Pour aller plus loin…

Découverte du module graphique Tkinter

Tkinter *(de l'anglais* ***Tool kit interface****) est le module graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques. L'un des avantages de* ***Tkinter*** *est sa portabilité sur les OS les plus utilisés par le grand public. Le module (classe)* ***Tkinter*** *contient des fonctions intégrées appelées méthodes permettant de réaliser des actions sur les fenêtres graphiques (objets).*

0. Quelques concepts liés à la programmation graphique

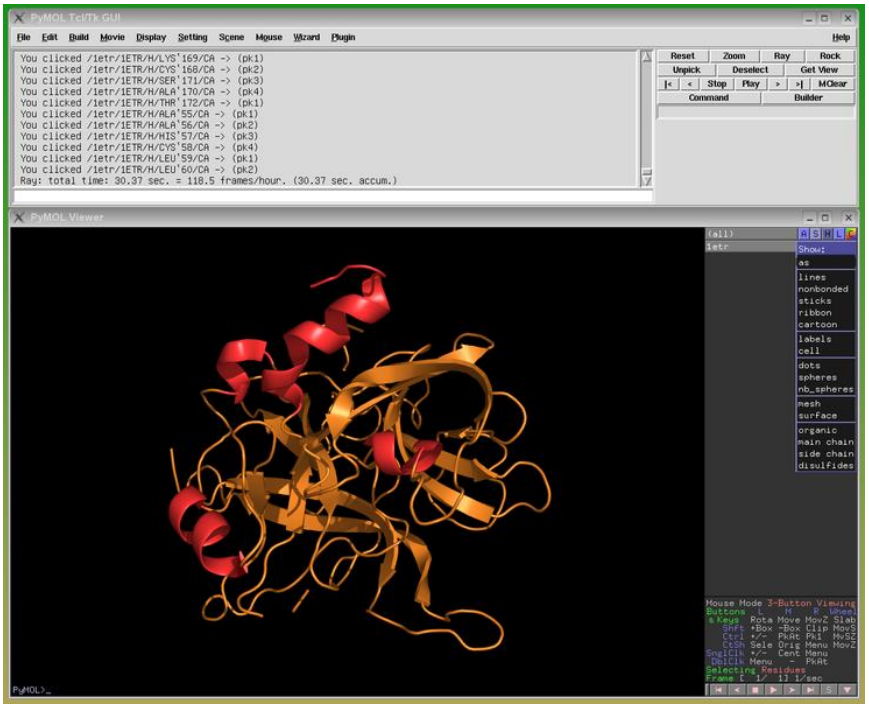
Lorsque l'on développe une **GUI** (**Graphical User Interface**), nous créons une fenêtre graphique contenant notre application, ainsi que des ***widgets*** inclus dans la fenêtre. Les ***widgets*** (***window gadget***) sont des objets graphiques permettant à l'utilisateur d'interagir avec le programme 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 vous avez 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*. 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'événements ».

Le gestionnaire d'événements 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.

Sites ressources : http://tkinter.fdex.eu/index.html

http://s15847115.domainepardefaut.fr/python/tkinter/index.html



GUI : le logiciel libre PYMOL de visualisation de molécules en 3D utilise Tkinter

1. Première fenêtre graphique : Hello World !

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..

Dans un **IDE** (Integrated Development Environment) PYTHON tel que **EduPython** ou **Spyder** par exemple, 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 ?
Q2. Expliquer le rôle des lignes 6,7 et 8.
Q3. Que se passe-t-il au niveau de l'affichage si on ne met pas la ligne 8 ?
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 ?
Q5. Qu'est-ce qu'un widget ? Quels sont ceux utilisés dans le programme ci-dessus ?
Q6. Quelle méthode est utilisée pour positionner les widgets ?
Q7. Que se passe-t-il si on oublie la ligne 22 ?

2. Connaître les widgets de base et savoir les positionner

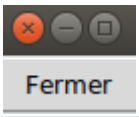
Pour créer un logiciel graphique vous devez ajouter dans une fenêtre des éléments graphiques que l'on nomme widget (window gadget).

2.1. Les widgets de base

■ Widget Button()

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

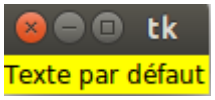
```
# Bouton de sortie
bouton=Button(fenetre, text="Fermer", command=fenetre.destroy)
bouton.pack()
```



■ 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()
```



■ Widget Entry()

L'équivalent d'un **input** dans une fenêtre graphique est le **widget Entry()** (entrée).

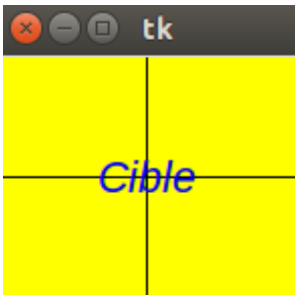
```
# Champ d'entrée
value = StringVar()
value.set("Texte par défaut")
entree = Entry(fenetre, textvariable=value, width=30)
entree.pack()
```

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()     # fenetre
```

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

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

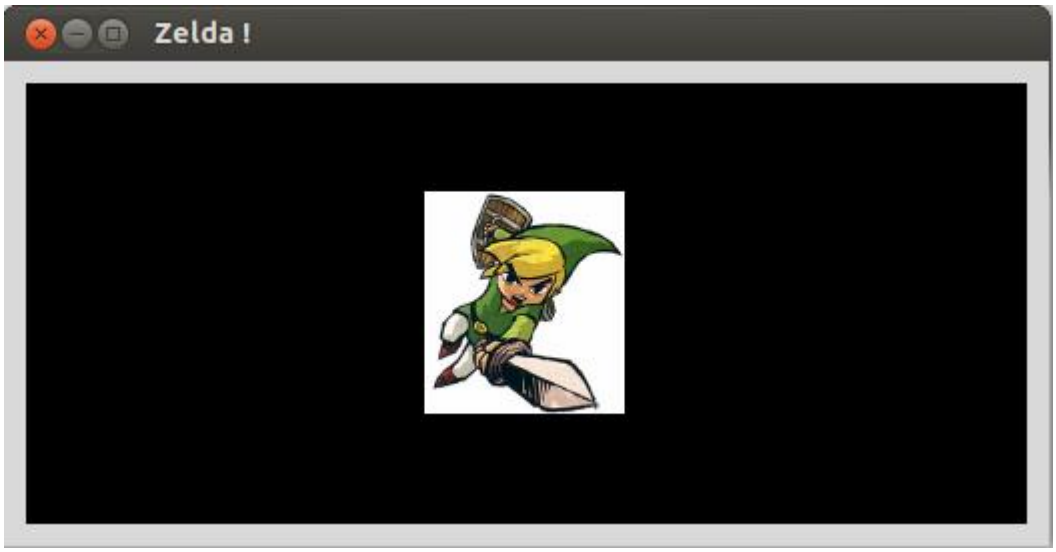
Pour supprimer un élément vous pouvez utiliser la méthode **delete :**

```
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 :

```
canevas = 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 :

Paramètres: **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.
tags – Si c'est une chaîne seule, elle sert à marquer (*tag*) l'image. Utiliser un tuple de chaînes pour lui attribuer plusieurs marques.

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

2.2. 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
```

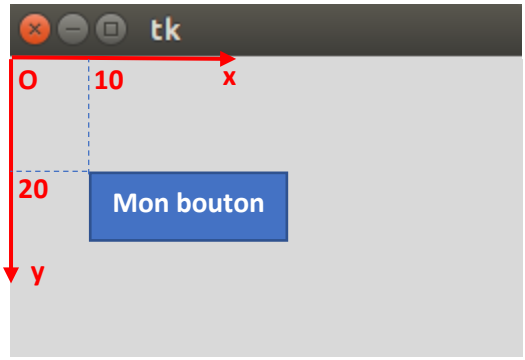
Exemple :

```
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 des 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.

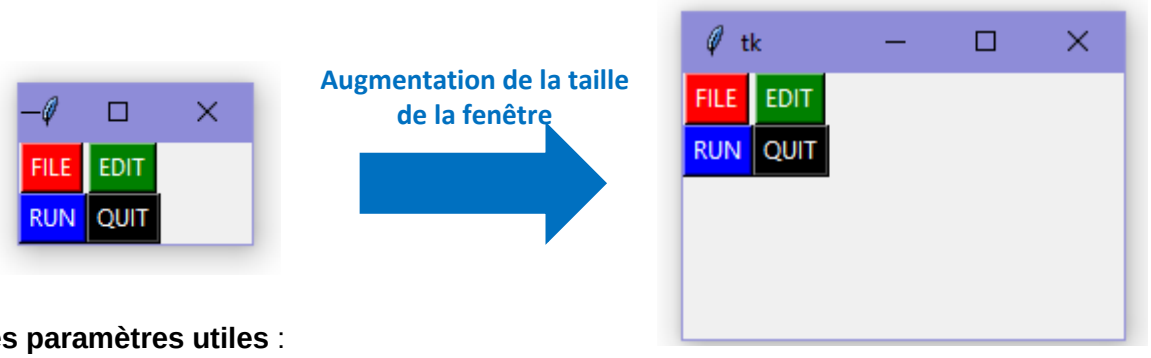
```
Button(fenetre, text ='Mon bouton').place(x=10, y=20)
```



■ 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 : Comme dans la plupart des cas, le premier élément n'est pas l'élément 1 mais l'élément 0.

```
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.

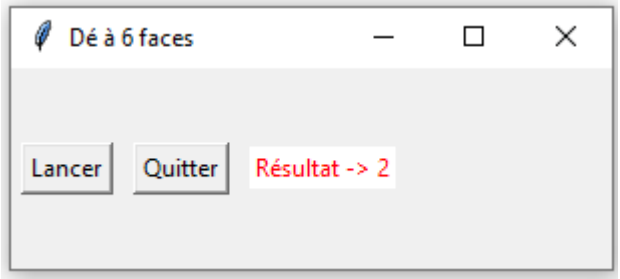
- **ipadx** – marge interne horizontale (en *x*). Cette dimension est ajoutée à l'intérieur du widget sur ses côtés gauche et droit.
- **ipady** – marge interne verticale (en *y*). Cette dimension est ajoutée à l'intérieur du widget sur ses côtés haut et bas.
- **padx** – Marge externe horizontale.
- **pady** – Marge externe verticale.

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

3. Deuxième fenêtre graphique : simuler un lancer de dé

Dans un **IDE** (Integrated Development Environment) PYTHON tel que **EduPython** ou **Spyder** par exemple, exécuter les lignes de code PYTHON suivantes :

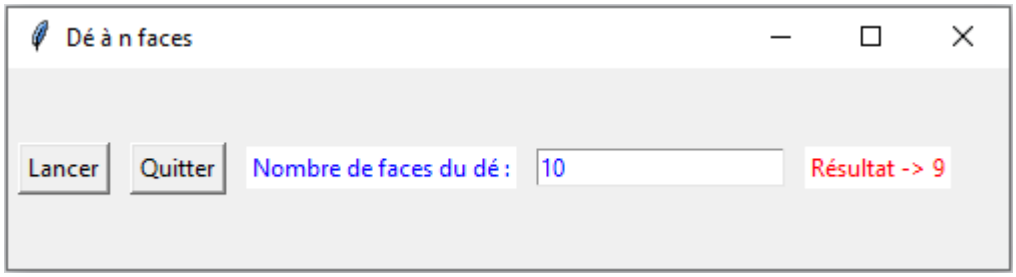
```
1  # -*- coding: utf-8 -*-
2
3  from tkinter import *
4  from random import *
5
6  # Définition de la fonction
7  def nouveauLance ():
8      nb = randint(1,6)
9      texte.set('Résultat -> ' + str(nb))
10
11 # Création de la fenêtre principale (Main Window)
12 fenetre = Tk()
16 fenetre.title('Dé 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()
```



Analyse du programme :

Q9. A quoi sert la fonction définie à la ligne 7 ?
Q10. Quelle est l'action du bouton à la ligne 17 ?
Q11. 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** ».

Q12. 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 :



FOCUS sur les variables de contrôle : les valeurs dans les widgets

Une **variable de contrôle** de Tkinter est un objet spécial qui se comporte comme une variable ordinaire de Python en ce sens que c'est un **conteneur** pour une valeur comme un nombre ou une chaîne de caractères.

Tout l'intérêt des variables de contrôle, c'est qu'elles peuvent-être partagées entre plusieurs widgets, et qu'elles se souviennent de tous les widgets qui les partagent à un moment donné. En particulier, cela signifie que si votre programme mémorise une **valeur** notée **val** dans une **variable de contrôle** notée **cont** en utilisant la méthode **cont.set(val)**, tous les widgets qui sont reliés à cette variable de contrôle sont automatiquement mis à jour sur l'écran.

Tkinter utilise des variables de contrôles pour réaliser de nombreuses fonctionnalités. Par exemple, vous pouvez relier un widget **Entry** (boîte de saisie) à un widget **Label** (étiquette) de telle sorte que quand l'utilisateur modifie le texte du premier, le second soit automatiquement mis à jour pour montrer le même texte.

Pour obtenir une variable de contrôle, utilisez l'un des constructeurs suivants, en fonction du type de valeur que vous souhaitez mémoriser :

```
val = DoubleVar()    # Mémorise un flottant; sa valeur par défaut est 0.0
val = IntVar()       # Mémorise un entier; sa valeur par défaut est 0
val = StringVar()    # Mémorise une chaîne de caractères; sa valeur par défaut est ''
```

Toutes les variables de contrôle disposent des 2 méthodes suivantes :

.get()
Retourne la valeur courante de la variable de contrôle.

.set(*valeur*)
Modifie la valeur courante de la variable de contrôle. Si les options d'un ou plusieurs widgets sont reliées à cette variable, ces widgets seront automatiquement mis à jour quand la boucle principale sera à nouveau en attente.

Voici quelques indications sur la façon dont les variables de contrôles sont utilisées avec certains widgets.

■ Button

Vous pouvez renseigner son option **textvariable** avec une StringVar. À chaque fois que cette variable est modifiée, le texte du bouton sera mis à jour pour afficher la nouvelle valeur. Il n'est pas nécessaire d'utiliser cela sauf si le texte du bouton doit changer au cours du temps: utilisez l'option **text** si l'étiquette du bouton ne change pas.

■ Entry

Positionnez son option **textvariable** avec une StringVar. Utilisez alors sa méthode get() pour récupérer le texte actuellement affiché dans la boîte de saisie. Vous pouvez aussi modifier ce texte en utilisant sa méthode .set().

■ Label

Positionnez son option **textvariable** avec une StringVar. En appelant sa méthode set(), vous pourrez modifier le texte affiché sur cette étiquette. Il n'est pas nécessaire d'utiliser cela ; si vous ne prévoyez pas de changer son texte, utilisez plutôt son option **text** pour renseigner son contenu.

4. Troisième fenêtre graphique : interface d'authentification

Dans un **IDE** (Integrated Development Environment) PYTHON tel que **EduPython** ou **Spyder** par exemple, exécuter les lignes de code PYTHON suivantes :

```
1  # -*- coding: utf-8 -*-
2
3  from tkinter import *
4  from tkinter.messagebox import *
5
6  # Définition de la fonction
7  def Verification():
8      if Motdepasse.get() == 'python123':
9          # le mot de passe est bon : on affiche une boîte de dialogue puis on  ferme la fenêtre
10         showinfo('Résultat','Mot de passe correct.\nAu revoir !')
11         Mafenetre.destroy()
12     else:
16         # le mot de passe est incorrect : on affiche une boîte de dialogue
14         showwarning('Résultat','Mot de passe incorrect.\nVeuillez recommencer !')
15         Motdepasse.set('')
16
17 # Création / positionnement de la fenêtre principale (Main Window)
18 Mafenetre = Tk()
19 Mafenetre.title('Identification requise')
20
21 # Création / positionnement d'un widget Label (texte 'Mot de passe')
22 Label1 = Label(Mafenetre, text = 'Mot de passe ')
23 Label1.pack(side = LEFT, padx = 5, pady = 5)
24
25 # Création / positionnement d'un widget Entry (champ de saisie)
26 Motdepasse= StringVar()
27 Champ = Entry(Mafenetre, textvariable= Motdepasse, show='*', bg ='yellow', fg='blue')
28 Champ.focus_set()
29 Champ.pack(side = LEFT, padx = 5, pady = 5)
30
31 # Création / positionnement d'un widget Button (bouton Valider)
32 Bouton = Button(Mafenetre, text ='Valider', command = Verification)
33 Bouton.pack(side = LEFT, padx = 5, pady = 5)
34
35 # Lancement du gestionnaire d'événements
36 Mafenetre.mainloop()
```

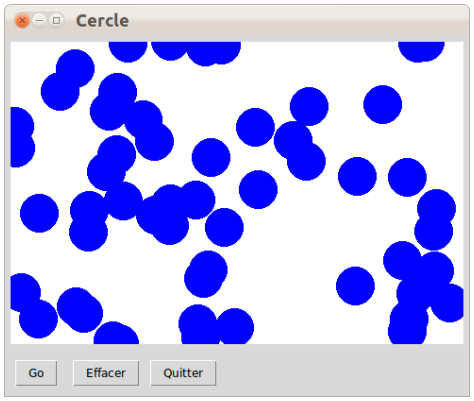
Les messages d'alertes reposent sur la boîte de dialogue **messagebox**.

Q13. Expliquer ce que réalise la fonction de la ligne 7.

5. Quatrième fenêtre graphique : widgets Canvas et Button

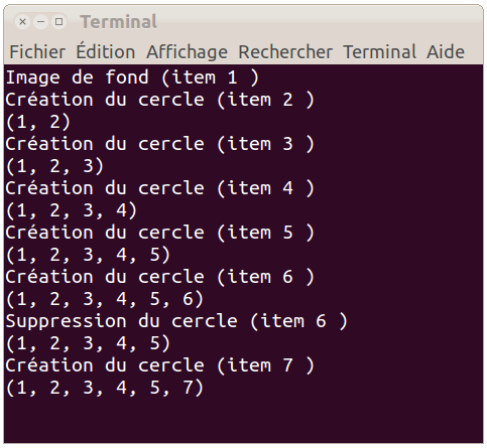
Le programme suivant dessine, à chaque clic sur le bouton Go, un disque de rayon 20 pixels à une position aléatoire :

```
# -*- coding: utf-8 -*-

from tkinter import *
from random import *

# Définition des fonctions
def Cercle():
    """ Dessine un cercle de centre (x,y) et de rayon r """
    x = randint(0,Largeur)
    y = randint(0,Hauteur)
    r = 20
    Canevas.create_oval(x-r, y-r, x+r, y+r, outline='blue', fill='blue')

def Effacer():
    """ Efface la zone graphique """
    Canevas.delete(ALL)

# Création de la fenêtre principale (Main Window)
Mafenetre = Tk()
Mafenetre.title('Cercle')

# Création / positionnement d'un widget Canvas (zone graphique)
Largeur = 480
Hauteur = 320
Canevas = Canvas(Mafenetre, width = Largeur, height =Hauteur, bg ='white')
Canevas.pack(padx =5, pady =5)

# Création / positionnement d'un widget Button (bouton Go)
BoutonGo = Button(Mafenetre, text ='Go', command = Cercle)
BoutonGo.pack(side = LEFT, padx = 10, pady = 10)

# Création / positionnement d'un widget Button (bouton Effacer)
BoutonEffacer = Button(Mafenetre, text ='Effacer', command = Effacer)
BoutonEffacer.pack(side = LEFT, padx = 5, pady = 5)

# Création / positionnement d'un widget Button (bouton Quitter)
BoutonQuitter = Button(Mafenetre, text ='Quitter', command = Mafenetre.destroy)
BoutonQuitter.pack(side = LEFT, padx = 5, pady = 5)

# Lancement du gestionnaire d'événements
Mafenetre.mainloop()
```



6. Cinquième fenêtre graphique : gestions des images sur un Canvas

On souhaite réaliser un programme affichant dans une fenêtre graphique l'image d'une cible sur laquelle on simulera des impacts de tirs. Les impacts seront des disques de rayon 10 pixels dont le centre sera généré aléatoirement sur la surface de l'image (550 px sur 550px) après chaque appui sur le bouton « Tirer ». L'image de la cible est téléchargeable ici. Le bouton « Effacer le dernier tir » devra permettre d'effacer la dernière action (pour cela, on se sert du numéro identifiant de chaque item d'un widget Canvas).



Q14. A l'aide du programme du § 5, implémenter un programme PYTHON répondant au cahier des charges.