Programmation événementielle

Dr. NSENGE MPIA HERITIER, PhD

Présentation

- La programmation événementielle est un paradigme de programmation dans lequel l'exécution d'actions est déclenchée automatiquement lorsqu'un événement survient.
- Un événement correspond en général à un changement d'état dans l'univers, ou bien à une intervention explicite de l'utilisateur (ou d'un système externe).
- On peut noter ces associations (événement, action) sous la forme :

  événement → action.

- La programmation événementielle est souvent utilisée dans les cas suivants :
  - La programmation d'automates (systèmes de régulation par exemple)
    - Par exemple : température < 20 → déclencher chauffage
  - La programmation d'interface graphique.
    - En effet, chaque action de l'utilisateur (clic souris, etc.) peut être vu comme un événement.

Présentation (Cont.)

- Avant d'entrer dans le vif du sujet, il est nécessaire de dire quelques mots sur la notion de programmation événementielle
  - Quand vous programmez en Python en mode console (sans utiliser d'interface graphique), votre programme s'exécute en commençant par la première ligne et en se terminant à la dernière ligne, comme dans cet exemple :

```
print("Ceci est ma première ligne, le programme commence")
a=5
b="Hello World!"
print(str(a)+" "+b)
print("Ceci est la dernière ligne, le programme s'arrête")
#Qui afficheront par ordre ce qui suit
```

```
Ceci est ma première ligne, le programme commence
5 Hello World!
Ceci est la dernière ligne, le programme s'arrête
```

- Le programme s'exécute de façon séquentielle, **ligne après ligne**, le programme à un début et une fin (**la dernière ligne de votre code sera la dernière ligne exécutée**).
- La programmation événementielle **fonctionne différemment**
  - en effet, ce type de programmation est basé sur une "boucle infinie", le programme boucle en attendant des évènements (action au niveau du clavier ou de la souris).

Présentation (Cont.)

- Dans le cas de Tkinter, c'est l'instruction "**mainloop()**" ("boucle principale" en français) qui permettra de rentrer dans cette boucle infinie (en attendant des évènements)
- La structure d'un programme événementielle est à peu près toujours la même :
  - on définit l'interface graphique (on met en place les boutons, les fenêtres...)
  - on définit les évènements (si l'utilisateur clique sur le bouton alors tu devras faire telle action...)
  - on fait entrer le programme dans la boucle principale (avec mainloop() dans le cas de Tkinter).

Saisissez, analysez et testez ce programme

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from tkinter import *
fen=Tk()
fen.mainloop()
```

- Explication du code ci-haut:
  - la première ligne permet de s'assurer de l'utilisation du bon interpréteur Python
  - la deuxième ligne permet d'utiliser l'UTF-8 pour l'encodage des caractères
  - **from tkinter import *** permet d'importer la bibliothèque Tkinter
  - **fen=Tk()** permet de définir la fenêtre principale de Tkinter (c'est cette fenêtre qui va accueillir les autres éléments (widgets) : boutons, zone de texte...)
  - **fen.mainloop()** le programme entre dans la boucle principale.
- Dans cet exemple, nous n'avons pas programmé d'évènements, en revanche, l'événement "clic de souris sur la croix en haut à droite de la fenêtre" est implémenté par défaut, il entraîne la sortie de la boucle principale et donc la fin du programme.

Les événements en TKinter

- En Tkinter, les événements sont des actions telles que des clics de souris, des pressions de touches, etc., qui se produisent pendant l'exécution de votre application.

- Vous pouvez associer des fonctions (appelées gestionnaires d'événements) à ces événements pour définir le comportement de votre application en réponse à ces actions.

Les événements en Tkinter (Cont.)

- Quelques événements courants en Tkinter :
  1. Clic de souris (<Button-1>, <Button-2>, <Button-3>) :

     Ces événements se produisent lorsque vous cliquez avec le bouton gauche, le bouton du milieu ou le bouton droit de la souris, respectivement.
  2. Pression de touche (<KeyPress>) : Cet événement se produit lorsque vous appuyez sur une touche du clavier.
  3. Libération de touche (<KeyRelease>) : Cet événement se produit lorsque vous relâchez une touche du clavier.
  4. Déplacement de la souris (<Motion>) : Cet événement se produit lorsque vous déplacez la souris sur un widget.
  5. Entrée dans un widget (<Enter>) : Cet événement se produit lorsque le pointeur de la souris entre dans un widget.
  6. Sortie d'un widget (<Leave>) : Cet événement se produit lorsque le pointeur de la souris quitte un widget.

Exemple de quelques événements (KeyRelease et ComboboxSelected)

```python
import tkinter as tk # Importe le module tkinter sous le nom tk, permettant d'accéder aux fonctionnalités de l'interface graphique Tkinter.
from tkinter import ttk #Importe le module ttk de tkinter, qui contient des widgets améliorés par rapport aux widgets standards.
#On définit une fonction on_entry_change qui sera appelée lorsque le texte dans la zone de texte (Entry) change.
#On prend un argument event, qui représente l'événement déclencheur (ici, la saisie d'une touche).
def on_entry_change(event):
    # Affiche un message dans la console indiquant que le texte a changé, suivi du texte actuel dans la zone de texte (Entry).
    print("Texte changé:", ceQuiestSaisi.get())
#On définit une fonction on_combobox_select qui sera appelée lorsque vous sélectionnez une option dans le menu déroulant (Combobox).
#On prend un argument event, qui représente l'événement déclencheur (ici, la sélection d'une option).
def on_combobox_select(event):
    # On affiche un message dans la console indiquant que l'option a été sélectionnée,
    #suivi de l'option actuellement sélectionnée dans le menu déroulant (Combobox).
    print("Option sélectionnée:", combobox.get())
#On crée une instance de la classe Tk, qui représente la fenêtre principale de l'application.
root = tk.Tk()
# Zone de texte. On crée un widget de zone de texte (Entry) dans la fenêtre principale.
ceQuiestSaisi = tk.Entry(root)
#On place le widget de zone de texte dans la fenêtre principale.
ceQuiestSaisi.pack()
#On associe la fonction on_entry_change à l'événement <KeyRelease> (relâchement d'une touche) de la zone de texte (Entry).
ceQuiestSaisi.bind("<KeyRelease>", on_entry_change)
# Combobox. On définit une liste d'options pour le menu déroulant (Combobox).
valeurs = ["Option 1", "Option 2", "Option 3"]
#On crée un widget de menu déroulant (Combobox) dans la fenêtre principale avec les options définies comme valeurs
combobox = ttk.Combobox(root, values=valeurs)
#On place le widget de menu déroulant dans la fenêtre principale.
combobox.pack()
#On associe la fonction on_combobox_select ci-haut à l'événement <<ComboboxSelected>> (sélection d'une option) du menu déroulant (Combobox).
combobox.bind("<<ComboboxSelected>>", on_combobox_select)
#On lance la boucle principale de l'interface graphique, permettant à l'application de détecter et de répondre aux événements utilisateur.
root.mainloop()
```

Événement du clavier

- Ci-dessous, on aperçoit une fenêtre Tkinter sans contenu.
  - Mais lorsqu'on appuie sur une touche du clavier, la console affiche un message indiquant sur quelle touche on a pressé.

```python
from tkinter import *
#On définit une fonction touche qui sera appelée lorsque l'événement de pression d'une touche est déclenché.
#Onp prend un argument event, qui représente l'événement déclencheur (ici, la touche pressée).
def touche(event):
    #On récupère le symbole de la touche pressée à partir de l'événement keysym et le stocke dans la variable t.
    t=event.keysym
    #On modifie le texte affiché dans le widget label pour indiquer quelle touche a été pressée.
    label.config(text="Touche %s pressée" % t)
#On crée une instance de la classe Tk, qui représente la fenêtre principale de l'application.
root = Tk()
#On crée un widget de type Label dans la fenêtre principale avec un texte initial vide.
label = Label(root, text="")
#On place le widget label dans la fenêtre principale.
label.pack()
#On associe la fonction touche à l'événement <Key> (pression d'une touche) de la fenêtre principale,
#de sorte que chaque fois qu'une touche est pressée, la fonction touche est appelée.
root.bind('<Key>', touche)
#On lance la boucle principale de l'interface graphique, permettant à l'application de détecter et de répondre aux événements utilisateur.
root.mainloop()
```

Récapitulatif des événement de la souris

- Ce tableau constitue le résumé des principaux événements liés à la souris que vous pouvez approfondir en lisant ici.

Nom de l'évenement de souris	Action
Button-1	Clic gauche
Button-3	Clic bouton droit
Button-2	Clic bouton central
Button-4	Molette vers le haut
Button-5	Molette vers le bas
ButtonRelease	Relâchement d'un bouton
Motion	Déplacement du curseur de la souris
MouseWheel	Roulette (pas sous Linux)
Button	Un des boutons

GUI avec python

- Python fournit diverses options pour développer des interfaces graphiques (GUI).
- Les plus importants sont:
  - **Tkinter** : Tkinter est une interface Python de la boîte à outils Tk GUI livrée avec Python.
  - **wxPython** : Ceci est une interface Python open source pour wxWindows.
  - **JPython** : JPython est une interface Python pour Java qui donne aux scripts Python un accès transparent aux bibliothèques de classes Java sur la machine locale.

Tkinter

- Tkinter est la bibliothèque GUI standard pour Python.
- Python lorsqu'il est utilisé avec Tkinter fournit un moyen rapide et facile de créer des applications GUI.
- Tkinter fournit une puissante interface, orientée objet à la boîte à outils Tk GUI.
- La création d'une application GUI à l'aide de Tkinter est une tâche facile. Tout ce que vous devez faire est d'effectuer les étapes suivantes :
- Importez le module **Tkinter**.
- Créez la fenêtre principale de l'application GUI.
- Ajoutez un ou plusieurs des widgets mentionnés ci-dessous à l'application GUI.
- Faite une boucle d'événement pour prendre des mesures contre chaque événement déclenché par l'utilisateur.

Tkinter(Cont.)

- Tkinter est appelé ainsi car c'est une interface Python vers Tk (Tool Kit), qui est un toolkit graphique développé à l'origine pour le langage de programmation **TCL** (Tool Command Language).

- Le nom "Tkinter" est en fait une contraction de "**Tk** interface".

- Tkinter est installé par défaut dans anaconda, si ce n'est pas le cas, lancez la commande suivante: pip install python-tk

Les widget Tkinter

- Pour créer un logiciel graphique vous devez ajouter dans une fenêtre des éléments graphiques que l'on nomme widget.

- Ce widget peut être tout aussi bien une liste déroulante que du texte.

Types des widgets

Widget	Description
Button	Le widget Button est utilisé pour afficher les boutons dans votre application.
Canvas	Le widget Canvas est utilisé pour dessiner des formes, telles que des lignes, des ovales, des polygones et des rectangles, dans votre application.
Checkbutton	Le widget Checkbutton est utilisé pour afficher un certain nombre d'options sous forme de cases à cocher. L'utilisateur peut sélectionner plusieurs options à la fois.
Radiobutton	Le widget Radiobutton est utilisé pour afficher un certain nombre d'options sous forme de boutons radio. L'utilisateur ne peut sélectionner qu'une seule option à la fois.
Entry	Le widget Entry est utilisé pour afficher un champ de texte sur une seule ligne pour accepter les valeurs d'un utilisateur.
Frame	Le widget Frame est utilisé comme conteneur pour organiser d'autres widgets.
Label	Le widget Label est utilisé pour fournir une étiquette sur une seule ligne pour d'autres widgets. Il peut également contenir des images.
Listbox	Le widget Listbox est utilisé pour fournir une liste d'options à un utilisateur.
Menubutton	Le widget Menubutton est utilisé pour afficher les menus de votre application.

Types des widgets (Cont.)

Widget	Description
Menu	Le widget Menu est utilisé pour fournir diverses commandes à un utilisateur. Ces commandes sont contenues dans Menubutton.
Message	Le widget Message est utilisé pour afficher les champs de texte multilignes pour accepter les valeurs d'un utilisateur.
Scale	Le widget Scale est utilisé pour fournir un curseur.
Scrollbar	Le widget Scrollbar est utilisé pour ajouter une fonction de défilement à divers widgets, tels que Listbox .
Text	Le widget Text est utilisé pour afficher le texte sur plusieurs lignes.
Toplevel	Le widget Toplevel est utilisé pour fournir un conteneur de fenêtre séparé.
Spinbox	Le widget Spinbox est une variante du widget Tkinter Entry standard, qui peut être utilisé pour sélectionner un nombre fixe de valeurs.
PanedWindow	Un PanedWindow est un widget conteneur qui peut contenir n'importe quel nombre de volets, disposés horizontalement ou verticalement.
Labelframe	Un labelframe est un widget conteneur simple. Son objectif principal est d'agir comme un séparateur ou un conteneur pour les dispositions de fenêtres complexes.
tkMessageBox	Ce module permet d'afficher des boîtes de message dans vos applications.

Création de l'instance de la classe **Tk**

- La ligne root = tk.Tk() crée une instance de la classe Tk dans Tkinter, qui représente la fenêtre principale de l'application graphique.
- Cette instance est souvent appelée "racine" (root en anglais) de l'interface graphique, car tous les autres widgets sont ajoutés à cette fenêtre ou à ses sous-fenêtres.
- Une fois que la fenêtre racine est créée, vous pouvez commencer à ajouter d'autres widgets et à définir la mise en page de votre application à l'intérieur de cette fenêtre.
- Code de création de la fenêtre principale:

```
import tkinter as tk
# Créer une fenêtre
root = tk.Tk()
```

Démarrage d'une application TK

- root.mainloop() est une méthode dans Tkinter qui démarre la boucle principale de l'application graphique.
- Cette boucle est responsable de la gestion des événements (comme les clics de souris, les pressions de touches, etc.) et de l'actualisation de l'interface utilisateur
- Une fois que **vous avez créé tous les widgets** et **défini la mise en page de votre interface graphique**, vous appelez root.mainloop() **pour démarrer l'application**.
  - À partir de ce moment, l'application est en attente d'événements (comme des clics de souris ou des pressions de touches) et répond en conséquence.
  - La boucle continue de s'exécuter jusqu'à ce que vous fermiez la fenêtre principale de l'application, ce qui arrête également la boucle et termine le programme.
  - **Exemple du code:**

```
import tkinter as tk
# Créer une fenêtre
root = tk.Tk()
# Placer les widgets dans la fenêtre
....
# Démarrer la boucle principale
root.mainloop()
```

Input/Output en TK

- En Tkinter, les widgets d'entrée/sortie les plus couramment utilisés sont les suivants :
  - **Entry** : Un champ de saisie simple qui permet à l'utilisateur de saisir du texte:

    entry = tk.Entry(root)
  - **Text** : Une zone de texte multi-lignes qui permet à l'utilisateur de saisir et d'afficher du texte:

    text = tk.Text(root)
  - **Label** : Un widget utilisé pour afficher du texte ou une image. Il est principalement utilisé pour la sortie:

    label = tk.Label(root, text="Bonjour mes amis!")
  - **Button** : Un bouton qui, lorsqu'il est cliqué, peut déclencher une action comme l'affichage d'un message:

    button = tk.Button(root, text="Clique moi", command=action_function)

Input/Output en TK (Cont.)

- Pour récupérer du texte saisi par l'utilisateur dans un Entry, vous pouvez utiliser la méthode get() de l'objet **Entry**.
- Par exemple :

```
nombre = tk.Entry(root)
user_input = nombre.get()
```

- Pour afficher du texte dans un **Text** ou un **Label**, vous pouvez utiliser la méthode insert() pour insérer du texte à une position spécifiée, ou la méthode config() pour modifier le texte existant.
- Par exemple :

```
text.insert(tk.END, "Hello, World!") # Insérer du texte à la fin du widget Text (tk.END)
label.config(text= "Nouveau text")
```

Gestionnaires de positionnement

- Tous les widgets Tkinter ont accès à des méthodes de gestion de géométrie spécifiques, qui ont pour but d'organiser les widgets dans la zone de widget parent.
- Tkinter expose les classes de gestionnaire de positionnement suivantes:
  - Pack
  - Grid
  - place.

Gestionnaires de positionnement (Cont.)

- **Pack()** :
  - La méthode pack() est une méthode de gestion de la mise en page dans Tkinter.
  - Elle est utilisée pour placer un widget (comme un bouton, une étiquette, etc.) dans un conteneur, tel qu'une fenêtre principale ou un cadre.
  - La méthode pack() organise les widgets de manière à ce qu'ils occupent l'espace minimum requis pour afficher tous les widgets sans superposition.
  - Par exemple, pour créer et afficher un simple bouton dans une fenêtre Tkinter en utilisant pack() :

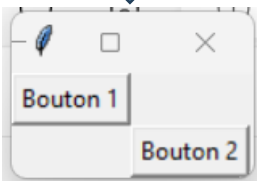
```python
import tkinter as tk
# Créer une fenêtre
root = tk.Tk()
# Créer un bouton
button = tk.Button(root, text="Université Shalom")
# Afficher le bouton en utilisant pack()
button.pack()
# Démarrer la boucle principale
root.mainloop()
```

- Dans cet exemple, button.pack() est utilisé pour placer le bouton dans la fenêtre principale root.
- La méthode pack() est simple à utiliser mais offre moins de contrôle sur la mise en page que d'autres méthodes telles que grid() et place().

Gestionnaires de positionnement (Cont.)

- **Grid()** :
  - La méthode grid() est une autre méthode de gestion de la mise en page dans Tkinter.
  - Contrairement à pack(), qui organise les widgets en fonction de leur ordre d'ajout, grid() permet de placer les widgets dans une grille (ou tableau) à deux dimensions.
  - Dans l'exemple ci-contre, button1.grid(row=0, column=0) place le premier bouton dans la première ligne et la première colonne de la grille, tandis que button2.grid(row=1, column=1) place le deuxième bouton dans la deuxième ligne et la deuxième colonne.

```python
import tkinter as tk
# Créer une fenêtre
root = tk.Tk()
# Créer deux boutons
button1 = tk.Button(root, text="Bouton 1")
button2 = tk.Button(root, text="Bouton 2")
# Placer les boutons dans une grille
button1.grid(row=0, column=0)
button2.grid(row=1, column=1)
# Démarrer la boucle principale
root.mainloop()
```

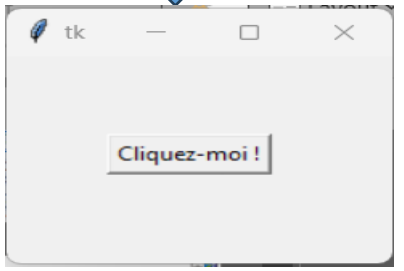


- Vous pouvez spécifier d'autres options comme sticky pour indiquer comment le widget doit remplir la cellule de la grille et padx et pady pour ajouter un espace autour du widget.

Gestionnaires de positionnement (Cont.)

- **Place()** :
  - La méthode place() est une autre méthode de gestion de la mise en page dans Tkinter. Contrairement à pack() et grid(), qui utilisent des algorithmes de disposition automatique, **place()** permet de positionner un widget de manière absolue en spécifiant les coordonnées **x** et **y** de son coin supérieur gauche.
  - Ci-contre un exemple simple qui utilise **place()** pour positionner un bouton à des coordonnées spécifiques dans une fenêtre Tkinter :

```python
import tkinter as tk
# Créer une fenêtre
root = tk.Tk()
# Créer un bouton
button = tk.Button(root, text="Cliquez-moi !")
# Positionner le bouton à des coordonnées spécifiques
button.place(x=50, y=50)
# Démarrer la boucle principale
root.mainloop()
```



- Dans cet exemple, button.place(x=50, y=50) place le bouton aux coordonnées **(50, 50)** dans la fenêtre.
- Vous pouvez également spécifier d'autres options telles que **width** et **height** pour définir la taille du widget, ou utiliser des valeurs relatives comme "50%" pour les positions et tailles.