Office de la formation professionnelle et de la promotion du travail

Rapport Interface Graphique python (Tkinter)

Le 06/08/2024

Option : DEVELOPPEMENT DIGITAL

ETABLIS PAR : Yasser KHIAT, Saad AIT ALLOUCH, Souhaib EDDAHMANI, Mohammed Reda MENYANI

Les applications graphiques avec Tkinter :

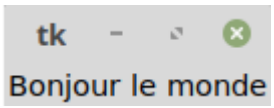
Tkinter est un module intégré à Python pour développer des applications graphiques. Ce module se base sur la bibliothèque graphique Tcl/Tk.

Un premier programme :

Le programme ci-dessous montre le principe de base de `tkinter` :

```
1  import tkinter as tk
2
3  app = tk.Tk()
4
5  message = tk.Label(app, text="Bonjour le monde")
6  message.pack()
7
8  app.mainloop()
```

Une fois lancé, ce programme fait apparaître une fenêtre avec le message « Bonjour le monde ».



DETAIL DU CODE

- ligne 1 : nous importons le module `tkinter`
- ligne 3 : nous créons un nouvel objet `Tk`. Cet objet représente la fenêtre principale de l'application graphique.
- ligne 5 : nous créons un composant graphique de type `Label`. Ce composant a la charge d'afficher un texte. Notez que l'on passe l'objet `app` comme premier paramètre de construction pour indiquer qu'il appartient à la fenêtre principale.
- ligne 6 : On appelle la méthode `pack` du composant `Label`. Cette méthode permet de calculer la taille du composant à l'écran (notamment pour pouvoir afficher correctement le texte).
- ligne 8 : On appelle la méthode `mainloop()`. C'est cette méthode qui affiche la fenêtre et lance la boucle d'événements.

Les objets d'une interface graphique :

Les fenêtres

La fenêtre principale d'une application Tkinter se construit avec le constructeur `Tk()`.

On peut modifier son apparence en passant certaines options lors de sa construction, ou bien après :

```
fenetre.title("Calculatrice ISN")
fenetre.minsize(300,200)
```

Les widgets :

Les éléments graphiques qui apparaissent dans une fenêtre sont nommés ***widgets***. Certains sont statiques (affichage simple) et d'autres permettent une interaction avec l'utilisateur.

Syntaxe commune :

```
Nom_du_widget(parent, [autres arguments])
```

Cadre

Un **cadre** (*frame*) est un conteneur pour d'autres *widgets*. Il permet de réaliser des groupes cohérents de widgets.

L'usage veut que l'on place au moins un un cadre dans la fenêtre de base, pour contenir les *widgets* de l'application :

```
cadre = Frame(fenetre)
cadre.pack()
```

Label

Les **labels** permettent d'afficher du texte dans une fenêtre. Le constructeur

`Label()`

accepte de nombreux arguments pour la mise en forme du texte. Mais on peut modifier les options de l'objet après sa construction grâce à la méthode

`.configure()`

Par exemple :

```
titre.configure(fg = 'red')
```

Entrée

Une **entrée** permet à l'utilisateur de saisir un texte sur une seule ligne.

Le constructeur

`Entry()`

accepte de nombreux arguments *(options d'apparence, ...)* dont le plus important est la **variable Tkinter**, objet de type `StringVar`, permettant de lier de façon dynamique le contenu de l'entrée au reste du programme.

```
# Saisie de l'expression mathématique : Entrée (Entry)
expression = StringVar()
expression.set("6*7")     # texte par défaut affiché dans l'entrée
entree = Entry(cadre, textvariable = expression, width = 30)
entree.pack()
```

Pour bien comprendre le comportement d'une **variable Tkinter**, il suffit de l'utiliser dans un autre *widget* (ci-dessous avec un

`Label`

):

```
# Résultat du calcul
sortie = Label(cadre, textvariable = expression)
sortie.pack()
```

Bouton

Les **boutons** permettent à l'utilisateur d'exécuter une *action*. Le constructeur

`Button()` attend donc un argument `Command` de type fonction.

```
# Bouton pour exécuter les calculs
bouton = Button(cadre, text = "Calculer", command = calculer)
bouton.pack()
```

L'inconvénient de la fonction

`calculer()`

, c'est qu'elle provoque une erreur dans les cas où l'expression n'est pas évaluable. Il faut donc prévoir cette éventualité :

```
def calculer():
    try:
        resultat.set(eval(expression.get()))
    except:
        pass
```

Bouton radio

Un **bouton radio** n'est en principe jamais seul : quand l'utilisateur doit faire un choix unique parmi plusieurs possibilités, on utilise typiquement un *groupe de boutons radio*.

Par exemple, si nous souhaitons contrôler le mode d'affichage du résultat :

```
# Boutons radio pour sélection du mode d'affichage
mode = StringVar()
bouton1 = Radiobutton(cadre, text="normal", variable=mode, value="n",
                      command = calculer)
bouton2 = Radiobutton(cadre, text="scientifique", variable=mode, value="s",
                      command = calculer)
bouton1.pack()
bouton2.pack()
bouton1.select()
```

Liste

Arrangement des widgets

Jusqu'à présent, tous les widgets se sont placés les uns au dessous des autres, dans l'ordre de leur création, et centrés horizontalement dans la fenêtre.

Il existe différentes méthodes pour gérer les emplacements des widgets dans une fenêtre :

- méthode `.place()` : positionnement absolu (à partir de coordonnées)
- méthode `.pack()` : un gestionnaire simple d'empilement de widgets
- méthode `.grid()` : arrangement des widgets dans une grille (ou un « tableau »)

Grid

La méthode

`.grid()`

permet d'arranger les widgets dans un « tableau », composé de lignes (*rows*), de colonnes (*columns*).

Un widget peut occuper plusieurs lignes et/ou plusieurs colonnes adjacentes (*rowspan*, *columnspan*).

Il peut être placé dans différentes positions (*sticky* ; N : nord, E : est, S : sud, W : ouest) dans sa cellule : sticky=NE (haut-droite), SE (bas-droite), SW (bas-gauche), ...

Exemples :

```
titre.grid(row=0, column=0, columnspan=4)
entree.grid(row = 1, column = 0, columnspan = 3, sticky = W+E)
...
```

Les événements

Fonctionnalités avancées

Surveillance d'une variable Tkinter

Il peut être particulièrement intéressant d'exécuter une action dès qu'une **variable Tkinter** a été modifiée. Pour cela, il faut lui attacher un **observateur** à l'aide de la méthode

```
expression.trace("w", calculer)
```

Fermer correctement une application Tkinter

```
from tkinter import *

##################################################################################
# Classe définissant l'objet représentant la fenêtre principale de l'application
##################################################################################
class Application(Tk):
    def __init__(self):
        Tk.__init__(self)
        self.title("Application du tonnerre !")   # Le titre de la fenêtre

        self.minsize(1550,850)      #taille de fenêtre

        # Une méthode séparée pour construire le contenu de la fenêtre
        self.createWidgets()

    # Méthode de création des widgets
    def createWidgets(self):
        self.grid()      # Choix du mode d'arrangement

        # Création des widgets
        # ...........
        # ...........
        # ...........

        # Un bouton pour quitter l'application
        self.quitButton = Button(self, text = "Quitter",
                                 command = self.destroy)
        self.quitButton.grid()

    # D'autres méthodes ....
    # ......................

app = Application()
app.mainloop()
```