TKINTER

Mémento

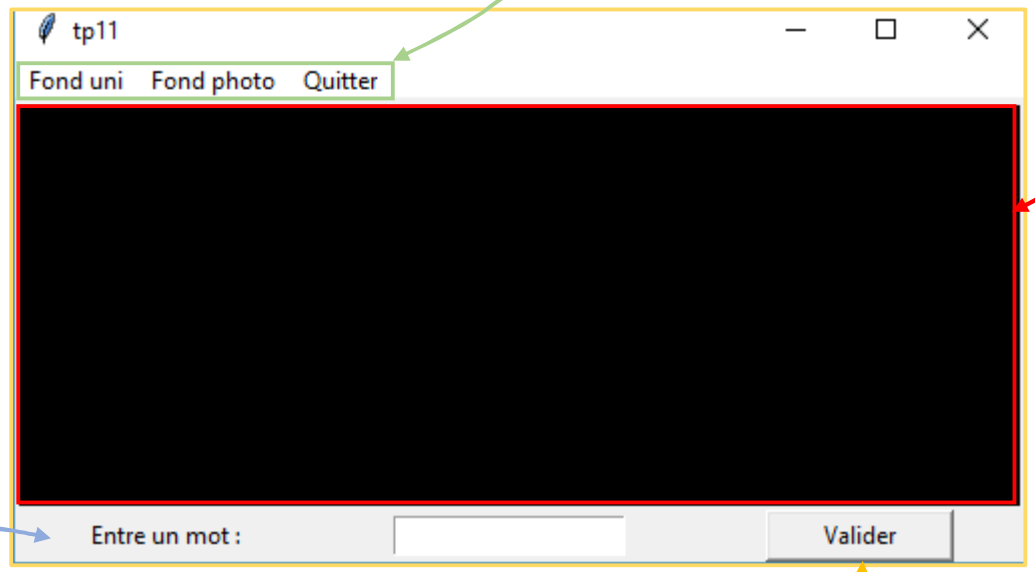
⇨ Modules à importer :

```
# Modules ------------------------------------------------------------
from tkinter import Tk , Menu , Canvas , Label , Entry , Text, StringVar ,Button
from PIL import Image, ImageTk
from random import randint
from winsound import PlaySound
```

Tk() : permet de créer un objet fenêtre Tkinter

Menu() : permet de créer un objet menu et de l'intégrer à la fenêtre Tkinter

Canvas() : permet de créer un objet capable d'afficher du contenu graphique et de l'intégrer à la fenêtre Tkinter



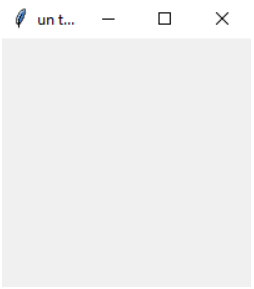
Label() : permet de créer un objet affichant un contenu texte et de l'insérer dans la fenêtre Tkinter

Button() : permet de créer un objet bouton et de l'intégrer dans la fenêtre Tkinter

Entry() ou Text() : permettent de créer un objet champs de saisi et de l'intégrer dans la fenêtre Tkinter

```
from PIL import Image, ImageTk   : permet de gérer les fichiers images
from random import randint       : permet de créer des nombres aléatoires
from winsound import PlaySound   : permet de gérer les fichiers sons
```

⇨ Créer une fenêtre Tkinter :



```
fenetre = Tk()
fenetre.title("un titre")

fenetre.mainloop()
```

Crée un objet Tkinter nommé ici *fenetre*.

Ajoute un titre à cet objet nommé *fenetre*
(*on applique la méthode title() à cet objet*)

A ajouter **en fin de programme principal**, pour maintenir l'objet nommé *fenetre* ouvert.

⇨ Créer un menu et l'insérer dans la fenêtre *Tkinter* :

On crée un objet de la classe Menu() appelé menu_A

On ajoute à cet objet menu_A une commande, en lui appliquant la méthode add_command()

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

def ma_fonction2(argument) :
    print("fonction2 : ",argument)

menu_A = Menu(fenetre)
menu_A.add_command(label= "Jaune", command = ma_fonction)
menu_A.add_command(label= "Vert", command = ma_fonction)

menu_B = Menu(fenetre)
menu_B.add_command(label="Potter Park",command= lambda:ma_fonction2("oui"))
menu_B.add_command (label= "Skate Park" ,command = lambda: ma_fonction2("non"))

mon_menu = Menu(fenetre)
mon_menu.add_cascade(label = "Fond uni", menu = menu_A)
mon_menu.add_cascade(label = "Fond photo", menu = menu_B)
mon_menu.add_command(label="Quitter",command = fenetre.destroy)

fenetre["menu"] = mon_menu
```

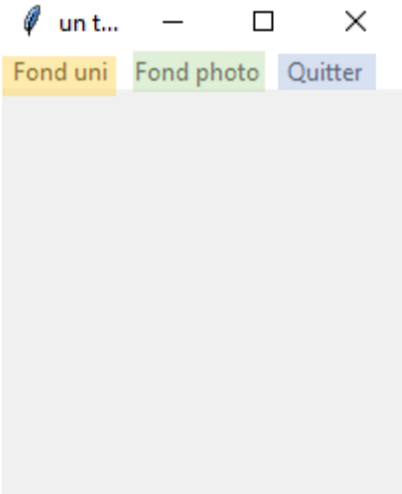
Fonction à appeler :

- Si cette fonction n'a pas d'arguments, on utilise la syntaxe :
  command = *nom_fonction*
- Sinon, on écrit :
  command = lambda : *nom_fonction(……..)*

Dans ces 2 cas, **il n'est pas possible de retourner une valeur**

On crée un objet de la classe Menu() appelé mon_menu

On ajoute à l'objet fenêtre Tkinter l'objet nommé mon_menu



⇨ Gérer des **Widgets dans une fenêtre *Tkinter*** nommée ci-dessous *fenetre* :

Objet de la classe *Canvas()*

- Créer un objet de la classe *Canvas()* :

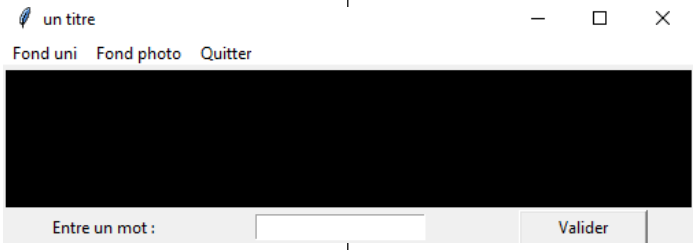
```
zone_graphique = Canvas(fenetre, width = 500, height = 200 , bg = 'black')
zone_graphique.grid(row = 0 , column = 0 , columnspan = 3 )
```

- Modifier les attributs d'un objet de la classe *Canvas()* :

```
zone_graphique.configure(bg = 'red' , height = 100)
```

- Supprimer un objet de la classe *Canvas()* :

```
zone_graphique.destroy()
```



Objet de la classe *Label()*

- Créer un objet de la classe *Label()* :

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

- Modifier les attributs d'un objet de la classe *Label()* :

```
mon_texte.configure(text = "ok")
```

- Supprimer un objet de la classe *Label()* :

```
mon_texte.destroy()
```

Objet de la classe *Entry()* ou *Text()*

Champs de saisi composé d'une ligne :

- Créer un objet de la classe *Entry()* :

```
champ_saisie = Entry(fenetre , textvariable = StringVar())
champ_saisie.grid(row = 1 , column = 1)
```

- Récupérer le texte saisi par l'utilisateur :

```
texte_saisi = champ_saisie.get()
```

Champs de saisi composé d'une zone multilignes :

- Créer un objet de la classe *Text()* :

```
champ_saisie = Text(fenetre , height = 2 , width = 20)
champ_saisie.grid(row = 1 , column = 1)
```

- Récupérer le texte saisi par l'utilisateur :

```
texte_saisi = champ_saisie.get("1.0", "end-1c")
```

Objet de la classe *Button()*

Si la fonction à appeler n'a pas d'arguments :

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

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

La fonction appelée ne peut pas retourner de valeur

Si la fonction à appeler a des arguments :

```
def ma_fonction2(argument) :
    print("fonction2 : ",argument)

bouton_valider = Button(fenetre, text = "Valider" ,width = 12 ,\
                        command = lambda : ma_fonction2('j'ai été appelé'))
bouton_valider.grid(row = 1, column = 2)
```

⇨ Gérer des **Items dans un Canvas** nommé ici zone_graphique :

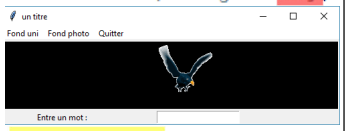
Item **Image**

- Créer un objet image Tkinter à partir d'un fichier *.jpg* ou *.png* :

```
obj = ImageTk.PhotoImage(Image.open('oiseau_0.png') , master = fenetre)
```

- **Créer** un item image à partir de cet objet :

```
mon_image = zone_graphique.create_image(200,2 , anchor = "nw" , image = obj)
```

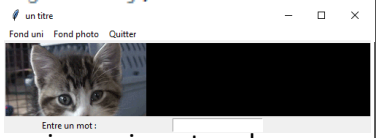


- **Récupérer** un attribut de l'item image :

```
valeur_ancre = zone_graphique.itemcget(mon_image,'anchor')
x , y = zone_graphique.coords(mon_image)
```

- **Modifier** un attribut de l'item image :

```
obj = ImageTk.PhotoImage(Image.open('chat_0.png') , master = fenetre)
zone_graphique.itemconfigure(mon_image , image = obj)
zone_graphique.coords(mon_image , 0 , 0 )
```



- **Supprimer** un item image :

```
zone_graphique.delete(mon_image)
```

on peut aussi supprimer tous les items :

```
zone_graphique.delete('all')
```

Item **Rectangle**

- **Créer** un item rectangle :

```
mon_rectangle = zone_graphique.create_rectangle(20,30,300,80,fill= "white",\
                                                width= 2, outline='red')
```

- **Récupérer** un attribut de l'item rectangle :

```
couleur = zone_graphique.itemcget(mon_rectangle,'fill')
x0 , y0 , x1 , y1 = zone_graphique.coords(mon_rectangle)
```

- **Modifier** un attribut de l'item rectangle :

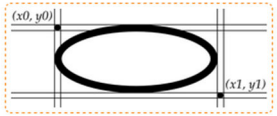
```
zone_graphique.itemconfigure(mon_rectangle,fill='yellow')
zone_graphique.coords(mon_rectangle ,0 , 10 , 400 , 20)
```

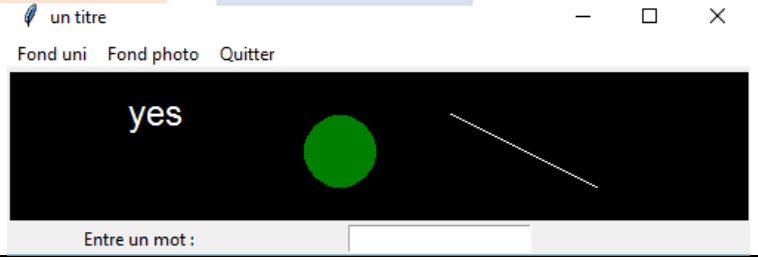
- **Supprimer** un item rectangle :

```
zone_graphique.delete(mon_rectangle)
```

Autres items : **Cercle** ou **Ligne** ou **Texte**

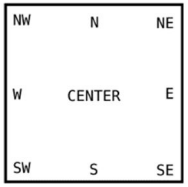
```
mon_cercle = zone_graphique.create_oval(200 , 30 , 250 , 80 , fill = "green")
ma_ligne = zone_graphique.create_line(300 , 30 , 400 , 80 , fill = "white")
mon_texte = zone_graphique.create_text(100 , 30 , anchor = "center", text = "yes",fill='white', font='arial 18')
```





La gestion de ces items est identique à celle des items rectangles.

Si la position de l'item est définie par 2 coordonnées, il faut définir la position de l'ancre :



⇨ Créer des **évènements** liés à une action de l'utilisateur sur la **souris** :

<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

```
def ma_fonction(event) :
    x_souris = event.x
    y_souris = event.y
    print("j'ai été appelé et les coordonnées de la souris sont : ", x_souris , y_souris)

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

On peut récupérer les coordonnées du pointeur de la souris

*Méthode bind() appliquée **sur le Canvas***

⇨ Créer des **évènements** liés à une action de l'utilisateur sur une touche du **clavier** :

<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>	

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

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

*On doit mettre en **argument** le **seul** mot clé **event***

*Lorsque la fonction est appelée, **on ne peut pas mettre d'argument**.*

*Méthode bind() appliquée **sur la fenêtre Tkinter***

⇨ Créer une **animation** en relançant régulièrement l'exécution d'une fonction :

```
def ma_fonction(n) :
    n = n + 1
    print("j'ai été appelé pour la ",n," ième fois")
    if n < 5 : fenetre.after(1000, lambda : ma_fonction(n))

ma_fonction(0)
```

```
>>> (executing file "memento_tkinter.py")
j'ai été appelé pour la  1  ième fois
j'ai été appelé pour la  2  ième fois
j'ai été appelé pour la  3  ième fois
j'ai été appelé pour la  4  ième fois
j'ai été appelé pour la  5  ième fois
```

On répète ici l'exécution de la fonction tous les 1000 ms

*Méthode after() appliquée **sur la fenêtre Tkinter***

On appelle la fonction la première fois

⇨ Jouer un fichier son au format .wav : On lance la fonction ***PlaySound()*** qui a été importée du module winsound

- Lire un fichier :

```
PlaySound("mon_fichier.wav" , winsound.SND_ASYNC)
```

- Stopper la lecture en cours :

```
PlaySound(None, winsound.SND_PURGE)
```