NSI	IHM & Web Interface graphique	TP – **Interface graphique avec Tkinter**

Objectifs :

- ⇨ Apprendre les principes de fonctionnement d'une interface graphique
- ⇨ Créer des interfaces graphiques simples pour de petits programmes

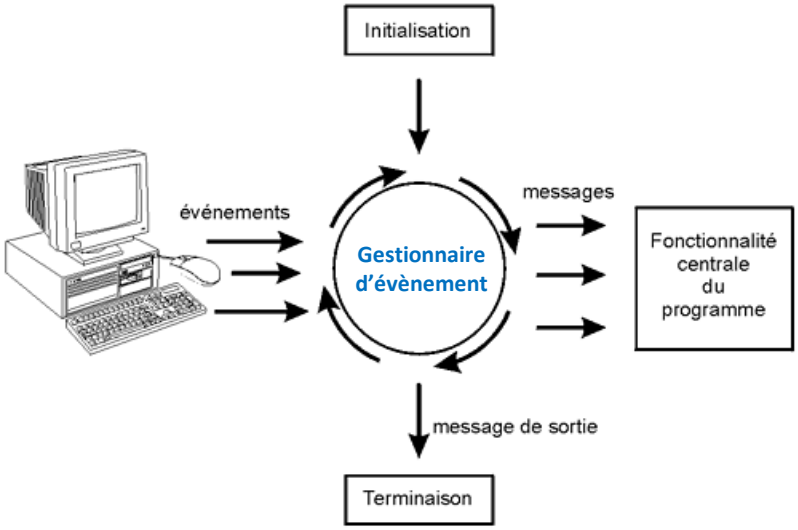
I – Principes généraux

1) Qu'est-ce qu'une interface graphique ?

On appelle ***interface*** (ou UI : **U**ser **I**nterface) tout composant logiciel qui fait la liaison entre deux entités. Ici la liaison est faite entre l'utilisateur (humain) et le programme python. En ce sens la console python est une interface. C'est même celle que nous utilisons depuis le début pour sa simplicité. Mais cette interface est assez simple et ne peut représenter que du texte. Une interface *graphique* (ou GUI : **G**raphical **UI**) permet de communiquer visuellement avec l'utilisateur de manière plus riche.
Il existe de nombreuses interfaces graphiques disponibles pour python (wxPython, PyQt, kivy, Pygame, …), mais nous étudierons ici plus précisément la plus répandue : Tkinter (pour **T**ool **k**it **inter**face).

2) Comment cela fonctionne-t-il ?

Dans un programme classique, les instructions s'enchaînent séquentiellement jusqu'à la fin du programme. Dans un programme faisant appel à une interface graphique, on doit changer de paradigme et passer à de la **programmation évènementielle**, comme en javascript. En effet, ici le programme doit réagir à des évènements liés principalement à l'utilisateur, comme : clic de la souris, déplacement de la souris, appui sur une touche, activation d'un élément graphique (bouton, menu, case à cocher, …) ou encore programmés à l'avance (déclenchement d'un minuteur).
Après sa phase d'initialisation, le programme de ce type se met en quelque sorte « en attente », et passe la main à un autre composant logiciel (ici l'interface graphique Tkinter), lequel est plus ou moins intimement intégré au système d'exploitation de l'ordinateur et « tourne » en permanence.



Le gestionnaire d'événements scrute sans cesse tous les périphériques (clavier, souris, etc.) et réagit immédiatement lorsqu'un événement y est détecté. Lorsqu'il détecte un événement, le gestionnaire vérifie si une fonction du programme a été « attachée » (bind) à cet évènement et si c'est le cas exécute la fonction en question. Dans le cas d'un programme avec interface graphique, l'ordre dans lequel les fonctions sont appelées n'est plus inscrit nulle part dans le programme. Ce sont les événements qui pilotent !

3) Constitution d'une interface graphique

a. Les widgets

Une interface graphique contient une ou plusieurs fenêtres qui sont créées vides[1]. On peut ensuite y rajouter des **éléments graphiques** ou Widgets (**Wi**ndows Ga**dgets**). Ces éléments peuvent être de plusieurs types :

[1] Par défaut il y aura uniquement une fenêtre. Si on veut en créer d'autres, il faut utiliser la classe `Toplevel` (voir ici).

Widgets d'interaction (entrée) :

- Bouton (`Button`)
- Case à cocher (`Checkbutton`)
- Bouton radio (`Radiobutton`)
- Champ de saisie uniligne (`Entry`) ou multilignes (`Text`)
- Liste de sélection (`Listbox`)
- Menu (`Menu`)
- Curseur (`Scale`)
- Choix d'un nombre (`Spinbox`)
- Onglets (`Notebook`)
- …

Widgets d'affichage (sortie) :

- Texte simple (`Label`)
- Zone de dessin (`Canvas`)
- Barre de progression (`Progressbar`)
- Séparateur (`Separator`)
- …

Widgets conteneur (pour organiser les widgets) :

- Conteneur rectangulaire simple (`Frame`) ou étiquetés (`NamedFrame`)
- Conteneur à panneaux (`PanedWindow`)

b. Attributs

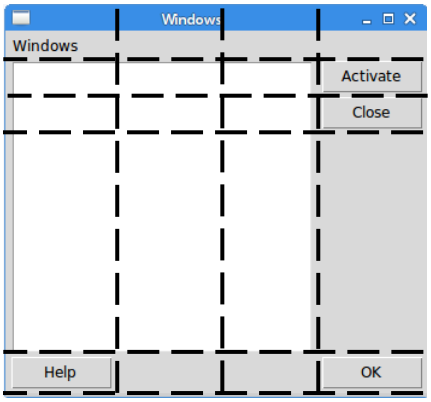
Les différents widgets d'une interface ont des attributs dont une partie est assez générique et que l'on peut modifier. Parmi les attributs génériques (que l'on retrouve dans à peu près tous les widgets), on peut trouver notamment :

- Le texte (`text=` chaîne de caractère écrite sur le widget)
- L'état (`state=` `"disabled"`, `"active"`, `"normal"`)
- Les dimensions (`width`, `height`, `borderwidth`, `padX` (marge horizontale), `padY`, …)
- Les couleurs (`bg` (fond), `fg` (avant), `highlightcolor`, …)
- Les polices (`family`, `size`, `weight`, …)
- L'ancrage (`anchor=` chaîne utilisant les points cardinaux (`n`,`s`,`e`,`o`) ou `center`) pour les images
- Le relief (`"raised"`, `"sunken"`, `"flat"`, `"groove"` ou `"ridge"`)
- L'icône (`bitmap=` au choix parmi 10 icônes standard ou `"@`*`fichier`*`.xbm"`)
- Le curseur de la souris au survol (`cursor=` `"arrow"`, `"circle"`, `"clock"`, `"cross"`, …)

Voir http://tkinter.fdex.eu/doc/sa.html (ou sa copie) pour des explications en français.

c. Placement des widgets

Le placement des widgets est confié à un *gestionnaire de placement* auquel on doit donner des directives précises.
Le gestionnaire le plus simple se contente de placer le widget précisément aux coordonnées x et y fournies avec la taille voulue. Mais ce n'est généralement pas ce qui est utilisé car l'interface graphique serait alors figée en taille alors qu'on souhaite généralement un contenu qui s'adapte à la taille de la fenêtre et de l'écran.
On a alors recours à d'autres gestionnaires de placement auxquels on va donner des instructions comme « place ce widget dans le coin supérieur gauche en divisant l'espace en 3 x 3 zones de taille égales ».



Placement des widgets de la fenêtre sur une grille étirable

Placer les widgets de manière judicieuse et adaptable est souvent la partie la

plus délicate de la création d'une GUI.

d. Fonction callback

Les fonctions « **callback** » sont les fonctions du programme qui sont appelées par le gestionnaire d'évènement quand un évènement qu'il doit gérer survient. Il exécute alors cette fonction (qui est généralement programmée par l'utilisateur) à laquelle il passe en argument un objet `event` décrivant l'évènement qui a déclenchée son exécution. Par exemple si c'est un appui sur une touche qui a déclenché l'évènement, `event` permettra de connaitre la touche en question et si une touche modificatrice (majuscule, Ctrl, Alt…) était appuyée en même temps. Pour certains widgets comme les boutons, l'association à une fonction callback peut se faire directement à la création (avec l'argument « `command=` »). Sinon il faut les lier après avoir défini le widget et la fonction.

Lorsque ces fonctions sont appelées par le gestionnaire d'évènement, il n'est pas possible de leur fournir d'autre argument que l'objet `event`. En conséquence *ces fonctions n'auront accès qu'aux variables* ***globales*** *du programme*.

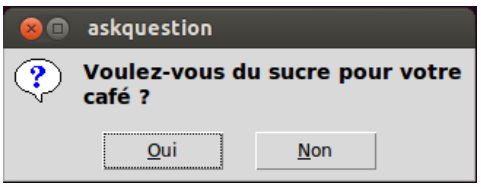
Rappel : En python, toutes les fonctions ont accès en lecture seule aux variables globales, mais si on veut modifier une variable globale, python crée à la place une variable locale qui va *masquer* la variable globale. Pour modifier une variable globale dans une fonction (ce qui nous sera souvent utile en programmation évènementielle), on doit utiliser le mot-clé `global`.

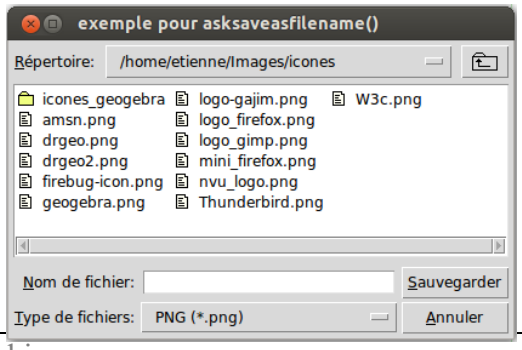
```
def f():
    b = a + 2
    print(b)  # Affiche 10

def g():
    global b
    b = a + 2

a = 8
b = 3
f()
print(b) # Ici b vaut toujours 3
g()
# Maintenant b vaut 10
```

e. Boîtes de dialogue

Les GUI permettent également d'afficher des boîtes de dialogue toutes faites de manière simple pour :

- Donner une information (`showinfo`, `showwarning`, `showerror`)
- Proposer un choix simple (`askquestion`, `askokcancel`, `askyesno`, `askretrycancel`)
- Demander une saisie (`askstring`, `askinteger`, `askfloat`)
- Sélectionner un fichier (`askopenfilename` et `asksaveasfilename`)
- Sélectionner une couleur (`askcolor`)





II - Mise en place avec Tkinter

1) Structure

Un programme avec interface graphique via Tkinter aura donc la structure suivante :

```
import tkinter as tk    # Importe la bibliothèque de fonction

fenetre = tk.Tk()       # Récupère l'objet fenêtre de base

# Création et ajout de tous les widgets
# et liens avec les fonctions de traitement

fenetre.mainloop()      # Passe la main au gestionnaire d'évènement

# Et ici le code exécuté lorsque la fenêtre principale sera fermée
```

La dernière instruction du programme est donc de lancer le gestionnaire d'évènement. Le programme ne récupèrera la main que lorsque des évènements auxquels il a lié des fonctions apparaissent. Il sera terminé lorsqu'on fermera la fenêtre principale (ce qui terminera la `mainloop()`).

2) Placement des éléments graphiques

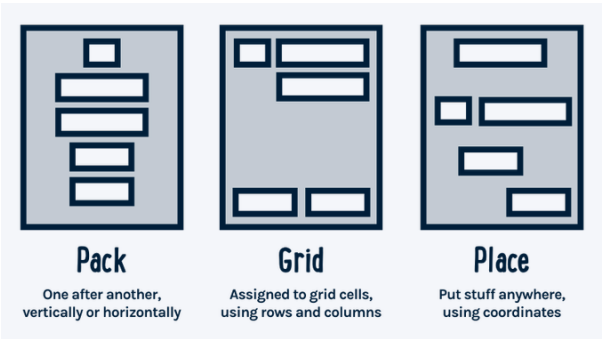
Le placement d'un widget se fait généralement en trois étapes :

1. Création de l'objet widget avec la méthode appropriée de l'objet `fenetre` obtenu par `tk.Tk()`. C'est à ce moment là qu'on indique le widget *parent* de notre nouveau widget (le widget à l'intérieur duquel va se placer le nouveau widget. Après cette étape, l'objet existe mais il n'est pas encore visible.

```
import tkinter as tk
fenetre = tk.Tk()

btn = tk.Button(fenetre, text="Test")
```

2. Placement de l'objet à l'intérieur de son conteneur avec une méthode de placement (`pack()`, `grid()` ou `place()`[2]). Après cette étape l'objet apparaît mais n'est pas vraiment actif.

```
btn.pack(side="right")
```



3. Rattachement d'un type d'évènement pour ce widget à une fonction callback avec la méthode `bind()` du widget. Voir http://tkinter.fdex.eu/doc/event.html ou sa copie pour davantage de détails.

```
btn.bind("<Button-3>",clic_droit)  # Fonction appelée en cas de clic droit
```

3) Variables de contrôle

Tkinter utilise des variables particulières que l'on peut lier à un ou plusieurs widgets et qui sont accessibles partout (comme des variables globales).

[2] Attention : On ne peut utiliser qu'un seul de ces gestionnaires à la fois dans un même conteneur (mais deux conteneurs différents peuvent avoir des gestionnaires différents).

Elles permettent de modifier le texte de l'interface (par exemple le texte d'un `Label` ou celui d'un `Button`), ou la valeur d'un `Scale` par exemple.
Pour créer une variable de contrôle `v`, on utilise :

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

Ensuite pour modifier sa valeur :

```
v.set(nouvellevaleur)
```

Le ou les objets liés à cette variable de contrôle seront automatiquement mis à jour sans que l'on n'ait rien de plus à faire.

Et pour récupérer sa valeur :

```
valeur_de_v = v.get()
```

On récupère alors la valeur de `v` au *moment de l'exécution de cette ligne de code*.

Exemple d'utilisation :

```
def depart():
    # On bascule le texte du bouton entre "START/stop" et "start/STOP"
    if texte_btn.get() == "START/stop" :
        texte_btn.set("start/STOP")
    else:
        texte_btn.set("START/stop")

texte_btn = tk.StringVar()
texte_btn.set("START/stop")
btn = tk.Button(fenetre, textvariable=texte_btn, command=depart).pack()
```

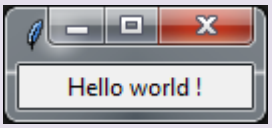
Remarque : Certains widgets ont une méthode `get()` qui permet de récupérer leur valeur interne et dans ce cas l'utilisation d'une variables de contrôle n'est pas indispensable.

4) Hello world !

Question 1 :

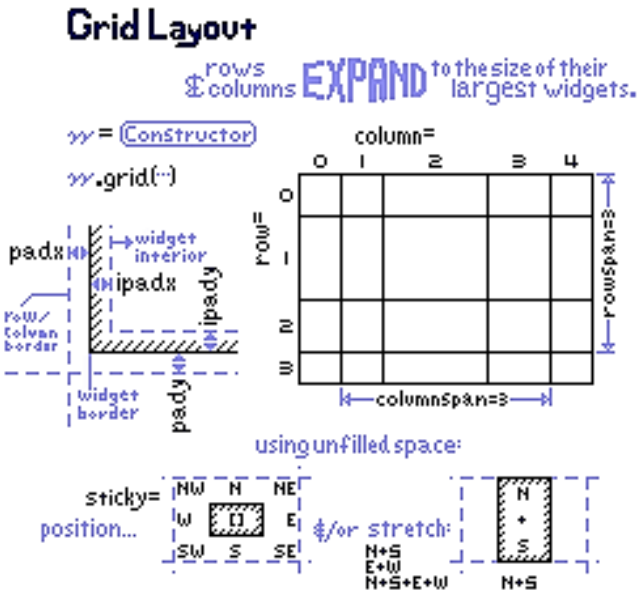
On va commencer par un traditionnel « hello world ! » pour s'assurer que tout fonctionne.

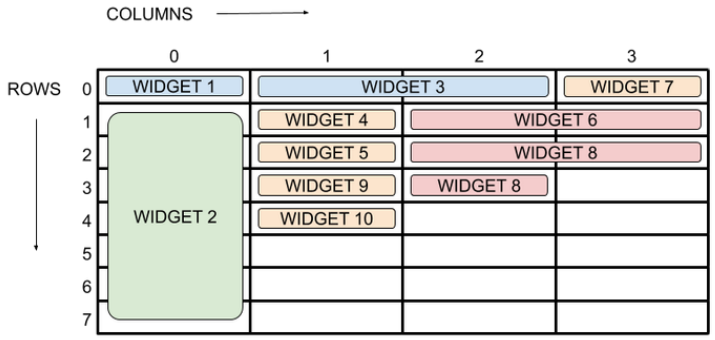
1) Ecrire un programme permettant de créer la fenêtre représentée ci-contre :
Essayez de redimmensionner la fenetre. Que remarquez-vous ?



2) Améliorer la fenêtre en fixant son titre à « message », sa taille initiale à 300x200, centrant le message et en l'affichant en police non proportionnelle (monospace), taille 20 gras et en rouge (voir deuxième fenêtre ci-contre).

Aide : Pour la question 2, vous pouvez vous aider de http://www.jchr.be/python/tkinter.htm#ouvrir pour modifier les caractéristiques de la fenêtre et http://www.jchr.be/python/tkinter.htm#fontes pour l'écriture. La couleur peut être réglée grâce à l'attribut `fg` (**f**ore**g**round color) à la création du widget. Pour le positionnement, le plus simple est d'utiliser le gestionnaire `pack()` (voir http://www.jchr.be/python/tkinter.htm#pack-grid) avec l'option permettant d'occuper tout l'espace.

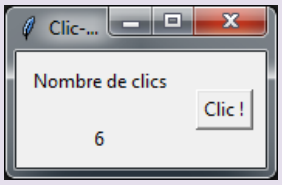




III - Entraînement

Question 2 :

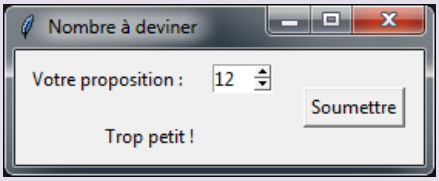
Créer une fenêtre avec un texte indiquant le nombre de clic, sa valeur qui vaut zéro au départ et un bouton intitulé « Clic ! ».
Rajouter le code permettant d'actualiser la valeur du compteur en l'incrémentant de 1 à chaque clic sur le bouton.



Aide : On peut utiliser une variable de contrôle de type `IntVar` pour stocker la valeur du compteur.
Pour lier l'action du clic à la fonction permettant d'actualiser le compteur, on peut soit utiliser l'argument `command=` à la création du bouton, soit utiliser la méthode `bind` avec l'argument `"<Button>"`. Cette dernière méthode permet de capturer les clics de tous les boutons (gauche, milieu et droit) et pas seulement le clic gauche.

Question 3 :

On souhaite faire une interface graphique pour le classique de programmation : le jeu consistant à tirer un nombre au hasard entre 0 et 20 et l'utilisateur doit deviner le nombre en un minimum d'essais, le programme indiquant à chaque fois si le nombre à deviner est plus grand ou plus petit que la proposition.
Ecrire le programme complet (interface graphique + programme de jeu) permettant de jouer au jeu avec une interface du type de celle représentée ci-contre. On pourra éventuellement rajouter un bouton pour relancer une nouvelle partie.



Aide : On peut utiliser une `Spinbox` pour que l'utilisateur donne sa proposition et afficher le message (trop grand, trop petit ou gagné) dans un `Label`.
Une fois le nombre secret tiré au hasard, le programme se met en attente dans la `mainloop()` et c'est la fonction liée au bouton « Soumettre » qui comparera la proposition avec le nombre secret et modifiera le message en conséquence.

Pour la prochaine question, nous allons utiliser un **canevas** (`Canvas`) qui est un élément graphique où on peut facilement tracer des formes.
Une fois le canevas mis en place, on peut tracer les différentes formes avec des méthodes du type `create_xyz` où `xyz` est le type de forme (`arc`, `oval`, `line`, `rectangle`, …). Les arguments de la méthode dépendent du type de forme (voir http://tkinter.fdex.eu/doc/caw.html ou sa copie), mais toutes ces méthodes renvoient un **identifiant de forme** (codé sous la forme d'un entier) qui permet par la suite de manipuler cette forme pour par exemple la mettre sur le devant, la déplacer ou l'effacer.

Question 4 :

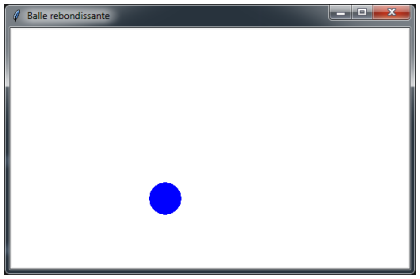
Dans ce programme, nous allons simplement animer une balle qui rebondi dans une boîte.
La balle sera tracée dans un objet `Canvas`.

1) Créer une fenêtre possédant un seul élément graphique de type `Canvas` de 500 x 300 pixels rempli en blanc (`bg = "white"`).
2) Ecrire une fonction `traceBalle(x, y, coul)` qui trace un disque de couleur `coul` de rayon 20 pixels aux coordonnées `x`, `y` et renvoie l'identifiant de la forme tracée. Appeler une fois la fonction avec des arguments quelconque avant la `mainloop()` pour vérifier son bon fonctionnement.

On va maintenant essayer d'animer la balle. Pour cela on utilisera 4 variables : `x`, `y`, `vx` et `vy`. `x` et `y` désignent la position actuelle de la balle et `vx` et `vy` respectivement sa vitesse horizontale et verticale.
Le principe de l'animation est le suivant :

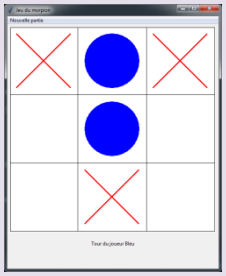
- La fonction `anime` est appelée toutes les 50 ms grâce à la méthode `after()` de la fenêtre principale
- La fonction `anime` commence par calculer la nouvelle position de la balle en ajoutant `vx` à `x` et `vy` à `y`.
- On déplace la forme de la balle dans le canevas d'un déplacement `vx` et `vy` en se servant de l'identifiant de la forme renvoyé au moment de sa création.
- On contrôle que `x` et `y` restent dans le cadre. Si ce n'est pas le cas, on inverse `vx`[3] (si c'est `x` qui n'est pas correct) et/ou `vy` (si `y` n'est pas correct).
- Enfin on programme la prochaine exécution de l'animation dans 50 ms en appelant la méthode `after()` juste avant la fin de notre fonction.

3) Ecrire la fonction anime ainsi que les quelques lignes de programme principal permettant de fixer la position et la vitesse initiales de la balle et de la tracer, puis lancer l'animation en utilisant la méthode `after()` juste avant de lancer la `mainloop()`.



[3] Ou plutôt on prend son opposé : `vx = - vx`.

Question 5 :



On va maintenant reprendre le jeu de morpion vu en début d'année, mais cette fois avec une interface graphique. Reprenez les trois fichiers de la correction ou vos trois fichiers à vous et modifiez-les pour obtenir un jeu avec interface graphique.

On pourra par exemple utiliser un canevas pour la grille de jeu et deux labels en-dessous pour donner des informations au joueur (tour du joueur, partie gagnée,…)

Il faudra reprendre fortement la partie de l'élève 1 (partie graphique) et également celle de l'élève 3 (moteur du jeu) pour l'adapter à une programmation évènementielle. La partie de l'élève 2 (vérification si on gagne) en revanche ne nécessite aucune modification.

Aides :

- On peut attacher une fonction `clic(evenement)` à l'action `"<Button-1>"` dans le canevas. C'est cette fonction qui gérera ce que gérait la boucle principale dans la version classique du morpion.
- On peut facilement récupérer les coordonnées du clic de la souris dans le canevas car elles correspondent aux propriétés `x` et `y` de l'objet evenement fourni à la fonction clic. Ainsi `evenement.x` renvoi l'abscisse du point cliqué.
- En divisant par *taille d'une case* (division entière avec `//`) la coordonnée x on peut en déduire la colonne cliquée (entre 0 et 2).
  Par exemple si la grille fait 600 pixels de large alors la taille d'une case est de 200 pixels. Si on clique en x=50, le calcul 50//200 donne 0 (colonne 0). Si on clique en x=354, le calcul 354//200 donne 1, donc colonne 1.
- Pour gérer le passage d'un joueur à un autre, on peut utiliser une variable globale `joueur` qui vaudra `0` ou `1`. On change alors de joueur simplement en faisant `joueur = 1 - joueur`.

Références :

http://tkinter.fdex.eu/ semble inaccessible.
Une version simplifiée sur https://profjahier.github.io/html/NSI/tkinter/doc_EFlorent_allegee/index-2.html
http://www.jchr.be/python/tkinter.htm
http://fsincere.free.fr/isn/python/cours_python_tkinter.php
https://sebsauvage.net/python/gui/index_fr.html
https://python.doctor/page-tkinter-interface-graphique-python-tutoriel
FAQ tkinter : https://tarball69.developpez.com/tutoriels/python/isn-passe-ton-faq/#LIII-C-1

Nombreux liens et explications sur tkinter et plus largement python :
http://pascal.ortiz.free.fr/contents/tkinter/tkinter/informations_generales_et_transversales.html

Plus en rapport avec la programmation évènementielle :
https://zestedesavoir.com/tutoriels/1729/programmation-avec-tkinter/programmation-evenementielle-avec-tkinter/

Documentation à télécharger :
http://ftp-developpez.com/michel-aubry/temp/tutoriels/python/tkinter-8-4-reference-interface-utilisateur-graphique-gui-pour-python/tkinter.pdf
http://www.maths-info-lycee.fr/pdfs/nsi_11_interfaces_graphiques.pdf

En anglais :
https://www.tutorialspoint.com/python/python_gui_programming.htm
https://www.pythontutorial.net/tkinter/
https://www.tcl.tk/man/tcl8.6.13/TkCmd/contents.html (référence très complète mais sans explications)