TD : Interfaces graphiques et interactions

L'objectif de ce sujet est de vous faire découvrir quelques notions supplémentaires concernant les interfaces graphiques, et notamment les possibilités d'interaction avec l'utilisateur.

I. Une classe héritée de `tkinter.Tk`

a. Syntaxe de base

En Python, il existe de nombreux outils pour réaliser de telles interfaces graphiques. Dans le cadre de ce cours, nous allons utiliser le module `tkinter`, qui a le gros avantage d'inclus par défaut dans les distributions de Python (il n'y a donc rien à installer de plus).

Toutes nos fenêtres graphiques seront donc créées à partir de la classe `tkinter.Tk` : plus précisément, pour créer une fenêtre graphique, nous allons créer une classe qui hérite de la classe `tkinter.Tk`. La syntase de base est la suivante :

```
import tkinter

class MaFenetreGraphique(tkinter.Tk):

def __init__(self, parent = None):
tkinter.Tk.__init__(self, parent)
# Declaration du contenu de la fenetre
self.mainloop()

# Methodes propres a la fenetre

MaFenetreGraphique()
```

Remarque : Comme indiqué dans le code ci-dessus, le constructeur de notre classe utilise deux arguments : `self`, et `parent`. Ce deuxième argument prend une valeur par défaut (ici `None`) ; nous verrons un peu plus loin à quoi cela correspond.

Question 1. Utiliser le code ci-dessus pour créer une classe héritée de la classe `tkinter.Tk`.

b. Déclaration du contenu de la fenêtre

L'étape suivante consiste à ajouter des éléments dans la fenêtre graphique.

Pour cela, il suffit de déclarer ces éléments en tant qu'attributs de cette nouvelle classe (dans le constructeur), et d'utiliser la méthode `pack` pour les ajouter dans la fenêtre principale.

Par exemple, pour ajouter un élément de type Canvas, on peut écrire :

```
self.monCanvas = tkinter.Canvas(self, width = 300, height = 200, background = "red")
self.monCanvas.pack()
```

Question 2. Utiliser la syntaxe précédente pour ajouter un élément de type `Canvas` dans votre fenêtre graphique. Ajouter ensuite plusieurs éléments de type `Canvas` : que peut-on en déduire sur le fonctionnement de la méthode `pack` ?

Question 3. Supprimer tous les objets de type `Canvas` sauf un. Modifier ensuite le constructeur pour lui ajouter deux paramètres `largeur` et `hauteur`, et les utiliser pour fixer les dimensions de l'élément de type `Canvas` restant.

Remarque : Il est ainsi possible de passer en paramètres différents aspects de la fenêtre graphique, comme par exemple la couleur de fond.

En Python, il est possible de définir des valeurs par défaut pour les arguments d'une fonction : si cette valeur n'est pas renseignée par l'utilisateur, c'est la valeur par défaut qui sera utilisée.

La syntaxe est relativement simple : il suffit d'ajouter un signe égal (=) suivie de la valeur par défaut après le nom de l'argument. Par exemple, `def direBonjour(prenom = "John")`. Il est même possible de fixer des valeurs par défaut à plusieurs arguments d'une même fonction : `def direBonjour(prenom = "John", nom = "Doe")`.

Si une fonction a plusieurs arguments, dont seuls certains ont des valeurs par défaut, ces éléments doivent être listés en dernier : `def direBonjour(age, prenom = "John", nom = "Doe")`

Question 4. Ajouter des valeurs par défaut au constructeur pour que l'utilisateur ne soit pas obligé de renseigner les dimensions de la fenêtre souhaitée.

II. Les composants de base

a. Les labels

Les labels sont des blocs de texte non éditables : ils sont généralement utilisés pour afficher une information à l'utilisateur.

Pour déclarer un label, on utilise la classe `tkinter.Label`, avec la syntaxe suivante :

```
self.monLabel = tkinter.Label(self, text="Texte du label")
self.monLabel.pack()
```

Question 5. Ajouter un label dans la fenêtre graphique.

b. Les champs de saisie

Les champs de saisie sont des blocs de texte éditables : ils permettent à l'utilisateur de renseigner une information qui sera ensuite utilisée par la machine.

Pour déclarer un champ de saisie, on utilise la classe `tkinter.Entry`, avec la syntaxe suivante:

```
self.monChampDeSaisie = tkinter.Entry(self)
self.monChampDeSaisie.pack()
```

Question 6. Ajouter un champ de saisie dans la fenêtre graphique.

c. Les boutons

Les boutons sont des blocs cliquables, qui vont permettre à l'utilisateur de donner des instructions à la machine.

Pour déclarer un bouton, on utilise la classe `tkinter.Button`, avec la syntaxe suivante :

```
self.monBouton = tkinter.Button(self, text="Texte du bouton")
self.monBouton.pack()
```

Question 7. Ajouter un bouton dans la fenêtre graphique.

III. Boutons et commandes

A ce stade, notre interface graphique contient différents types d'éléments, mais les interactions restent limitées : quels que soient les textes entrés dans les champs de saisie ou les clics effectués sur les boutons, rien ne se passe.

a. L'argument `command`

Nous allons donc définir quel est le comportement attendu au clic sur un bouton : pour cela, on va modifier la déclaration des boutons dans le constructeur, pour y ajouter l'argument `command` :

```
self.monBouton1 = tkinter.Button(self, text="Texte du bouton 1", command = self.MaFonction1)
```

La fonction à appeler, ici `MaFonction1`, doit être l'une des méthodes définies dans cette classe.

Remarque : Comme cette fonction fait partie de la classe, elle a accès à tous les attributs de cette classe. On peut donc par exemple tracer un rectangle sur l'élément de type `Canvas` en écrivant `self.monCanvas.create_rectangle(10,20, 120, 200, fill="red")`.

Question 8. Créer une méthode qui trace un disque à une position aléatoire sur l'élément de type `Canvas`, et modifier le constructeur pour appeler cette méthode en cas de clic sur le bouton.

b. Méthodes héritées

Notre fenêtre graphique hérite des méthodes définies pour la classe mère `Tk.inter` : par exemple, l'instruction `self.quit()` a pour effet d'utiliser la méthode `quit` définie dans la classe `Tk.inter`, et de provoquer la fermeture de la fenêtre.

Question 9. Ajouter un bouton "Quitter" dans la fenêtre graphique, qui appelle une méthode chargée de fermer la fenêtre.

c. Accéder et modifier les éléments textuels

Plusieurs approches sont possibles pour accéder ou modifier les éléments textuels présents dans la fenêtre graphique :

- Pour modifier le texte d'un label, on peut écrire
  `self.monLabel["text"] = monNouveauTexte`
- Pour obtenir le texte actuellement affiché sur un label, on peut écrire
  `self.monLabel["text"]`
- Pour obtenir le contenu d'un champ de saisie, on peut écrire
  `self.monChampDeSaisie.get()`
- Pour modifier le contenu d'un champ de saisie, on peut écrire

```
self.monChampDeSaisie.delete(0, tkinter.END)
self. monChampDeSaisie.insert(0, str(self.longueurListe))
```

Question 10. Ajouter un bouton dans la fenêtre graphique, qui appelle une méthode chargée d'afficher un texte de votre choix sur le label.

Question 11. Ajouter un nouveau bouton et un nouveau label, qui doit afficher le nombre de clics déjà effectués sur ce bouton.

Indice : Vous pouvez tout à fait déclarer des attributs de type `int`, `float` ou `str` pour votre classe.

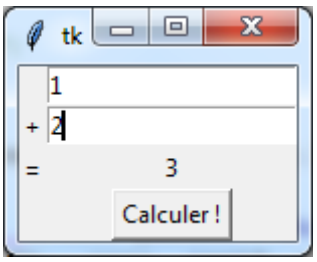
Question 12. Ajouter un bouton qui copie le contenu du bloc de saisie sur l'un des labels.

IV. Mise en page

Pour améliorer la mise en page des éléments, on peut utiliser la méthode `grid` au lieu de la méthode `pack`. Cette méthode permet en effet de répartir automatiquement les éléments dans une grille, en spécifiant pour chacun d'eux la ligne et la colonne où on souhaite les afficher.

Par exemple, si l'on écrit `self.monBouton.grid(row = 2, column = 3)`, l'élément `monBouton` sera positionné sur la 3e ligne (indice 2) et la 4e colonne (indice 3).

Question 13. Créer une nouvelle interface graphique contenant 2 champs de saisie, plusieurs labels, et un bouton, afin d'obtenir le résultat suivant :



★ **Question 14.** Créer une nouvelle interface graphique représentant une calculatrice simplifiée :

- 10 boutons pour chacun des chiffres de 0 à 9
- 4 boutons pour les 4 opérateurs + - * /
- Un bouton =
- Un label pour afficher la saisie en cours
- Un label pour afficher les résultats

V. Clics sur un élément de type `Canvas`

Pour certaines interfaces graphiques, il peut être intéressant de détecter les clics de l'utilisateur sur un élément de type `Canvas`.

Pour cela, il y a deux choses à faire :

- Ajouter une instruction de la forme `self.monCanvas.bind("<Button-1>", self.onClick)` après la déclaration de l'objet de type `Canvas`.
- Définir la méthode `onClick` dans la classe.

La méthode `onClick` doit avoir deux arguments : `self`, et `event`. Ce deuxième argument, de type `tkinter.Event`, contient les coordonnées du clic dans le `Canvas` : pour y accéder, il suffit alors d'écrire `event.x` et `event.y`.

Question 15. Créer une nouvelle interface graphique contenant un élément de type Canvas et un label, et utiliser la syntaxe ci-dessus pour afficher le texte « Clic ! » sur le label lorsque l'utilisateur clique sur le canvas.

Question 16. Modifier le code de la question précédente pour afficher les coordonnées du clic dans le label lorsque l'utilisateur clique sur le canvas.

★ **Question 17.** Créer une interface graphique pour représenter une partie de Tic-Tac-Toe, et permettre aux utilisateurs de jouer leurs coups en cliquant sur les cases de la grille :