Chapitre 14 - Bibliothèque Tkinter

La bibliothèque *Tkinter* de Python permet de créer une fenêtre graphique dans laquelle on peut insérer des images, des formes géométriques, des boutons, des champs de saisies, … On retrouve un peu un environnement graphique d'une page web, avec les différences suivantes :

- La fenêtre graphique est ici de taille fixe, ce qui simplifie le positionnement des éléments,
- Le contenu est géré en python, ce qui nous permettra d'utiliser l'ensemble des fonctionnalités pythons vues jusqu'à présent pour réaliser des applications interactives.

1- IMPORTATION DES BIBLIOTHEQUES :

On importe en début de script les classes Tk, Canvas, Label, Text et Button de la bibliothèque tkinter et Image, ImageTk de la bibliothèque PIL. Cette dernière permet d'insérer des images dans les fenêtres graphiques « tkinter ».

```
# Modules -----------------------------------------------------
from tkinter import Tk , Canvas , Label , Text , Button
from PIL import Image, ImageTk   # pip install pillow
from random import randint

# Fonctions ---------------------------------------------------



# Main --------------------------------------------------------

```

2- CREATION DE LA FENETRE TKINTER :

```
# Fonctions ---------------------------------------------------------------
def creer_fenetre() :
    fenetre = Tk()
    fenetre.title("tp11")
    return fenetre


# Main --------------------------------------------------------------------
fenetre = creer_fenetre()


fenetre.mainloop()
```

On crée un **objet** tkinter (Tk) que l'on appelle ici : *fenetre*

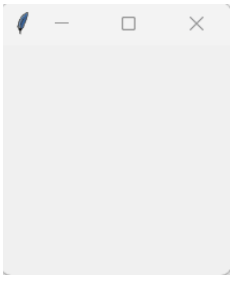
Permet de rajouter un titre en haut de la fenêtre

On retourne cet objet afin qu'il puisse être utilisé dans le programme principal

Permet de maintenir la fenêtre ouverte

Attention : La variable qui contient l'objet Tk() doit être retournée dans le programme principal.

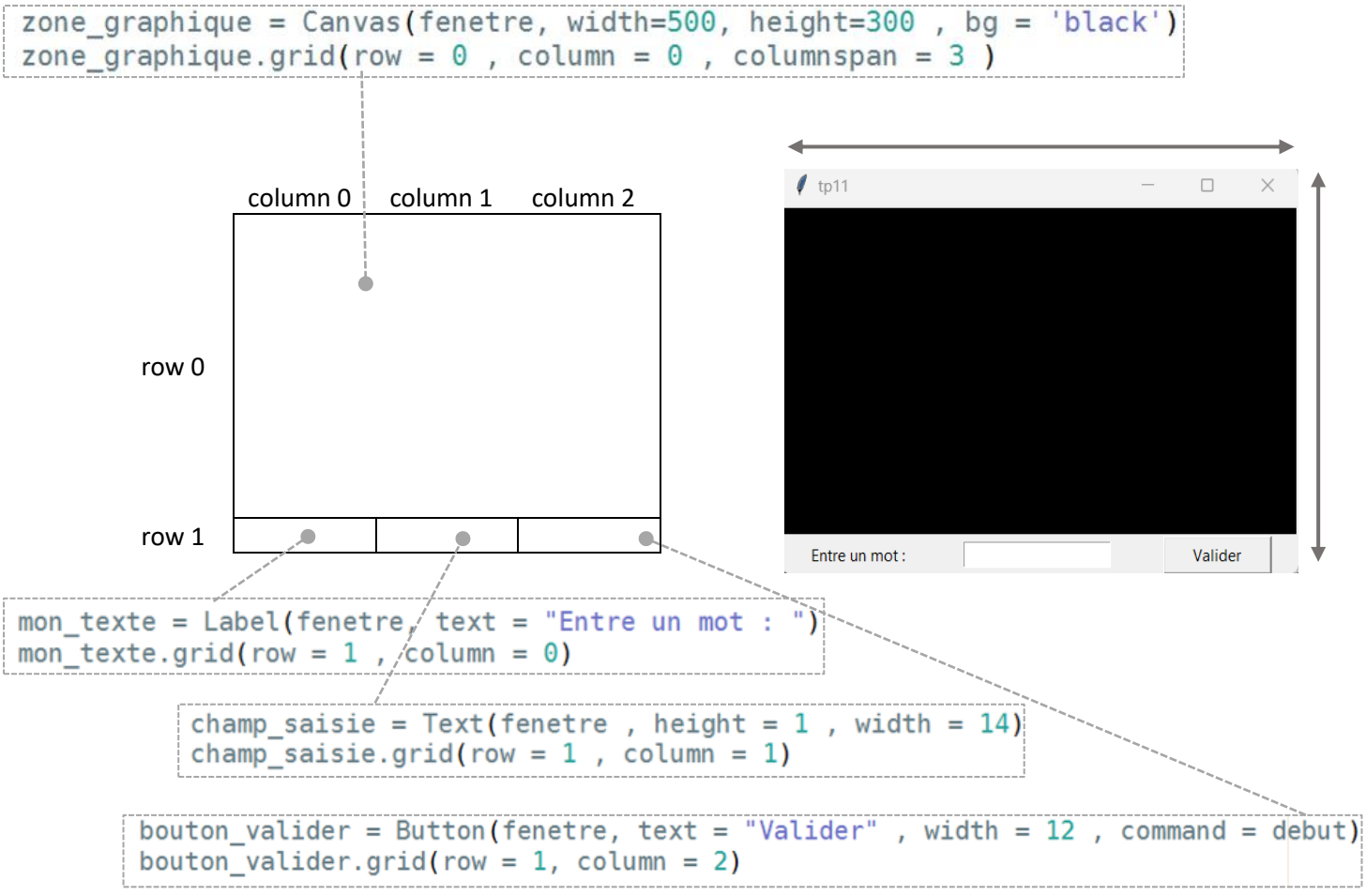
Taille de la fenêtre : La taille de la fenêtre n'est pas à préciser. Elle dépendra de la taille des éléments graphiques qui y seront insérés.



3- CREATION DES WIDGETS :

```
def creer_widgets() :
    zone_graphique = Canvas(fenetre, width=500, height=300 , bg = 'black')
    zone_graphique.grid(row = 0 , column = 0 , columnspan = 3 )

    mon_texte = Label(fenetre, text = "Entre un mot : ")
    mon_texte.grid(row = 1 , column = 0)

    champ_saisie = Text(fenetre , height = 1 , width = 14)
    champ_saisie.grid(row = 1 , column = 1)

    bouton_valider = Button(fenetre, text = "Valider" , width = 12 , command = debut)
    bouton_valider.grid(row = 1, column = 2)

    return zone_graphique, mon_texte, champ_saisie, bouton_valider


def debut():
    print("tu as cliqué sur valider")
```

```
# Main ----------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

fenetre.mainloop()
```



Attention : Les variables qui contiennent les différents widgets doivent être retournées dans le programme principal.

POSITIONNER des widgets dans une fenêtre tkinter :

- Syntaxe : *nomDuWidget.grid(row = … , column = …)*
- Arguments optionnels :
  - *rowspan = n* : fusionne *n* lignes
  - *columnspan = n* : fusionne *n* colonnes

MODIFIER un widget : *nomDuWidget.configure(argument = ….)*

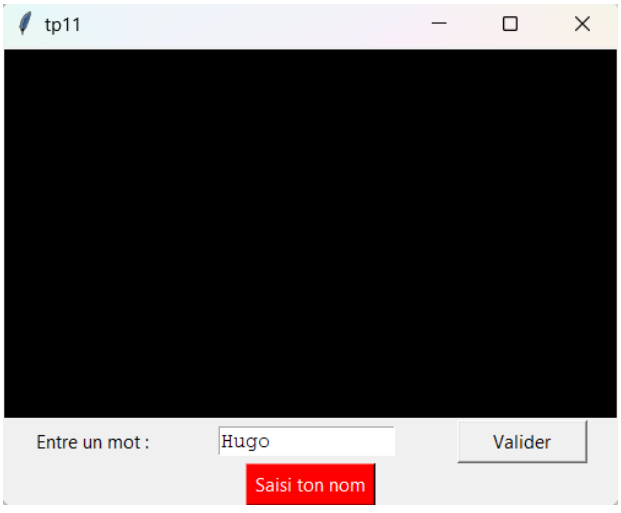
RECUPERER le texte d'un champ de saisi : *nomDuWidget.get("1.0", "end-1c")*

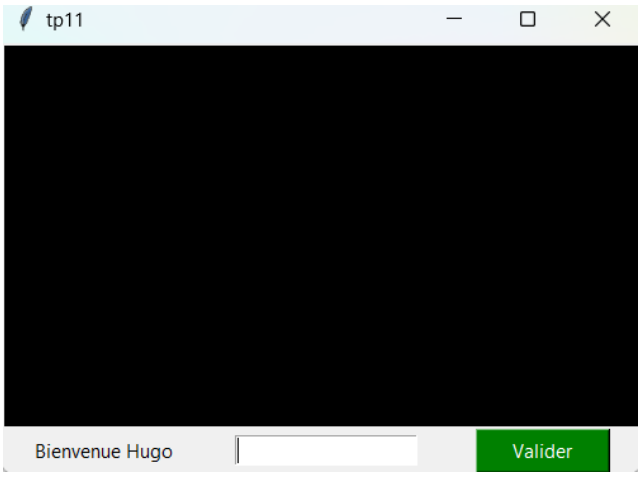
SUPPRIMER un widget : *nomDuWidget.destroy()*

Exemple :

```
def autreFonction() :
    texteSaisi = champ_saisie.get("1.0", "end-1c")
    texteSaisi = "Bienvenue " + texteSaisi
    mon_texte.configure(text = texteSaisi)
    bouton_valider.configure(bg = 'green' , fg="white")
    autreBouton.destroy()
    champ_saisie.delete("1.0", "end-1c")
```

```
# Main ------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

autreBouton = Button(fenetre, text = "Saisi ton nom" , width = 12 , command = autreFonction ,
bg = 'red' , fg='#FFFFFF')
autreBouton.grid(row = 2, column = 0, columnspan = 3)
```





Gérer la taille de la fenêtre et du Canvas :

```
# Fonctions -------------------------------------------------------------
def creer_fenetre() :
    fenetre = Tk()
    w, h = fenetre.winfo_screenwidth(), fenetre.winfo_screenheight()
    fenetre.geometry(f"{w}x{h}")
    fenetre.title("tp11")
    return fenetre

def creer_widgets() :
    w, h = fenetre.winfo_screenwidth(), fenetre.winfo_screenheight()
    zone_graphique = Canvas(fenetre, width=w, height=h-200 , bg = 'black')
    zone_graphique.grid(row = 0 , column = 0 , columnspan = 3 )
```

4- INSERTION D'UNE IMAGE DANS LE CANVAS :

Les images contenues dans un fichier au format *.png* ou *.jpg* doivent être dans un premier temps être converties par la méthode *open()* de la classe *Image*, en objet de la bibliothèque *pillow* (*PIL*).
Par exemple la ligne `pic = Image.open("l_0.png")` permet de créer un objet de la bibliothèque *pillow* dans lequel se trouve toutes les informations contenues dans le fichier '*l_0.png*' . Cet objet est placé ici dans la variable nommée *pic* .

Avec cet objet, on peut utiliser toutes les méthodes disponibles dans la bibliothèque *pillow*. Par exemple, il est possible de redimensionner l'image ou de la tourner sans que le fichier '*l_0.png*' soit modifié :

- `pic = pic.resize((100,50))` permet de modifier la largeur de l'image à 100 px et sa hauteur à 50 px,
- `pic = pic.rotate(45)` permet de la tourner de 45° dans le sens trigonométrique.

Malheureusement, cet objet *pillow* n'est pas compatible avec les éléments graphiques utilisés par la bibliothèque *tkinter*. Il est nécessaire de refaire une conversion en utilisant cette fois-ci la méthode *PhotoImage()* de la classe *ImageTk*.

Par exemple la ligne `pic = ImageTk.PhotoImage(pic , master = fenetre)` permet de modifier le contenu de la variable *pic* pour y placer un autre objet de la bibliothèque pillow, qui sera à présent compatible avec la bibliothèque tkinter.

Les 2 conversions détaillées ci-dessus peuvent être exécutées sur la même ligne. Cela donne :

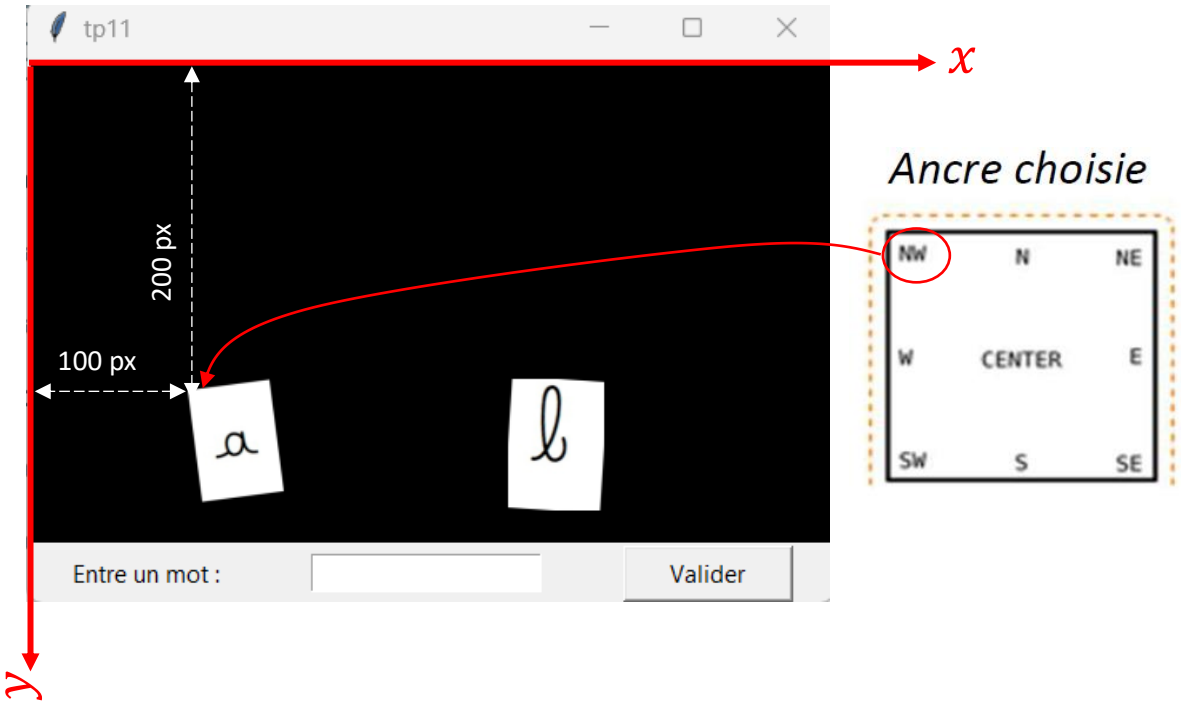
```
pic = ImageTk.PhotoImage(Image.open("l_0.png") , master = fenetre)
```

Attention, l'objet contenu ici dans *pic* **doit être retourné** dans le **programme principal**.

Dans l'exemple ci-dessous, ces opérations de conversion sont réalisées dans la fonction *creerImagesTk()* pour ici 3 images . Les images converties sont placées dans le dictionnaire *dicImagesTk* avec comme clé le nom du fichier qui contient l'image. Ce dictionnaire est créé dans le programme principal, donc pas la peine ici d'y retourner les objets images. Ce dictionnaire sera ensuite accessible en lecture et écriture dans toutes les fonctions.

```
def creerImagesTk() :
    dicImagesTk["l_0.png"] = ImageTk.PhotoImage(Image.open("l_0.png") , master = fenetre)
    dicImagesTk["l_1.png"] = ImageTk.PhotoImage(Image.open("l_1.png") , master = fenetre)
    dicImagesTk["l_2.png"] = ImageTk.PhotoImage(Image.open("l_2.png") , master = fenetre)


def insererImages() :
    num = zone_graphique.create_image(100, 200 ,anchor = "nw",image = dicImagesTk["l_0.png"])
    imDansCanvas.append(num)
    num = zone_graphique.create_image(300, 200 ,anchor = "nw",image = dicImagesTk["l_1.png"])
    imDansCanvas.append(num)


# Variables globales ------------------------------------------------
dicImagesTk = {}
imDansCanvas = []

# Main --------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()
creerImagesTk()
insererImages()
```

Après cette étape de conversion, il faut à présent **insérer** les images dans le *Canvas* nommé dans l'exemple ci-dessus *zone_graphique*. On utilise alors la méthode *create_image()* qui renvoie un nombre entier. Ce nombre est une sorte de **numéro d'identification**. Il s'agit de le conserver afin de pourvoir ultérieurement retirer cette image du *Canvas* ou simplement la modifier.




```
num = zone_graphique.create_image(100, 200 ,anchor = "nw",image = dicImagesTk["l_0.png"])
```

Contenu de la liste *imDansCanvas* :

```
>>> imDansCanvas
[1, 2]
```

6- MODIFICATION OU SUPPRESSION D'UNE IMAGE DEJA INSEREE DANS LE CANVAS :

On suppose qu'une image dont le numéro est **num** a été insérée dans `zone_graphique`

```
num = zone_graphique.create_image(100, 200 ,anchor = "nw",image = dicImagesTk["l_0.png"])
```

- Pour récupérer le nom de l'objet image tkinter :

```
pic = zone_graphique.itemcget(num,"image")
```

- Pour récupérer l'ancre de cette image :

```
pic = zone_graphique.itemcget(num,"anchor")
```

- Pour modifier le fichier relié à cette image :

```
zone_graphique.itemconfigure(num,image = dicImagesTk["l_2.png"])
```

- Pour modifier l'ancre de cette image :

```
pic = zone_graphique.itemconfigure(num,anchor="sw")
```

- Pour récupérer les coordonnées de l'ancre de cette image :

```
x,y = zone_graphique.coords(num)
```

- Pour modifier les coordonnées de l'ancre de cette image à $x = 50$ et $y = 100$:

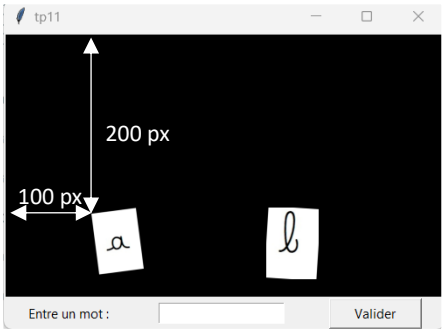
```
zone_graphique.coords(num, 50 , 100)
```

- Pour supprimer cette image du Canvas :

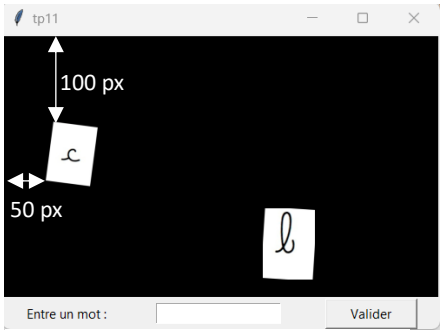
```
zone_graphique.delete(num)
```

On peut par exemple réaliser les opérations suivantes :

```
def modifierImages() :
    num = imDansCanvas[0]
    zone_graphique.itemconfigure(num,image = dicImagesTk["l_2.png"])
    x,y = zone_graphique.coords(num)
    zone_graphique.coords(num, 50 , 100)

def supprimeImage() :
    num = imDansCanvas[1]
    zone_graphique.delete(num)
    del imDansCanvas[1]

# Variables globales ----------------------------------------------
dicImagesTk = {}
imDansCanvas = []

# Main ------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()
creerImagesTk()
insererImages()
modifierImages()
supprimeImage()
```



Etat initial



Après `modifierImages()`

Après `supprimeImage()`

7- INSERTION D'UN TEXTE DANS LE CANVAS :

⇨ Pour insérer un texte dans le Canvas nommé ici *zone_graphique*, on utilise la méthode *create_text()*. Elle renvoie un numéro d'identification qui permet ensuite de modifier le contenu et la position de ce texte. Ce nombre entier est ci-dessous stocké dans la variable nommée *num* :

```
num = zone_graphique.create_text(10, 50, anchor = "nw",
text='Bienvenue\ndans mom monde', fill='#fb8007', font='Arial 20')
```

⇨ Pour modifier le contenu, la taille, la couleur, la font du texte, on utilise la méthode *itemconfigure() :*

```
zone_graphique.itemconfigure(num ,  text = 'yes' ,
fill="#ffffff" , font = "Times 80")
```

⇨ Pour modifier les coordonnées de l'ancre du texte, on utilise la méthode *coords()* :

```
zone_graphique.coords(num , 200 , 100)
```

⇨ Pour récupérer la couleur du texte, on utilise toujours la méthode *itemcget()* :

```
couleur = zone_graphique.itemcget(num,"fill")
```

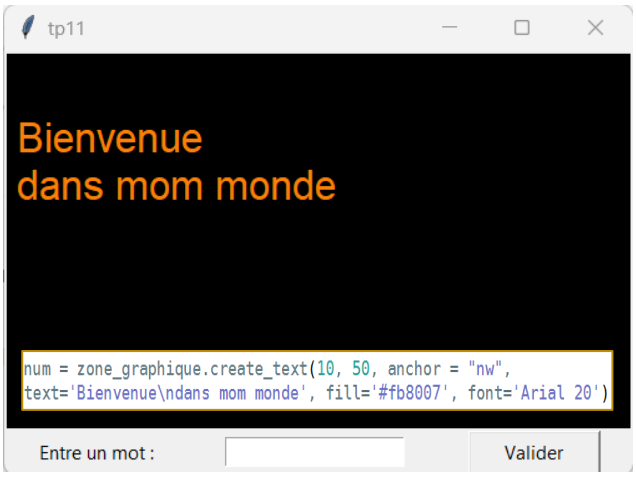
⇨ Pour récupérer le contenu du texte, on utilise toujours la méthode *itemcget()* :

```
t = zone_graphique.itemcget(num,"text")
```

⇨ Pour récupérer les coordonnées de l'ancre du texte, on utilise toujours la méthode *coords()* :

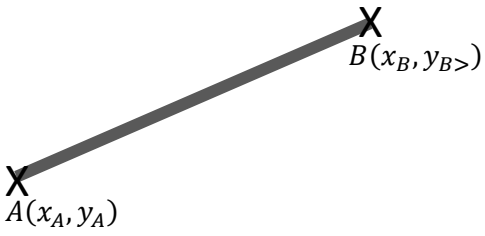
```
x,y = zone_graphique.coords(num)
```

⇨ Pour supprimer le texte, on utilise la méthode *delete()* : `zone_graphique.delete(num)`



8- INSERTION D'UN SEGMENT DANS LE CANVAS :

⇨ Pour insérer un segment de droite entre 2 points A et B de coordonnées $A(x_A, y_A)$ et $B(x_B, y_B)$ dans le Canvas nommé ici *zone_graphique*, on utilise la méthode *create_line()*. Elle renvoie un numéro d'identification qui permet ensuite de modifier ce segment. Ce nombre est ici stocké dans la variable nommée *num* :




```
num = zone_graphique.create_line(10, 10, 400, 200, width=4, fill='#ffffff')
```

⇨ Pour modifier la taille, la couleur, …. , on utilise la méthode *itemconfigure() :*

```
zone_graphique.itemconfigure(num ,  fill="#00ff00")
```

⇨ Pour modifier les coordonnées des points A et B, on utilise la méthode *coords()* :

```
zone_graphique.coords(num , 10 , 300 , 400 , 10)
```

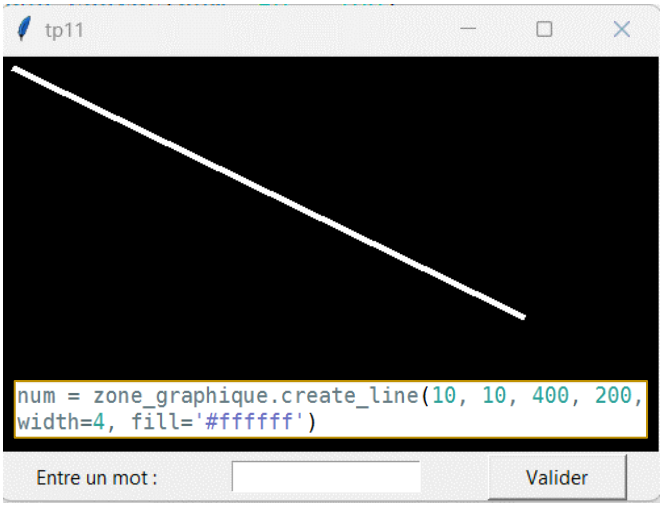
⇨ Pour récupérer les coordonnées des points A et B, on utilise toujours la méthode *coords()* :

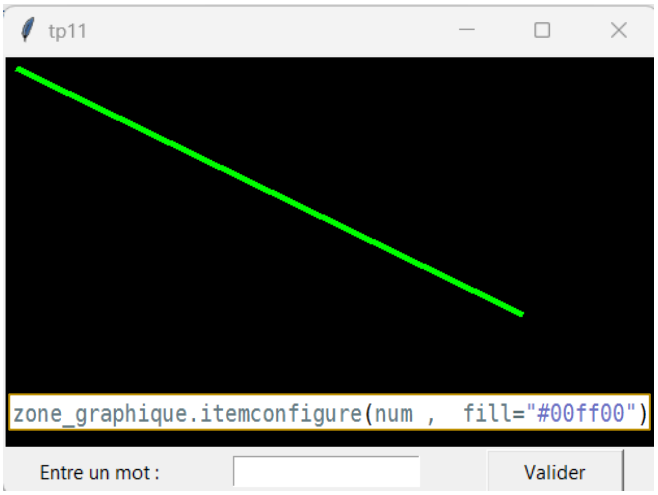
```
xa,ya,xb,yb = zone_graphique.coords(num)
```

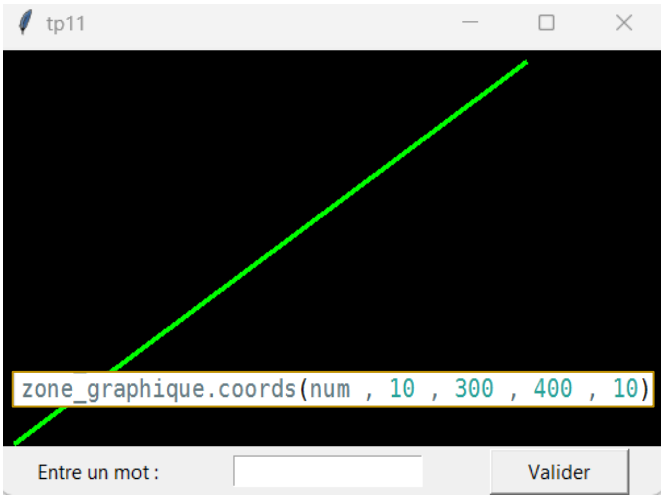
⇨ Pour récupérer la couleur, …, on utilise la méthode *itemcget() :*

```
couleur = zone_graphique.itemcget(num,"fill")
```

⇨ Pour supprimer le segment, on utilise la méthode *delete()* : `zone_graphique.delete(num)`

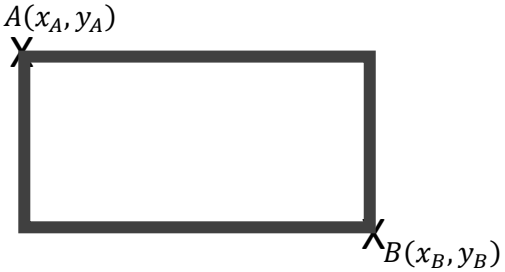






9- INSERTION D'UN RECTANGLE DANS LE CANVAS :

⇨ Pour insérer un rectangle défini par la position des points A et B de coordonnées $A(x_A, y_A)$ et $B(x_B, y_B)$ dans le Canvas nommé ici *zone_graphique*, on utilise la méthode *create_rectangle()*. Elle renvoie un numéro d'identification qui permet ensuite de modifier ce rectangle.



Ce nombre entier est ci-dessous stocké dans la variable nommée *num* :

```
num = zone_graphique.create_rectangle(100, 100, 400, 200, width=4,
outline = "#00ff00", fill='#ffffff')
```

⇨ Pour modifier la taille, la couleur, …, on utilise la méthode *itemconfigure() :*

```
zone_graphique.itemconfigure(num ,  fill="#00ff00")
```

⇨ Pour modifier les coordonnées des points A et B, on utilise la méthode *coords()* :

```
zone_graphique.coords(num , 10 , 300 , 400 , 10)
```

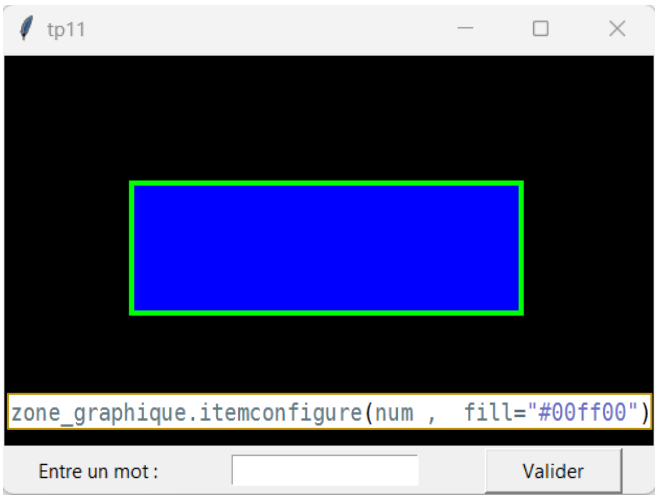
⇨ Pour récupérer les coordonnées des points A et B, on utilise toujours la méthode *coords()* :

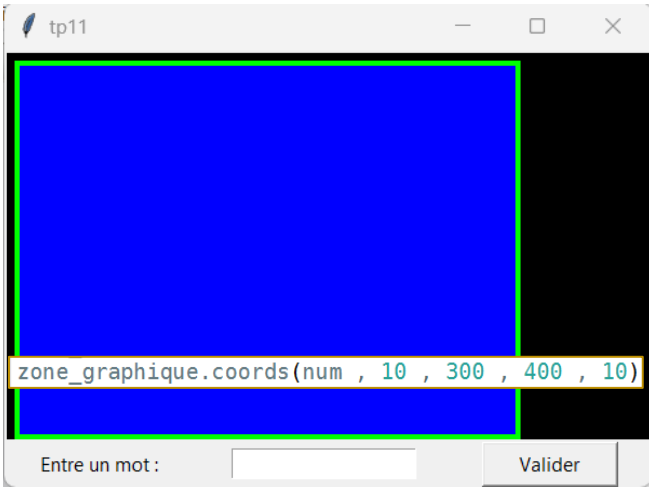
```
xa,ya,xb,yb = zone_graphique.coords(num)
```

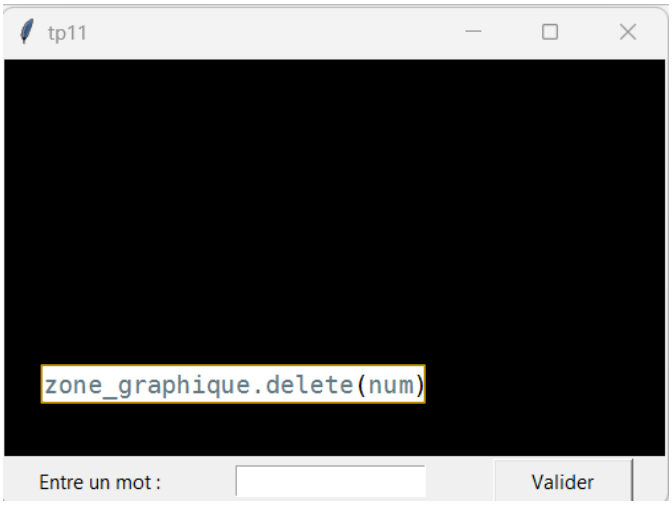
⇨ Pour récupérer la couleur, …, on utilise la méthode *itemcget () :*

```
couleur = zone_graphique.itemcget(num,"fill")
```

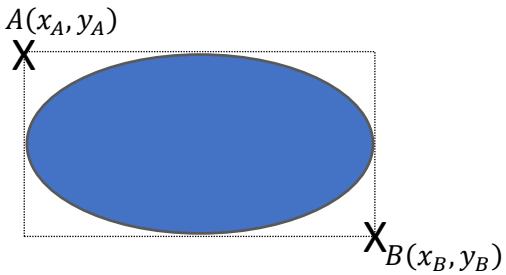
⇨ Pour supprimer le rectangle, on utilise la méthode *delete()* : `zone_graphique.delete(num)`







10-INSERTION D'UN OVALE DANS LE CANVAS :



⇨ Pour insérer un oval défini par la position des points A et B de coordonnées $A(x_A, y_A)$ et $B(x_B, y_B)$ dans le Canvas nommé ici *zone_graphique*, on utilise la méthode *create_oval()*. Elle renvoie un numéro d'identification qui permet ensuite de modifier cet oval.

Ce nombre entier est ci-dessous stocké dans la variable nommée *num* :

```
num = zone_graphique.create_oval(100, 100, 400, 200, width=4,
outline = "#00ff00", fill='#ffffff')
```

⇨ Pour modifier la taille, la couleur, on utilise la méthode *itemconfigure() :*

```
zone_graphique.itemconfigure(num ,  fill="#00ff00")
```

⇨ Pour modifier les coordonnées des points A et B, on utilise la méthode *coords()* :

```
zone_graphique.coords(num , 10 , 300 , 400 , 10)
```

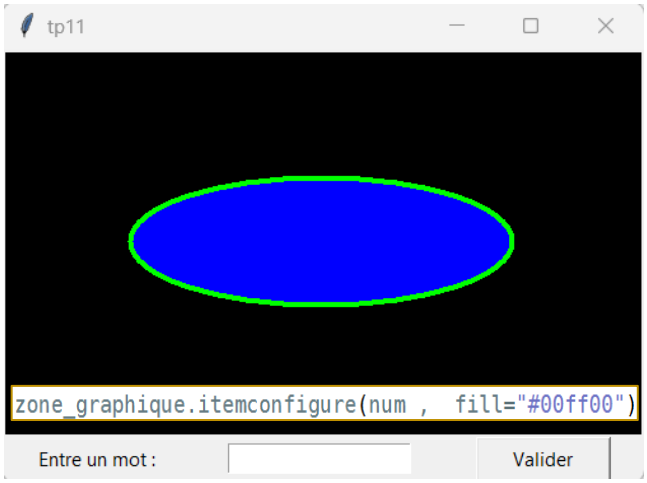
⇨ Pour récupérer la couleur, …, on utilise la méthode *itemcget () :*

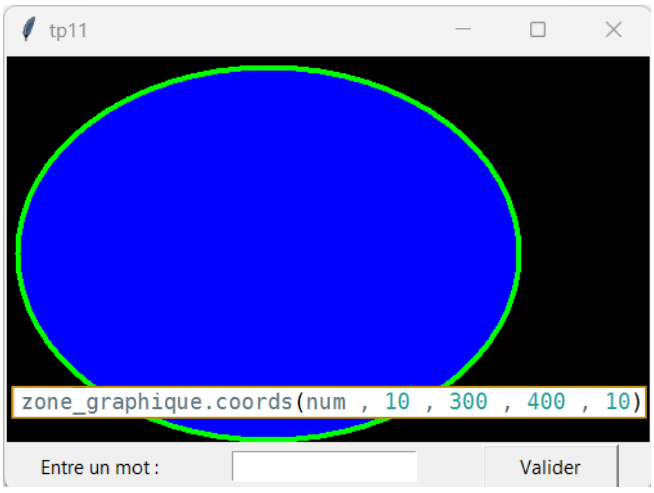
```
couleur = zone_graphique.itemcget(num,"fill")
```

⇨ Pour récupérer les coordonnées des points A et B, on utilise toujours la méthode *coords()* :

```
xa,ya,xb,yb = zone_graphique.coords(num)
```

⇨ Pour supprimer l'oval, on utilise la méthode *delete()* : `zone_graphique.delete(num)`





11-SUPPRESSION DE TOUS LES ELEMENTS INSERES DANS LE CANVAS :

Il est possible de supprimer tous les éléments déjà insérés dans le Canvas en utilisant la méthode *delete()* avec comme argument le string ***'all'***. Si le Canvas se trouve dans la variable *zone_graphique*, cela donne :

```
zone_graphique.delete("all")
```

Remarque : ne pas confondre avec la méthode *destroy()* qui détruirait le widget Canvas :

```
zone_graphique.destroy()
```

12-GESTION DES EVENEMENTS :

Lors de la lecture d'un fichier .py, les instructions pythons écrites sont lues de haut en bas, **et sont exécutées une seule fois**. Si l'on veut créer une application interactive, il est nécessaire de créer des évènements.

Ces évènements sont mémorisés à l'unique lecture du fichier .py . Ils associent une fonction appelée **callback** à une action de l'utilisateur sur le clavier ou sur la souris.

a. GESTION DES EVENEMENTS SOURIS :

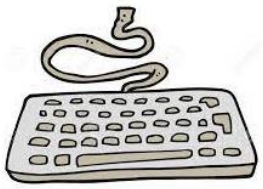
On déclare un évènement souris en utilisant la méthode *bind()* sur l'objet *Canvas* nommé ici *zone_graphique*. La fonction callback est nommée *ma_fonction()* dans l'exemple ci-dessous. **On ne peut pas mettre d'arguments à cette fonction**. Si cela est nécessaire, on utilise des variables globales. Il est possible de récupérer les coordonnées de la souris lorsque l'évènement se produit.

```
def ma_fonction(event) :
    xSouris = event.x
    ySouris = event.y
    print("je suis sur les coordonnées",xSouris,ySouris)


# Main ----------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

zone_graphique.bind("<ButtonPress-1>",ma_fonction)
```

La syntaxe des différentes actions souris qu'il est possible d'exploiter sont données ci-contre :

<ButtonPress-1>	**Appui sur le bouton gauche de la souris**
<ButtonRelease-2>	**Relâchement du bouton gauche de la souris**
<Double-Button-1>	**Double clic sur le bouton gauche.**
<Motion>	**Déplacement** de la souris
<B1-Motion>	Déplacement de la souris avec le bouton gauche appuyé.
<Enter>	**Entrée dans le widget** de la souris
<Leave>	**Sortie du widget** de la souris

b. GESTION DES EVENEMENTS CLAVIER :



On déclare un évènement clavier en utilisant la méthode *bind()* sur l'objet *Tk* nommé ici *fenetre*. La fonction callback est nommée mon_*action()* dans l'exemple ci-dessous :

```
def mon_action(event) :
    print("j''ai été appelé")

# Main ----------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

fenetre.bind("<Up>",mon_action)
```

La syntaxe pour repérer quelques touches du clavier est donnée ci-contre :

<KeyPress>	**Appuie d'une touche** quelconque du clavier
<KeyRelease>	**Relâchement d'une touche** quelconque du clavier
a, A, 1, 2, +, * , …	Surveille **les touches indiquées.**
<Right>, <Down>,<Up>, <Left>	On peut combiner ces touches par un tiret : '<Controle-Up>'
<Alt>,<Shift>,<Control>	

Voir sur le net pour les autres touches.

13-GESTION DES ANIMATIONS :

Pour certaines applications, il est nécessaire de relancer l'exécution de fonctions régulièrement. On parle alors d'animation. Pour répéter l'exécution d'une fonction, nommée *mon_animation()* dans l'exemple ci-dessous, on exécute la méthode *after()* sur l'objet *Tk* appelé ici *fenetre* :

```
def mon_animation() :
    global n
    print(f"je suis exécuté pour la {n}ième fois")
    n = n + 1
    if n < 10 : fenetre.after(1000 , mon_animation)

# Variables globales ----------------------------------------------------
n = 0

# Main ----------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()

mon_animation()
```

L'exécution de ce script donne dans la console :

```
je suis exécuté pour la 0ième fois
je suis exécuté pour la 1ième fois
je suis exécuté pour la 2ième fois
je suis exécuté pour la 3ième fois
je suis exécuté pour la 4ième fois
je suis exécuté pour la 5ième fois
je suis exécuté pour la 6ième fois
je suis exécuté pour la 7ième fois
je suis exécuté pour la 8ième fois
je suis exécuté pour la 9ième fois
```

Le temps indiqué dans les arguments de la méthode *after()* est en millisecondes.

Remarque : Il est possible d'avoir des arguments dans la fonction qui est répétée. On peut par exemple améliorer le code précédent en plaçant la variable n en argument de la fonction répétée. Cette solution évite l'utilisation d'une variable globale :

```
def mon_animation(n) :
    print(f"je suis exécuté pour la {n}ième fois")
    n = n + 1
    if n < 10 : fenetre.after(1000 , mon_animation , n)

# Main ----------------------------------------------------------------
fenetre = creer_fenetre()
zone_graphique,mon_texte,champ_saisie,bouton_valider = creer_widgets()
mon_animation(1)
```

L'exécution de ce script donne dans la console :

```
je suis exécuté pour la 1ième fois
je suis exécuté pour la 2ième fois
je suis exécuté pour la 3ième fois
je suis exécuté pour la 4ième fois
je suis exécuté pour la 5ième fois
je suis exécuté pour la 6ième fois
je suis exécuté pour la 7ième fois
je suis exécuté pour la 8ième fois
je suis exécuté pour la 9ième fois
```

14-GESTION DU SON AVEC LA BIBLIOTHEQUE VLC:

⇨ Pour pouvoir lire des fichiers audios au format wav ou mp3, on importe déjà la class *MediaPlayer* de la librairie vlc : `from vlc import MediaPlayer  # pip install python-vlc`

⇨ Pour pouvoir lire le fichier *mon_son.mp3* , on crée dans un premier temps un objet de la class MediaPlayer à partir du fichier .mp3 ou .wav : `mon_son = MediaPlayer("mon_son.wav")`

⇨ Pour activer la lecture, on exécute : `mon_son.play()`

⇨ Pour arrêter la lecture, on exécute : `mon_son.stop()`

15-GESTION DU SON AVEC LA BIBLIOTHEQUE WINSOUND :

⇨ Pour pouvoir lire des fichiers audios au format wav, on importe la class *PlaySound* de la librairie winsound :

```
from winsound import PlaySound, SND_ASYNC # pip install winaudio
```

⇨ Pour pouvoir lire le fichier *mon_son.wav* : `PlaySound("mon_son.wav", SND_ASYNC)`

⇨ Pour arrêter la lecture, on exécute : `PlaySound(None, SND_PURGE)`

⇨ Pour créer des fichiers de sons purs (notes de musiques), changer le format ou raccourcir des fichiers existants, utiliser un logiciel son tel que *audiacity* (logiciel libre).