FICHE PYTHON 13 :
INTERFACE GRAPHIQUE 2 - PLACEMENT (4 pages)

Bibliothèque à importer pour créer des applications graphiques	`#!/usr/bin/env python`
	`# -*- coding: utf-8 -*-`
	`from tkinter import *`
	.

Création d'une fenêtre	`fen_princ = Tk()`
	`# Insérer ici vos différents widgets`
	`fen_princ.mainloop()`

Modification du titre et de la taille de la fenêtre	`fenetre = Tk()`
	`fenetre.title("NOM DE VOTRE PROGRAMME")`
	`fenetre.geometry("500x500")`

Destruction d'une fenêtre	`fenetre.destroy()`

Méthode pack()

Code de base	`bou1.pack()` La méthode **pack()** est rapide à utiliser mais pas pratique avec plusieurs widgets.
`pack(side=LEFT)`	SI vous voulez placer le widget sur le côté gauche de la fenêtre.
`pack(side=RIGHT)`	SI vous voulez placer le widget sur le côté droit de la fenêtre.
`pack(side=TOP)`	SI vous voulez placer le widget en début de fenêtre.
`pack(side=BOTTOM)`	SI vous voulez placer le widget en fin de fenêtre.
`pack(fill=NONE)`	Pour ne pas utiliser le remplissage. Valeur par défaut.
`pack(fill=X)`	Pour utiliser le remplissage maximum sur la place disponible horizontalement
`pack(fill=Y)`	Pour utiliser le remplissage maximum sur la place disponible verticalement
`pack(fill=BOTH)`	Pour prendre le reste de la place disponible

Méthode grid()

Code de base	`bou1.grid()` Le principe est simple : on crée des lignes (**`row`**) et des colonnes (**`column`**). La taille des cellules n'est pas uniforme, elles sont fixées en fonction de la place disponible et des arguments transmis. **Numérotation** : Comme dans la plupart des cas, le premier élément n'est pas l'élément 1 mais l'élément 0.

Exemple	`bou1.grid(row=0,column=0)`
	`bou2.grid(row=0,column=1)`
	`bou3.grid(row=1,column=0)`
	`bou4.grid(row=1,column=1)`
	 

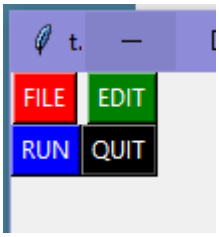
sticky

Ce paramètre permet d'ancrer le widget de façon un peu plus précise.

Il faut utiliser les 4 points cardinaux en anglais (N (North), S(South), W(West) ou E(East)) ou une combinaison des 4 : N+E, S+W ...

Ainsi, **`grid(sticky=E)`** ou **`grid(sticky=W+N)`** sont valides.

```
bou1.grid(row=0,column=0)
bou2.grid(row=0,column=1,sticky=E)
bou3.grid(row=1,column=0)
bou4.grid(row=1,column=1)
```



On remarquera que EDIT est maintenant parfaitement aligné à droite par rapport à QUIT.

rowspan

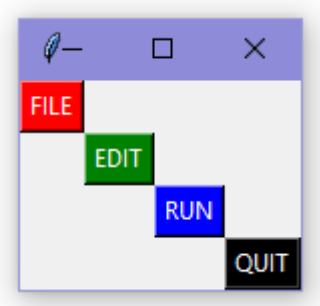
columnspan

On peut imposer à un widget de prendre un peu plus de place si nécessaire.

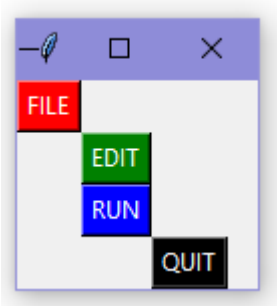
Le paramètre nommé **rowspan** indique le nombre de lignes sur lesquelles un widget peut s'étaler.

L'attribut **columnspan** fait la même chose pour les colonnes.

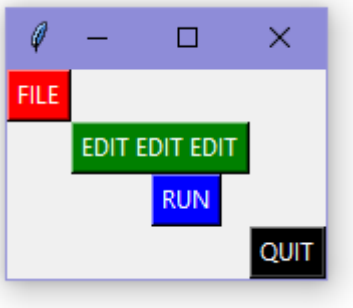
```
bou1.grid(row=0,column=0)
bou2.grid(row=1,column=1)
bou3.grid(row=2,column=2)
bou4.grid(row=3,column=3)
```




```
bou1.grid(row=0,column=0)
bou2.grid(row=1,column=1,columnspan=2)
bou3.grid(row=2,column=2)
bou4.grid(row=3,column=3)
```

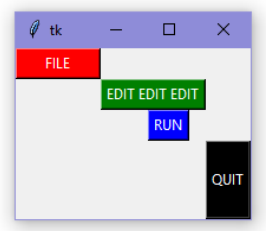


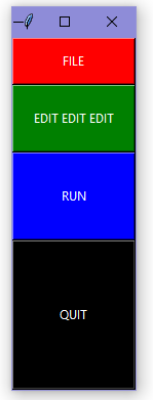

```
bou1.grid(row=0,column=0)
bou2=Button(fenetre, text="EDIT EDIT
EDIT", fg="white", bg="green", command =
boutonFourreTout)
bou3.grid(row=2,column=2)
bou4.grid(row=3,column=3)
```



Les marges internes : ipadx et ipady pour pack() et grid()

Présentation	Pour augmenter la taille interne du widget, nous allons utiliser `ipadx=10` pour créer 10 pixels de plus horizontalement et `ipady=10` pour créer 10 pixels de plus verticalement.

	`bou1.grid(row=0,column=0, ipadx=20)`
	`bou2.grid(row=1,column=1,columnspan=2)`
	`bou3.grid(row=2,column=2)`
	`bou4.grid(row=3,column=3,ipady=20)`

	bou1.pack(fill=X, ipady=10) bou2.pack(fill=X, ipady=20) bou3.pack(fill=X, ipady=30) bou4.pack(fill=X, ipady=60)

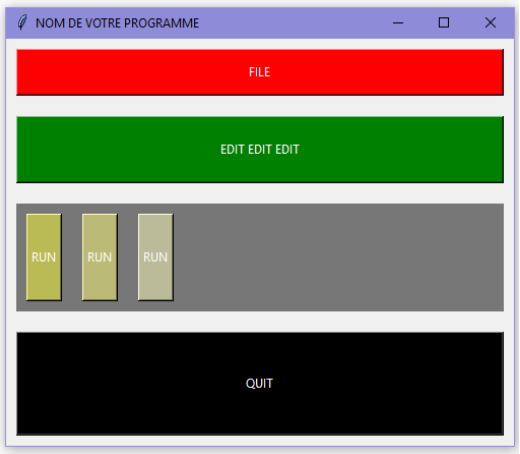
Les marges externes : padx et pady pour pack() et grid()

Présentation	C'est exactement la même syntaxe que avec ipad mais sans le i pour interne : cette fois, on écarte les autres widgets en les repoussant. Mais la marge d'espace entre les deux widgets n'aura pas la couleur de background de widget mais celle de la fenêtre elle-même

	`bou1.pack(fill=X, ipady=10, padx=10,pady=10)`
	`bou2.pack(fill=X, ipady=20, padx=10,pady=10)`
	`bou3.pack(fill=X, ipady=30, padx=10,pady=10)`
	`bou4.pack(fill=X, ipady=60, padx=10,pady=10)`

Les cadres : les frames

Présentation	Un cadre (frame) consiste à créer une sous-zone dans votre fenêtre. Vous pourrez placer votre cadre et le remplir ensuite à votre manière.

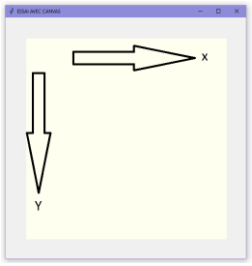


Si on lit le code ci-contre, on voit que :

- On crée un Frame nommé **zone2**.
- On y rattache bou3a.
- On y rattache bou3b.
- On y rattache bou3c.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from tkinter import *
# DEFINITIONS DES FONCTIONS
def boutonFourreTout():
    fenetre.destroy()
    return(0)
# CORPS PRINCIPAL DU PROGRAMME
fenetre = Tk()
fenetre.title("NOM DE VOTRE PROGRAMME")
fenetre.geometry("500x400")

bou1 = Button(fenetre, text="FILE", fg="white",
bg="red", command = boutonFourreTout)
bou2 = Button(fenetre, text="EDIT EDIT EDIT",
fg="white", bg="green", command = boutonFourreTout)

zone2 = Frame(fenetre, bg='#777777')
bou3a = Button(zone2, text="RUN", fg="white",
bg="#BBBB55", command = boutonFourreTout)
bou3b = Button(zone2, text="RUN", fg="white",
bg='#BBBB77', command = boutonFourreTout)
bou3c = Button(zone2, text="RUN", fg="white",
bg='#BBBB99', command = boutonFourreTout)

bou4 = Button(fenetre, text="QUIT", fg="white",
bg="black", command = boutonFourreTout)
bou1.pack(fill=X, ipady=10, padx=10,pady=10)
bou2.pack(fill=X, ipady=20, padx=10,pady=10)
zone2.pack(fill=Y, padx=10,pady=10)
bou3a.pack(side=LEFT, fill=Y, ipady=30,
padx=10,pady=10)
bou3b.pack(side=LEFT, fill=Y, ipady=30,
padx=10,pady=10)
bou3c.pack(side=LEFT, fill=Y, ipady=30,
padx=10,pady=10)
bou4.pack(fill=X, ipady=60, padx=10,pady=10)

fenetre.mainloop()
```

Méthode place() qui s'applique à tous les widgets, donc aux canvas également.

place() *à la place de la méthode pack()* 	Plutôt que pack() ou grid(), on peut utiliser cette méthode qui va permet de placer votre widget au pixel près. `monWidget.place(x=100,y=100)`