

1. Python script for database connectivity. CSV loading, Perform CRUD operations.
CSV should be in (utf-8) database in (latin-9). Read config file using argparse then pass that to required functions. CSV should be more than 100000 rows

```
from google.colab import drive
drive.mount('/content/drive')
import configparser
import pandas as pd
import pyodbc
import argparse

config = configparser.ConfigParser()
config.read('config.ini')

def create_connection(server, database, username, password):
    conn = None
    try:
        conn = pyodbc.connect('DRIVER={SQL Server};SERVER=' + server +
                               ';DATABASE=' + database +
                               ';UID=' + username +
                               ';PWD=' + password)
        print("Database connection established.")
    except pyodbc.Error as e:
        print(e)
    return conn

def create_table_from_dataframe(conn, df):
    try:
        cursor = conn.cursor()
        cursor.execute("IF OBJECT_ID('movies', 'U') IS NOT NULL DROP TABLE
movies")

        column_types = {
            'Title': 'NVARCHAR(MAX)',
            'Genre': 'NVARCHAR(MAX)',
            'Tags': 'NVARCHAR(MAX)',
            'Languages': 'NVARCHAR(MAX)',
            'Country_Availability': 'NVARCHAR(MAX)',
            'Runtime': 'NVARCHAR(MAX)',
            'Director': 'NVARCHAR(MAX)',
            'Writer': 'NVARCHAR(MAX)',
            'Actors': 'NVARCHAR(MAX)',
            'View_Rating': 'NVARCHAR(MAX)',
            'IMDb_Score': 'NVARCHAR(MAX)',
```

```

        'Awards_Received': 'NVARCHAR(MAX)',
        'Awards_Nominated_For': 'NVARCHAR(MAX)',
        'Boxoffice': 'NVARCHAR(MAX)',
        'Release_Date': 'NVARCHAR(MAX)',
        'movies_Release_Date': 'NVARCHAR(MAX)',
        'Production_House': 'NVARCHAR(MAX)',
        'movies_Link': 'NVARCHAR(MAX)',
        'Summary': 'NVARCHAR(MAX)',
        'Series_or_Movie': 'NVARCHAR(MAX)',
        'IMDb_Votes': 'NVARCHAR(MAX)',
        'Image': 'NVARCHAR(MAX)'
    }
    create_table_sql = f"CREATE TABLE movies ({', '.join([f'{col} {col_type}'
for col, col_type in column_types.items()])})"
    cursor.execute(create_table_sql)
    conn.commit()
    print("Table created successfully based on DataFrame schema.")
except pyodbc.Error as e:
    print(e)

def load_data(conn, df):
    try:
        cursor = conn.cursor()
        for _, row in df.iterrows():
            row_values = [str(value) for value in row]
            cursor.execute("INSERT INTO movies VALUES ({})".format(','.join('?' *
len(row_values))), tuple(row_values))
            conn.commit()
        print("Data loaded from DataFrame to database successfully.")
    except pyodbc.Error as e:
        print(e)

def read_data(conn):
    try:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM movies")
        rows = cursor.fetchall()
        for row in rows:
            print(row)
    except pyodbc.Error as e:
        print(e)

def delete_data(conn, id):

```

```

try:
    cursor = conn.cursor()
    cursor.execute("DELETE FROM movies WHERE Title=?", (id,))
    conn.commit()
    print("Data deleted successfully.")
except pyodbc.Error as e:
    print(e)

def update_data(conn, id, column, new_value):
    try:
        cursor = conn.cursor()
        cursor.execute(f"UPDATE movies SET {column}=? WHERE Title=?", (new_value,
id))
        conn.commit()
        print("Data updated successfully.")
    except pyodbc.Error as e:
        print(e)

def main():
    parser = argparse.ArgumentParser(description="Database CRUD operations with
movie data")
    parser.add_argument("--config-file", type=str, help="Config file containing
database connection information")
    parser.add_argument("--csv-file", type=str, help="CSV file containing movie
data")
    parser.add_argument("--read", action="store_true", help="Read data from the
database")
    parser.add_argument("--update", nargs='+', type=str, help="Update data in the
database (provide id, column, and new value)")
    parser.add_argument("--delete", type=str, help="Delete data from the database
by id")
    args = parser.parse_args()

    df = pd.read_csv(args.csv_file)
    username = config.get('Settings', 'username')
    password = config.get('Settings', 'password')
    driver_name = config.get('Settings', 'DRIVER_NAME')
    database_name = config.get('Settings', 'DATABASE_NAME')
    server_name = config.get('Settings', 'SERVER_NAME')
    conn = create_connection(server_name, database_name, username, password)
    if conn is not None:
        create_table_from_dataframe(conn, df)
        load_data(conn, df)

```

```

    if args.read:
        read_data(conn)

    if args.update:
        update_data(conn, args.update[0], args.update[1], args.update[2])

    if args.delete:
        delete_data(conn, args.delete)

    conn.close()
    print("Database connection closed.")

if __name__ == '__main__':
    main()

```

2. Python code execution flow explanation

It is first compiled into bytecode, which is then executed by the Python interpreter. However, this execution is done line by line, which is very time-consuming, so some compilers like just-in-time compilers are added to try to speed things up.

3. List partition question

```

lst = [1,3,5,7,9,11,13]

below5 = [x for x in lst if x < 5]
equal5 = [x for x in lst if x == 5]
above5 = [x for x in lst if x > 5]

print("Less than 5: ", below5)
print("Equal to 5: ", equal5)
print("Greater than 5: ", above5)

```

4. Use Enum in python

```

from enum import Enum

class color(Enum):
    red = 1

```

```
green = 2
blue = 3


print(color.red.value)
print(color.green.value)
print(color.blue.value)
```

5. Prepare notes for itertools functions which are commonly used

- `repeat()`: Repeats a value a specified number of times (n) or indefinitely if n is not provided.
- `compress()`: Filters elements from an iterable based on a predicate function.
- `groupby()`: Groups elements from an iterable based on a key function, returning an iterator of key-value pairs. Useful for data analysis and processing tasks.
- `permutations()`: Generates all possible permutations (orderings) of elements from an iterable, with an optional parameter for specifying the number of elements per permutation.
- `combinations()`: Generates all possible combinations (selections without order) of elements from an iterable, with a specified number of elements per combination.


6. Example of pass by reference and pass by value

- Pass By Reference

```
 a=[1,3,5,7,9,11,13]
b=a

print(id(a))
print(id(b))

b.append(10)
print(a)
print(b)
```

```
 132788385635136
132788385635136
[1, 3, 5, 7, 9, 11, 13, 10]
[1, 3, 5, 7, 9, 11, 13, 10]
```

- Pass By Value

```

import copy

a=[1,3,5,7,9,11,13]
b=copy.copy(a)
c=copy.deepcopy(a)

print("A: ",id(a))
print("B: ",id(b))
print("C: ",id(c))

a.append(10)
b.append(20)
c.append(30)

print(a)
print(b)
print(c)

```

```

A: 132788385644416
B: 132788385640576
C: 132788385636416
[1, 3, 5, 7, 9, 11, 13, 10]
[1, 3, 5, 7, 9, 11, 13, 20]
[1, 3, 5, 7, 9, 11, 13, 30]

```

7. Demo of Union operator using typing library

```

from typing import Union

UserId = Union[Name, Age]

user_id_name: UserId = "Aryan"
user_id_age: UserId = 23

def get_user_info(user_id: UserId) -> str:
    if isinstance(user_id, Name):
        return f"User name: {user_id}"
    elif isinstance(user_id, Age):
        return f"User age: {user_id}"
    else:
        raise ValueError("Invalid user ID type")

user_info_name = get_user_info(user_id_name)
user_info_age = get_user_info(user_id_age)

print(user_info_name)
print(user_info_age)

```

8. How to check if variable is iterable python

```
def is_iterable(var):
    try:
        iter(var)
        return True
    except TypeError:
        return False

lst = [1, 2, 3]
stri = "Hello"

print(is_iterable(lst))
print(is_iterable(stri))
```

9. What are various ways to implement string formatting in python?

- F-strings:
 - Allows for direct embedding of expressions within curly braces.
 - F “Hello, my name is {name}.”
- format() method:
 - The method is called on the string literal and the values are passed as arguments, with curly braces as placeholders.
 - “Hello, my name is {}.”
- String Concatenation:
 - Using +, we can concatenate strings
 - “Hello, my name is ” + name + “.”

10. Example of any() and all()

```
[12] lst = [1,3,5,7,9,11,13]

below5 = [x for x in lst if x < 5]
equal5 = [x for x in lst if x == 5]
above5 = [x for x in lst if x > 5]

print("Less than 5: ", below5)
print("Equal to 5: ", equal5)
print("Greater than 5: ", above5)
print("Any Values above 5? ", any(x > 5 for x in lst))
print("All Values above 5? ", all(x > 5 for x in lst))
```

```
➡ Less than 5: [1, 3]
   Equal to 5: [5]
   Greater than 5: [7, 9, 11, 13]
   Any Values above 5? True
   All Values above 5? False
```