Interfaces graphiques

Les interfaces graphiques, appelées GUI (graphical user interface), utilisent une bibliothèque appelée tkinter. Il existe d'autres types d'interfaces graphiques, qui sont en général plus compliquées.
Citons tout de même *pygame*, qui, comme son nom l'indique, est particulièrement adaptée pour créer des jeux, avec interfaces graphiques, gestion du temps, gestion de la musique. Il est déconseillé d'utiliser pygame tant que vous n'avez pas des notions de base sur les interfaces graphiques. Par ailleurs, l'installation pour usage avec la dernière version de python n'est pas forcément simple, la dernière version de python supportée par pygame étant en général plus ancienne **Il est impératif de ne pas mélanger les bibliothèque GUI** : pas de tkinter et pygame dans le même programme. A priori, tkinter fait tout ce que fait pygame. Certains conseillent de tout faire avec tkinter, et de n'utiliser aucune autre bibliothèque GUI, quel que soit le programme que l'on souhaite faire.
Les GUI plantent assez facilement, ne vous inquiétez pas si vous voyez apparaitre un message dont la teneur est "dead kernel, will restart automatically". Après quelques instants, ou bien après une relance du programme, une bulle "kernel restart" bleue devrait apparaitre en haut à droite (à côté de python 3). Si ce n'est pas le cas, commencez par sauver votre travail. Puis allez dans l'onglet "Home", et fermez le TD : cocher la case correspondante, "shutdown" en haut de la page. Fermez ensuite l'onglet du TD, et relancez-le à partir de "Home".
Les éléments d'une GUI sont appelés widgets (windows gadgets), que ce soit un champ de texte, de saisie, un bouton, ou même un clic de souris..

Afficher une fenêtre

L'exemple suivant crée une fenêtre avec saisie, affichage et modification de texte. La fenêtre est crée en dehors du navigateur utilisé pour ce cours. Elle figure sous le nom tkinter ou tk (dans la barre des tâches sous Windows, idem pour les autres systèmes).

[ ]
```
from tkinter import *      # import de la bibliothèque graphique

#########  FONCTIONS
def afficher(event):
    """
    Fonction qui transforme le texte dans champ_label2
    @faux parametre : "event" n'est pas un parametre passe en
argument,
       cela désigne juste l'evenement qui se produit
       lorsque l'on appuie sur la touche entree
    @faux return : la chaine de caracteres stockee dans
champ_label2 est modifiee avec
       le texte recupere par la methode "texte.get()"
    => On ne specifie pas les fonctions GUI comme on le fait pour
une fonction normale
```

```
        - On donne son effet
        - S'il y a des "vrais" paramètres et/ou un vrai "retour",
on le(s) spécifie
        - Sinon on ne met pas de @param ni @return
    """
    champ_label2.configure(text="bonjour "+texte.get())
    return()

#########  PROGRAMME PRINCIPAL
# On crée une fenêtre, racine de notre interface
fenetre = Tk()

# On crée un label (ligne de texte) de titre
# Note : le premier paramètre passé a Label est notre
#        interface racine, la fenetre creee initialement
champ_label = Label(fenetre, text="C tro bo lol !")

champ_label.pack()   # On affiche le label dans la fenêtre, au
milieu par defaut

# On affiche un label, puis un champ de saisie
champ_label2 = Label(fenetre, text="Rentrez votre nom")
champ_label2.pack(side = LEFT)  # On affiche le label dans la
fenêtre, a gauche

texte = Entry(fenetre, width=30)    # le champ de saisie donne
une variable nommee texte
texte.bind("<Return>",afficher)     # quand on appuie sur entree,
on apelle fonction "afficher"
texte.pack()                        # On affiche le texte dans la
fenêtre

# on rajoute un bouton "quitter", avec une commande
fenetre.destroy au nom assez explicite !
bouton_quitter = Button(fenetre, text="Quitter",
command=fenetre.destroy)
bouton_quitter.pack()     # On affiche le bouton dans la fenêtre

# On démarre la boucle Tkinter qui s'interrompt quand on ferme la
fenêtre
fenetre.mainloop()
```

Vous remarquez que les widgets (éléments de la GUI) sont "packés", c'est-à-dire positionnés, les uns au dessus des autres, dans l'ordre de leur apparition dans le programme. La méthode utilisée est donc `.pack()`. Diverses méthodes permettent de mieux préciser l'emplacement des différents composants de l'interface.
L'exemple suivant, basé sur la conversion farenheit/celsius, montre l'usage d'une grille avec la méthode `.grid()`.

```
[ ]  from tkinter import *

     ##### FONCTIONS

     def convertir(event):
         """
         convertit une temperature de fahrenheit en celsius
         """
         fahr=float(temperature.get())
         celsius = 0.55556*(fahr-32)
         champ_label3.configure(text="En degres Celsius cela donne
     "+str(celsius))
         return()

     ##### PROGRAMME PRINCIPAL
     fenetre = Tk()        # On crée une fenêtre, racine de notre
     interface

     # On crée tous les labels (les textes) ainsi que tous les
     elements de la fenetre
     champ_label = Label(fenetre, text="Utilitaire de conversion")
     champ_label2 = Label(fenetre, text="Rentrez une temperature en
     degres farenheit")
     champ_label3 = Label(fenetre, text="")
     temperature = Entry(fenetre)
     temperature.bind("<Return>",convertir)
     bouton_quitter = Button(fenetre, text="Quitter",
     command=fenetre.destroy)
         # la commande fenetre.quit devrait fonctionner mais non
     # un bouton qui devrait fonctionner ci-dessous mais non, probleme
     de typage
     #bouton_convertir = Button(fenetre, text="Convertir en degres
     Celsius", command=convertir(temperature))



     # On affiche tout, avec la methode grid qui range les elements
     dans des lignes et colonnes
     champ_label.grid(row=0)
     #bouton_convertir.grid(row=0, column = 1)    # le bouton qui
     devrait fonctionner...
     champ_label2.grid(row=1, sticky = W)
     champ_label3.grid(row = 2,sticky = W)
     temperature.grid(row=1, column=1)
     bouton_quitter.grid(row=2, column = 1)


     # On démarre la boucle Tkinter qui s'interrompt quand on ferme la
     fenêtre
     fenetre.mainloop()
```

Dessiner dans une fenêtre

L'exemple suivant trace des formes aléatoires simples (segments, ovales, rectangles), dans une fenêtre.

[ ]

```
#### BIBLIOTHEQUES
from tkinter import *
from random import *

# FONCTIONS
def change_couleur():
    """
    changement aleatoire de la couleur de trace
    """
    pal=
['purple','cyan','maroon','green','red','blue','orange','yellow']
    c = randrange(8)
    coul = pal[c]
    return(coul)

def choix_alea():
    """
    choix aléatoire de la forme, de sa couleur, de sa position et
de ses dimensions
    """
    couleur = change_couleur()
    alea = randint(1,3)
    if alea == 1:
        x1 = randint(10,390)
        y1 = randint(10,390)
        x2 = randint(10,390)
        y2 = randint(10,390)
        dessiner_ligne(x1,y1,x2,y2,couleur)
    elif alea == 2:
        x1 = randint(10,320)
        y1 = randint(10,320)
        x2 = x1 + randint(5,70)
        y2 = y1 + randint(5,70)
        dessiner_ovale(x1,y1,x2,y2,couleur)
    else:
        x1 = randint(10,320)
        y1 = randint(10,320)
        x2 = x1 + randint(5,70)
        y2 = y1 + randint(5,70)
        dessiner_rectangle(x1,y1,x2,y2,couleur)
    return()

def dessiner_ligne(x1,y1,x2,y2,couleur):
    """
    trace d'une ligne dans un canevas
```

```
        """
        largeur = randint(1,4)
        can1.create_line(x1,y1,x2,y2,width = largeur,fill=couleur)
        return()

    def dessiner_ovale(x1,y1,x2,y2,couleur):
        """
        trace d'un ovale dans un canevas
        """
        can1.create_oval(x1,y1,x2,y2,fill=couleur)
        return()

    def dessiner_rectangle(x1,y1,x2,y2,couleur):
        """
        trace d'un rectangle dans un canevas
        """
        can1.create_rectangle(x1,y1,x2,y2,fill=couleur)
        return()

    ######
    #
    # PROGRAMME PRINCIPAL
    #
    ########

    fenetre = Tk()   # On crée une fenêtre, racine de notre interface

    # creation des widgets "esclaves" de la fenetre principale
    can1 = Canvas(fenetre, bg='dark grey', height = 400, width = 400)
    can1.pack(side=LEFT)

    # on rajoute un bouton "quitter".
    bouton_quitter = Button(fenetre, text="Quitter",
    command=fenetre.destroy)
    bouton_quitter.pack(side=BOTTOM)

    # bouton de trace de ligne
    bouton_dessiner = Button(fenetre, text="Tracer une forme",
    command=choix_alea)
    bouton_dessiner.pack()

    # On crée et affichage du titre
    champ_label = Label(fenetre, text="Mon dessin il est tro bo !")
    champ_label.pack(side = TOP)

    # On démarre la boucle Tkinter qui s'interrompt quand on ferme la
    fenêtre
    fenetre.mainloop()
```

Récupérer les coordonnées d'un clic de souris

L'exemple suivant trace une grille dans une fenêtre, puis trace un cercle centré dans la grille au clic de souris (utile pour les plateaux de jeux !).

[ ]

```
# BIBLIOTHEQUE
from tkinter import *
from random import *

# FONCTIONS

def dessiner_ligne(xA,yA,xB,yB,couleur):
    """
    trace d'une ligne dans un canevas
    entre les points de coordonnees (xA,yA) et (xB,yB), de
couleur "couleur"
    """
    can1.create_line(xA,yA,xB,yB,width=2,fill=couleur)
    return

def change_couleur():
    """
    changement aleatoire de la couleur de trace
    """
    pal=
['purple','cyan','maroon','green','red','blue','orange','yellow']
    c = randrange(8)
    coul = pal[c]
    return coul

def trace_grille():
    """
    trace d'une grille vierge dans le canevas
    """
    global can1
    can1.delete(ALL)
    x1 = 10
    y1 = 370
    x2 = 10
    y2 = 10
    coul = change_couleur()
    for i in range(10):
        dessiner_ligne(x1,y1,x2,y2,coul)
        x1 = x1 + 40
        x2 = x2 + 40

    x1 = 10
    y1 = 10
```

```
        x2 = 370
        y2 = 10
        coul = change_couleur()
        for i in range(10):
            dessiner_ligne(x1,y1,x2,y2,coul)
            y1 = y1 + 40
            y2 = y2 + 40
        return()


    def pointeur(event):
        """
        dessine un rond dans la grille ou un carre
        """
        global can1
        chaine.configure(text="clic detecte en X = "+
    str(event.x)+",et Y = "+str(event.y))
        # dessiner un bateau 1 case eventuellement.
        if event.x > 10 and event.x < 370 and event.y > 10 and
    event.y < 370:
            cx = (event.x-10)//40*40 + 30     # coordonnees du centre
    du cercle
            cy = (event.y-10)//40*40 + 30
            coul = change_couleur()
            can1.create_oval(cx-15,cy-15,cx+15,cy+15,fill=coul)
        return

    ######
    #
    # programme principal
    #
    ########


    fenetre = Tk()

    # creation des widgets "esclaves" de la fenetre principale
    can1 = Canvas(fenetre, bg='dark grey', height = 420, width = 420)
    can1.bind("<Button-1>",pointeur)
    can1.pack(side=LEFT)

    chaine = Label(fenetre)
    chaine.pack(side = BOTTOM)

    # on rajoute un bouton "quitter".
    bouton_quitter = Button(fenetre, text="Quitter",
    command=fenetre.destroy)  #fenetre.quit
    bouton_quitter.pack(side=BOTTOM)

    # bouton "dessiner la grille vierge"
    bouton_grille = Button(fenetre, text="Tracer une grille",
    command=trace_grille)
```

```
bouton_grille.pack(side=TOP)

# mettre une image idiote en fond : le chemin de l'image sur le
disque dur doit être precise
#photo = PhotoImage(file="martine-vista-geek.jpg")         # seul
le format gif est bien supporte
#item = can1.create_image(200,200, image=photo)

champ_label = Label(fenetre, text="Grilles")

# On affiche le label dans la fenêtre
champ_label.pack(side = TOP)


# On démarre la boucle Tkinter qui s'interrompt quand on ferme la
fenêtre
fenetre.mainloop()
```

Menus

L'exemple suivant montre la construction des menus.

[ ]
```
from tkinter import *
def donothing():
   filewin = Toplevel(root)
   button = Button(filewin, text="Bouton qui ne fait rien")
   button.pack()
   return()

root = Tk()     # la fenetre s'appelle ici root (racine), le nom
n'a pas d'importance
menubar = Menu(root)    # on construit une barre
filemenu = Menu(menubar, tearoff=0) # le menu fichier n'est pas
detachable
filemenu.add_command(label="New", command=donothing)
filemenu.add_command(label="Open", command=donothing)
filemenu.add_command(label="Save", command=donothing)
filemenu.add_command(label="Save as...", command=donothing)
filemenu.add_command(label="Close", command=donothing)

filemenu.add_separator()

filemenu.add_command(label="Quit", command=root.destroy)
menubar.add_cascade(label="File", menu=filemenu)

editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Undo", command=donothing)
```

```
editmenu.add_separator()

editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
editmenu.add_command(label="Paste", command=donothing)
editmenu.add_command(label="Delete", command=donothing)
editmenu.add_command(label="Select All", command=donothing)

menubar.add_cascade(label="Edit", menu=editmenu)
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)
menubar.add_cascade(label="Help", menu=helpmenu)

root.config(menu=menubar)
root.mainloop()
```

Exercice

Reprendre un programme simple (le jeu "deviner un nombre" est parfait), et le faire avec une interface graphique, qui comprendra aussi bien un menu qu'un bouton pour lancer la partie.

Compléments : gestion du temps et des fenêtres secondaires

L'exemple suivant montre comment créer une fenêtre secondaire, et comment gérer une boucle temporelle. Cet exemple est complexe : vous pouvez le faire tourner immédiatement pour en voir l'effet. Néanmoins, passez à la suite (lecture et écriture dans un fichier) avant de le comprendre et/ou de le modifier, qui présente des connaissances plus importantes à maîtriser. En effet, comme le titre l'indique, ce paragraphe-ci est un complément.

[ ]
```
from tkinter import *
from time import sleep      # cette fonction permet d'arreter
momentanement un programme

def clavier(event):
    """
    Fonction qui gere l'evenement (event) "frappe sur une touche"
    Cree une fenetre fille, affiche un message dedans
    et lance une boucle d'animation
    """
```

```
    global fen_fille    # la fenetre fille est une variable
globale
                        # pourquoi ? je ne sais pas
    global texte        # le texte est une variable globale, pour
plus
                        # de simplicite
    touche = event.keysym       # recupere la touche
    canevas.unbind("<Key>")      # empeche l'event clavier de se
produire
                                # pendant l'execution de cette
fonction
    fen_fille = Toplevel()      # creation de la fenetre fille
    # ci-dessous affichage d'un label dans la fenetre fille
    taille =12
    texte = "AAAHHH appui interdit sur la touche "+touche
    champ_label = Label(fen_fille, text=texte,
                        font=("Helvetica", taille),fg = "red",
                        bg = "yellow")
    champ_label.pack()
    # appel de la boucle pour animation
    modif_fen(champ_label,taille)
    return()

def modif_fen(champ_label,taille):
    global texte
    # modification du texte
    champ_label.configure(text=texte,font=("Helvetica", taille),
                          fg = "red",bg = "yellow")
    champ_label.pack()
    if taille < 50:     # boucle d'animation
       taille = taille + 1
       # ci-dessous modification de la fenetre apres 20
millisecondes
       # observez le passage des parametres de cette methode:
       # d'abord la duree d'attente, puis on reappelle la
fonction
       # modif_fen (recursivite), et enfin les parametres de
modif_fen,
       # qui ne sont pas passes comme d'habitude
       fen_fille.after(20, modif_fen,champ_label,taille)
    else:           # fin de boucle d'animation
       sleep(1.5)   # le programme se met en veille 1,5 seconde
       canevas.bind("<Key>",clavier) # on autorise a nouveau le
clavier
       fen_fille.destroy()
    return()

root = Tk()

canevas = Canvas(root, width=500, height=500)
canevas.focus_set()      # precise que c'est le canvas qui
recuperera les evenements
```

```
                        # cela sert surtout lorsqu'il y a
plusieurs elements de GUI
                        # (canevas, boutons, etc...)
canevas.bind("<Key>", clavier)
canevas.pack()

root.mainloop()
```

Un site qui présente les bases de tkinter, simple et bien fait à la fois: http://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel
On y trouve d'autres widgets notamment.

Une présentation plus complexe avec la méthode .grid()

Dans l'exemple suivant, on utilise à nouveau la méthode `.grid()`. L'aspect recherché est le suivant (*éventuellement, remarquez dans le code que le texte prend 3 colonnes de large... sinon c'est moche ; d'où la nécessité de tester malgré tout*).

	Colonne 0	**Colonne 1**	**Colonne 2**	**Colonne 3**	**Colonne 4**	**Colonne 5**
Ligne 0	Etiquette 1	Saisie 1				
Ligne 1	Etiquette 2	Saisie 2	Graphique			
Ligne 2	Etiquette 3	Saisie 3				
Ligne 3	Affichage des résultats en texte					
Ligne 4	Bouton					

Cet exemple montre aussi l'utilisation de la bibliothèque matplotlib, pour les courbes et graphiques scientifiques. Accpetez le code correspondant sans essayer d'en saisir le sens. L'idée n'est pas de comprendre en détail le fonctionnement de ce programme, mais de pouvoir utiliser ce programme comme modèle si nécessaire.
Dans ce programme, la séparation des parties "métier" et "graphique" est respectée : aucun calcul n'est fait dans la partie graphique, et aucun affichage/saisie n'est effectué dans la partie calcul/métier.
Remarque : le bouton "quitter" peut être capricieux, il peut être ncécessaire de redémarrer le noyau Python.

```
[ ]   #!/usr/bin/env python3
      # -*- coding: utf-8 -*-
      """
      Created on Tue Mar 31 18:05:09 2020

      @author: Frédéric Mandon
      """

      from tkinter import *      # import de la bibliothèque graphique
      from random import randint

      import numpy as np               # bibliothèque (de calcul
      scientifique) utilisée pour le graphique
      import matplotlib.pyplot as plt  # bibliothèque de tracé de
      courbes et graphiques scientifiques

      #########  FONCTIONS PARTIE METIER
      def calcul(n_faces,n_des,n_jets):
          """
          Simule des jets de dés. Partie "métier" du programme
          @param n_faces : entier > 0 nombre de faces des dés
          @param n_des : entier > 0 nombre de dés
          @param n_jets : entier > 0 nombre de jets
          @return resultats : liste d'entiers entre 0 et n_jets,
      résultats possibles des jets de dés
                      (avec 4 dés à 8 faces, les entiers seront compris
      entre 4 et 32)
          @return effectifs : liste des effectifs, entiers entre 0 et
      nb_jets, correspondants à chaque résultat de la liste
                             "resultat"
          @returnfrequences : liste des frequences, flottants entre 0
      et 1, correspondants à chaque résultat de la liste
                           "resultat"
          @return somme_eff : entier égal à nb_jets, somme des entiers
      du tableau "effectifs".
                      Présent uniquement pour la vérification, et à
      visée pédagogique.
          @return somme_freq : floattant théoriquement égal à 1, somme
      des flottants du tableau "frequences".
                      Avec les arrondis peut en différer légérement.
                      Présent uniquement pour la vérification, et à
      visée pédagogique.
          """
          resultats =[i for i in range(n_des,n_faces*n_des + 1)]
          effectifs =[0]*len(resultats)

          for i in range(n_jets):
              lance = 0
              for j in range(n_des):
                  lance = lance + randint(1,n_faces)
```

```
            effectifs[lance - n_des] = effectifs[lance - n_des] + 1

        frequences = effectifs[:]
        somme_eff = 0   #verification des effectifs et des fréquences
        somme_freq = 0
        for i in range(len(frequences)):
            frequences[i] = frequences[i]/n_jets
            somme_eff = somme_eff + effectifs[i]
            somme_freq = somme_freq + frequences[i]
        return(resultats,effectifs,frequences,somme_eff,somme_freq)

    ######### FONCTIONS PARTIE GRAPHIQUE

    def ausuivant1(event):
        """
        Met le focus (le curseur) sur le champ de saisie du nombre de
    dés
        """
        texte2.focus_set()
        return()

    def ausuivant2(event):
        """
        Met le focus (le curseur) sur le champ de saisie du nombre de
    jets
        """
        texte3.focus_set()
        return()

    def afficher(event):
        """
        Affichage des résultats. Partie "graphique" du programme
        """
        # on efface la partie texte avec la liste des résultats
        texte.delete(1.0, END)
        texte.tag_add("tout", 1.0, END)
        texte.tag_config("tout", background="white")
        # on pourrait aussi effacer la mise en forme avec les tags
    crées précédemments, lorsque l'on fait plusieurs simulations.
        # Ca demande de créer une liste des tags

        # on récupère les valeurs dans les champs de saisie. Ce sont
    des chaînes, on les convertit en entiers.
        nb_faces = int(texte1.get())
        nb_des = int(texte2.get())
        nb_jets = int(texte3.get())

        # appel de la partie "métier" qui réalise les calculs
        (result,effectif,freq,s_eff,s_freq) =
    calcul(nb_faces,nb_des,nb_jets)
```

```
    # mise en forme de la zone de texte où figure la liste des
résultats, effectifs et fréquences
    chaine = str('{:10}|{:10}|
{:10}\n'.format('Résultat','Effectif','Fréquence')) # colonnes de
largeur 10 séparées par une barre verticale |

                # soit 32 caractères, se finit par un saut de
ligne \n
    texte.insert(INSERT, chaine) # affichage du titre
    texte.tag_add("titre", "1.0", "1.32") # tag pour la mise en
forme du titre ligne 1 sur 32 caractères (de 1.0 à 1.32)
    texte.tag_config("titre", background="light grey", underline
= 1) # mise en forme du titre sur le tag titre défini à la ligne
précédente


    # affichage dans la zone de texte des résultats ligne par
ligne
    for i in range(len(result)):
        str_freq = str(round(freq[i],5))
        chaine = str('{:-10}|{:-10}|
{:>10}\n'.format(result[i],effectif[i],str_freq))
        texte.insert(INSERT, chaine)

    # mise en forme en gris clair une ligne sur deux (à faire
après l'affichage des lignes)
    for i in range(len(result)):
        if (i + 1)%2 == 1:
            tag_name = "ligne"+str(i + 3)
            tag_debut = str(i + 3) + ".0"
            tag_fin = str(i + 3)+ ".32"
            texte.tag_add(tag_name, tag_debut, tag_fin)
            texte.tag_config(tag_name, background="light grey")

    # Mise en forme (soulignement) de la dernière ligne des
résultats
    tag_name = "ligne"+str(len(result) + 1)
    tag_debut = str(len(result) + 1) + ".0"
    tag_fin = str(len(result) + 1)+ ".32"
    texte.tag_add(tag_name, tag_debut, tag_fin)
    texte.tag_config(tag_name, underline = 1)

    # affichage de la ligne de vérification des sommes
effectifs/fréquences
    str_s_freq = str(round(s_freq,5))
    chaine = str('{:10}|{:-10}|
{:>10}\n'.format('Sommes',s_eff,str_s_freq))
    texte.insert(INSERT, chaine)
    # puis mise en forme éventuelle
    if result[-1]%2 == 1:
        tag_name = "ligne"+str(len(result)+2)
        tag_debut = str(len(result)+2) + ".0"
```

```
            tag_fin = str(len(result)+2) + ".32"
            texte.tag_add(tag_name, tag_debut, tag_fin)
            texte.tag_config(tag_name, background="light grey")

    # construction du graphique avec les bibliothèques numpy et
matplotlib/pyplot
    y_pos = np.arange(len(result))
    axe_x = result[:]

    fig = plt.figure(figsize=(cote_fig,cote_fig))
    plt.bar(y_pos, freq, align='center', alpha=0.5)
    plt.xticks(y_pos, axe_x)
    plt.ylabel('Fréquence')
    titre = str(nb_jets) + " simulations de " + str(nb_des) + "
dés à " + str(nb_faces) + " faces"
    plt.title(titre)

    # enregistrement du graphique dans un fichier externe (du
type 1000_jets_3_dés_6_faces.png) avec la bibliothèque pyplot
    nom_fichier = str(nb_jets) + "_jets_" + str(nb_des) + "_dés_"
+ str(nb_faces) + "_faces.png"
    plt.savefig(nom_fichier, edgecolor='k')

    #nettoyage de l'objet graphique (au cas où on fait plusieurs
simulation, sinon les graphiques se superposent) avec la
bibliothèque pyplot
    plt.clf()

    # affichage du graphique dans le canevas : on ouvre le
fichier externe et on l'affichage dans le canevas
    canevas.delete(ALL)     # d'abord on efface ce qu'il y a déjà
    graphe = PhotoImage(file = nom_fichier) # puis on l'ouvre
    item =
canevas.create_image(dpi*cote_fig/2,dpi*cote_fig/2,image =
graphe) # le centre de l'image est au centre du canevas
    canevas.image = graphe # et on l'affiche

    return()

#########  PROGRAMME PRINCIPAL (partie graphique)

print("en cas de plantage sous EduPython notamment, \
si >>> n'apparait pas après avoir quitté, \
clic droit ici > moteur Python > réinitialiser le moteur")
print("Sous Jupyter : aller dans la page Home > Running, fermer
le notebook, puis le redémarrer.")
dpi = 100       # points par pouce
cote_fig = 5    # coté de l'image (carrée) en pouces

fenetre = Tk()
fenetre.title("Simulation de jets de dés")
fenetre.geometry("900x550") #800x550 sous OsX
```

```
champ_label = Label(fenetre, text="Simulation de jets de dés")


champ_label1 = Label(fenetre, text="Nombre de
faces",justify=LEFT,padx = 5)
champ_label1.grid(row = 0,column = 0, sticky = W)
texte1 = Entry(fenetre, width=10)
texte1.bind("<Return>",ausuivant1)  # taper sur la touche entrée
déplace le curseur dans le champ de saisie suivant
texte1.grid(row = 0,column = 1, sticky = E)
texte1.delete(0, END)
texte1.insert(0, 6)
texte1.focus_set()

champ_label2 = Label(fenetre, text="Nombre de
dés",justify=LEFT,padx = 5)
champ_label2.grid(row = 1,column = 0, sticky = W)
texte2 = Entry(fenetre, width=10)
texte2.bind("<Return>",ausuivant2)  # taper sur la touche entrée
déplace le curseur dans le champ de saisie suivant
texte2.grid(row = 1,column = 1, sticky = E)
texte2.delete(0, END)
texte2.insert(0, 3) # a enlever si pas de valeur par défaut

champ_label3 = Label(fenetre, text="Nombre de
jets",justify=LEFT,padx = 5)
champ_label3.grid(row = 2,column = 0, sticky = W)
texte3 = Entry(fenetre, width=10)
texte3.bind("<Return>",afficher)    # taper sur la touche entrée
lance le calcul et l'affichage des résultats
texte3.grid(row = 2,column = 1, sticky = E)
texte3.delete(0, END)
texte3.insert(0, 1000) # a enlever si pas de valeur par défaut

texte = Text(fenetre,padx = 5)

# mise en forme de la zone de texte où figure la liste des
résultats, effectifs et fréquences
chaine = str('{:10}|{:10}|
{:10}\n'.format('Résultat','Effectif','Fréquence')) # colonnes de
largeur 10 séparées
                                                    # par une
barre verticale | soit 32 caractères au total
texte.insert(INSERT, chaine) # affichage du titre
texte.tag_add("titre", "1.0", "1.32") # tag pour la mise en forme
du titre ligne 1 sur 32 caractères (de 1.0 à 1.32)
texte.tag_config("titre", background="light grey", underline = 1)
# mise en forme du titre sur le tag titre défini à la ligne
précédente

texte.grid(row = 3,column = 0, columnspan = 3)
```

```
canevas = Canvas(fenetre, width = dpi*cote_fig+ 10, height =
dpi*cote_fig +10)
# DANS la ligne suivante, remplacer padx = 150 -sous windows- par
padx= 40 sous OsX
canevas.grid(row=1, column=2, columnspan=4, rowspan=4,
sticky=W+E+N+S, padx= 150, pady=5)
canevas.create_rectangle(10,10,dpi*cote_fig,dpi*cote_fig)

bouton_quitter = Button(fenetre, text="Quitter",
command=fenetre.destroy)
bouton_quitter.grid(row = 4,column = 0)


fenetre.mainloop()
fenetre.quit()
```

[**Frederic Mandon**](mailto:frederic.mandon@ac-montpellier.fr), Lycée Jean Jaurès - Saint Clément de Rivière - France (2015-2020)