

UAA4 Projet de programmation impérative python en utilisant la librairie graphique tkinter

0 Introduction

Bibliographie

1. Chapitre 8 sur Tkinter du manuel "Apprendre à programmer en Python de Gérard Swinnen".

<https://dn790000.ca.archive.org/0/items/apprendre-a-programmer-avec-python-3/apprendre-a-programmer-avec-python-3.pdf>

2. Documentation Tkinter de Pascal Ortiz.

<https://upinfo.univ-cotedazur.fr/~obaldellon/L1/py/tkinter.pdf>

GUI et gestion d'évènements

Tkinter (Tool Kit INTERface) est le module graphique d'origine pour python permettant la création d'interfaces graphiques.

Lorsque l'on développe une **GUI** (**Graphical User Interface**), nous créons une fenêtre graphique contenant notre application, ainsi que des **widgets** inclus dans la fenêtre. Les **widgets** (**WInDow GadgET**) sont des objets graphiques permettant à l'utilisateur d'interagir avec le programme Python de manière conviviale.

L'utilisation d'une GUI va amener une nouvelle manière d'aborder le déroulement d'un programme, il s'agit de la programmation dite « **événementielle** ». Jusqu'à maintenant tu as programmé « linéairement », c'est-à-dire que les instructions du programme principal s'enchaînaient les unes derrière les autres (avec bien sûr de possibles appels à des fonctions. Avec une GUI, **l'exécution est décidée par l'utilisateur en fonction de ses interactions avec les différents widgets**, par exemple : un clic sur un bouton de souris, le déplacement de la souris, l'appui sur une touche du clavier ...

Comme c'est l'utilisateur qui décide quand et où il clique dans l'interface, il va falloir mettre en place ce qu'on appelle un « gestionnaire d'évènements ».

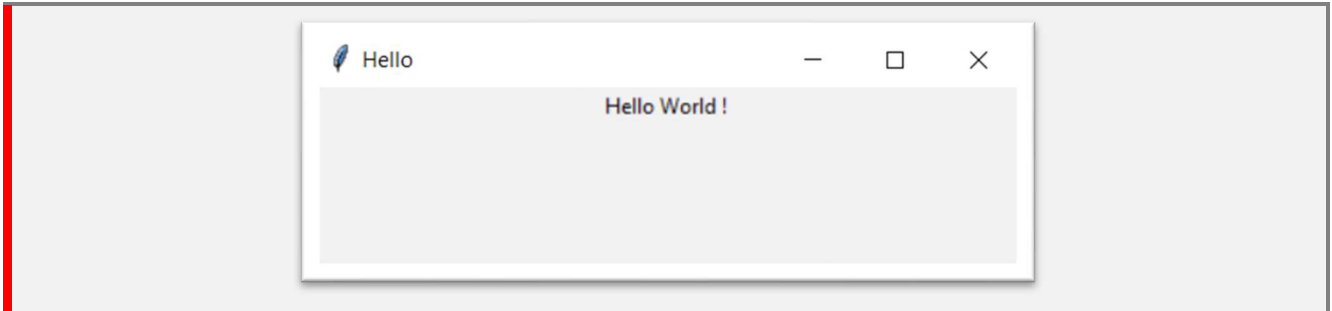
Le **gestionnaire d'évènements** est une sorte de « **boucle infinie** » qui est à l'affût de la moindre action de la part de l'utilisateur. C'est lui qui effectuera une action lors de l'interaction de l'utilisateur avec chaque **widget** de la GUI. Ainsi, l'exécution du programme sera réellement guidée par les actions de l'utilisateur.

1 Prise en main de tkinter

1.1 Test1 - Hello world, affichage de texte: widget Label

- ✓ Crée un fichier `tk1_hello_world_Text.py`, ouvre-le avec visual studio et copie le code expliqué ci-dessous.

Résultat avec l'application de bureau



Détail et explication du code

- ✓ Importe la librairie tkinter

```
from tkinter import *
```

- ✓ Crée une variable **fenetre** qui est une fenêtre graphique dans laquelle tu pourras placer différents éléments ou widgets : boutons (Button), étiquette (Label), zone de texte (Entry) ...
Tu pourras aussi gérer de nombreux événements : clic ou déplacement de la souris, appuie sur une touche du clavier ...

```
fenetre = Tk()
```

- ✓ Change le **titre** de la fenêtre et donne-lui une largeur de 400 pixels et une hauteur de 100 pixels

```
fenetre.title("Hello")  
fenetre.geometry('400x100')
```

- ✓ Crée un widgets **Label** appelé **message** pour afficher à l'écran Hello world !

```
message = Label(fenetre, text="Hello World !")
```

- ✓ **Positionne** le label message dans la fenêtre avec **pack**.

```
message.pack()
```

- ✓ Lance le **gestionnaire d'évènement**

```
fenetre.mainloop()
```

Code complet

```
from tkinter import *

# Création de la fenêtre graphique
fenetre = Tk()
fenetre.title("Hello")
fenetre.geometry('400x100')

# Création des widgets
bonjour = Label(fenetre, text="Hello World !")

# Positionnement des widgets sur la fenêtre graphique avec pack()
bonjour.pack()

# Lancement du gestionnaire d'évènements
fenetre.mainloop()
```

Exercice

- ✓ Rajoute un 2^{ème} label « Comment ça va ? » et observe où il se place.
- ✓ En t'aidant de la page 2 du **memo « Enjoliver vos widgets »**, et de visual studio code (avec l'aide lorsque tu tapes ton code), change la **taille du texte en 14, bleu, avec un fond jaune**.

1.2 Test2 – Positionnement des widgets sur la fenêtre graphique : pack, grid

- ✓ Crée un fichier tk2_Row_Column.py et écris 3 labels "Hello World !", "Comment ça va ?", "T'es sûr que tu vas bien ?".

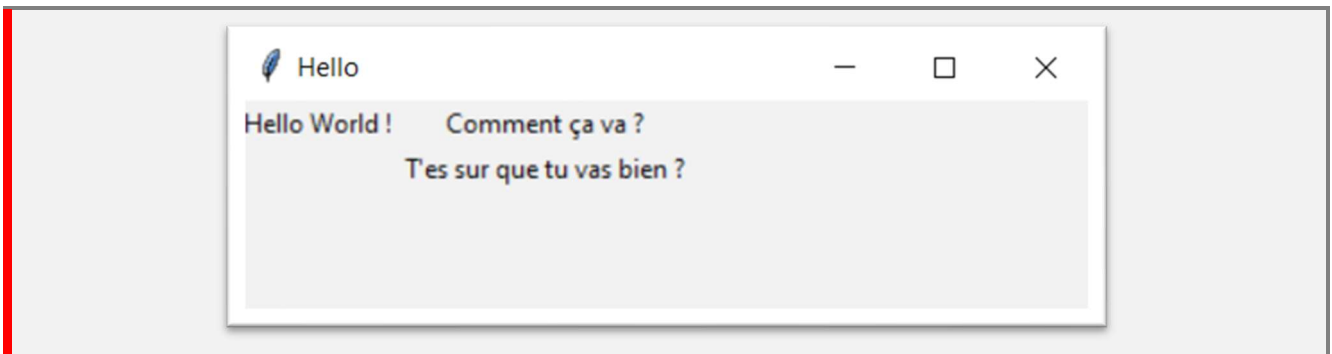
grid()

L'instruction **pack** positionne les labels les uns sous les autres.

Change pack par **grid** : la fenêtre est divisée virtuellement en plusieurs zones correspondant aux cases d'une grille. Chaque case est repérée par sa ligne et sa colonne, en commençant la numérotation en haut à gauche en (0,0). La taille des cases s'adapte aux dimensions de l'objet qu'elle contient.

Mets le 1^{er} label en haut à gauche, à côté à droite et le 3^{ème} en dessous à gauche.

Résultat avec l'application de bureau



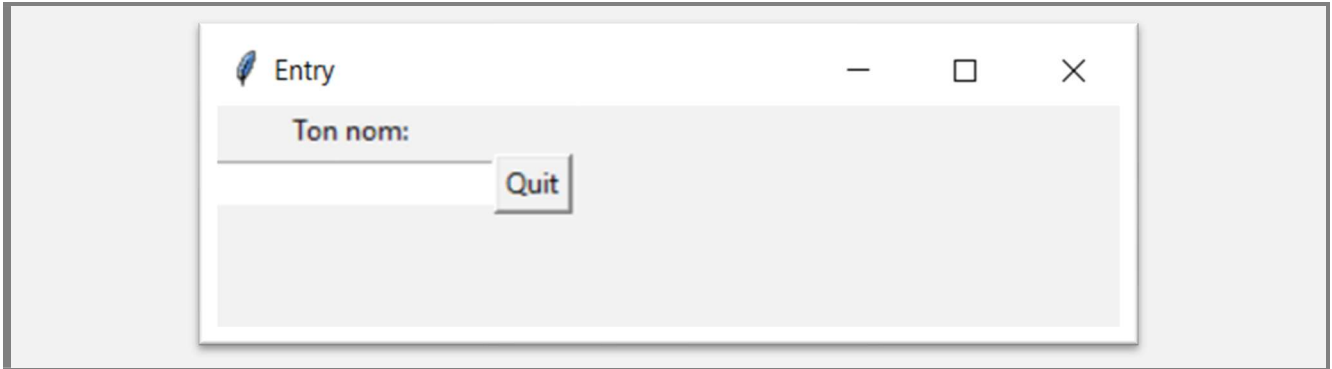
Extrait du code

```
bonjour.grid(row=0, column=0)
politesse.grid(row=0, column=1)
inquiétude.grid(row=1, column=1)
```

1.3 Test3 - Lecture clavier : widgets Entry et boutons Button

- ✓ Crée un fichier `tk3_TextField_ElevatedBoutons.py`, ouvre-le avec visual studio et copie le code expliqué ci-dessous.

Résultat avec l'application de bureau



Détail et explication du code

- ✓ Crée un champ dans lequel tu pourras rentrer ton nom. Le widget **Entry** est l'équivalent d'un `input()` dans une fenêtre graphique.

```
champ = Entry(fenetre)
```

- ✓ Rajoute un bouton pour fermer la fenêtre. Le widget **Button** permet de proposer une action à l'utilisateur.

```
bouton = Button(fenetre, text="Dis mon nom !", command=fenetre.destroy)
```

Code complet

```
from tkinter import *

# Création de la fenêtre graphique
fenetre = Tk()
fenetre.title("Entry")
fenetre.geometry('400x100')

# Création des widgets
consigne = Label(fenetre, text="Ton nom:")
champ = Entry(fenetre)
bouton = Button(fenetre, text="Dis mon nom !", command=fenetre.destroy)

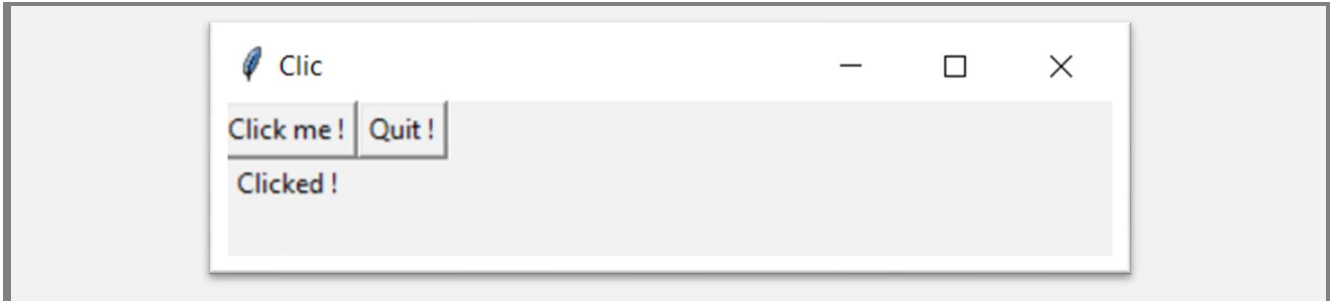
# Positionnement des widgets sur la fenêtre graphique avec grid()
consigne.grid(row=0,column=0)
champ.grid(row=1,column=0)
bouton.grid(row=1,column=1)

# Lancement du gestionnaire d'évènements
fenetre.mainloop()
```

1.4 Test4 - Gestion d'évènement : click souris

Crée un fichier `tk4_click_souris.py`, ouvre-le avec visual studio et copie le code expliqué ci-dessous. Le but est d'afficher « **Clicked** » lorsque l'on fait un clic souris sur le bouton.

Résultat avec l'application de bureau



La programmation évènementielle

Quand on utilise une interface graphique, la plupart du temps, il ne se passe rien : le programme attend que tu interviennes en écrivant quelque chose ou en cliquant sur un bouton ; tes interventions qui font réagir le programme sont des évènements. La surveillance des évènements est réalisée par la boucle d'évènements **mainloop** : c'est une boucle infinie qui scanne (ou capturent) les évènements et réagit à eux.

Par exemple, dans le code ci-dessous, lorsque tu cliques sur le bouton, la fonction `affiche_message` est appelée et affiche « clicked ! ».

Dans un 1^{er} temps, contente-toi d'afficher le message dans le terminal

```
# Définition de la fonction:
def affiche_message():
    print("Clicked !")

# Création des widgets
bouton_clic = Button(fenetre, text="Click me !", command=affiche_message)
label_message=Label(fenetre)
```

Modification d'une option de Label avec `config`

```
# Définition de la fonction:
def affiche_message():
    label_message.config(text="Clicked !")

# Création des widgets
bouton_clic = Button(fenetre, text="Click me !", command=affiche_message)
label_message=Label(fenetre)
```

Modification du Label avec la variable de contrôle **StringVar**

```
# Définition de la fonction:
def affiche_message():
    message.set("Clicked !")

# Création des widgets
bouton_clic = Button(fenetre, text="Click me !", command=affiche_message)
message = StringVar()
label_message = Label(fenetre, textvariable=message)
```

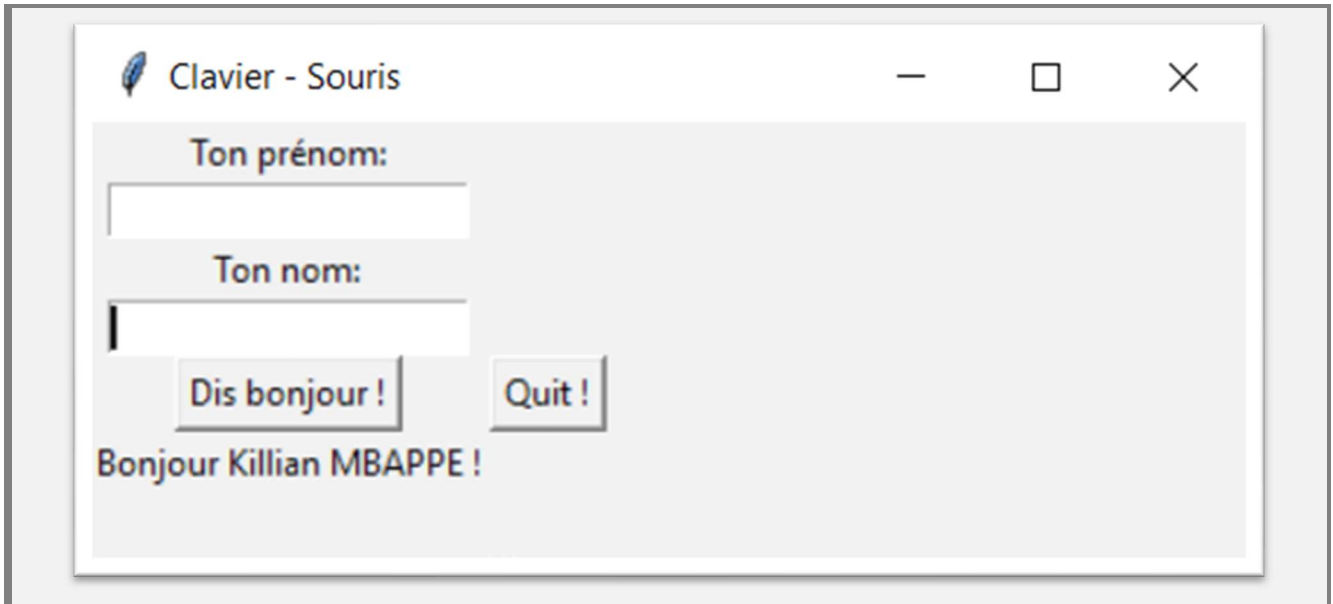
Exercice

Modifie le code de la fonction pour compter le nombre de clics.

1.5 Test5 - Exercice : lecture clavier, clic souris

Crée un fichier **tk5_clavier_souris.py**, ouvre-le avec visual studio et fais l'exercice ci-dessous. Le but est d'afficher quand on clique sur le bouton les prénoms et noms tapés au clavier .

Résultat avec l'application de bureau



1.6 Test6 - Exercice : compteur - / +

Crée un fichier **tk6_compteur.py**, ouvre-le avec visual studio et fais l'exercice ci-dessous. Le but est d'afficher un compteur qui va diminuer ou augmenter quand on clique sur le bouton - ou +.

- ✓ Crée 3 widgets: label_compteur, bouton_moins et boutons_plus
 - Le compteur est centré, a 100px de largeur
 - Les boutons affichent une icône moins / plus et appellent les fonctions minus_click ou plus_click
- ✓ Crée les fonctions minus_click ou plus_click qui vont diminuer ou augmenter la valeur du compteur.
- ✓ Rajoute un titre à la page
- ✓ Ajuste les contrôles dans une colonne, centrée au milieu de la page

1.7 Dessiner sur une toile: widget Canvas

Crée un fichier `tk7_canvas.py`, ouvre-le avec visual studio et copie le code expliqué ci-dessous.

- ✓ Crée un widget **Canvas** de **400x400**: il s'agit d'une zone graphique dans lequel tu peux dessiner ou écrire ce que tu veux (lignes, rectangles, cercles ...)

```
xmax, ymax=400,400  
can1=Canvas(fenetre, bg='grey',height=ymax,width=xmax)
```

- ✓ Rajoute à côté un bouton pour fermer la fenêtre.
- ✓ Trace un trait horizontal à 10px du bas de la fenêtre avec **create_line**.

```
can1.create_line(0,390,xmax,390,width=2,fill="red")
```

- ✓ Trace un carré bleu de 100px de côté en haut à gauche avec **create_rectangle**.

```
rect=can1.create_rectangle(0,0,100,100, fill='blue')
```

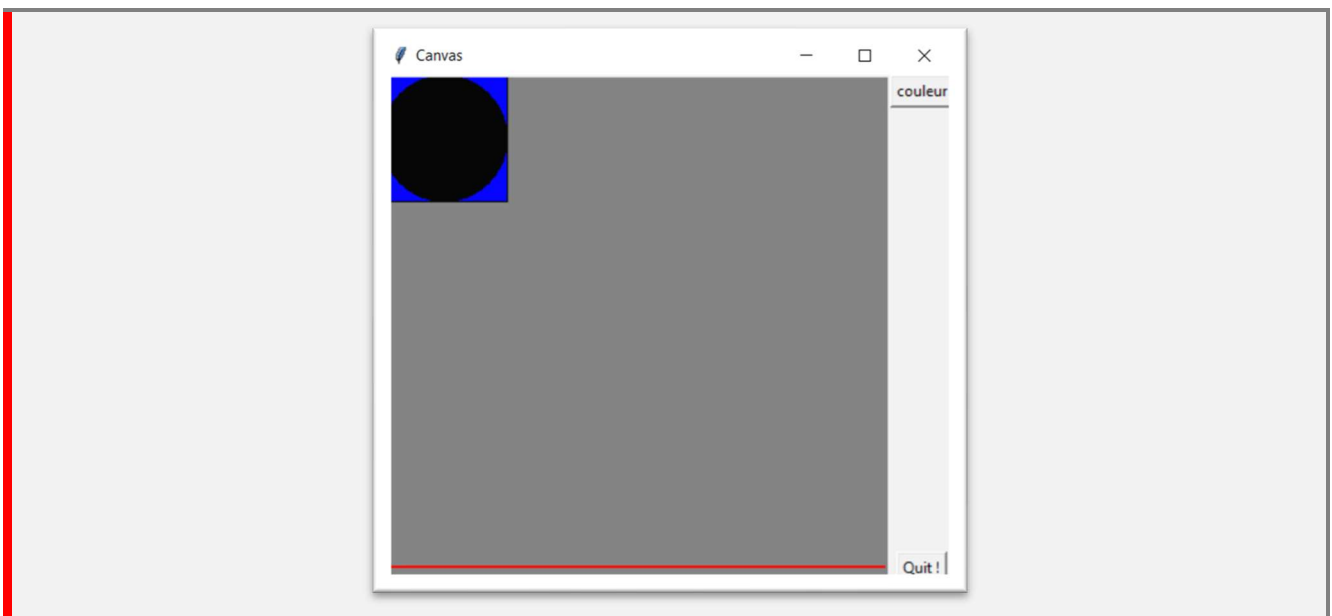
- ✓ Trace un cercle rempli en noir de même taille par-dessus avec **create_oval**.

```
cercle=can1.create_oval(0,0,100,100, fill='black')
```

- ✓ Rajoute un bouton qui appelle la fonction `change_couleur` afin de changer la couleur du cercle en rouge avec **itemconfig**

```
def change_couleur():  
    can1.itemconfig(cercle,fill='red')
```

Résultat avec l'application de bureau



Code complet

```
from tkinter import *
# Création de la fenêtre graphique
fenetre = Tk()
fenetre.title("Canvas")
xmax, ymax=400,400

# Définition des fonctions
def change_couleur():
    can1.itemconfig(cercle,fill='red')

# Création des widgets
can1=Canvas(fenetre, bg='grey',height=ymax,width=xmax)
can1.create_line(0,390,xmax,390,width=2,fill="red")
rect=can1.create_rectangle(0,0,100,100, fill='blue')
cercle=can1.create_oval(0,0,100,100, fill='black')

bouton_couleur=Button(fenetre, text="couleur",command=change_couleur)

bouton_quit = Button(fenetre, text="Quit !", command=fenetre.destroy)

# Positionnement des widgets
can1.pack(side=LEFT)
bouton_couleur.pack()
bouton_quit.pack(side=BOTTOM)

# Lancement du gestionnaire d'évènements
fenetre.mainloop()
```

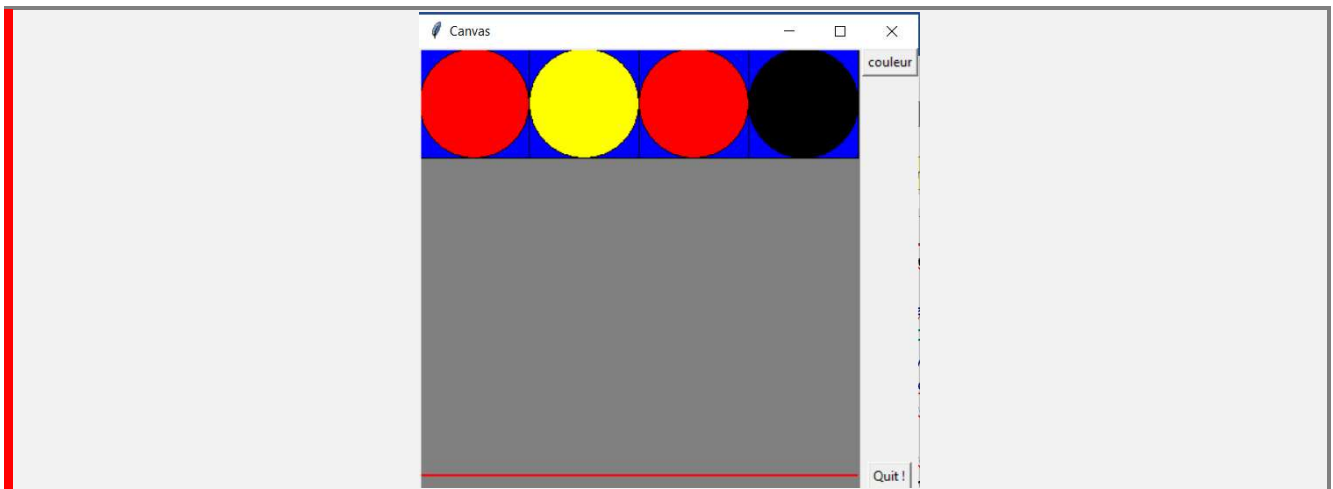
- ✓ Modifie le code pour qu'à chaque clic, la couleur change entre le jaune et le rouge.

```
def change_couleur():  
    can1.itemconfig(cercle,fill='red')
```

1.8 Dessiner sur une toile: widget Canvas (bis)

- ✓ Crée un fichier **tk8a_canvas.py**, ouvre-le avec visual studio et copie le code expliqué ci-dessous.
- ✓ Repars du code précédent pour afficher 4 carrés bleus / cercles noirs côte à côte.

Résultat avec l'application de bureau



- ✓ Pour cela crée une fonction `creat_widgets(can1)` qui re-affiche les 4 carrés à chaque évènement.

```
coul_def={0:'black', 1:'yellow', 2:'red'}  
coul_sel=[0,0,0,0] # changement de la couleur avec coul_def[cou_sel]  
  
# Définition des fonctions  
def creat_widgets(can1):  
    can1.delete()  
    rect=can1.create_rectangle(0,0,100,100, fill='blue')  
    cercle=can1.create_oval(0,0,100,100, fill=coul_def[coul_sel[0]])  
    ...
```

- ✓ Une autre façon de faire est créer une liste contenant les widgets cercle et de modifier la couleur avec **itemconfig**
- ✓ Profites-en pour pouvoir modifier la couleur avec un clic souris sur le cercle
- ✓ Crée les widgets avec une boucle for

```

coul_def={0:'black', 1:'yellow', 2:'red'}
coul_sel=[0,0,0,0] # changement de la couleur avec coul_def[cou_sel]
cercle_lst=[]
imax=4

# Définition des fonctions
def change_couleur(event):
    global coul_sel

    x, y = event.x, event.y
    i,j= x//100, y//100
    print(x, y, x//100, y//100)
    if coul_sel!=1:
        coul_sel=1
    else:
        coul_sel=2
    can1.itemconfig(cercle_lst[i],fill=coul_def[coul_sel])

# Création des widgets
can1=Canvas(fenetre, bg='grey',height=ymax,width=xmax)

for i in range(imax):
    rect=can1.create_rectangle(i*100,0,(i+1)*100,100, fill='blue')
    cercle=can1.create_oval(i*100,0,(i+1)*100,100, fill=coul_def[coul_sel[0]])
    cercle_lst.append(cercle)

bouton_couleur=Button(fenetre, text="couleur",command=change_couleur)

```