Music Quiz Game - Computer Science NEA

Structure:

This project is broken down into 3 files:

- main.py - Main Code
- db_manager.py - Contains functions for interacting with SQLite 3.0 Database
- nea.db - SQLite 3.0 Database file - contains Scores, Songs and Logins

main.py:

main.py is where the game logic can be found. It's the file that is run in order to start the game, and contains the startup and shutdown logic.

The first thing that happens in this file is `try_login()` is called.

```
def try_login():
    username = str(input("\nEnter your username to login: $ "))
    password = str(input(f"Enter your password for \"{username}\": $ "))

    return db_manager.check_login(username, password), username, password
```

This function queries the SQL database using the db_manager.py file and the `check_login()` function to check if the login is valid. The returned variables are assigned as can be seen below, and used to check if the user is authenticated:

```
logged_in, username, password = try_login()

while not logged_in: ##Infinite loop unitl logged in
    logged_in, username, password = try_login()
```

If the user enters the correct login information, they are welcomed, and the variable `score` is initialised

```
print(f"Login Successful - Hello {username}")
score = 0
```

This is where the main game loop begins:

```
while True: ##Initialize main game loop
```

```
    all_songs = db_manager.get_songs()
    song = random.choice(all_songs) ##var song is tuple with structure (id, 'song1', 'artis

    song_name = song[1]
    song_artist = song[2]
    first_letters = []

    if player_guess().lower() != song_name.lower(): ##First Attempt
        print("\nincorrect! 1 attempt left")
        time.sleep(1)
        if player_guess().lower() != song_name.lower(): ##Second Attempt
            nea_exit(username, score)
        else:
            score += 1
            print("\ncorrect! +1 score")
    else:
        score += 2
        print("\ncorrect! +2 score")
```

In the first line of the loop, `all_songs` is assigned to a nested list from the database, containing all of the songs and artists.

A random song is chosen with

```
song = random.choice(all_songs)
```

From this, the `song_name`, `song_artist` and `first_letters` lists are initialised

```
song_name = song[1]
song_artist = song[2]
first_letters = []
```

From this point, `player_guess()` is called in an `if` statement:

```
def player_guess():
    print(f"\nArtist Name: {song_artist}")
    print(f"First letter of song: {song_name[:1]}")
    guess = str(input("Your Guess $ "))

    return guess
```

This function is just to prevent repeating code, and queries the user for their guess based on the first letter and artist name of the song.

string indicies are used with `song_name[:1]` to get the first letter of the string

back to the if statement:

```
if player_guess().lower() != song_name.lower():
```

here the player guess to compared with the song name to check if they are the same. if they are, 2 score is added for being the first guess:

```
else:
    score += 2
    print("\ncorrect! +2 score")
```

however if it's not, the user is given another chance:

```
if player_guess().lower() != song_name.lower(): ##Second Attempt
            nea_exit(username, score)
```

As can be seen here, if the second attempt is wrong, the exit logic function is run.

```
def nea_exit(username, score):
    ##display current score, update scores db, display top 5 scores from db
    db_manager.submit_to_leaderboard(username, str(score))
    print("\nGame Over!")
    print(f"You got {str(score)}!\n")
    print("Top 5 scores:")
    for i in db_manager.get_top_5():
        print(f"{i[1]}: {i[2]}")
    exit()
```

This function is the last peice of code run, as it ends with a call to `exit()` which terminates the program.

On the first line, the new score is submitted to the database with `submit_to_leaderboard()`

```
db_manager.submit_to_leaderboard(username, str(score))
```

Then, the player's score is printed, and the top 5 scores are prepared to be printed.

```
print("\nGame Over!")
print(f"You got {str(score)}!\n")
print("Top 5 scores:")
```

To acutally get the top 5 scores, the `get_top_5()` function is called, and the results are iterated

over:

```
for i in db_manager.get_top_5():
        print(f"{i[1]}: {i[2]}")
```

In this context, var i is a list of a score, containing `(id, name, score)`. Therefore when `i[1]` is printed, it prints the name. This applies similarly to the score with `i[2]`.

Finally, the `exit()` function is called. This terminates the program.

```
exit()
```

db_manager.py

`db_manager.py` is the file containing all of the functions that actually interact with the database using SQL queries.

check_login()

the first function to be defined in this file is `check_login()`

```
def check_login(username, password): ##checks for username and password combo in sqlite dat
    conn = sqlite3.connect('nea.db')
    c = conn.cursor()

    c.execute("SELECT * FROM logins WHERE username = ? AND password = ?", (username, passwo
    rows = c.fetchall()
    conn.close()

    if len(rows) == 0: ##var rows contains all instances of matching usr:pass combos theref
        return False
    else:
        return True
```

This function takes 2 positional arguments: `username` and `password`. The purpose of the function is to check if the `username` and `password` combination is valid.

In the first 2 lines, a connection to the database file is established:

```
conn = sqlite3.connect('nea.db')
c = conn.cursor()
```

The defined cursor is then used to execute the SQL query:

```
SELECT * FROM logins WHERE username = ? AND password = ?
```

This query selects all rows from the `logins` table where the `username = ?` and the `password = ?`. The ?'s in this context are passed through to the sqlite3 module where the `username` and the `password` variables are substituted in respectively in order to prevent potential SQL vunerabilities.

The rows from the result of this are fetched as can be seen, and the file is closed:

```
rows = c.fetchall()
conn.close()
```

Finally, an `if` statement checks the length of the resulting rows. If the length of them is 0, the login is incorrect, and False is returned. Therefore if the length is greater than 0 True is returned.

get_songs()

The next function is `get_songs()`. This function returns a nested list of all the songs and artists, and does not have any arguments.

```
def get_songs(): ##return nested list of all song names and artists
    conn = sqlite3.connect('nea.db')
    c = conn.cursor()

    c.execute("SELECT * FROM songs")
    rows = c.fetchall()
    conn.close()

    return rows
```

similarly to before, the connection to the database is establised and the cursor is defined:

```
conn = sqlite3.connect('nea.db')
c = conn.cursor()
```

After this, the SQL Query `SELECT * FROM songs` is executed, and the result is sored in the variable `rows` before the file is closed.

```
c.execute("SELECT * FROM songs")
rows = c.fetchall()
conn.close()
```

From this point, the variable `rows` is returned.

submit_to_leaderboard

The function `submit_to_leaderboard()` takes 2 positional arguments: `username` and `score`.

```
def submit_to_leaderboard(username, score): ##add score to leaderboard table
    conn = sqlite3.connect('nea.db')
    c = conn.cursor()

    c.execute("SELECT * FROM leaderboard WHERE username = ?", (username,))
    rows = c.fetchall()

    if len(rows) == 0:
        c.execute("INSERT INTO leaderboard(username, score) VALUES(?, ?)", username, str(sc
    else:
        if int(score) < int(rows[0][2]):
            conn.close()
            return
        else:
            c.execute("UPDATE leaderboard SET score = ? WHERE username = ?", (str(score), u

    conn.commit()
    conn.close()
```

After the SQL file is opened:

```
conn = sqlite3.connect('nea.db')
c = conn.cursor()
```

The `leaderboard` table is then queried for all of its rows matching the `username` of the user

```
SELECT * FROM leaderboard WHERE username = ?
```

This is done to check if the user already exists in the leaderboard so their score can be updated, and if not they can be added.

This is then stored in `rows` and if the user does exist (equal to 0), the following SQL string is executed:

```
INSERT INTO leaderboard(username, score) VALUES(?, ?)
```

This inserts the `username` with the `score` into the `leaderboard` table

However if the user does exist, the SQL commands `UPDATE` is used instead but, before it can be updated the score is checked with the user's old score to check if it's a highscore or not.

```
if int(score) < int(rows[0][2]):
```

```
    conn.close()
    return
```

This prevents the user's old highscore from being overwritten. If the score is a highscore, it can be updated:

```
UPDATE leaderboard SET score = ? WHERE username = ?
```

This updates the column `score` for the user in the row where the `username` matches.

get_top_5()

This function returns the top 5 scores from the `leaderboard` table in the form of a nested list.

```
def get_top_5(): ##returns top 5 from leaderboard in nested list
    conn = sqlite3.connect('nea.db')
    c = conn.cursor()

    c.execute("SELECT * FROM leaderboard ORDER BY -ABS(score) LIMIT 5") ##SQL: top 5 ordere
    rows = c.fetchall()

    return rows
```

In the first lines the SQL database is opened and the following SQL query is executed:

```
SELECT * FROM leaderboard ORDER BY -ABS(score) LIMIT 5
```

This selects all rows from the `leaderboard` table, ordered by `-ABS(score)` with a limit of 5

`-ABS(score)` means the absolute value of score. This is required as `score` is stored as a `VARCHAR` in the SQL table. The `-` in from means decending.

`LIMIT 5` means stop at 5 results. This is to get the top 5.

The final row returns the `rows`.

nea.db

This file is a SQLite 3.0 database file. It contains 3 tables:

- leaderboard
- logins
- songs

leaderboard

This table has the columns `id`, `username` and `score`

logins

This table has the columns `id`, `username` and `password`

songs

This table has the columns `id`, `song_name` and `artist_name`