PROGRAMMATION PYTHON (Notions Avancées)

Par LOYEM Anderson

A Qui S'adresse Cette Formation ?

Prerequis essentiel : **BASES DE LA PROGRAMMATION AVEC PYTHON**

PLAN

GESTION DES EXCEPTIONS

DATE ET HEURE

GESTION DES FICHIERS

CONNECTIVITÉ BASES DE DONNÉES

Gestion des Exceptions

Gestion des Exceptions

- Une **exception** peut être définie comme une condition inhabituelle dans un programme entraînant une interruption du déroulement du programme.

- Chaque fois qu'une exception se produit, le programme arrête l'exécution, et ainsi le code supplémentaire n'est pas exécuté. Par conséquent, les erreurs d'exécution qui ne peuvent pas gérer le script Python constituent une exception. Une exception est un objet Python qui représente une erreur

- Python fournit un moyen de gérer l'exception afin que le code puisse être exécuté sans aucune interruption. Si nous ne gérons pas l'exception, l'interpréteur n'exécute pas tout le code qui existe après l'exception.

- **Erreurs de Syntaxe**
- **Erreurs logiques (Exceptions)**

Gestion des Exceptions

- **ZeroDivisionError:** Se produit lorsqu'un nombre est divisé par zéro.
- **NameError:** Cela se produit lorsqu'un nom n'est pas trouvé. Cela peut être local ou global.
- **IndentationError:** Si une indentation incorrecte est donnée.
- **IOError:** Cela se produit lorsque l'opération d'entrée-sortie échoue.
- **EOFError:** Cela se produit lorsque la fin du fichier est atteinte et que des opérations sont en cours d'exécution.

Gestion des Exceptions

```
a = int(input("Entrez a:")) # a : 2

b = int(input("Entrez b:"))  # b : 0

c = a/b   # 2/0

print("a/b = %d" %c)     #ZeroDivisionError:

division by zero
```

```
#autre code:

print("Voici la suite de mon programme.")
```

```
try:

    a = int(input("Entrez a:"))

    b = int(input("Entrez b:"))

    c = a/b

except:

    print("Pas de division par zéro")
```

TP1

Ecrire un programme python qui effectue la division entre 2 nombres et lève les exceptions `ArithmeticError, NameError.`

Gestion des Exceptions

A Retenir

- Python nous permet de ne pas spécifier l'exception avec l'instruction **except.**
- Nous pouvons déclarer plusieurs exceptions dans l'instruction except car le bloc try peut contenir les instructions qui lèvent les différents types d'exceptions.
- Nous pouvons également spécifier un bloc **else** avec l'instruction try-except, qui sera exécutée si aucune exception n'est déclenchée dans le bloc try.
- Les instructions qui ne lèvent pas l'exception doivent être placées dans le bloc **else.**

Gestion des Exceptions

```
try:
    # code

except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>) :
    # code

else:
    # code

# code
```

```
try:
    # code

except Exception 1:
    # code

except Exception 2:
    # code

else:
    # code
```

Gestion des Exceptions(try… finally)

- Python fournit l'instruction **finally** facultative, qui est utilisée avec l'instruction **try**. Elle est exécuté quelle que soit l'exception qui se produit, aussi utilisé pour libérer la ressource externe.
- Le bloc **finally** fournit une garantie de l'exécution.

```
try:
    fileptr = open("file2.txt","r")
    try:
        fileptr.write("Hi I am good")
    finally:
        fileptr.close()
        print("file closed")
except:
    print("Error")
```

TP2

Ajouter un bloc `finally` au TP1.

Gestion des Exceptions(Custom Exceptions)

- Python nous permet de créer nos **exceptions** qui peuvent être déclenchées à partir du programme et interceptées à l'aide de la clause except.

- Une exception définie par l'utilisateur hérite de la classe Exception.

- Lorsque nous développons un grand programme Python, il est recommandé de placer toutes les exceptions définies par l'utilisateur que notre programme déclenche dans un fichier séparé. De nombreux modules standard le font. Ils définissent leurs exceptions séparément comme **exceptions.py** ou **errors.py** (généralement mais pas toujours).

Gestion des Exceptions(Custom Exceptions)

```
class ErreurCode(Exception):
    def __init__(self, data):
        self.data = data


try:
    raise ErreurCode(2000)
except ErreurCode as ae:
    print("Received error:", ae.data)
```

Gestion des Exceptions (Custom Exceptions)

Exemple d'un programme permettant de deviner un nombre spécifier dans le code et retourner une erreur si le nombre est plus grand ou plus petit.

```
class Erreur(Exception):
    """ class de base pour les autres erreurs"""
    pass

class ValueTropPetiteErreur(Erreur):
    """survient quand la valeur entre est petite"""
    pass

class ValueTropLargeErreur(Erreur):
    """survient quand la valeur entre est grande"""
    pass
```

```
number = 10
while True:
    try:
        i_num = int(input("Entrez un nombre : "))
        if i_num < number :
            raise ValueTropPetiteErreur
        elif i_num > number:
            raise ValueTropLargeErreur
        break
    except ValueTropPetiteErreur:
        print("cette valeur est petite, essaye
encore")
        print()
    except ValueTropLargeErreur:
        print("cette valeur est grande, essaye
encore")
        print()
print("félicitation! vous avez trouvé le nombre
correct")
```

TP3

Ecrire un programme qui permet d'enregistrer les Joueuses d'une équipe Professionnelle de football féminin. Une **Joueuse** est Représentée par son **nom, age, sexe, Numero.** Réalisez les tâches suivante:

- Créez la classe **Joueur,** ayant un constructeur a plusieurs parametre.
- Créez un module **erreurs.py** qui aura les classes **Erreur , ErreurMinAge, ErreurMaxAge, ErreurSexe**
  - **Erreur :** classe d'exception de base pour les autres classes (Hérite de la classe Exception)
  - **ErreurMinAge :** est évoqué lorsque l'âge du joueur est inférieur à **18**
  - **ErreurMAxAge:** est évoqué lorsque l'âge du joueur est supérieur à **35**
  - **ErreurSexe :** est évoqué lorsque le sexe mentionner est different de « F » ou « f »

dans le fichier **main.py**, créez des objets de la classe joueur et le code qui permettra de verifier que l'age du joueur est **>=18 et <35** que le sexe est « F » ou « f » car il s'agit d'une equipe feminine; Utilisez **Exception as e** pour recuper les autres type d'erreurs (erreur imbrique dans python) pouvant apparaitre a l'execution du programme.

DATE & TIME

DATE & TIME

Python permet au module **datetime** de fonctionner avec des **dates et des heures** réelles. Dans les applications du monde réel, nous devons travailler avec la date et l'heure. Python nous permet de programmer notre script pour qu'il s'exécute à un moment donné.

En Python, la date n'est pas **un type de données**, mais nous pouvons travailler avec les objets de date en important le module nommé **datetime**, **date**, **time** et **calendar.**

DATE & TIME

Les classes de **datetime** sont classées parmi les six classes principales.

- **date** - C'est une date idéale naïve. Elle est compose de l'année, du mois et du jour en tant que attributs.
- **time** - C'est un moment parfait, en supposant que chaque jour compte précisément **24 * 60 * 60 secondes**. Elle a **heure, minute, seconde, microseconde et tzinfo** comme attributs.
- **datetime** - Il s'agit d'un regroupement de **date et d'heure**, ainsi que les attributs **année, mois, jour, heure, minute, seconde, microseconde et tzinfo.**
- **timedelta** - Elle représente la différence entre deux instances de date, heure ou datetime avec une résolution de l'ordre de la microseconde.
- **tzinfo** - fournit des objets d'information sur le fuseau horaire.
- **timezone** - C'est la classe qui implémente la classe de base abstraite tzinfo.

DATE & TIME (strftime())

- La méthode **strftime ()** renvoie une chaîne représentant la date et l'heure en utilisant un objet **date, heure ou datetime.**

```
from datetime import datetime
now = datetime.now() # date et temps courantes
date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
print("date et temps : ", date_time) #    date et temps : 12/24/2018, 04:59:31
```

DATE & TIME (strftime() & strptime())

Directives	Sens	Exemple
%a	Nom abrégé du jour de la semaine.	Sun, Mon, ...
%A	Nom complet du jour de la semaine.	Sunday, Monday, ...
%b	Nom du mois abrégé.	Jan, Feb, ..., Dec
%B	Nom complet du mois.	January, February, ...
%w	Jour de la semaine sous forme de nombre décimal.	0, 1, ..., 6
%c	Représentation appropriée de la date et de l'heure.	Mon Sep 30 07:06:05 2013
%x	Représentation de date appropriée de la langue locale.	09/30/13
%X	Représentation temporelle appropriée des paramètres régionaux.	07:06:05

DATE & TIME (strptime())

La méthode **strptime()** crée un objet **datetime** à partir de la chaîne de caractères donnée.

Remarque: Vous ne pouvez pas créer d'objet datetime à partir de chaque chaîne**. La chaîne doit être dans un certain format.**

Ex: “ 01 Jan, 2020 ” , “10 MAI, 2010”

"12/11/2018 09:15:32", etc...

```
from datetime import datetime


date_string = "21 June, 2018"

print("date_string =", date_string)

print("type de date_string =", type(date_string) )


date_object = datetime.strptime(date_string, "%d %B, %Y")

print("date_object =", date_object)

print("type de date_object =", type(date_object) )
```

DATE & TIME

```
# imprimer la date courante
import datetime
date_ = datetime.datetime.today().strftime('%Y-%m-%d %H:%M:%S')
print(date_)
```

```
# effectuer la différence entre deux date timedelta

from datetime import datetime, timedelta

ini_time_for_now = datetime.now()
print ("initial_date", str(ini_time_for_now))

future_date_after_2yrs = ini_time_for_now + timedelta(days = 730)

future_date_after_2days = ini_time_for_now + timedelta(days = 2)

print('future_date_after_2yrs:', str(future_date_after_2yrs))
print('future_date_after_2days:', str(future_date_after_2days))
```

DATE & TIME

```
 # imprimer le calendrier de 2023

import calendar
year = 2023
print(calendar.calendar(year))
```

```
# timezone
# imprimer la date dans un fuseau horaire précis London

import pytz
from datetime import datetime, timezone

utc_dt = datetime.now(timezone.utc)

paris = pytz.timezone("Europe/Paris")
london = pytz.timezone("Europe/London")

print("Paris time {}".format(utc_dt.astimezone(paris).isoformat()))
print("London time {}".format(utc_dt.astimezone(london).isoformat()))
```

DATE & TIME (TP4)

créer une **horloge numérique** qui retourne à un instant **t** l'heure des fuseaux horaire (**Africa/Douala**) et celle de (**Europe/Paris**).

- Retournez l'heure au format **Heure:Minute:Seconde**
- Les 02 résultats doivent être affichés sur la même ligne. **Ex:**

  Heure Douala : 23:58:07 <--> Heure Paris : 00:58:07
- Utilisez **flush** pour effacer le contenu de l'écran , puis '**\r**' pour revenir a la ligne précédente.
- Ne pas afficher le résultat sur une autre ligne lorsque l'heure augmente d'une seconde. Tout doit être affiché sur la même ligne. **Ex:**

  Heure Douala : 23:58:08 <--> Heure Paris : 00:58:08 doit remplacer le résultat précédent sur la même ligne et ainsi de suite.
- Suspendre l'exécution du programme pendant 1 seconde avant d'afficher le prochain résultat

Gestion des Fichiers

Gestion des Fichiers

Les **opérations** sont effectuées sur les fichiers selon l'ordre suivant:

1. **Ouvrir un fichier**
2. **Lecture ou écriture (Exécution d'une opération)**
3. **Fermez le fichier**

```
fichier1 = open("fichier.txt","r")

if fichier1:

   print("ouvert avec succès")

fichier1.readlines()

fichier1.close() # fermez le fichier
```

Gestion des Fichiers

fichier_object = open(<nom_fichier>, <mode_access>, <buffering>)

	Mode d'accès	Description
1	**r**	Il ouvre le fichier en mode lecture seule. Le pointeur de fichier existe au début. Le fichier est par défaut ouvert dans ce mode si aucun mode d'accès n'est passé.
2	**r+**	Il ouvre le fichier pour lire et écrire les deux. Le pointeur de fichier existe au début du fichier.
3	**w**	Il ouvre le fichier en écriture uniquement. Il écrase le fichier s'il existe précédemment ou en crée un nouveau s'il n'existe aucun fichier portant le même nom. Le pointeur de fichier existe au début du fichier.
4	**w+**	Il ouvre le fichier pour écrire et lire les deux. Il est différent de w +en ce sens qu'il écrase le fichier précédent s'il en existe un alors que r +n'écrase pas le fichier précédemment écrit. Il crée un nouveau fichier si aucun fichier n'existe. Le pointeur de fichier existe au début du fichier.
5	**a**	Il ouvre le fichier en mode ajout. Le pointeur de fichier existe à la fin du fichier précédemment écrit s'il en existe. Il crée un nouveau fichier s'il n'existe aucun fichier portant le même nom.

Gestion des Fichiers

with open(<nomfichier>, <accessmode>) as <pointeurfichier>:

6	**a+**	Il ouvre un fichier pour ajouter et lire. Le pointeur de fichier reste à la fin du fichier si un fichier existe. Il crée un nouveau fichier s'il n'existe aucun fichier portant le même nom.
7	**rb**	Il ouvre le fichier en lecture seule au format binaire. Le pointeur de fichier existe au début du fichier.
8	**rb+**	Il ouvre le fichier pour lire et écrire à la fois au format binaire. Le pointeur de fichier existe au début du fichier.
9	**wb**	Il ouvre le fichier pour écrire uniquement au format binaire. Il écrase le fichier s'il existe précédemment ou en crée un nouveau si aucun fichier n'existe. Le pointeur de fichier existe au début du fichier.
10	**wb+**	Il ouvre le fichier pour écrire et lire les deux au format binaire. Le pointeur de fichier existe au début du fichier.
11	**ab**	Il ouvre le fichier en mode ajout au format binaire. Le pointeur existe à la fin du fichier précédemment écrit. Il crée un nouveau fichier au format binaire s'il n'existe aucun fichier portant le même nom.

Gestion des Fichiers

```
# écrire dans fichier
if __name__=='__main__':
    fichier1 = open("file.txt", "w+")

    if fichier1 :
        print("votre fichier est ouvert")
        fichier1.write("test")
    fichier1.close()
```

```
 # lire le contenu d'un fichier
 f = open("file.txt", "r")
 print(f.read())
 print(f.readline())
```

```
# Loading a JSON File to a Python Dictionary
# my-file.json content {}
import json

with open('my-file.json', 'r') as file:
    data = json.loads(file)

print(data)
```

Gestion des Fichiers (OS Module)

Le module **Python OS** permet une interaction avec le système d'exploitation.

Le module os fournit les fonctions impliquées dans les opérations de traitement de fichiers comme le changement de nom, la suppression, etc.

Il nous fournit la méthode rename () , remove(), mkdir(), getcwd() etc.

La méthode **check_call ()** du module **subprocess** est utilisée pour exécuter un script Python et écrire la sortie de ce script dans un fichier.

Gestion des Fichiers

Quelques Méthodes Du module OS :

```
import os

os.getcwd()  # retourne le repertoire courant
os.chdir() # permet de changer de repertoire

os.listdir() # retourne la liste de tout le contenu d'un repertoire

os.mkdir() # cree un nouveau repertoire

os.rename() # renome un fichier

os.remove() # supprimer un fichier

os.rmdir() # supprimer un repertoire vide
```

```
import subprocess


with open("output.txt", "w") as f:

    subprocess.check_call(["python",
"file.py"], stdout=f)
```

Gestion des Fichiers (TP5)

Créez un programme qui prend le **nom de l'utilisateur**, effectue des **opérations arithmétiques (addition)** de base puis retourne une trace des opérations effectuées dans un **fichier historique.txt** de la sorte **: nom user, operation, date et temps.** le fichier historique.txt doit se trouver dans un répertoire **logs** créé dans le programme.

Ex : nom-utilisateur : **Toto**

opération : **1 + 2 = 3**

date et heure : **2021-04-20 12:36**

****

nom-utilisateur : **Ali**

opération : **4 + 2 = 6**

date et heure : **2021-05-10 11:01:23**

****

BASES DE DONNÉES AVEC PYTHON

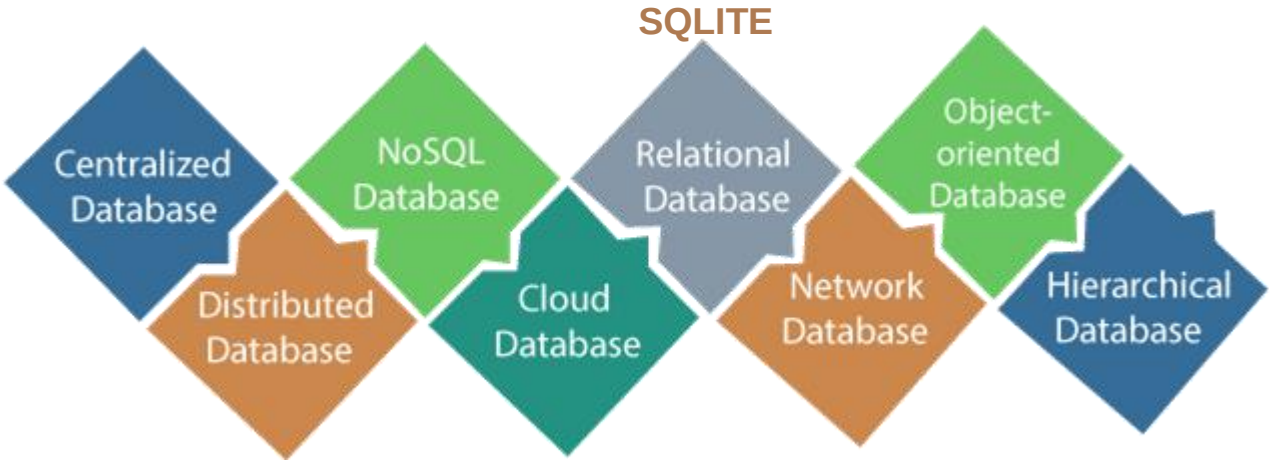
BASES DE DONNÉES AVEC PYTHON

Une Donnée est une collection d'une petite unité d'information distincte. Elle peut être utilisée sous diverses formes telles que du texte, des nombres, des supports, des octets, etc., elle peut être stockée dans des morceaux de papier ou de mémoire électronique, etc.

- Une **base de données** est une collection organisée de données, de sorte qu'elle peut être facilement accessible et gérable.
- On peut organiser les données en tables, lignes, colonnes et les indexer pour faciliter la recherche d'informations pertinentes.
- **L'objectif principal** de la base de données est d'exploiter une grande quantité d'informations en stockant, en récupérant et en gérant des données.
- **Les gestionnaires de bases de données** créent une base de données de telle sorte qu'un seul ensemble de logiciels donne accès aux données à tous les utilisateurs.

BASES DE DONNÉES AVEC PYTHON

TYPES DE BASES DE DONNÉES



BASES DE DONNÉES AVEC PYTHON

SGBD	Système de Fichiers
Le SGBD est une collection de données. Dans le SGBD, l'utilisateur n'est pas obligé d'écrire les procédures.	Le système de fichiers est une collection de données. Dans ce système, l'utilisateur doit rédiger les procédures de gestion de la base de données.
Le SGBD donne une vue abstraite des données qui cache les détails.	Le système de fichiers fournit le détail de la représentation des données et du stockage des données.
Le SGBD fournit un mécanisme de reprise après incident, c'est-à-dire qu'il protège l'utilisateur contre la défaillance du système.	Le système de fichiers n'a pas de mécanisme de plantage, c'est-à-dire que si le système se bloque lors de la saisie de certaines données, le contenu du fichier sera perdu.
Le SGBD fournit un bon mécanisme de protection.	Il est très difficile de protéger un fichier sous le système de fichiers.
Le SGBD contient une grande variété de techniques sophistiquées pour stocker et récupérer les données.	Le système de fichiers ne peut pas stocker et récupérer efficacement les données.
Le SGBD prend en charge l'accès simultané aux données en utilisant une forme de verrouillage.	Dans le système de fichiers, l'accès simultané présente de nombreux problèmes tels que la redirection du fichier tandis que d'autres suppriment certaines informations ou mettent à jour certaines informations.

BASES DE DONNÉES AVEC PYTHON

SGBD Relationnelle

Tous les systèmes de gestion de base de données modernes tels que **SQL, MS SQL Server, IBM DB2, ORACLE, My-SQL et Microsoft Access** sont basés sur **SGBDR**.

Il est appelé système de gestion de base de données relationnelle (**SGBDR**) car il est basé sur un modèle relationnel introduit par **E.F. Codd.**

Les données sont représentées en termes de tuples (**lignes**) dans le **SGBDR**.

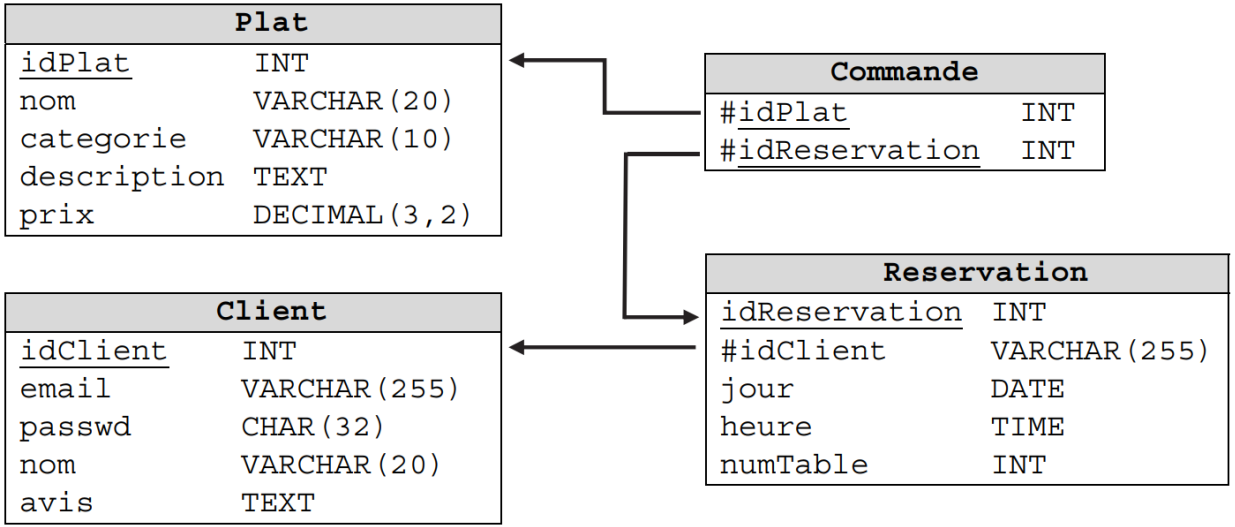
La base de données relationnelle est la base de données la plus couramment utilisée. Elle contient un certain nombre de tables et chaque table a sa propre **clé primaire**.

En raison d'un ensemble de tableaux organisés, les données sont facilement accessibles dans le **SGBDR**, grâce au **SQL**.

SQL est l'acronyme de Structured Query Language. Il est utilisé pour stocker et gérer des données dans un système de gestion de base de données relationnelle (**SGBDR**).

BASES DE DONNÉES AVEC PYTHON

SGBD Relationnelle



BASES DE DONNÉES AVEC PYTHON

Langage de définition des données

Create
Alter
Drop
Truncate
Rename
Comment

Langage de manipulation des données

Select
Insert
Update
Delete
Lock Table

Langue de contrôle des données

Grant
Revoke

Langue de contrôle des transactions

Commit
Rollback

BASES DE DONNÉES AVEC PYTHON

CAS PRATIQUE

BASES DE DONNÉES AVEC PYTHON

sqlite

- Télécharger sqlite studio et installer https://sqlitestudio.pl/
- Création d'une base de données
- Connexion à une BD en python
- Insertion
- Modification
- Suppression
- Lecture

BASES DE DONNÉES AVEC PYTHON

```
# creation d'une base de données et ajout d'une table

import sqlite3

connexion = sqlite3.connect("school.db")
curseur = connexion.cursor()

###Exécution unique
curseur.execute('''CREATE TABLE IF NOT EXISTS students(
                id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
                pseudo TEXT,
                valeur INTEGER
                )'''
            )
curseur.close()
```

BASES DE DONNÉES AVEC PYTHON

```
## Création de deux tables avec une clé étrangère
curseur.execute('''CREATE TABLE classroom (
                    id   INTEGER PRIMARY KEY,
                    libelle TEXT NOT NULL
                )'''
            )
connexion.commit()
curseur.execute('''CREATE TABLE IF NOT EXISTS students(
                id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
                pseudo TEXT,
                valeur INTEGER,
                class_id  INTEGER NOT NULL,
                FOREIGN KEY (class_id) REFERENCES classroom (class_id)
            )'''
            )
connexion.commit()
```

BASES DE DONNÉES AVEC PYTHON

```
### Insertion d'une donnée
donnee = ("toto", 10000)
curseur.execute('''INSERT INTO students (pseudo, valeur)
VALUES (?, ?)''',donnee)
connexion.commit()
```

```
### Insertion multiple de données
donnees = (
{"psd" : "toto", "val" : 1000},
{"psd" : "tata", "val" : 750},
{"psd" : "titi", "val" : 500}
)
curseur.executemany("INSERT INTO students (pseudo,
valeur) VALUES (:psd, :val)", donnees)
connexion.commit()
```

BASES DE DONNÉES AVEC PYTHON

```
## Lire tout les items de la table
curseur.execute("SELECT * FROM scores")
resultats = curseur.fetchall()
for resultat in resultats:
    print(resultat)
```

```
# Sélectionner un item dans la DB via son pseudo
donnee = ("titi", )
curseur.execute("SELECT valeur FROM scores WHERE pseudo = ?",donnee)
result = curseur.fetchone()
print(result)
while result:
    print(result)
result = curseur.fetchone()
```

BASES DE DONNÉES AVEC PYTHON

```
# Modifier un item dans la BD
task = (2, '2015-01-04', '2015-01-06', 2)
sql = ''' UPDATE tasks
              SET priority = ? ,
                  begin_date = ? ,
                  end_date = ?
              WHERE id = ?'''
cur.execute(sql, task)
conn.commit()
```

```
## Supprimer un item dans la BD
def delete_item(id):
    sql_update_query = """DELETE from
SqliteDb_developers where id = ?"""
    cursor.execute(sql_update_query, (id,))
    sqliteConnection.commit()
    print("Record deleted successfully")
```

BASES DE DONNÉES AVEC PYTHON

(Exercice)

Créez une une base de données ayant une table **Voiture** représentée par : **id,nom, couleur, vitesse_max**

- Insérez les données dans la table **Voiture**

  EX: **Toyota, rouge, 2000 / Mercedes , Rouge , 2500 / etc.**
- Recuperez le nom de toutes les voitures de couleur **Rouge** puis affichez a l'ecran.

  Ex: **Toyota , Mercedes**

MINI PROJET

Créez un programme python permettant la gestion d'une salle de classe. Une salle de classe(nom, moyenne) comporte plusieurs étudiants (nom, date de naissance, matricule, note). L'utilisateur doit pouvoir ajouter un nouvel étudiant, le modifier et le supprimer, afficher la liste de la classe.

Utilisez toutes les notions vues :

- Classe : étudiant, personne (classe abstraite), classe,
- Héritage, abstraction
- Dictionnaire
- Boucle
- Fonctions
- Modules

Insérer l'étudiant et la classe dans une base de données sqlite