Université de Bordeaux
Semestre 2

2021-2022

TD4 : Application Intéractive

Compétences

- Notion d'IHM
- Architecture MVC
- Le module tkinter

Application Intéractive

C'est application avec laquelle l'utilisateur peut interagir. L'application effectue des traitements en réponse aux actions de l'utilisateur. Ces interactions se font la plupart du temps à travers une interface graphique également appelée interface homme-machine (IHM).

PARTIE A

IHM

Les IHM servent à rendre les programmes plus interactifs. Elles simplifient la vie des utilisateurs mais elles demandent plus de temps pour les concevoir et les développer. Un programme sans interface exécute des instructions les unes à la suite des autres. Avec une IHM, le programme attend un événement - pression d'une touche du clavier, clic de souris - pour exécuter un bout de code. C'est comme si le programme avait une multitude de points d'entrée.
Il existe plusieurs modules pour développer une interface graphique.

- tkinter : le plus simple mais limité. Visuellement, tkinter est moins joli que d'autres extensions mais c'est un package facile à installer. Pour cette raison, c'est celui que nous utiliserons.
- wxPython : est une librairie communautaire.
- PyQt5 : est une librairie professionnelle bien plus fournie et bien plus complète.

La conception d'une interface graphique se déroule généralement selon deux étapes. La première consiste à dessiner l'interface, c'est-à-dire choisir une position pour les objets de la fenêtre (boutons, zone de saisie, liste déroulante...). C'est la partie visible par l'utilisateur, elle est généralement appelée **front**. La seconde étape définit le fonctionnement de la fenêtre, c'est-à-dire associer à chaque objet des fonctions qui seront exécutées si tel événement se réalise (pression d'un bouton, pression d'une touche...). C'est la partie invisible par l'utilisateur, elle est appelée **back**.

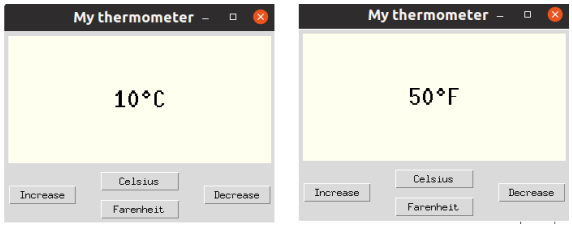
Exemple : My Thermometer

Les interfaces graphiques sont composées d'objets ou widgets ou contrôles. Voici quelques exemples de widgets :

- **Label** : A widget used to display text on the screen
- **Button** : A button that can contain text and can perform an action when clicked
- **Entry** : A text entry widget that allows only a single line of text
- **Text** : A text entry widget that allows multiline text entry
- **Frame** : A rectangular region used to group related widgets or provide padding between widgets

Ce lien https://realpython.com/python-gui-tkinter/ donne une description détaillée et des exemples d'utilisation de ces widgets. Pour prendre un bon départ, il est important d'avoir une idée précise de la structure de son IHM et des différents widgets dont vous allez avoir besoin.
Un autre lien en français (à partir de la page 49) : https://hal.archives-ouvertes.fr/hal-02126596/document

Pour notre premier exemple, nous avons besoin d'une application qui affiche la température, permet de la modifier et de l'afficher soit en degré Celsius soit en degré Fahrenheit.



1. Détaillez les tâches pour satisfaire à l'exigence du besoin exprimé.

2. Téléchargez l'archive `src_student.zip`
3. Créez un répertoire *thermometerV1* et décompressez l'archive dans ce répertoire.
4. À l'aide du code proposé :
   (a) repérez les instructions qui permettent d'obtenir un widget correspondant à la fenêtre de notre application. Quel est son type ?
   (b) déterminez les différents widgets utilisés pour cette application,
   (c) expliquez en quoi consiste la programmation événementielle d'un bouton.
5. Implémentez chacune de vos tâches.

PARTIE B

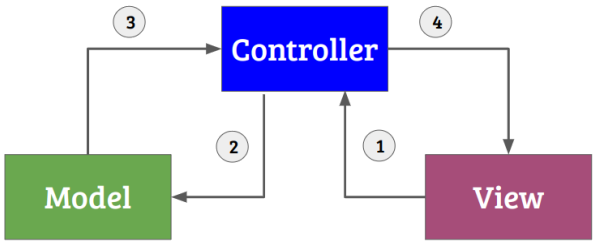
Architecture

Lors de l'élaboration d'une application interactive avec IHM, les développeurs ont noté des problèmes de conception récurrents et similaires. Il devenait donc nécessaire de conceptualiser la conception de manière à réutiliser les mêmes réponses à chaque fois que les mêmes problèmes réapparaissaient. En termes d'architecture d'application, c'est ici que les **design patterns (patrons de conception)** interviennent et nous permettent, avec des méthodes et des outils déjà éprouvés, d'aborder les problématiques de conception d'une application.
Un des patterns, communément utilisé, et qui comporte de nombreuses variantes, est connue sous l'acronyme **MVC** qui signifie **Model - View - Controller.** Dans cette architecture on divise le code des applications en entités distinctes (modèles, vues et contrôleurs) qui communiquent entre-elles au moyen de divers mécanismes (invocation de méthodes, génération et réception d'événements, etc.).

Le pattern MVC

Ce pattern permet de structurer l'application en séparant :

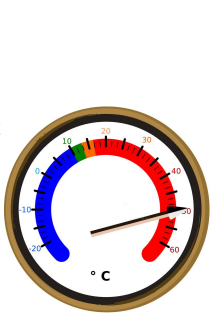
- Les données et leurs traitements : **Le Modèle**
- L'affichage des données pour l'utilisateur : **La Vue**
- Le comportement de l'application face aux actions de l'utilisateur : **Le Contrôleur**

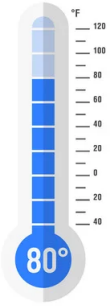


Programmation événementielle :

1. Action utilisateur,
2. `Traitement` des données,
3. Notification,
4. `Update` de la vue.

1. Téléchargez l'archive `MVC_student.zip`
2. Créez un répertoire *thermometerV2* et décompressez l'archive dans ce répertoire.
3. Quelles sont les instructions qui permettent de lancer l'application ?
4. À quoi correspond `la vue` de cette application ?
5. Quelle méthode permet de mettre `la vue` à jour ? À quelle classe appartient cette méthode ?
6. Quel objet permet de mettre `la vue` à jour ? Quel est le rôle de cet objet ?
7. Quels sont les traitements qui justifient la mise à jour de la `la vue` ?
8. Plusieurs vues peuvent-elles partager le même modèle ? Justifier
9. Complétez votre code pour satisfaire l'exigence de la PARTIE A.
10. Ajoutez la possibilité d'obtenir la température en ^{o}K (Kelvin).
11. Implémentez une vue différente, sans modifier `model` et `controller`





Deux autres `vues` possibles.