Course: Python language

Ali ZAINOUL <ali.zainoul.az@gmail.com>

for NEEDEMAND - IBM - LEARNQUEST
August 12, 2024

** PyQt **

Introduction à Conda (Anaconda)

- Conda est un gestionnaire de paquets et un environnement de gestion open-source.
- Il permet de gérer des installations de paquets logiciels ainsi que leurs dépendances.
- Anaconda est une distribution de Conda qui inclut plus de 1 500 paquets scientifiques et analytiques.
- Conda supporte plusieurs langages de programmation, y compris Python, R, Ruby, Lua, Scala, Java, JavaScript, C/C++, FORTRAN.

Commandes de base à connaître

- **Créer un environnement :** `conda create --name myenv`
- **Activer un environnement :** `conda activate myenv`
- **Désactiver un environnement :** `conda deactivate`
- **Lister les environnements :** `conda env list`
- **Supprimer un environnement :** `conda remove --name myenv --all`
- **Installer un paquet :** `conda install package_name`
- **Mettre à jour un paquet :** `conda update package_name`
- **Supprimer un paquet :** `conda remove package_name`

Gestion des environnements Conda

- **Créer un environnement :** `conda create --name myenv`
- **Activer un environnement :** `conda activate myenv`
- **Désactiver un environnement :**`conda deactivate`
- **Lister les environnements :** `conda env list`
- **Supprimer un environnement :** `conda remove --name myenv --all`
- **Sauvegarder un environnement :**
  `conda env export > environment.yml`
- **Recréer un environnement à partir d'un fichier YAML :**`conda env create -f environment.yml`

Introduction à pip

- `pip` est le gestionnaire de paquets par défaut pour Python.
- Il permet d'installer et de gérer des paquets Python depuis le Python Package Index (PyPI).
- `pip` est souvent utilisé avec des environnements virtuels pour isoler les dépendances des projets.

Commandes de base à connaître

- **Créer un environnement** : `python -m venv myenv`
- **Activer un environnement** : `# Sur Windows`
  ```
  myenv\Scripts\activate
  # Sur macOS/Linux
  source myenv/bin/activate
  ```
- **Mettre à jour pip** : `pip install --upgrade pip`
- **Installer un paquet** : `pip install package_name`
- **Mettre à jour un paquet** : `pip install --upgrade package_name`
- **Supprimer un paquet** : `pip uninstall package_name`
- **Lister les paquets installés** : `pip list`
- **Afficher les informations sur un paquet** : `pip show package_name`
- **Générer un fichier de dépendances** : `pip freeze > requirements.txt`
- **Installer les paquets depuis un fichier de dépendances** : `pip install -r requirements.txt`
- **Désactiver un environnement** : `deactivate`

Gestion des environnements pip

- **Créer un environnement** : `python -m venv myenv`
- **Activer un environnement** :
```
# Sur Windows
myenv\Scripts\activate
# Sur macOS/Linux
source myenv/bin/activate
```
- **Mettre à jour pip** : `pip install --upgrade pip`
- **Installer un paquet** : `pip install package_name`
- **Mettre à jour un paquet** : `pip install --upgrade package_name`
- **Supprimer un paquet** : `pip uninstall package_name`
- **Générer un fichier de dépendances** : `pip freeze > requirements.txt`
- **Installer les paquets depuis un fichier de dépendances** : `pip install -r requirements.txt`
- **Désactiver un environnement** : `deactivate`

Introduction à PyQt

- PyQt est un ensemble de liaisons Python pour Qt, un framework C++ pour la création d'interfaces utilisateur graphiques.
- Il permet de créer des applications multiplateformes avec une seule base de code.
- PyQt est disponible sous les licences GPL et commerciale.
- **Documentation:** Tutorial PyQt6

Installation de PyQt via pip

- **Créer un environnement :**

```
python -m venv myenv
```

- **Activer l'environnement :**

```
\# Sur Windows
myenv\Scripts\activate

\# Sur macOS/Linux
source myenv/bin/activate
```

- **Mettre à jour pip :**

```
pip install --upgrade pip
```

- **Installer PyQt6 :**

```
pip install PyQt6
```

- **Lister les paquets installés :**

```
pip list
```

- **Désactiver l'environnement :**

```
deactivate
```

Installation de PyQt via Conda

- **Créer un environnement :**

```
conda create --name myenv
```

- **Activer l'environnement :**

```
conda activate myenv
```

- **Installer PyQt :**

```
conda install pyqt
```

- **Lister les paquets installés :**

```
conda list
```

- **Désactiver l'environnement :**

```
conda deactivate
```

- **Supprimer l'environnement :**

```
conda remove --name myenv --all
```

Premier programme en PyQt

- Voici un exemple simple de création d'une fenêtre avec PyQt.

```
1  import sys
2  from PyQt6.QtWidgets import QApplication, QWidget
3
4  app = QApplication(sys.argv)
5
6  window = QWidget()
7  window.setWindowTitle('Hello, PyQt!')
8  window.setGeometry(100, 100, 280, 80)
9  window.show()
10
11 sys.exit(app.exec())
```

Introduction à PyQt

- PyQt est un binding Python pour la bibliothèque Qt, utilisée pour développer des interfaces graphiques.
- Qt est une bibliothèque C++ largement utilisée pour le développement d'applications avec des interfaces utilisateur riches.
- PyQt permet de créer des applications GUI multiplateformes en utilisant Python.
- Il existe plusieurs versions de PyQt correspondant aux versions de Qt (par exemple, PyQt5, PyQt6).

La notion de package PyQt

- **PyQt5** et **PyQt6** sont les versions les plus courantes.
- **PyQt5** : Correspond à Qt5, disponible pour Python 2 et Python 3.
- **PyQt6** : Correspond à Qt6, uniquement disponible pour Python 3.
- **PySide** : l'alternative libre à PyQt, maintenue par The Qt Company.

Les modules de base de PyQt

- **QtWidgets** : Contient les widgets et les contrôles de l'interface utilisateur.
- **QtGui** : Fournit des classes pour la gestion des graphiques 2D, des images et des polices.
- **QtCore** : Contient les classes de base telles que les timers, les threads, et les événements.
- **QtNetwork** : Fournit des classes pour les communications réseau.
- **QtSql** : Permet l'intégration des bases de données SQL.
- **QtMultimedia** : Gestion des médias (audio et vidéo).

Les classes de base de PyQt

- **QApplication** : Gère l'application entière, initialisation et gestion des ressources.
- **QMainWindow** : Fournit une fenêtre principale avec une barre de menus, une barre d'outils et une zone centrale.
- **QWidget** : Classe de base pour tous les objets d'interface utilisateur.
- **QPushButton** : Représente un bouton cliquable.
- **QLabel** : Affiche du texte ou des images statiques.
- **QLineEdit** : Permet la saisie de texte sur une seule ligne.
- **QTextEdit** : Permet la saisie de texte multi-lignes avec formatage.
- **QVBoxLayout** : Disposition verticale des widgets.
- **QHBoxLayout** : Disposition horizontale des widgets.
- **QFormLayout** : Disposition en forme de formulaire, souvent utilisée pour les entrées utilisateur.
- **QGraphicsScene** : Fournit une surface pour dessiner des objets graphiques, utilisée avec **QGraphicsView** pour des affichages graphiques avancés.

Concepts de PyQt

- **Widgets** : Composants de l'interface utilisateur comme boutons, labels, et champs de texte.
- **Layouts** : Gestion de la disposition des widgets dans les fenêtres.
- **Signaux et Slots** : Mécanisme pour la communication entre les objets et la gestion des événements.
- **Événements** : Gestion des interactions utilisateur, comme les clics de souris et les frappes clavier.
- **Styles** : Personnalisation de l'apparence des widgets à l'aide de feuilles de style.

Introduction à l'Event Loop

- L'**Event Loop** est un mécanisme central dans la programmation événementielle, particulièrement utilisé dans les applications graphiques.
- Dans PyQt, l'Event Loop gère les événements, tels que les clics de souris, les touches de clavier, ou les actions déclenchées par l'utilisateur.
- Il fonctionne en écoutant les événements, en les plaçant dans une file d'attente, puis en les distribuant aux widgets ou composants appropriés.

Le Fonctionnement de l'Event Loop

- Lorsque l'application PyQt est démarrée, l'Event Loop commence à fonctionner en appelant la méthode `app.exec()`.
- Cette boucle d'événements continue de tourner jusqu'à ce que `app.quit()` soit appelé.
- Pendant son exécution, l'Event Loop récupère les événements dans la file d'attente et les distribue aux widgets concernés.
- Chaque widget dans PyQt peut réagir à ces événements grâce aux signaux et slots, ou en surchargeant des méthodes d'événements spécifiques.

Les Signaux et Slots dans l'Event Loop

- Les `signaux` et `slots` sont des mécanismes de communication entre objets dans PyQt.
- Un signal est émis lorsqu'un événement particulier se produit, par exemple, un bouton est cliqué.
- Un slot est une fonction ou une méthode qui est appelée en réponse à ce signal.
- L'Event Loop écoute les signaux et appelle les slots correspondants, permettant aux objets de réagir aux événements en temps réel.

Gestion des Événements Personnalisés

- En plus des événements standards, PyQt permet de créer et de gérer des événements personnalisés.
- Ces événements peuvent être utilisés pour des actions spécifiques à l'application qui ne sont pas couvertes par les événements standards.
- Les événements personnalisés sont généralement dérivés de la classe `QEvent`.
- L'Event Loop peut être configuré pour écouter et gérer ces événements en surchargeant la méthode `event()` d'un widget.

La Méthode `exec()` et le Bouclage d'Événements

- La méthode `exec()` démarre l'Event Loop en entrant dans un état de bouclage constant, où l'application attend des événements.
- Pendant ce bouclage, l'application devient réactive et peut interagir avec l'utilisateur.
- La méthode `exec()` est souvent appelée après avoir configuré tous les widgets et signaux/slots de l'application.
- Elle ne retourne qu'une fois que l'Event Loop est terminé, généralement par un appel à `app.quit()`.

Terminaison de l'Event Loop

- L'Event Loop se termine lorsque la méthode `app.quit()` est appelée.
- Cela peut être déclenché par une action utilisateur (comme fermer la fenêtre principale) ou par le code (comme après un certain événement).
- La fin de l'Event Loop entraîne la fermeture de l'application, en nettoyant tous les widgets et en libérant les ressources utilisées.
- C'est un processus critique qui doit être géré avec soin pour éviter les fuites de mémoire et d'autres problèmes liés aux ressources.

Exemples de Scénarios d'Utilisation

- **Traitement en arrière-plan** : L'Event Loop peut être utilisé pour gérer des tâches longues en arrière-plan sans bloquer l'interface utilisateur.
- **Interactions utilisateur** : Gérer les clics de boutons, les sélections dans les listes, et les entrées de texte en temps réel.
- **Mises à jour dynamiques** : L'Event Loop permet de mettre à jour l'interface utilisateur en fonction des données entrantes, comme des flux en temps réel.

Bonnes Pratiques pour l'Utilisation de l'Event Loop

- **Éviter les blocages** : Ne pas exécuter de tâches longues directement dans l'Event Loop, utiliser des threads ou des timers.
- **Gérer correctement la terminaison** : S'assurer que tous les événements sont bien traités avant de quitter l'Event Loop.
- **Optimiser les performances** : Minimiser le travail dans chaque itération de l'Event Loop pour garder l'interface utilisateur fluide et réactive.
- **Utiliser les signaux et slots** : Tirer parti de ce mécanisme pour une gestion efficace des événements.

Conclusion et Importance de l'Event Loop

- L'Event Loop est le cœur de toute application PyQt, assurant la gestion des événements et la réactivité de l'interface utilisateur.
- Une compréhension approfondie de l'Event Loop est essentielle pour développer des applications performantes et fiables.
- En appliquant les bonnes pratiques, on peut éviter les blocages et maximiser la réactivité de l'application.