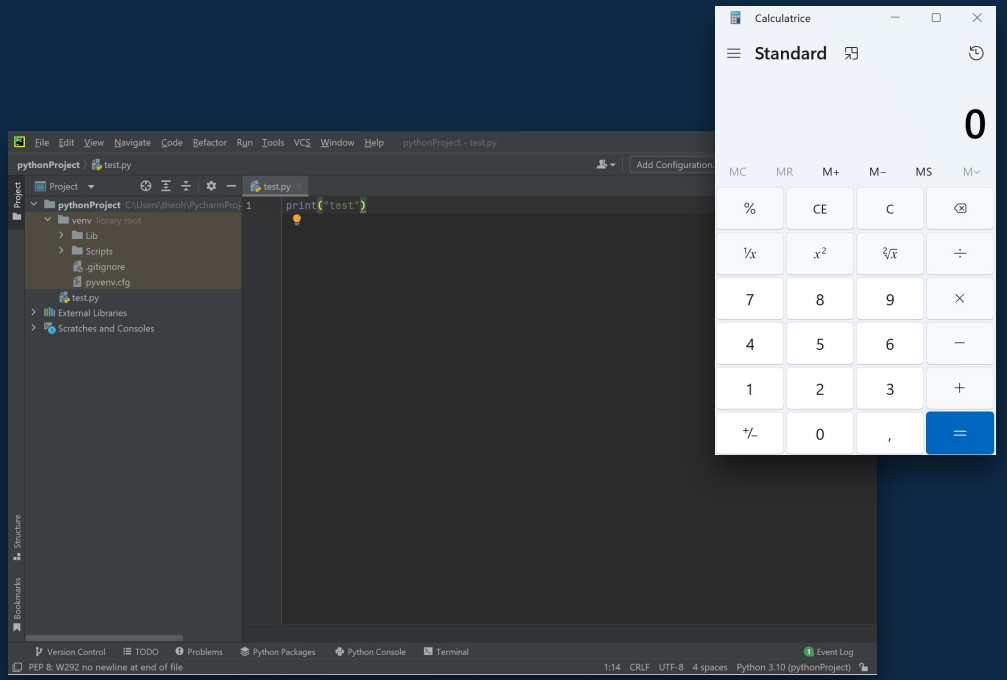
Formation POEC Cybersécurité

Théo Hubert

Concept

- *Interface graphique*

L'interface graphique désigne la **manière dont est présenté un logiciel à l'écran** pour l'utilisateur. C'est le positionnement des éléments : menus, boutons, fonctionnalités dans la fenêtre. Une interface graphique bien conçue est ergonomique et intuitive afin que l'utilisateur la comprenne tout de suite.



Concept

- *Interface graphique*

Python comprend une large gamme d'implémentations d'interface disponibles, de **Tkinter** (il est livré avec Python) à une variété de solutions multiplateformes, telles que **PyQt5**, qui est connue pour ses widgets plus sophistiqués et son look élégant.

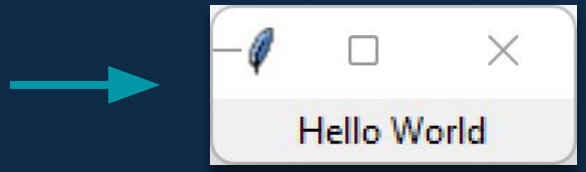
Syntaxe

- *Introduction - Tkinter*

Tkinter est un module intégré à Python pour développer des applications graphiques. Ce module se base sur la bibliothèque graphique Tcl/Tk.

Premier programme :

```
from tkinter import *

fenetre = Tk()

label = Label(fenetre, text="Hello World")
label.pack()

fenetre.mainloop()
```



Syntaxe

- *Les objets - Tkinter*

Les interfaces graphiques sont composées **d'objets aussi appelés widgets ou contrôles**. Ils reçoivent des événements, en un sens, ce sont ces objets qui pilotent un programme ou qui le contrôlent.

Par exemple, ci-dessous on fait créer un objet label qui permet de créer une zone de texte

```
zone_texte = tkinter.Label (text = "zone de texte")
```

Voici la démarche complète pour faire apparaître le label au sein de la GUI.

```
import tkinter          # import de tkinter
root = tkinter.Tk ()    # création de la fenêtre principale
# ...
obj = tkinter.Label (text = "zone de texte")
# ...
obj.pack ()             # on ajoute l'objet à la fenêtre principale
root.mainloop ()        # on affiche enfin la fenêtre principal et on attend
                        # les événements (souris, clic, clavier)
```

Syntaxe

- *Window - Widget*

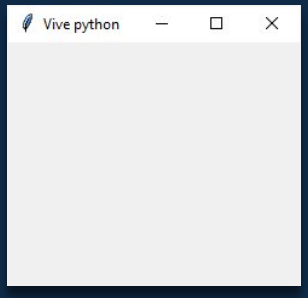
La fenêtre est le widget qui permet de créer la fenêtre de l'interface graphique. Le nom par défaut d'une fenêtre est tk.

Voici la syntaxe pour créer ce widget :

```
from tkinter import *

window = Tk()
```

Remarque : Pour changer le nom d'une fenêtre il faut faire appel à la fonction **title()**

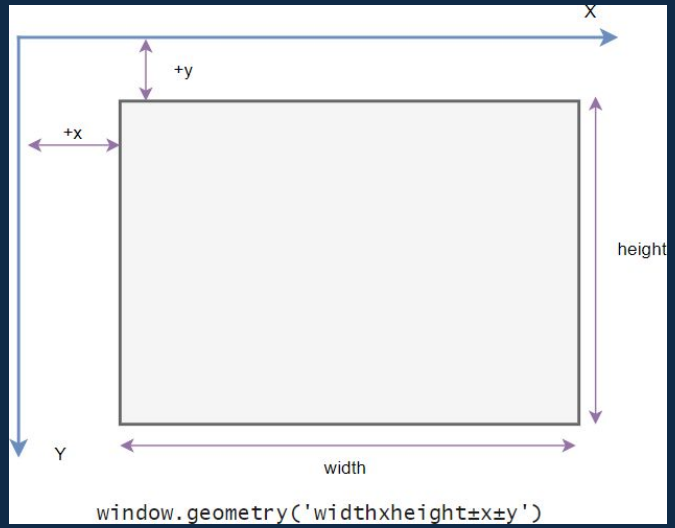
```
from tkinter import *

root = Tk()
root.title("Vive python")

root.mainloop()
```



Syntaxe

- *Window - Widget*

Remarque : Pour modifier la taille et la position de la fenêtre il faut faire appel à la fonction **geometry()**

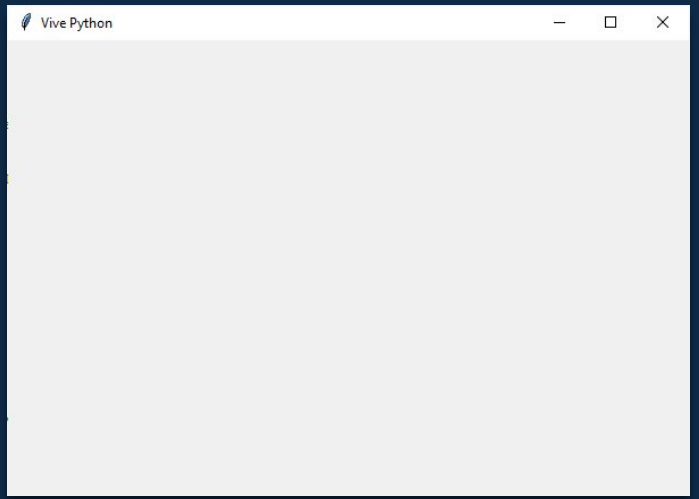


Syntaxe

- *Window - Widget*

Remarque : Pour modifier la taille et la position de la fenêtre il faut faire appel à la fonction **geometry()**

```
from tkinter import *

root = Tk()

root.title('Vive Python')
root.geometry('600x400+50+50')

root.mainloop()
```

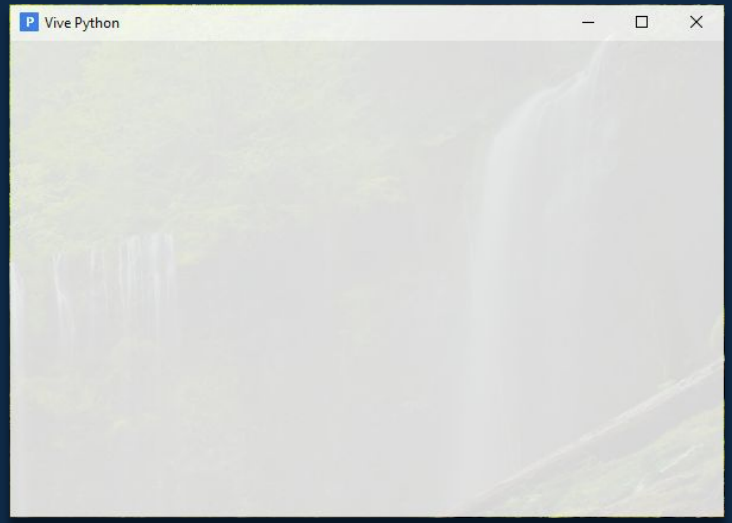


Syntaxe

- *Window - Widget*

Remarque : Il également possible de modifier la transparence d'une fenêtre grâce à la fonction **attributes()** ou de modifier son icône grâce à la fonction **iconbitmap()…**

```
from tkinter import *

root = Tk()

root.title('Vive Python')
root.geometry('600x400+50+50')

root.attributes('-alpha', 0.9)
root.iconbitmap('pythontutorial.ico')

root.mainloop()
```



Syntaxe

- *Label - Widget*

Un Label est un widget Tkinter standard utilisé pour **afficher un texte ou une image** à l'écran. Un Label ne peut afficher du texte que dans une seule police. Le texte affiché par ce widget peut être mis à jour à tout moment.

Il est également possible de souligner une partie du texte et d'afficher le texte sur plusieurs lignes

Voici la syntaxe pour créer ce widget :

```
l = Label(master, option = value, ...)
```

- **master** : **Fenêtre parent**.
- **options** : Liste des **options** les plus couramment utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules.

Syntaxe

- *Label - Widget*

Exemples :

```
gui = Tk()
label = Label(gui, text="Ceci est un test")
label.pack()
gui.mainloop()
```


```
gui = Tk()
label = Label(gui, text="Ceci est un test")
label.pack()
gui.mainloop()
```

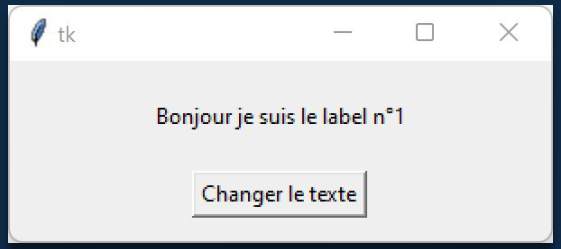

```
gui = Tk()
label = Label(gui, text="Ceci est un test")
label.pack()
gui.mainloop()
```

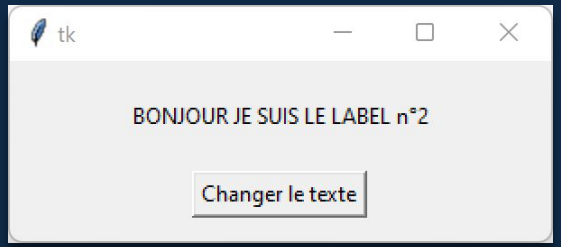
Syntaxe

- *Label - Widget*

En utilisant **StringVar()** il est possible de faire varier le contenu d'un label.

```
import tkinter as tk

def changeText():
  text.set("BONJOUR JE SUIS LE LABEL n°2")

gui = tk.Tk()
gui.geometry('300x100')

text = tk.StringVar()
text.set("Bonjour je suis le label n°1")

label = tk.Label(gui, textvariable=text)
label.pack(pady=20)

button = tk.Button(gui, text="Changer le texte", command=changeText)
button.pack()

gui.mainloop()
```





Syntaxe

- *Label - Widget*

bitmap	Définissez cette option sur un objet bitmap ou image et le Label affichera ce graphique.
bd	Largeur de bordure en pixels. La valeur par défaut est 2.
bg	Couleur de fond normale.
cursor	Si vous définissez cette option sur un nom de curseur (arrow, dot etc.), le curseur de la souris changera pour ce modèle lorsqu'il se trouvera sur le label.
fg	Couleur normale du premier plan (texte).
font	Police de texte à utiliser pour le label.
height	Hauteur du Label selon les lignes de texte (pour les Labels textuels) ou en pixels (pour les images).
text	Pour afficher une ou plusieurs lignes de texte dans un widget Label, définissez cette option sur une chaîne contenant le texte. Le caractère (« \n ») forceront un saut de ligne.
image	Image à afficher sur le label (au lieu du texte).
justify	• **LEFT** pour justifier le texte à gauche de chaque ligne; • **CENTER** pour centrer le texte; • **RIGHT** pour justifier le texte à droite.

Source : https://waytolearnx.com/2020/06/label-tkinter-python-3.html

Syntaxe

- *Button - Widget*

Un bouton a pour but de faire le lien entre une fonction et un clic de souris. Pour créer un bouton il faut créer un objet à partir de la classe **Button**. Il est possible d'attacher une fonction ou une méthode à un bouton qui est appelé automatiquement lorsque vous cliquez sur le bouton, pour cela il faut utiliser l'argument **command**.

Voici la syntaxe pour créer ce widget :

```
btn = Button(master, option = value, ...)
```

- **master** : **Fenêtre parent**.
- **options** : Liste des **options** les plus couramment utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules.

Syntaxe

- *Button - Widget*

Exemples :

```
from tkinter import *

fenetre = Tk ()    # création de la fenêtre principale

bouton=Button(fenetre, text="Fermer", command=fenetre.quit)
bouton.pack()

fenetre.mainloop ()
```


```
from tkinter import *

fenetre = Tk ()

def msgCallBack():
   messagebox.showinfo("Alerte", "Attention il y a eu une intrustion!")
btn = tkinter.Button(fenetre, text ="Cliquez ici!", command = msgCallBack)

btn.pack()
gui.mainloop()
```

Syntaxe

- *Button - Widget*

activebackground	Couleur d'arrière-plan lorsque le bouton est sous le curseur.
activeforeground	Couleur du premier plan lorsque le bouton est sous le curseur.
bd	Largeur de bordure en pixels. La valeur par défaut est 2.
bg	Couleur de fond normale.
command	Fonction ou méthode à appeler lorsqu'on clique sur le bouton.
fg	Couleur normale du premier plan (texte).
font	Police de texte à utiliser pour l'étiquette du bouton.
height	Hauteur du bouton selon les lignes de texte (pour les boutons textuels) ou en pixels (pour les images).
highlightcolor	La couleur du focus lorsque le widget a le focus.
image	Image à afficher sur le bouton (au lieu du texte).
justify	• **LEFT** pour justifier le texte à gauche de chaque ligne; • **CENTER** pour centrer le texte; • **RIGHT** pour justifier le texte à droite.
width	Largeur du bouton selon les lettres (si vous affichez du texte) ou en pixels (si vous affichez une image).

Source : https://waytolearnx.com/2020/06/button-tkinter-python-3.html

Syntaxe

- *Entry - Widget*

Entry est un widget utilisé pour permettre à l'utilisateur de **saisir une chaîne de caractères**. Ce widget permet à l'utilisateur de saisir une ligne de texte.

Voici la syntaxe pour créer ce widget :

```
e = Entry(master, option = value, ...)
```

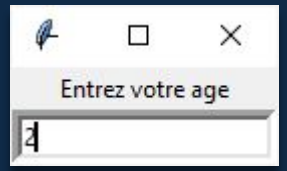
- **master** : **Fenêtre parent**.
- **options** : Liste des **options** les plus couramment utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules.

Syntaxe

- *Entry - Widget*

Exemples :

```
gui = Tk()
l = Label(gui, text = "Entrez votre age")
e = Entry( gui, bd = 5)
l.pack()
e.pack()
gui.mainloop()
```




```
gui = Tk()
l = Label(gui, text = "Entrez votre age : ")
e = Entry( gui, bd = 5)
l.pack(side=LEFT)
e.pack(side=RIGHT)
gui.mainloop()
```


```
gui = Tk()
l = Label(gui, text = "Entrez votre age : ")
e = Entry( gui, bd = 5, fg="green", bg="black")
l.pack(side=LEFT)
e.pack(side=RIGHT)
gui.mainloop()
```

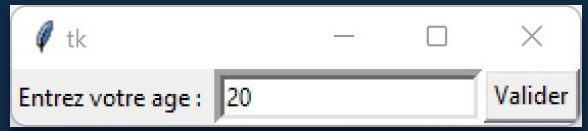
Syntaxe

- *Entry - Widget*

Exemples :

Le widget Entry possède la méthode **get()** pour récupérer la valeur saisi par l'utilisateur.

```
from tkinter import *

gui = Tk()

def getEntry():
    print(entry.get())
    print(type(entry.get()))

l1 = Label(gui, text="Entrez votre age : ")
frame = Frame(gui)
entry = Entry(frame, bd=5)
b1 = Button(frame, text="Valider", command=getEntry)

l1.pack(side=LEFT)
frame.pack(side=RIGHT)
entry.pack(side=LEFT)
b1.pack(side=RIGHT)

gui.mainloop()
```




```
==== RESTART: C:
20
<class 'str'>
20
<class 'str'>
20
<class 'str'>
20
<class 'str'>
```

Syntaxe

- *Entry - Widget*

bd	Largeur de bordure en pixels. La valeur par défaut est 2.
bg	Couleur de fond normale.
command	Fonction ou méthode à appeler à chaque fois que l'utilisateur modifie l'état du widget Entry.
fg	Couleur normale du premier plan (texte).
font	Police de texte à utiliser pour le widget Entry.
height	Hauteur du bouton selon les lignes de texte (pour les boutons textuels) ou en pixels (pour les images).
highlightcolor	La couleur du focus lorsque le widget a le focus.
justify	• **LEFT** pour justifier le texte à gauche de chaque ligne; • **CENTER** pour centrer le texte; • **RIGHT** pour justifier le texte à droite.
exportselection	Par défaut, si vous sélectionnez du texte dans un widget Entry, il est automatiquement exporté vers le presse-papiers. Pour éviter cette exportation, utilisez exportselection = 0.

Source : https://waytolearnx.com/2020/06/entry-tkinter-python-3.html

Syntaxe

- *ListBox - Widget*

Listbox est un widget qui permet d'**afficher une liste d'éléments** à partir de laquelle un utilisateur peut **sélectionner** un certain nombre d'éléments.

Voici la syntaxe pour créer ce widget :

```
liste = Listbox(master, option = value, ...)
```

- **master** : **Fenêtre parent**.
- **options** : Liste des **options** les plus couramment utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules.

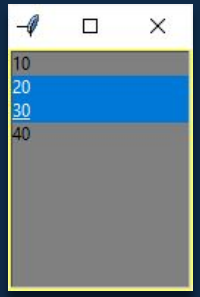
Syntaxe

- *ListBox - Widget*

Exemples :

```
gui = Tk()
liste = Listbox(gui)
liste.insert(1, "Bleu")
liste.insert(2, "Rouge")
liste.insert(3, "Vert")
liste.insert(4, "Jaune")
liste.insert(5, "Orange")
liste.insert(6, "Noir")
liste.pack()
gui.mainloop()
```


```
gui = Tk()
liste = Listbox(gui, selectmode=MULTIPLE, highlightcolor="yellow", bg="grey")
liste.insert(1, 10)
liste.insert(2, 20)
liste.insert(3, 30)
liste.insert(4, 40)
liste.pack()
gui.mainloop()
```



Syntaxe

- *ListBox - Widget*

bg	Couleur de fond normale.
fg	Couleur normale du premier plan (texte).
font	Police de texte à utiliser pour la liste.
height	Nombre de lignes (pas de pixels!) Affichées dans la zone de liste. La valeur par défaut est 10.
highlightcolor	La couleur du focus lorsque le widget a le focus.
selectmode	Détermine le nombre d'éléments pouvant être sélectionnés et la manière dont les éléments sont sélectionnés: • **BROWSE** Vous ne pouvez sélectionner qu'une seule ligne dans la liste. Si vous cliquez sur un élément, puis faites glisser vers une autre ligne, la sélection suivra la souris. C'est la valeur par défaut. • **SINGLE** Vous ne pouvez sélectionner qu'une seule ligne et vous ne pouvez pas faire glisser la souris. • **MULTIPLE** Vous pouvez sélectionner n'importe quel nombre de lignes à la fois. Cliquer sur une ligne permet de choisir si elle est sélectionnée ou non. • **EXTENDED** Vous pouvez sélectionner n'importe quel groupe de lignes adjacentes à la fois en cliquant sur la première ligne et en faisant glisser jusqu'à la dernière ligne.

Source : https://waytolearnx.com/2020/06/listbox-tkinter-python-3.html

Syntaxe

- *Frames - Widget*

Le widget Frame est très important. Il permet de **regrouper et d'organiser d'autres widgets** d'une manière simple. Il fonctionne comme un **conteneur**, qui est responsable de l'organisation de la position des autres widgets.

Il utilise des zones rectangulaires à l'écran pour organiser la mise en page de ces widgets.

Voici la syntaxe pour créer ce widget :

```
frame = Frame(master, option = value, ...)
```

- **master** : **Fenêtre parent**.
- **options** : Liste des **options** les plus couramment utilisées pour ce widget. Ces options peuvent être utilisées sous forme de paires clé-valeur séparées par des virgules.

Syntaxe

- *Frame - Widget*

Exemple :

```
root = Tk()
frame = Frame(root)
frame.pack()

bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )

redbutton = Button(frame, text="Rouge", bg="red", fg="white")
redbutton.pack( side = LEFT)

greenbutton = Button(frame, text="Marron", bg="brown", fg="white")
greenbutton.pack( side = LEFT )

bluebutton = Button(frame, text="Bleu", bg="blue", fg="white")
bluebutton.pack( side = LEFT )

blackbutton = Button(bottomframe, text="Noir", bg="black", fg="white")
blackbutton.pack( side = BOTTOM)

root.mainloop()
```

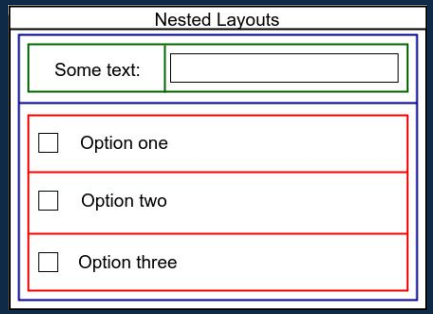
Syntaxe

- *Layout*

Lorsque l'on créer une interface graphique on utilise des **layouts** et un layout manager. Les layouts sont des **objets non visibles** qui permettent d'organiser et de **positionner** au bon endroits les différents **widgets/composants** de notre interface.

Tkinter comprend 3 layouts managers "built-in" :

- **pack**
- **grid**
- **place**



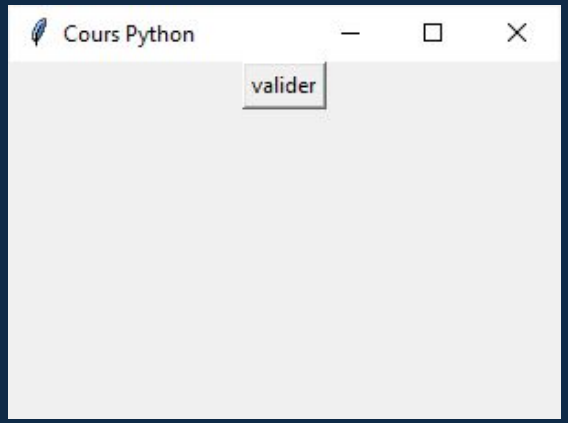
Syntaxe

- *Layout - pack*

pack est l'un des layout manager les plus utilisé.

- Il place automatiquement les widgets dans la fenêtre en fonction de l'espace disponible.
- Il organise les widgets dans des blocs horizontaux et verticaux.

```
from tkinter import *

window = Tk()

window.title('Cours Python')
window.geometry('300x200+50+50')
btn = Button(window, text="valider")
btn.pack()

window.mainloop()
```



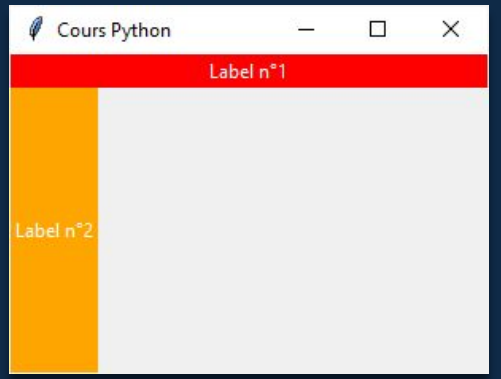
Syntaxe

- *Layout - pack*

pack() possède plusieurs attributs optionnels :

- side : permet de positionner un widget à l'aide des mots clés suivants LEFT, RIGHT, TOP, BOTTOM
- fill : permet de forcer un widget à remplir un espace vertical ou horizontale ou les 2 (X, Y, BOTH).
- padx, pady : permet de décider de l'espacement horizontal et vertical entre les widgets
- ipadx, ipady : permet de décider de l'espacement horizontal et vertical entre le label et le bord du widget.

```
window = Tk()

window.title('Cours Python')
window.geometry('300x200+50+50')

label1 = Label(window, text="Label n°1", fg="white", bg="red")
label1.pack(fill=X)

label2 = Label(window, text="Label n°2", fg="white", bg="orange")
label2.pack(fill=Y, side=LEFT)

window.mainloop()
```



Syntaxe

- *Layout - pack*

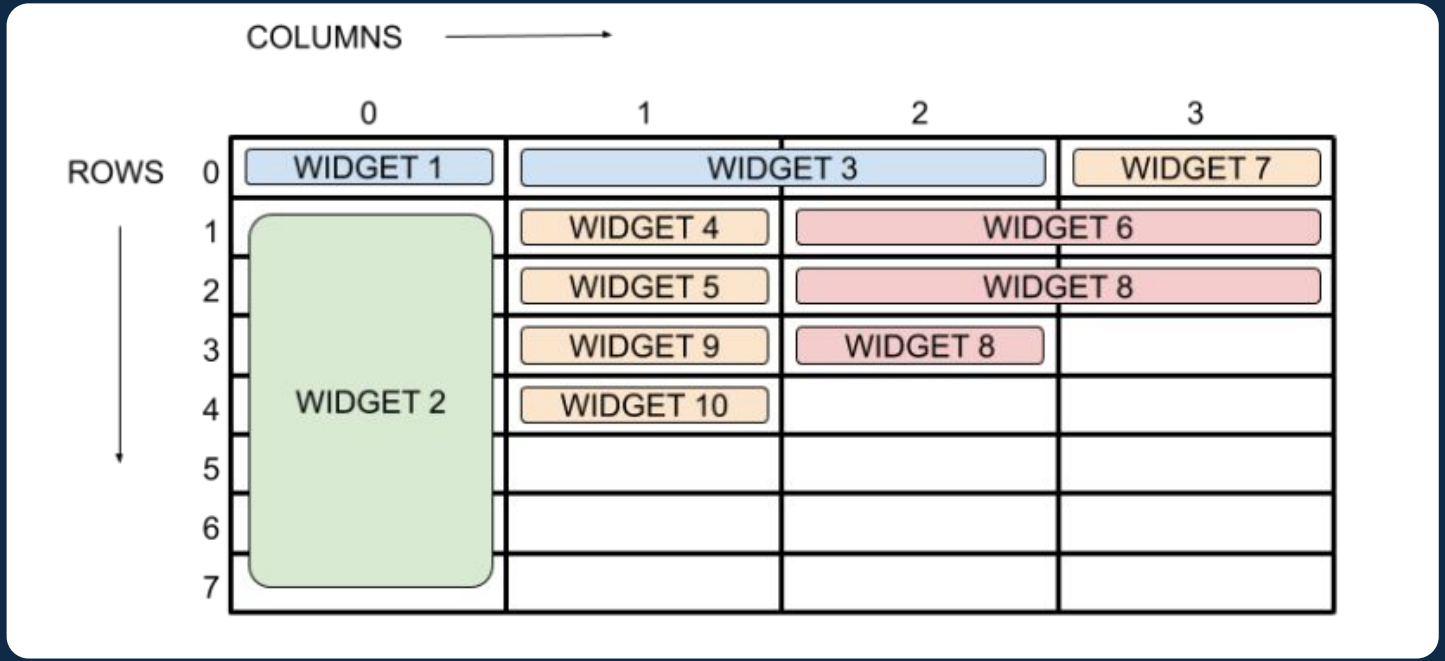
grid est également l'un des layout manager les plus utilisé. Il permet de positionner les widgets and utilisant un système géométrique en 2 dimensions avec à la fois des lignes et des colonnes.

Tout comme pack il possède des arguments optionnels :

- row, column : Permet de préciser l'index de la ligne et de la colonne.
- padx, pady : Permet de préciser l'espace vertical et horizontal
- columnspan : Permet de fusionner des colonnes
- rowspan : Permet de fusionner des lignes

Syntaxe

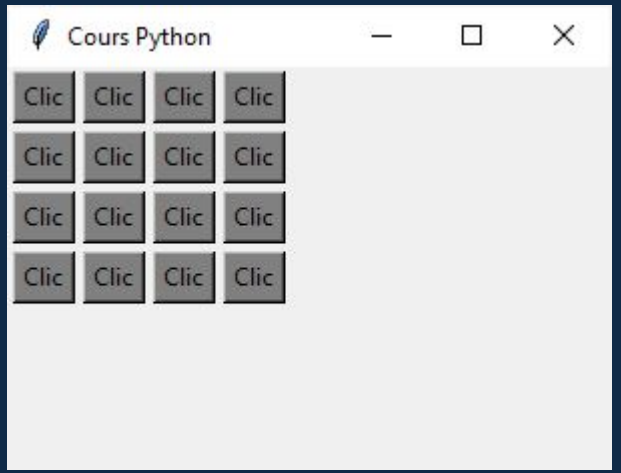
- *Layout - pack*



Syntaxe

- *Layout - pack*

```
from tkinter import *

window = Tk()

window.title('Cours Python')
window.geometry('300x200+50+50')

for i in range(4):
    for j in range(4):
        button = Button(window, text="Clic", bg="grey")
        button.grid(padx=2, pady=2, row=i, column=j)

window.mainloop()
```



Syntaxe

Travaux Pratique

Références

Histoire de la POO :
https://bpesquet.developpez.com/tutoriels/csharp/programmation-orientee-objet-csharp/?page=initiation-a-la-programmation-orientee-objet
https://www.jedha.co/blog/quest-ce-que-la-programmation-orientee-objet

POO/Procédurale :
https://waytolearnx.com/2018/09/difference-entre-programmation-procedurale-et-orientee-objet.html
https://practicalprogramming.fr/paradigme-programmation-orientee-objet-et-programmation-fonctionnelle

Intérêt de la POO :
https://www.powerpress.fr/ads/avantages-de-la-programmation-orientee-objet/

Classes abstraites :
https://pythonforge.com/classes-abstraites-en-python/#:~:text=Les%20classes%20abstraites%20sont%20des,'a%20pas%20d'impl%C3%A9mentation.

Fin

La suite :
Programmation Orienté Objet