

TD 10 : Découverte du module graphique Tkinter

Le module **Tkinter** de Python permet de créer des interfaces graphiques (GUI : Graphical User Interface). De nombreux composants graphiques (ou widgets) sont disponibles dans le module **Tkinter** : fenêtre (classe Tk), bouton (classe Button), case à cocher (classe Checkbutton), étiquette (classe Label), zone de texte simple (classe Entry), menu (classe Menu), zone graphique (classe Canvas), cadre (classe Frame)...

On peut également gérer de nombreux événements : clic sur la souris, déplacement de la souris, appui sur une touche du clavier, top d'horloge..

Exécuter les lignes de code PYTHON suivantes :

```
1  # -*- coding: utf-8 -*-
2
3  from tkinter import *
4
5  # Création de la fenêtre graphique
6  fenetre = Tk()
7  fenetre.title("Hello")
8  fenetre.geometry('640x480')
9  fenetre.resizable(width=False, height=False)
10
11 # Création des widgets
12 label = Label(fenetre, text="Hello World !")
16 bouton = Button(fenetre, text="Quitter", command=fenetre.destroy)
14 champ = Entry(fenetre)
15
16 # Positionnement des widgets sur la fenêtre graphique avec la méthode pack()
17 label.pack()
18 bouton.pack()
19 champ.pack()
20
21 # Lancement du gestionnaire d'évènements
22 fenetre.mainloop()
```

Analyse du programme :

- Q1. Expliquer le rôle de la ligne 3 du code ci-dessus ? *elle importe la lib de Tkinter*
- Q2. Expliquer le rôle des lignes 6, 7 et 8. *elles permettent de créer, mettre un titre et la dimensionner*
- Q3. Que se passe-t-il au niveau de l'affichage si on ne met pas la ligne 8 ? *la fenêtre est toute petite car selon nos défin.*
- Q4. Que se passe-t-il au niveau de l'affichage si on modifie la valeur des booléens à True dans l'instruction de la ligne 9 ? *cela permet de changer la taille de la fenêtre*
- Q5. Qu'est-ce qu'un widget ? Quels sont ceux utilisés dans le programme ci-dessus ? *permet l'affichage de diff*
- Q6. Quelle méthode est utilisée pour positionner les widgets ? *pack()* *le titre, le bouton, info*
- Q7. Que se passe-t-il si on oublie la ligne 22 ? *la fenêtre se lance pas* *le champ de texte quitter*

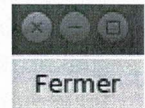
Connaître les widgets de base et savoir les positionner

Pour créer un logiciel graphique vous devez ajouter dans une fenêtre des éléments graphiques que l'on nomme widget (window gadget).

■ Widget Button()

Les **boutons** permettent de proposer une action à l'utilisateur. Dans l'extrait de code -dessous, on associe ci au **widget Button()** la fermeture de la fenêtre dans laquelle il s'exécute.

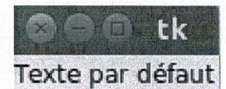
```
# Bouton de sortie
bouton=Button(fenetre, text="Fermer", command=fenetre.destroy)
bouton.pack()
```



■ Widget Label()

Les **labels** (étiquettes) sont des espaces prévus pour écrire du texte. Le **widget Label()** sert souvent à décrire un widget comme une entrée (voir ci-dessous Entry).

```
# Etiquette de texte
label = Label(fenetre, text="Texte par défaut", bg="yellow")
label.pack()
```

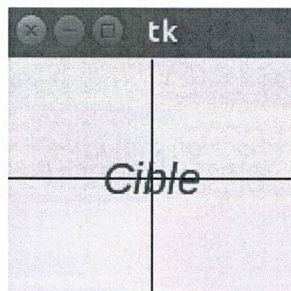


D'autres widgets sont décrits ici : <https://python.doctor/page-tkinter-interface-graphique-python-tutoriel>

■ Widget Canvas()

Le widget **Canvas()** (toile, tableau en français) est un espace dans lequel vous pouvez dessiner ou écrire ce que vous voulez.

```
# Canevas (toile)
canvas = Canvas(fenetre, width=150, height=120, background='yellow')
ligne1 = canvas.create_line(75, 0, 75, 120)
ligne2 = canvas.create_line(0, 60, 150, 60)
txt = canvas.create_text(75, 60, text="Cible", font="Arial 16 italic", fill="blue")
canvas.pack()
```



Vous pouvez créer d'autres éléments:

create_arc()	# arc de cercle
create_bitmap()	# bitmap
create_image()	# image
create_line()	# ligne
create_oval()	# ovale
create_polygon()	# polygone
create_rectangle()	# rectangle
create_text()	# texte
create_window()	# fenêtre

Si vous voulez changer les coordonnées d'un élément créé dans le canevas, vous pouvez utiliser la méthode **coords** :

```
canvas.coords(élément, x0, y0, x1, y1)
```

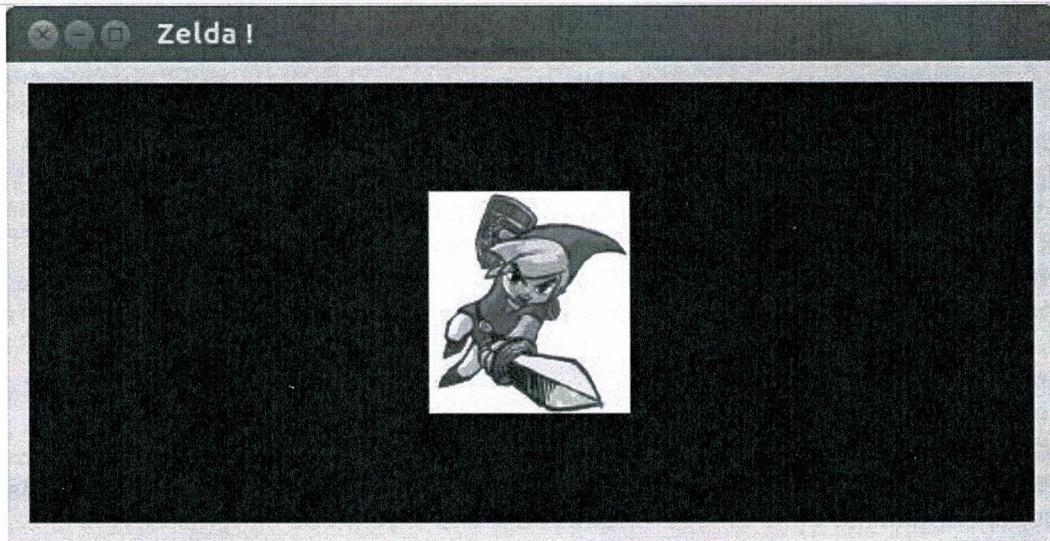
Pour supprimer un élément vous pouvez utiliser la méthode **delete** :

```
canvas.delete(élément)
```

Il est également possible de déposer sur un canevas une image de type gif (uniquement).

Extrait de code à utiliser pour déposer au centre du canevas l'image zelda.gif se trouvant dans le même répertoire que le programme Python :

```
canveas = Canvas(fenetre, width=w, height=h, bg="black")
#...
img = PhotoImage(file='zelda.gif')
can.create_image(w/2, h/2, image=img)
#...
```



Pour afficher une image sur un canevas, utiliser:

Canvas.create_image(x, y, option, ...)

Retourne l'identifiant numérique de l'item image créé sur le canevas appelant.

L'image est positionnée relativement au point (x, y). Ces options sont :

activeimage – Image à afficher lorsque la souris survole l'item. Pour les valeurs possibles, voir l'option image ci-dessous.

anchor – Par défaut, vaut 'center' ce qui signifie que le texte est centré par rapport à la position (x, y). Par exemple, si anchor='s', l'image sera positionnée de sorte que le point (x, y) soit situé au milieu de son bord supérieur (sud).

disabledimage – Image à afficher lorsque l'item est inactif (à l'état 'disabled'). Pour les valeurs possibles, voir image ci-dessous.

image – L'image à afficher.

state – 'normal' par défaut. Mettre cet option à 'disabled' pour l'empêcher de réagir à la souris, la mettre à 'hidden' pour la rendre invisible.

D'autres widgets sont décrits ici : <https://python.doctor/page-tkinter-interface-graphique-python-tutoriel>

Positionner les widgets sur la fenêtre graphique

Tkinter dispose de 3 méthodes pour placer les widgets sur la fenêtre graphique : **pack()**, **place()**, **grid()**. Pour une fenêtre graphique donnée, on doit choisir l'une des 3 méthodes de positionnement (on ne peut pas mixer les trois). La méthode **grid()** offre le meilleur compromis en terme d'ergonomie.

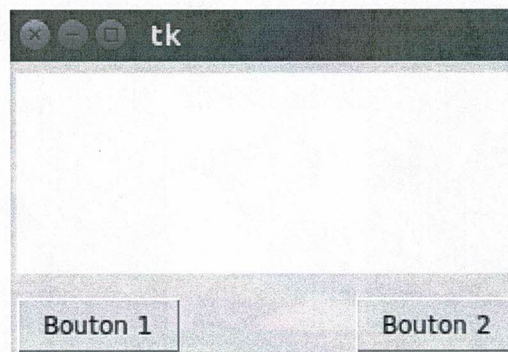
■ A) Méthode **pack()** : placement par blocs

C'est la méthode la plus simple. Avantage : elle permet de positionner automatiquement les widgets sur la fenêtre. Inconvénient : elle manque de souplesse pour faire du sur-mesure. En outre, elle est peu pratique lorsqu'il y a beaucoup de widgets à positionner. Il est possible de préciser la place des widgets à l'aide du paramètre **side** :

```
side=TOP      # haut
side=LEFT     # gauche
side=BOTTOM   # bas
side=RIGHT    # droite
```

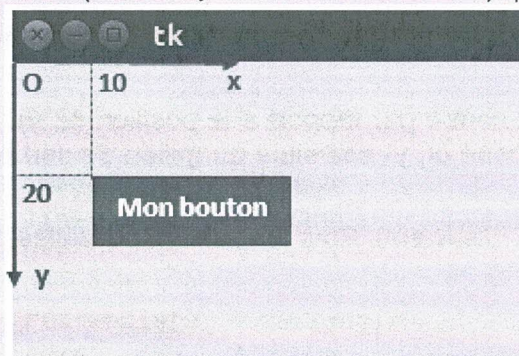
Exemple :

```
Canvas(fenetre, width=250, height=100, bg='ivory').pack(side=TOP, padx=5, pady=5)
Button(fenetre, text='Bouton 1').pack(side=LEFT, padx=5, pady=5) Button(fenetre,
text='Bouton 2').pack(side=RIGHT, padx=5, pady=5)
```



■ B) Méthode **place()** : placement dans un repère cartésien **Attention** : l'origine du repère est situé dans le coin supérieur gauche de la fenêtre graphique et l'axe ordonnées est orienté vers le bas. Cette méthode permet de faire du sur-mesure mais est relativement lourde à maintenir en cas d'ajout de nouveaux widgets sur la fenêtre. Les widgets positionnés de cette manière resteront fixes après redimensionnement de la fenêtre.

```
Button(fenetre, text='Mon bouton').place(x=10, y=20)
```



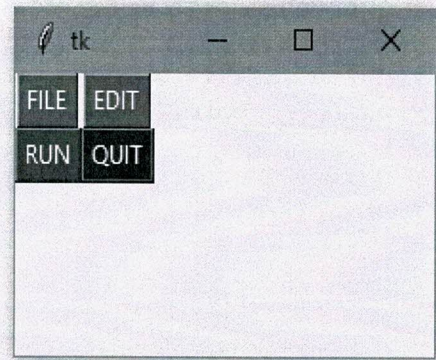
■ C) Méthode grid() : placement sur une grille

La méthode **grid()** (grille) est un bon compromis entre les méthodes **pack()** et **place()**.

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 :

```
boutonRouge.grid(row=0,column=0) boutonVert.grid(row=0,column=1) boutonBleu.grid(row=1,column=0) boutonNoir.grid(row=1,column=1)
```



Quelques paramètres utiles :

- **sticky** – Cette option détermine la façon de distribuer l'espace inoccupé par un widget à l'intérieur d'une cellule. Si vous ne donnez aucune valeur à l'attribut **sticky**, le comportement par défaut est de centrer le widget dans sa cellule. Vous pouvez positionner le widget dans un des coins de la cellule en indiquant **sticky=N+E** (nord-est : en haut à droite), **S+E** (en bas à droite), **S+W** (en bas à gauche), ou **N+W** (en haut à gauche). Le paramètre **sticky** utilise les 4 points cardinaux en anglais : W pour West, E pour East, N pour North, S pour South.
- **columnspan** – Normalement un widget occupe seulement une cellule. Cependant, vous pouvez regrouper (fusionner) plusieurs cellules d'une ligne en indiquant via **columnspan** le nombre de cellules à regrouper. Par exemple, `mon_widget.grid(row=0, column=2, columnspan=3)` aura pour effet de placer `mon_widget` dans une cellule qui s'étale sur les colonnes 2, 3 et 4 de la ligne 0.
- **rowspan** – Normalement un widget occupe seulement une cellule. Cependant, vous pouvez regrouper plusieurs cellules d'une colonne en indiquant via **rowspan** le nombre de cellules à fusionner. Cette option peut être utilisée en combinaison avec **columnspan** afin de préciser un bloc de cellules. Par exemple, `mon_widget.grid(row=3, column=2, rowspan=4, columnspan=5)` aura pour effet de placer `mon_widget` dans une zone obtenue en fusionnant 20 cellules, avec les numéros de lignes 3 - 6 et les numéros de colonnes 2 - 7.

Q8. Modifiez le programme du paragraphe 1 afin de tester les méthodes de positionnement `place()` et `grid()` et leurs différents paramètres.

Deuxième fenêtre graphique : simuler un lancer de dé

Exécuter les lignes de code PYTHON suivantes :

```

1  # -*- coding: utf-8 -*-
2
3  from tkinter import *
4  from random import *
5
6  # Définition de la fonction
7  def nouveauLance ():
8      nb = randint(1,6)
9      texte.set('Résultat -> ' + str(nb))
10
11 # Création de la fenêtre principale (Main Window)
12 fenetre = Tk()
16 fenetre.title('Dé à 6 faces')
14 fenetre.geometry('300x100')
15
16 # Création d'un widget Button (bouton Lancer)
17 boutonLancer = Button(fenetre, text='Lancer', command = nouveauLance)
18 # Positionnement du widget avec la méthode pack()
19 boutonLancer.pack(side = LEFT, padx = 5, pady = 5)
20
21 # Création d'un widget Button (bouton Quitter)
22 BoutonQuitter = Button(fenetre, text='Quitter', command = fenetre.destroy)
23 # Positionnement du widget avec la méthode pack()
24 BoutonQuitter.pack(side = LEFT, padx = 5, pady = 5)
25
26 # Création d'un widget Label (texte 'Résultat -> x')
27 texte = StringVar()
28 labelResultat = Label(fenetre, textvariable = texte, fg='red', bg='white')
29 # Positionnement du widget avec la méthode pack()
30 labelResultat.pack(side = LEFT, padx = 5, pady = 5)
31
32 # Appel de la fonction
33 nouveauLance()
34
35 # Lancement du gestionnaire d'événements
36 fenetre.mainloop()

```

Q9. A quoi sert la fonction définie à la ligne 7 ? Quelle est l'action du bouton à la ligne 17 ?

Q10. Rechercher la signification de l'instruction StringVar() de la ligne 27. Vous pourrez vous aider du document de la page suivante intitulée « **FOCUS sur les variables de contrôle : les valeurs dans les widgets** ».

Q11. Modifier et adapter le programme précédent de manière à ce que la fenêtre graphique comporte un champ demandant à l'utilisateur de rentrer le nombre de faces (4, 6, 8, 10, 12, 14, 16, 20) du dé. Le résultat du tirage au sort devra tenir compte du choix de l'utilisateur. L'interface (500 px sur 100 px) devra avoir l'aspect suivant :

