Langages de scripts (LS4)

TP 7 – Tkinter – interfaces graphiques en PYTHON

Exercice 1 – *Une première interface : « Hello, World ! ».*

Voici le code d'un script PYTHON qui crée une fenêtre et y affiche le message « Hello, World ! » :

```
#!/usr/bin/env python

# importer la bibliothèque Tkinter
from Tkinter import *

# créer la fenêtre principale
top = Tk()

# créer un cadre dans cette fenêtre
frame = Frame(top)
frame.pack()

# créer une étiquette dans ce cadre
label = Label(frame,
    text="Hello, World")
label.pack()

# créer un bouton sortie lié à la fonction destroy
button = Button(frame,
    text="Exit",
    command=top.destroy)
button.pack()

# lancer la boucle principale du programme
top.mainloop()
```

Prenez bien note de la façon dont des options peuvent être passées à la plupart des fonctions Tkinter, par exemple l'option text est ici passée aux constructeurs de Label et Button.

1. Récupérez le fichier `monitLS4/hello.py` et testez-le, premièrement en le recopiant ligne par ligne dans l'interpréteur. Avant d'évaluer la dernière instruction, essayez d'utiliser le bouton. Que constatez-vous ? Testez enfin ce script en le lançant comme un exécutable.

2. Ajoutez un bouton étiqueté « Date » qui, lorsqu'il est cliqué, provoque l'affichage de la date et l'heure actuelles dans la fenêtre. Vous pourrez par exemple utiliser la fonction datetime.datetime.today() du module `datetime`[1].

Exercice 2 – *Automates.* **Exercice pouvant compter pour le contrôle continu.**

Cet exercice en deux parties a pour objectif de vous faire programmer une classe Python simpliste permettant de représenter des automates finis, et une petite interface graphique pour fournir des mots en entrée à un automate, effectuer le calcul correspondant (pas à pas ou d'une traite) et afficher le résultat du calcul (mot rejeté ou accepté).

Première partie : la classe Automaton

1. Créez la classe Automaton, et son constructeur acceptant trois arguments (état initial, liste d'états finaux et fonction de transition). Veillez à choisir une représentation adaptée pour les attributs de votre classe. On supposera que les automates créés seront toujours *déterministes*.
2. Programmez la méthode `can_step(c)` qui permet de tester si une transition est possible depuis l'état courant de l'automate en lisant un caractère `c`.
3. Programmez la méthode `step(c)` qui effectue une transition de l'automate par `c` en supposant qu'elle est possible.
4. Programmez la méthode `reset` qui réinitialise l'automate, et la méthode `is_final` qui teste si l'automate est dans un état acceptant et renvoie le booléen correspondant.

Seconde partie : interface graphique Voici la capture d'écran d'une interface simpliste permettant de tester l'appartenance d'un mot donné au langage reconnu par un automate fini :

Sur cette image, l'état courant de l'automate est l'état 2, le mot « aaa » a déjà été lu et il reste à lire le mot « bbab ».

5. Commencez par créer un élément de type `Button` et associez-lui l'action de fermer la fenêtre de votre application.
6. Ajoutez à votre fenêtre un élément de type `Entry` (en lecture seule, c'est à dire non modifiable par l'utilisateur) affichant l'état actuel de l'automate.

[1] `http://docs.python.org/lib/module-datetime.html`

7. Ajoutez le bouton « Pas à pas » et associez-le à une fonction qui appelle la méthode `step` de la classe Automaton, met à jour le champ contenant le reste du mot à lire ainsi que l'affichage de l'état courant.
8. Ajoutez un bouton « Calcul » dont l'action associée consiste à lire en une seule fois autant de lettres que possible du mot fourni.

   Ici encore, mettez à jour le mot restant à lire ainsi que l'état courant. Affichez également un message approprié selon que le mot est accepté ou rejeté (vous pouvez pour cela utiliser des éléments Label ou des boîtes de dialogue[2]).
9. Ajoutez enfin un bouton « RàZ » de remise à zéro de l'application.

Améliorations Voici quelques propositions d'améliorations possibles pour votre application. Choisissez-en deux et implémentez-les :

Améliorations de la classe Automaton :
- Autorisez la création d'automates non-déterministes et créez une méthode permettant de tester si un automate est bien déterministe, et une méthode permettant de déterminiser l'automate.
- Créez une méthode permettant de minimiser un automate déterministe.
- Créez une méthode `accepts` prenant une chaîne en paramètre et déterminant directement si le mot qu'elle représente est accepté. Attention, votre méthode doit aussi fonctionner sur un automate non-déterministe (sans déterminisation préalable !).

Améliorations de l'interface graphique :
- Ajoutez des éléments permettant de saisir directement l'ensemble des transitions de l'automate sans redémarrer l'application.
- Ajoutez des mécanismes permettant de rendre indisponibles (grisés) les boutons de calcul pas à pas ou en un seul coup si la liste de lettres à lire est vide.
- Ajoutez un élément Label permettant d'afficher la partie du mot lue jusqu'à présent, tel que cela est fait sur la capture d'écran ci-avant.
- Etudiez les options de la fonction `pack()` (notamment les options `side` et `anchor`) pour imiter la disposition de la capture d'écran. Inventez une autre disposition utilisant la fonction `grid()`.
- Ajoutez des raccourcis clavier permettant d'activer les boutons de votre application en frappant simplement une touche.

[2] Obtenues par exemple grâce à la fonction `showinfo(titre, texte)` du module `tkMessageBox`