Exercice : Transmission de données de température via TCP et sockets en Python

Objectif

Créer une application client-serveur en Python utilisant les sockets TCP pour transmettre des données de température au format JSON. Le serveur devra sauvegarder les données reçues soit dans un fichier, soit dans une base de données (au choix).

Consignes

1. Côté Client

- Écrire un script Python qui :
  - Se connecte à un serveur via TCP (adresse et port à définir).
  - Génère ou saisit une température (par exemple, un nombre flottant).
  - Prépare un message JSON contenant la température et un identifiant de capteur.
  - Envoie ce message au serveur.

2. Côté Serveur

- Écrire un script Python qui :
  - Écoute sur un port TCP.
  - Accepte les connexions entrantes.
  - Reçoit les messages JSON envoyés par les clients.
  - Parse les données JSON pour extraire la température et l'identifiant.
  - Sauvegarde chaque donnée reçue :
    - soit dans un fichier texte (par exemple, `donnees.txt`), chaque ligne correspondant à une mesure,
    - soit dans une base de données SQLite (table avec au moins les colonnes : identifiant, température, timestamp).

Exemple de message JSON

```
{
    "capteur_id": "capteur_1",
    "temperature": 23.5
}
```

Livrables

- `client.py`

- `serveur.py`
- Fichier de sauvegarde ou base de données

Extension : Programmation Orientée Objet (POO)

Réalisez une version orientée objet de l'application :

Par exemple

- **Côté Client** :
  - Créez une classe `CapteurTCP` qui gère la connexion, la génération et l'envoi des données.
  - La méthode principale doit permettre d'envoyer une mesure de température.
- **Côté Serveur** :
  - Créez une classe `ServeurTCP` qui gère l'écoute, la réception et la sauvegarde des données.
  - La méthode principale doit permettre de lancer le serveur et de traiter les connexions entrantes.

Livrables supplémentaires :

- `client_poo.py`
- `serveur_poo.py`
- Documentation rapide des classes et méthodes utilisées.

Héritage

Travail demandé

Proposez une extension de votre application orientée objet en utilisant l'héritage :

Par exemple

- **Côté Client** :
  - Créez une classe de base `Capteur` qui contient les attributs et méthodes communs à différents types de capteurs (par exemple, identifiant, génération de mesure).
  - Créez une classe dérivée `CapteurTCP` qui hérite de `Capteur` et ajoute les fonctionnalités liées à la connexion et à l'envoi via TCP.
- **Côté Serveur** :
  - Créez une classe de base `Serveur` qui gère les fonctionnalités générales d'un serveur (écoute, gestion des connexions).
  - Créez une classe dérivée `ServeurTCP` qui hérite de `Serveur` et ajoute la gestion spécifique des messages JSON et de la sauvegarde des données.

Expliquez brièvement l'intérêt de l'héritage dans ce contexte et documentez les classes créées.

Extension : Gestion des threads et utilisation des classes

Objectif

Adapter votre application orientée objet pour permettre la gestion simultanée de plusieurs connexions ou tâches grâce à l'utilisation des threads en Python.

Consignes

- **Côté Serveur** :
    - Modifiez la classe `ServeurTCP` pour qu'elle puisse gérer plusieurs clients en parallèle, chaque connexion étant traitée dans un thread distinct.
    - Utilisez le module `threading` de Python pour lancer un nouveau thread à chaque connexion entrante.
    - Assurez-vous que la sauvegarde des données est thread-safe (utilisez des verrous si nécessaire).
- **Côté Client** :
    - (Optionnel) Permettez l'envoi de plusieurs mesures en parallèle en utilisant des threads, par exemple pour simuler plusieurs capteurs.

Programme principal

- Créez un programme principal (`main.py` ou intégré dans vos scripts existants) qui instancie les classes et lance les threads nécessaires.
- Documentez la structure de votre programme principal et expliquez comment les threads sont créés et gérés.

Livrables supplémentaires

- Code source modifié avec gestion des threads.
- Documentation rapide expliquant l'organisation des classes et la gestion des threads.
