Ressource : https://docs.python.org/fr/3/library/sqlite3.html
Dans la suite, on utilise des objets du module `sqlite3` et leurs méthodes.
Vous manipulez déjà des objets en Python avec le type `list` (et même d'autres types).
Le mode de fabrication d'un objet est défini dans une `class`.
La classe contient un constructeur (une fonction) qui définit les attributs que possède l'objet.
La classe contient des méthodes (des fonctions) qui définissent des actions que peut exécuter cet objet.
Pour les listes Python, `list()` est un constructeur. `append()`, `pop()` sont des méthodes de cette objet.
Vous aurez une explication plus précise prochainement.

I Créer et connecter une base de données dans un programme

```
import sqlite3

def creer_connexion(db_file):
    """ cree une connexion a la base de donnees SQLite
        specifiee par db_file
    parametre db_file: fichier BD
    return: objet connexion ou None
    """
    conn = sqlite3.connect(db_file)
    #On active les foreign keys
    conn.execute("PRAGMA foreign_keys = 1")
    return conn

def main():
    # Definition du nom de la base
    database = "mabase.db"

    # creer une connexion a la BD (si le fichier mabase.db n'existe pas, il est cree)
    conn = creer_connexion(database)

    # fermeture de la connexion
    conn.close()
```

- conn est un objet (une instance) de la classe connexion, il est construit par l'appel au constructeur `connect`
- `execute()`, `close()` sont des méthodes de la classe connexion.

II Exécuter un un fichier SQL

```
def majBD(conn, file):
    """ execute dans la base de donnee les requetes contenues
    dans le fichier file
    parametres :
        conn : l'objet connexion a la base de donnees
        file : nom du fichier contenant les requetes
    return : None
    """
    # Lecture du fichier et placement des requetes dans un tableau
    fichier = open(file, 'r')
    chaine = fichier.read()
    fichier.close()
    requetes = createSql.split(";")

    # Execution de toutes les requetes du tableau
    cursor = conn.cursor()
```

```
    for req in requetes:
        cursor.execute(req)

    # commit des modifications
    conn.commit()
```

- Les 4 premières lignes permettent de lire le contenu du fichier et de créer la liste `requetes` des requêtes contenu dans le fichier et séparées par un « ; ».
- Pour exécuter une requête, on a besoin d'un curseur de la base de données.
- On parcourt ensuite la liste et l'on exécute chaque requête `cursor.execute(req)`
- La dernière ligne permet de s'assurer que le fichier `mabase.db` a bien été modifié.

III Interroger la base et afficher le résultat des requêtes

```
def select_tous_animaux(conn):
    """
    parametre conn: objet connexion
    return:
    """
    cur = conn.cursor()
    cur.execute("SELECT * FROM LesAnimaux")

    liste = cur.fetchall()

    for elt in liste:
        print(elt)
```

- `fetchall()` renvoie la liste des tuples résultats de la requête.

IV Programme principal

```
def main():
    database = "zoo.db"

    # creer une connexion a la BD
    conn = creer_connexion(database)

    # remplir la BD
    print("1. On cree LesAnimaux zoo et on les initialise avec des premieres valeurs.")
    majBD(conn, "zoo.sql")

    # lire la BD
    print("2. Liste de tous les animaux")
    select_tous_animaux(conn)

    # commit des modifications eventuelles
    conn.commit()

    # fermeture de la connexion
    conn.close()
```

V Se méfier des entrées utilisateur

1 Une requête qui demande à l'utilisateur une entrée

Sur le web, de nombreuses fonctionnalités nécessitent l'authentification d'un utilisateur. L'utilisateur est alors invité à saisir son identifiant et son mot de passe dans une formulaire. Les données sont alors envoyées au serveur via la méthode POST associées aux clé `login` et `password` et comparées au contenu de la base de données contenant les utilisateurs.
La chaine de la requête SQL peut par exemple être :

```
"SELECT *
FROM Utilisateurs
WHERE Login = '"+resultat['login']+"' AND Password = '"+resultat['password']+"'"
```

pour obtenir le nom et le prénom de l'utilisateur.
Dans la suite, on utiliser un mini site qui est relié à une base de données qui contient une table Utilisateurs et les données ci-dessous :

Login	Password	Nom	Prénom	Privilèges
LukeS	Luke5	Skywalker	Luke	user
LeiaS	Le1a5	Skywalker	Leïa	admin
Ackbar	Am1ral	Ackbar	Gial	user
admin	@dmin	Skywalker	Anakin	admin
AnakS	@n@k1n	Skywalker	Anakin	user

2 Un danger, l'injection SQL (et hors de ce cours c'est interdit de le faire)

Pour la suite, vous allez utiliser un site web local avec un serveur Python Flask.
Dans une console, sélectionner le répertoire `Site_injection` comme répertoire courant et démarrer le site avec la commande `python3 run.py` (attention, la bibliothèque Flask doit être installée).
Ouvrir ensuite un navigateur et saisir dans la barre d'adresse l'adresse qui a été donnée dans la console.

a Tester le fonctionnement

Tester le bon fonctionnement en choisissant le login et le mot de passe d'un utilisateur de la base de données puis avec un mot de passe erroné.

b Première tentative d'injection

Tester maintenant en entrant en login :
'OR 1=1 - - (sans espace entre les deux tirets)
Quel est l'effet ?

En utilisant la requête du **1.**, Qu'elle est la chaine qui a été transmise et a été sollicitée au SGBD avec un tel argument.

A quelle requête simplifiée est-elle équivalente ?

Pourquoi renvoie-t-elle le résultat obtenu ?

c Deuxième tentative pour viser un utilisateur particulier

On peut varier l'injection si l'on connait le login d'un utilisateur.
Tester maintenant en entrant en login :
admin' - - (sans espace entre les deux tirets)
Quel est l'effet ?

3 Se prémunir de l'injection SQL avec les requêtes "paramétrées"

Depuis 20 ans que ce danger est apparu, les liaisons vers les SGBD ont été sécurisées avec des requête paramétrées.
Pour sqlite, la méthode execute peut prendre 2 paramètres :

- Une chaine dans laquelle les « valeurs attendues » seront remplacées par des point d'interrogation « ? »
- Un tuple correspondant au nombre de point d'interrogation dans la chine de la requête.

Ainsi, les paramètres sont pris comme tels et ne modifient pas la structure de la chaine.

1. En console, arrêter le serveur Flask avec Ĉ (control C)
2. Dans le répertoire Site_injection/Projet/modele, ouvrir avec un éditeur de texte le fichier `model.py`.
   (a) Dans la fonction `db_identification` remplacer la ligne
   `chaine = ...`
   par
   `chaine = "SELECT * FROM Utilisateurs WHERE Login = ? AND Password = ?"`
   (b) Remplacer
   `cur.execute(chaine)`
   par
   `cur.execute(chaine,(resultat['login'], resultat['password']))`
3. Sauvegarder les modifications.
4. En console, redémarrer le serveur avec `python3 run.py`
5. Tester avec un utilisateur existant
6. Tester avec une tentative d'injection SQL