Interfaces graphiques en Python 3 avec Tkinter

Ce TP se fait seul, sur la VM et il vous permettra d'avancer à votre rythme tout en découvrant quelques possibilités de Tkinter pour faire ses propres logiciels, voire ses propres jeux (Pygame est conseillé pour faire des jeux). Parlez moi du logiciel d'effaroucheur dans mon jardin, du logiciel d'apprentissage des tables de calcul (addition, soustraction, multiplication, division) et du logiciel de course à pieds par exemples

1. Introduction

Dans toute cette séance pratique de découverte de Tkinter, et sauf mention contraire, il vous sera demandé de travailler dans une console interactive (Idle). En mode interactif, la construction de l'interface graphique (souvent abrégée en GUI, pour Graphical User Interface en anglais) est automatique.
Chaque fois que l'on voudra exploiter la bibliothèque Tkinter, il faudra commencer par l'importer, le plus souvent en totalité (attention, l'import indiqué ci-dessous ne fonctionne qu'en Python 3.x : en Python 2.x, la bonne syntaxe sera `from Tkinter import *`, avec un T majuscule) :

```
>>> from tkinter import *
# Création de la fenêtre de base de l'application
>>> f=Tk()
# On lui donne un nom
>>> f.title("Création de fenêtre pour l'ADESFA")
```

Remarque : la Programmation Orientée Objet (POO) est le style de programmation de prédilection pour les interfaces graphiques (cela permet notamment d'éviter de s'encombrer des variables globales, même si Tkinter offre - dans les cas simples - des moyens de contourner cette difficulté). Vous aurez donc très souvent à faire appel aux notations usuelles en POO : objet.méthode. Pour faire simple : un objet est une portion de code autonome. Il est défini via une classe, qui, chaque fois qu'elle est instanciée, crée un objet. Pour vous donner une image, une classe est un moule à gâteau, un objet est un gâteau, et on peut créer plusieurs gâteaux (éventuellement un peu différents) à partir du même moule. Une méthode est une fonction embarquée dans l'objet qui permet d'agir sur lui (en particulier de le modifier) « de l'intérieur ».

2. Spécificité d'un programme « graphique »

Dans un script, il faudra systématiquement démarrer la « boucle principale » à l'aide d'une instruction du type `f.mainloop()` (où f désigne la fenêtre principale du programme). La méthode mainloop provoque le démarrage du gestionnaire d'événements associé à la fenêtre : indispensable pour que votre application soit « à l'affût » des clics de souris, des pressions sur les touches du clavier, etc. C'est un peu cette instruction qui « met en marche » votre application graphique, en lui permettant d'interagir avec l'utilisateur.
On parle dans cette situation de « programmation événementielle », dans laquelle une boucle infinie attend sans arrêt qu'on la sollicite pour réagir, par opposition à la « programmation séquentielle » dans laquelle un programme a vocation à se terminer à un moment (qui, si l'on s'en donnait la peine, pourrait être) connu d'avance.

3. Un premier widget

Un widget est un WIndow gaDGET. Une GUI est composée de divers widgets, nous en verrons quelques uns (et vous en découvrirez d'autres par vous-mêmes).

```
# Création d'un label (une zone d'affichage), dépendant de la fenêtre f
>>> l=Label(f, text="Mon premier widget")
# Mais pourquoi donc rien ne s'affiche ?
```

Rien ne s'affiche parce que le widget n'est pas encore positionné dans la fenêtre. Il y a en effet plusieurs façons de placer les éléments constitutifs de la GUI au sein de la fenêtre principale. Tkinter connaît 3 gestionnaires de positionnement : pack, grid et place. On utilisera surtout pack dans les cas simples : c'est le plus basique et facile à utiliser (il « empile » les widget l'un sous l'autre en colonne, ou l'un à côté de l'autre en ligne). Pour des GUI plus complexes, on utilisera surtout grid.

```
# Insertion pour affichage du label dans la fenêtre f
>>> l.pack()
# Note : la géométrie de la fenêtre s'en trouve affectée !
```

On peut évidemment contrôler le « format » de la fenêtre : définir sa taille, décider si l'utilisateur est autorisé à la redimensionner (et si oui, dans quelle direction).

```
# Altérer la géométrie d'une fenêtre
>>> f.geometry('200x30')
# L'approche suivante est meilleure
>>> f.geometry('{}x{}'.format(300, 50))
# Interdire le redimensionnement d'une fenêtre
>>> f.resizable(0,0)
# Autoriser le redimensionnement d'une fenêtre en largeur
>>> f.resizable(1,0)
# Autoriser le redimensionnement d'une fenêtre en hauteur
>>> f.resizable(0,1)
# Autoriser le redimensionnement d'une fenêtre dans les 2 directions
>>> f.resizable(1,1)
```

Vous remarquerez que le label reste « statique » : il ne change pas de taille lorsque la fenêtre est redimensionnée.
On peut là encore contrôler comment le widget se comporte en fonction des modifications subies par la fenêtre.
Mais on va d'abord modifier sa couleur de fond afin de rendre ces changements plus visibles.

```
# Configuration a posteriori du label : définition d'un couleur de fond
>>> l.configure(background="light blue")
# On veut que l'espace vide autour du widget soit équitablement réparti
>>> l.pack(expand=1)
# On veut que notre label occupe l'espace jusqu'aux bords de la fenêtre
>>> l.pack(fill="x")
# Essayer de redimensionner la fenêtre horizontalement, puis verticalement
>>> l.pack(fill="y")
# Réessayer de redimensionner la fenêtre
>>> l.pack(fill="both")
# Et maintenant ?
```

4. Détruire ou « oublier » un widget ou une fenêtre

Oublier un widget est différent de le détruire : il cesse d'être affiché, mais peut être réaffiché à discrétion. On pourrait par exemple oublier le label l en faisant `l.forget()`.
Attention : ici, la méthode forget est un raccourci pour pack_forget ! Il faut utiliser la méthode grid_forget lorsqu'on a recourt au gestionnaire de placement grid. Une possibilité intéressante de ce dernier gestionnaire est l'oubli temporaire : avec la méthode grid_remove, le widget est retiré de l'affichage mais toutes ses options sont mémorisées (y compris sa position dans la grille). Il suffit d'un nouvel appel à sa méthode grid et il sera réaffiché exactement de la même manière qu'avant son oubli (pack ne possède pas d'équivalent).
Détruire un widget est définitif (il faut le recréer intégralement si l'on veut le retrouver). On utilise pour cela la méthode destroy, par exemple `l.destroy()`.
Pour détruire la fenêtre f (c'est l'objet que l'on veut détruire alors : `f.detroy()`)
Remarque : les méthodes pack_slaves et grid_slaves renvoient la liste des widget « enfants » d'un widget donné (afin de les supprimer ou de les oublier).

5. Dessiner !

Le widget canevas
On va donc créer et afficher un canevas : il s'agit d'une zone dans laquelle on peut « dessiner ».

```
>>> c=Canvas(f, background="yellow")
>>> c.pack()
# Réglages a posteriori (on peut aussi décider de faire ces réglages lors de la création du Canvas,
c'est une question de point de vue et de lisibilité)
>>> c.configure(background="light green")
# Il y a 2 approches possibles
>>> c['background']="light yellow"
>>> c.configure(height="800")
>>> c.configure(width="800")
# On peut évidemment régler plusieurs paramètres en une seule instruction
>>> c.configure(width=600, height=400)
```

6. Les formes de base

Le canevas de Tkinter permet d'utiliser différents objets graphiques : lignes, rectangles, polygones, ovales, arc, images, textes et sous-fenêtres. Nous allons en expérimenter quelques unes.

```
# Création d'un rectangle rempli avec la couleur "light blue" (bleu léger)
>>> rr=c.create_rectangle(10, 50, 100, 200, fill="light blue")
# L'affichage du rectangle est immédiat (l'objet « cannevas » est directement
# modifié « en place »)
# Noter que rr est un simple entier, qui repère l'objet du cannevas
# Grâce à ce numéro, on peut retrouver l'objet graphique et le modifier
>>> c.itemconfigure(rr, fill="red")
# (Re)Trouver le type d'un objet et la valeur d'un de ses paramètres
>>> c.type(rr)
>>> c.itemcget(rr,"fill")
# Création d'autres figures, par exemple un ovale (sert aussi pour les cercles)
>>> o=c.create_oval(50,100,250,150,fill="yellow")
>>> rb=c.create_rectangle(20,60,150,210,fill="blue")
>>> rv=c.create_rectangle(250,250,100,100,fill="green")
>>> rn=c.create_rectangle(0,500,100,100,fill="black")
```

7. Les déplacements

Une fois un objet créé, on peut le déplacer facilement de manière relative (par rapport à sa position de départ, donc : il s'agit d'une translation). Cette approche a l'avantage d'éviter tout risque de modifier par erreur les proportions de l'objet.

```
# Déplacement d'un objet par décalage à partie de sa position actuelle
>>> c.move(rr,200,0)
>>> c.move(rr,0,100)
```

Mais pourquoi l'ovale n'apparaît-il pas ? Parce que les formes se recouvrent. On peut donc gérer la « profondeur d'affichage » (passage au plan supérieur / inférieur d'un objet graphique).

Cela se fait à l'aide des méthodes tag_lower et tag_raise. Par exemple :

```
>>> c.tag_raise(o)
>>> c.tag_lower(rn)
```

Remarques
1. ces méthodes peuvent également prendre en argument un tag (une chaîne de caractère identifiant un ou plusieurs objets graphiques). On peut notamment placer un objet par rapport à un autre...
2. on peut aussi repositionner un objet graphique de façon absolue en modifiant ses coordonnées. Mais il faut alors faire bien attention, car en cas d'erreur on risque de modifier les proportions de l'objet !

8. Détecter des chevauchements

C’est utile, par exemple dans un jeu, pour détecter les collisions…

```
>>> c.find_overlapping(0,0,100,100)
# Si un seul élément est retourné dans le tuple, il n'y a pas de chevauchement
# Si le tuple compte 2 éléments ou plus, c'est le cas...
>>> c.find_overlapping(0,0,200,100)
>>> c.find_overlapping(200,200,250,250)
>>> c.find_overlapping(300,300,350,350)
```

9. Trouver et altérer les coordonnées d’un objet du canevas

Ces deux actions se réalisent à l’aide de la méthode coords.

```
# Trouver les coordonnées d'un objet d'un canevas
>>> c.coords(o)
# Modifier les coordonnées d'un objet d'un canevas (2 approches possibles).
# Notez que la géométrie de l'objet peut changer si l'on n'y prend pas garde !
>>> c.coords(o, 200, 300, 250, 350)
>>> c.coords(o, (200, 50, 350, 100))
```

10. Divers

```
# Liste de tous les objets graphiques du cannevas
>>> c.find_all()
# Détruire un objet graphique
>>> c.delete(3)
# Forcer la réactualisation du canevas. Peut être utile pour des questions de
# performances, et donc de fluidité d'affichage (vous pourrez lire à ce sujet
# http://effbot.org/tkinterbook/canvas.htm#performance-issues)
>>> c.update()
```

Bon à savoir : un système de « tag » permet d’étiqueter les objets graphiques du cannevas (un tag est une simple chaîne de caractères) : un objet peut avoir plusieurs tags différents, et un même tag peut être associé à plusieurs objets différents.
On peut alors sélectionner des objets d’après leur(s) tag(s) : pratique pour faire évoluer plusieurs éléments formant un tout logique. Typiquement, dans un jeu vidéo, cela pourrait s’avérer utile de pouvoir sélectionner les éléments graphiques composant un avatar (ou une portion d’avatar), ou des éléments du décor…

Un peu de pratique : faire rebondir un objet sur les bords du canevas

```
>>> from random import randint as r
>>> Dx, Dy = r(1,20), r(1,20)
>>> while True:
        c.move(o, Dx, Dy)
        if c.coords(o)[0] <= 0 or c.coords(o)[2] >= c.winfo_width():
                Dx *= -1
        if c.coords(o)[1] <= 0 or c.coords(o)[3] >= c.winfo_height():
                Dy *= -1
        # Nécessaire ici car la boucle principale de la GUI n'est pas lancée
        c.update()
```

Vous devriez pouvoir interrompre la boucle infinie en pressant simultanément les touches Ctrl + C.

11. Projet

Voici la vidéo d'un projet d'étudiant ici et le programme vous est offert ci-dessous :

```
from tkinter import *
import csv

top = Tk()
top.title("Logiciel de nom")
top.geometry("500x800")

listbox = Listbox(top)

filepath='SNPI3.csv'
File = open(filepath)
Reader = csv.reader(File)
Data = list(Reader)

maliste_nom=[]
for x in list (range(0,len(Data))):
  maliste_nom.append(Data[x][0])

maliste_prenom=[]
for x in list (range(0,len(Data))):
  maliste_prenom.append(Data[x][1])

def recherche():
  Listbox3.delete(0,END)
  for i in range(len(maliste_nom)):
    Listbox1.itemconfig(i,{'bg':'White'})
    Listbox2.itemconfig(i,{'bg':'White'})

  for i in range(len(maliste_nom)):
    if recherche_lettre.get() in maliste_nom[i]:
      Listbox3.insert(i,maliste_nom[i])
      Listbox1.itemconfig(i,{'bg':'OrangeRed3'})

  for i in range(len(maliste_prenom)):
    if recherche_lettre.get() in maliste_prenom[i]:
      Listbox3.insert(i,maliste_prenom[i])
      Listbox2.itemconfig(i,{'bg':'OrangeRed3'})

def ajout():

  maliste_nom.append(ajout_nom.get().upper())
  Listbox1.insert(len(maliste_nom)+1,ajout_nom.get().upper())
  maliste_prenom.append(ajout_prenom.get().capitalize())
  Listbox2.insert(len(maliste_prenom)+1,ajout_prenom.get().capitalize())
  with open('SNPI3.csv', mode='w') as employee_file:
    employee_writer = csv.writer(employee_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)
    for i in range (len(maliste_nom)):
      employee_writer.writerow([maliste_nom[i],maliste_prenom[i]])

bouton1 = Button(top, text ="Recherche", command=recherche)
bouton2 = Button(top, text ="Ajout", command=ajout)

Label(top, text="Nom:").grid(row=0, column=0)
Label(top, text="Prénom:").grid(row=0, column=1)
Label(top, text="Résultat Recherche:").grid(row=0, column=2)

Listbox1 = Listbox(top, height=len (maliste_nom)+5)

for i in range (len (maliste_nom)):
  Listbox1.insert(i,maliste_nom[i])

Listbox1.grid(row=1, column=0)

Listbox2 = Listbox(top, height=len (maliste_prenom)+5)
for i in range (len (maliste_prenom)):
  Listbox2.insert(i,maliste_prenom[i])

Listbox2.grid(row=1, column=1)
bouton1.grid (row=21, column=1)
Listbox3 = Listbox(top)
Listbox3.grid(row=1, column=2)
bouton2.grid (row=22, column=2)
```

```
recherche_lettre=Entry(top)
recherche_lettre.grid (row=21, column=0)

ajout_nom=Entry(top)
ajout_nom.grid (row=22, column=0)
ajout_prenom=Entry(top)
ajout_prenom.grid (row=22, column=1)

top.mainloop()
```

Le fichier snpi3.csv a ce format :
SNOECK,Olivier
PERE,Noel
PERE,Biloutte

Je vous propose d'utiliser le fichier Excel de notre formation ADESFA et de modifier le programme fourni afin de :

- permettre à l'utilisateur de sélectionner un pays (au moyen d'une case à cocher) et d'afficher les noms, prénoms des collègues de ce pays
- permettre à l'utilisateur de sélectionner un module du cours et d'afficher les noms, prénoms des collègues de ce pays

Note sur le projet : ce serait très sympa d'imaginer un site internet écrit en Python (avec Flask) qui permet cette fonctionnalité. L'intérêt d'avoir un site internet écrit avec Flask est qu'il fonctionne sur une Raspberry pi et c'est une fonctionnalité très intéressante pour de l'IoT. C'est ce que j'utilise pour piloter mon système de chauffage (2 chaudières en parallèle, les circulateurs et une pompe à eau) à mon domicile.