
| | |
|---|---|
| Student name: | Bùi Nguyên Bảo Khang |
| ID number: | GCS220046 |
| Academic year: | 2024-2025 |
| Module: | **COMP1752: Object Oriented Programming** |
| Module Assessment Title: | PlayMusic app: JukeBox |
| Submission Date: | 25/11/2024 |
| Due Date: | 04/12/2024 |

Link Github: https://github.com/Khang-IM/PlayMusic-app-JukeBox

**Table of Contents**

1. Introduction ..... 5
2. How it works ? ..... 5
2.1. Design LibraryItem Class. ..... 5
2.2. Design Track Library ..... 7
3. Interface and functions. ..... 8
3.1. View Tracks ..... 8
3.1.1. Interface ..... 8
3.1.2. Functions ..... 9
3.2. Create Track List ..... 10
3.2.1. Interface ..... 10
3.2.2. Functions ..... 11
3.3. Update Track ..... 17
3.3.1. Interface ..... 17
3.3.2. Function ..... 18
3.4. Get Top Tracks From API ..... 22
3.4.1. Interface ..... 22
3.4.2. Function ..... 23
3.5. Listen Track Online ..... 24
3.5.1. Interface ..... 24
3.5.2. Function ..... 24
4. Testing ..... 25
4.1. Pytest ..... 25
4.2. GUI Test ..... 26
5. Conclusion ..... 31

**IMAGE LIST**

Figure 1: Class LibraryItem ..... 6
Figure 2: The program's main JukeBox window ..... 8
Figure 3: View Tracks interface ..... 8
Figure 4: Function list_tracks_clicked load all tracks into txt element ..... 9
Figure 5: Function set_text ..... 9
Figure 6: Function view_tracks_clicked ..... 9
Figure 7: When the track is found ..... 10
Figure 8: When track is not found ..... 10
Figure 9: Create Track interface ..... 11
Figure 10: Add_track function ..... 12
Figure 11: The interface after successfully ..... 12
Figure 12: In case track number already exists ..... 13
Figure 13: In case the data is not fully entered ..... 13
Figure 14: In case rating is not a integer ..... 14
Figure 15: Function play_list ..... 14
Figure 16: Results after playing the list for the first time ..... 15
Figure 17: Results after playing the list for the second time ..... 16
Figure 18: Function clear_input ..... 16
Figure 19: Function reset ..... 17
Figure 20: After reset ..... 17
Figure 21: Update track interface ..... 18
Figure 22: Function update_rating ..... 19
Figure 23: Before update ..... 19
Figure 24: After update ..... 20
Figure 25: In case track number does not exist ..... 20
Figure 26: In case new rating is not integer ..... 21
Figure 27: In case new rating < 0 or new rating > 10 ..... 21
Figure 28: Screen of create track after update. ..... 22
Figure 29: Result top tracks of Adele ..... 22

Figure 30: The fetch_top_track function returns a track list of songs by artist name. .....23
Figure 31: Get top tracks function interacts with interface .....23
Figure 32: Interface if view tracks after update .....24
Figure 33: Function view track in lastfm .....25
Figure 34: Listen track online .....25
Figure 35: Using pytest for testing LibraryItem.....26
Figure 36: Result of test suite.....26

# 1. Introduction

This basic music player application, built with Python and the tkinter library, offers a straightforward and interactive interface for managing and enjoying a personal music library. Designed to focus on essential features, the application enables users to browse through their existing music collection, create new song entries, and update details for songs already in the library.

At its core, the music player includes a library view where users can see their entire collection in a list format, making it easy to locate and manage tracks. Users can add new songs to the list, complete with metadata such as title, artist, and genre. If any song details need to be modified—such as changing a rating or updating the play count—users can quickly edit the entry through the update functionality. Additionally, the app includes basic playback features, allowing users to play songs directly from the interface without needing to switch between multiple applications.

Built with tkinter, this application serves as an excellent example of how Python can be used to develop simple yet effective GUI applications. Not only does it showcase the versatility of tkinter for creating interactive interfaces, but it also demonstrates how Python's modularity supports adding custom features, such as a track rating or play count increment, to enhance user engagement. This project is ideal for beginners interested in learning about GUI development in Python and provides a strong foundation for those looking to expand into more complex multimedia applications.

# 2. How it works?

This project is an Object-Oriented Programming (OOP) project. It follows the principles of OOP by using classes and objects to model the music library and its functionality. Here is how it fits into OOP concepts

# 2.1. Design LibraryItem Class.

First, the LibraryItem class represents individual music items in the library, such as songs or tracks. This class is designed to encapsulate essential attributes and behaviors for each music item, including the title, artist, rating, and play count. By defining these characteristics within the LibraryItem class, we can manage each track as an object with easily accessible properties and behaviors, facilitating organized management of a music library.

```python
class LibraryItem:
    def __init__(self, name, artist, rating=0):
        self.name = name
        self.artist = artist
        self.rating = rating
        self.play_count = 0

    1 usage (1 dynamic)
    def info(self):
        return f"{self.name} - {self.artist} {self.stars()}"

    1 usage
    def stars(self):
        stars = ""
        for i in range(self.rating):
            stars += "*"
        return stars
```

Figure 1: Class LibraryItem

This class will contain the following properties:

- name: This property stores the title of the music item, allowing users to identify songs by their specific name.
- artist: This property contains the name of the artist associated with the music item, providing additional context for each song.
- rating: This optional property is initialized to 0 by default, representing the user's rating of the music item. This property helps users rate songs based on their preferences. It will be an integer.
- play_count: This property tracks the number of times a music item has been played. This property is initialized to 0 and can be incremented each time a music item is played, allowing users to see which songs they listen to the most

And methods are also defined with their own functions as follows:

- info(): This method returns a string representation of the music item, combining the name, artist, and visual representation of the rating. It provides a user-friendly format for displaying music item details.
- stars(): The stars() method generates a string representation of the rating using asterisks (*). This allows for an intuitive, visual representation of the song's rating, where each asterisk corresponds to one rating point.

With these attributes and methods, the LibraryItem class provides a well-organized structure for representing individual songs in a music library. The class allows for easy manipulation and retrieval of song information, making it a foundational component in the music player application.

## 2.2. *Design Track Library*

In track_library.py, this code acts as the main manager of the music library, organizing songs into LibraryItem objects stored in a dictionary. It is organized into several separate functions, which take in and return corresponding values.

- library dictionary: Stores each LibraryItem with a unique key, allowing for easy retrieval and management of music items.
- list_all(): Generates a formatted string listing all items in the library, displaying the key, name, artist, and rating of each song.
- get_name(key): Retrieves the name (title) of the song associated with the specified key. Returns None if the key is not found.
- get_artist(key): Returns the artist's name for the specified song key. If the key is invalid, returns None.
- get_rating(key): Retrieves the rating of the song identified by the given key. Returns -1 if the key does not exist.
- set_rating(key, rating): Updates the rating of the song associated with the given key. If the key is invalid, the function does nothing.
- get_play_count(key): Returns the number of plays for the song identified by the specified key. If the key is missing, returns -1.
- increment_play_count(key): Increases the number of plays for the specified song by 1. If the key does not exist, it does nothing.

### 2.3. User Interaction

After defining the classes and methods, the main user interface has been built with Tkinter. The main window will be called JukeBox. And it contains 3 buttons that will lead to 3 child windows with separate functions.

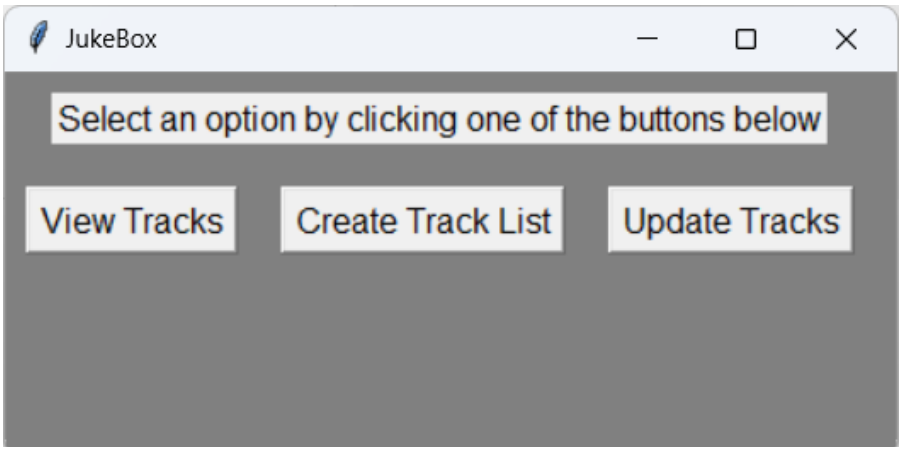


Figure 2: The program's main JukeBox window

When the user clicks on one of the three buttons: View Tracks, Create Track List or Update Tracks; the program opens a new window corresponding to that button while keeping the JukeBox window active. Details about the three new windows will be presented in the following section.

# 3. Interface and functions.

## 3.1. View Tracks

### 3.1.1. Interface

The TrackViewer class sets up a basic GUI for viewing music tracks. It includes buttons to list all tracks and view details for a specific track number entered by the user. Track details are shown in scrollable text areas, and a status label at the bottom displays feedback messages. When the window opens, all tracks are listed automatically.

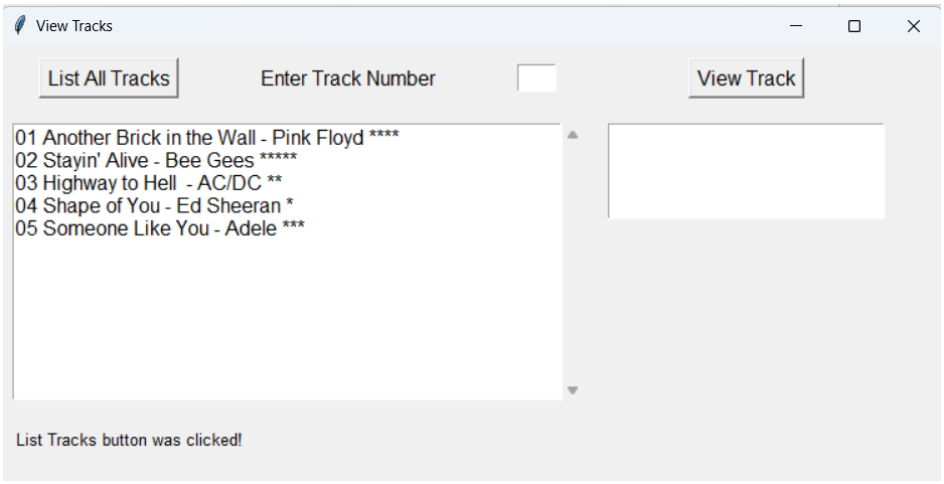


Figure 3: View Tracks interface

## 3.1.2. Functions

When launching the View Tracks screen, the tracks that have been created in the track library will be called via the list_all() method mentioned in 2.2. Define a function list_tracks_clicked that can be called and passed as an argument.

```
def list_tracks_clicked(self):
    track_list = lib.list_all()
    set_text(self.list_txt, track_list)
    self.status_lbl.configure(text="List Tracks button was clicked!")
```

Figure 4: Function list_tracks_clicked load all tracks into txt element

In this function, we use a set_text function which clears all the existing data in the text area and inserts the new content at the beginning.

```
def set_text(text_area, content):
    text_area.delete("1.0", tk.END)
    text_area.insert(1.0, content)
```

Figure 5: Function set_text

The function view_tracks_clicked is responsible for handling the action when the user clicks the "View Track" button, người dùng sẽ cần nhập number của track vào text box. Sau khi click.

```
def view_tracks_clicked(self):
    key = self.input_txt.get()
    name = lib.get_name(key)
    if name is not None:
        artist = lib.get_artist(key)
        rating = lib.get_rating(key)
        play_count = lib.get_play_count(key)
        track_details = f"{name}\n{artist}\nrating: {rating}\nplays: {play_count}"
        set_text(self.track_txt, track_details)
    else:
        set_text(self.track_txt, content: f"Track {key} not found")
    self.status_lbl.configure(text="View Track button was clicked!")
```

Figure 6: Function view_tracks_clicked

If the track is found, it formats and displays the track details (name, artist, rating, play count) in the track_txt text area using the set_text() function as shown in figure 7. If the track is not found, it displays an error message as shown in figure 8.

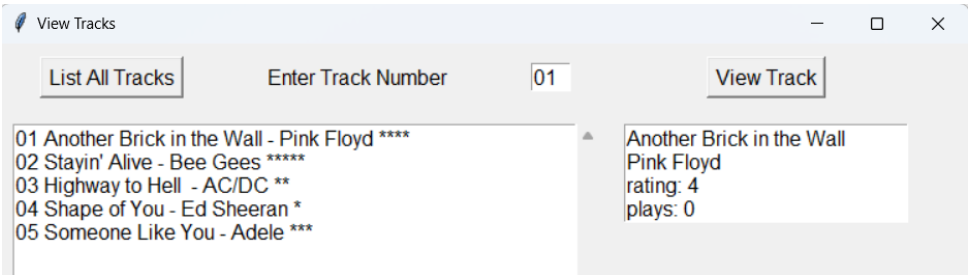


Figure 7: When the track is found

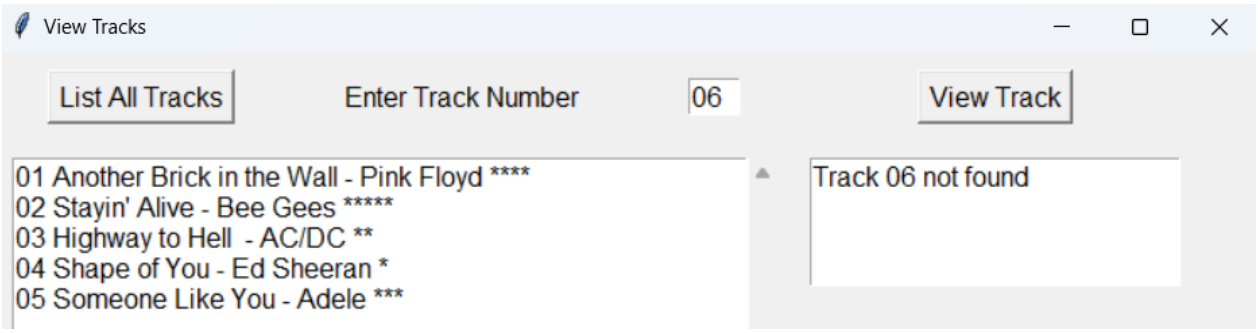


Figure 8: When track is not found

## 3.2. Create Track List

### 3.2.1. Interface

The create track list window interface is designed so that users can enter the necessary information of the song and add it to the current library if the conditions are satisfied. It will have labels corresponding to the inputs, text boxes that allow users to enter values into it.

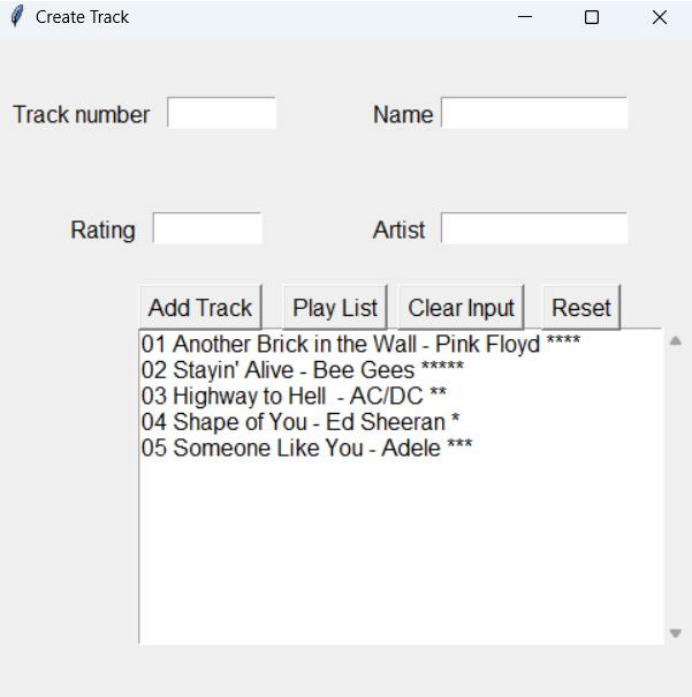


Figure 9: Create Track interface

The window also contains a text area that displays the existing music library. The buttons with corresponding names and functions are:

- Add Track: Add a track to the track library if the track number does not exist in the library
- Play List: each track on the playlist should have its play count value incremented by 1
- Clear input: clear the data currently in the input text boxes
- Reset: reset the playlist and clear the text area

## 3.2.2. Functions

*a) Add track*

Use the same set_text function as in ViewTracks to display data into the text area. The add_track function receives data from the text boxes, then checks if the conditions are met. If the conditions are met, the new track will be added and the text area will be reloaded with the new song library.

```
def add_track(self):
    track_number = self.txt_track_number.get( index1: "1.0",  index2: "end-1c").strip()
    track_name = self.txt_name.get( index1: "1.0",  index2: "end-1c").strip()
    artist = self.txt_artist.get( index1: "1.0",  index2: "end-1c").strip()
    rating = self.txt_rating.get( index1: "1.0",  index2: "end-1c").strip()

    if not track_number or not track_name or not artist or not rating:
        messagebox.showwarning( title: "Input Error",  message: "Please fill in all fields!")
        return
    try:
        rating = int(rating)
    except ValueError:
        messagebox.showwarning( title: "Input Error",  message: "Rating must be a number!")
        return
    if track_number not in lib.library:
        new_track = LibraryItem(track_name, artist, rating)
        lib.library[track_number] = new_track
        self.list_tracks_clicked()
    else:
        messagebox.showerror( title: "Error",  message: f"Track number '{track_number}' already exists!")
```

Figure 10: Add_track function

The interface after successfully adding a new track looks like this:

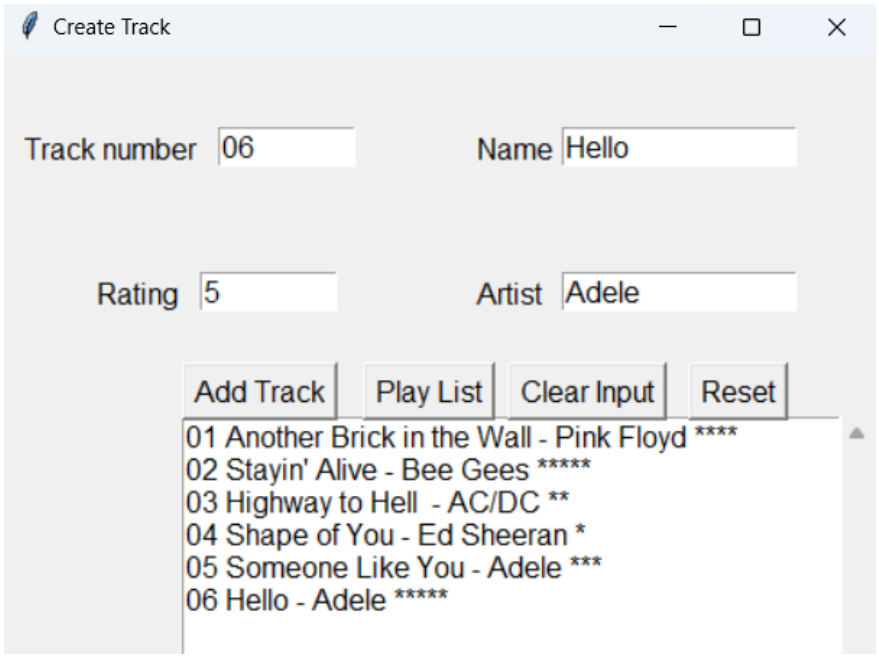


Figure 11: The interface after successfully

In addition, the error cases are as follows:

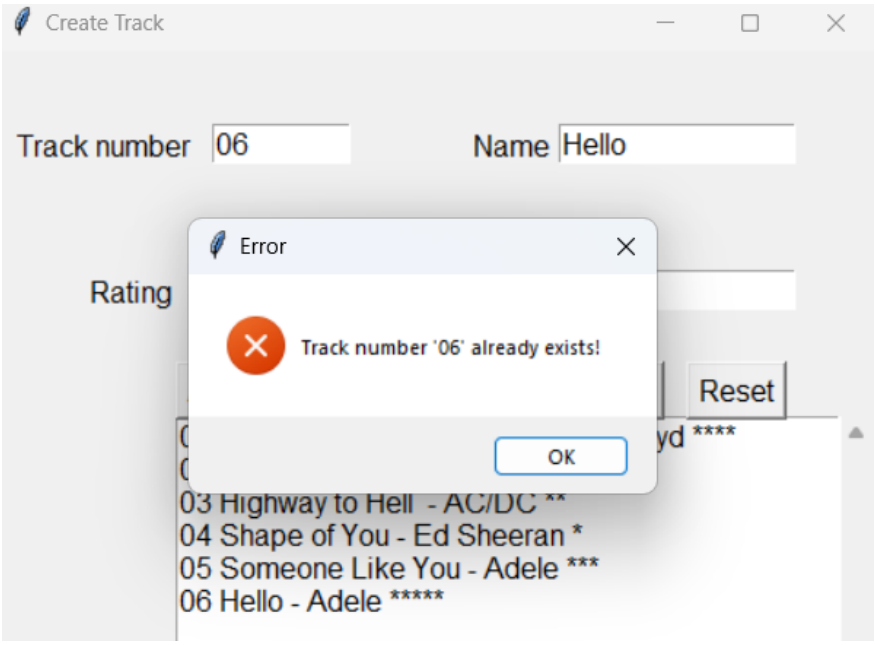


Figure 12: In case track number already exists

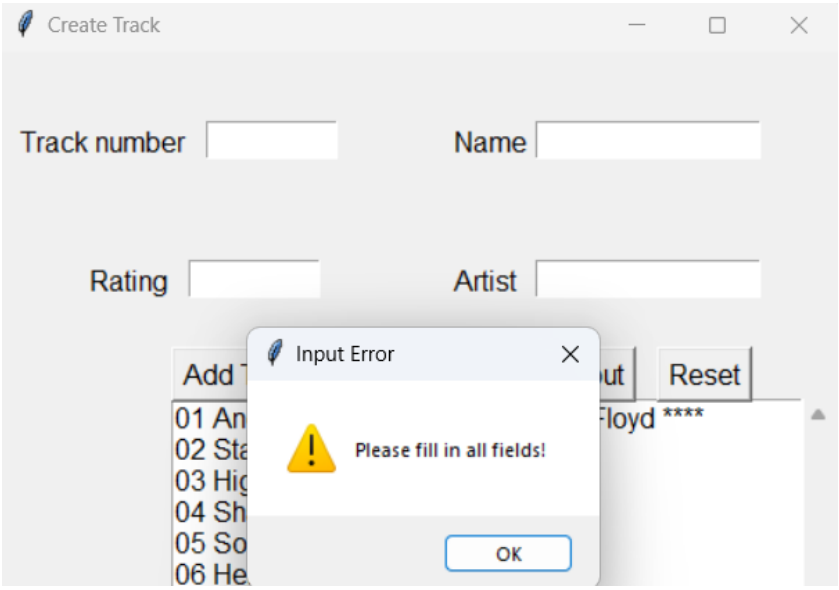


Figure 13: In case the data is not fully entered

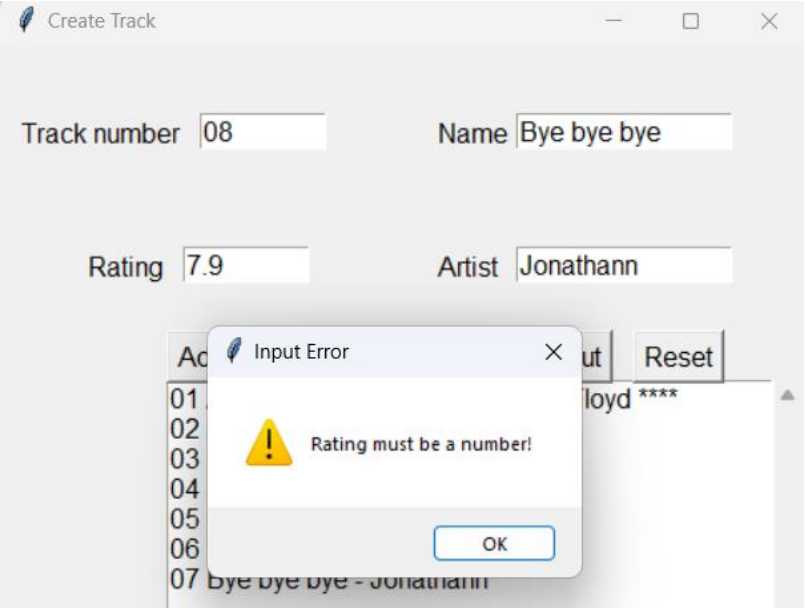


Figure 14: In case rating is not a integer

*b) Play list*

The play list button will call the get_play_count function from the previously defined library, looping through the existing track numbers to get the number of times played. After clicking, all songs will have their play count increased by 1 using the increment_play_count function.

```
def play_list(self):
    message = ""
    for key in lib.library:
        play_count = lib.get_play_count(key)
        message += f"Track {key}: Play count = {play_count}\n"

        lib.increment_play_count(key)
        updated_play_count = lib.get_play_count(key)
        message += f"Track {key}: Play count updated to = {updated_play_count}\n\n"
    messagebox.showinfo( title: "Track Play Count", message)
```

Figure 15: Function play_list

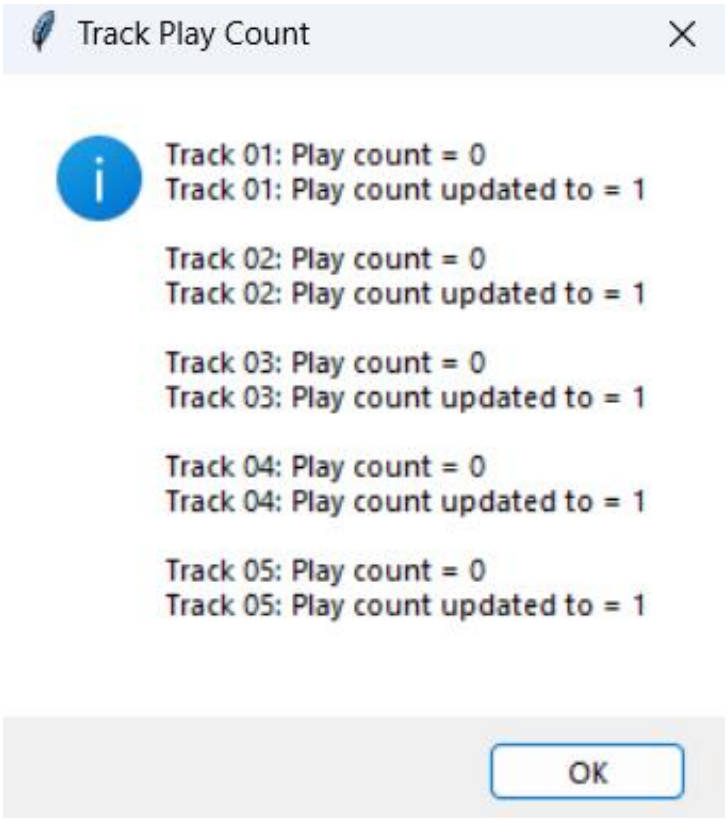


Figure 16: Results after playing the list for the first time

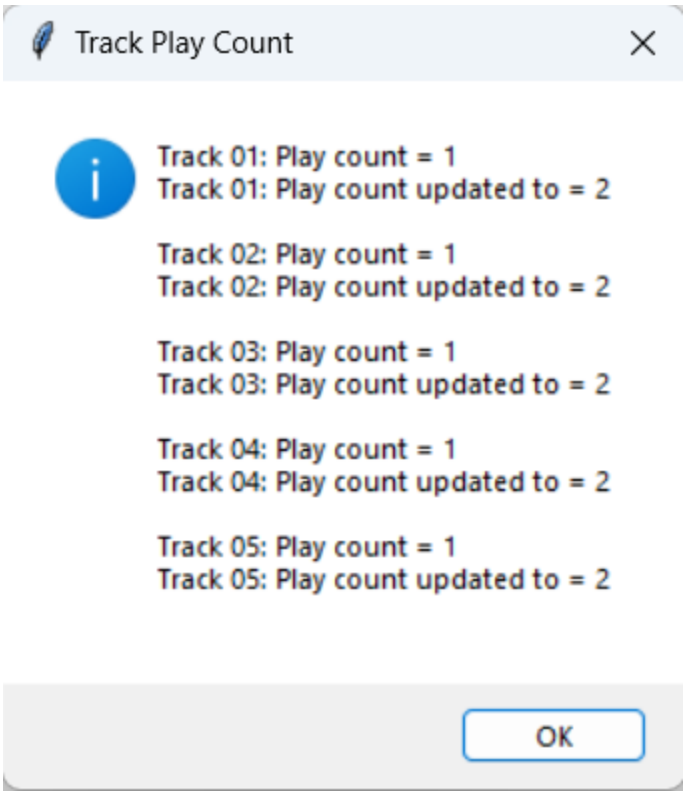


Figure 17: Results after playing the list for the second time

*c) Clear input*

This button is a quick utility for users to clear all data being entered to re-enter.

```
def clear_input(self):
    self.txt_track_number.delete( index1: "1.0",  index2: "end")
    self.txt_name.delete( index1: "1.0",  index2: "end")
    self.txt_rating.delete( index1: "1.0",  index2: "end")
    self.txt_artist.delete( index1: "1.0",  index2: "end")
```

Figure 18: Function clear_input

After clicking, the data being entered will be cleared and returned to a blank cell.

*d) Reset*

The reset button will reset the playlist and clear the currently loaded text area.

```
def reset(self):
    set_text(self.list_txt,  content: "")
    messagebox.showinfo( title: "Notification", message: "Reset sucessfully !!")
```

Figure 19: Function reset

After reset, the playlist will be cleared and the screen will have the result as figure 20

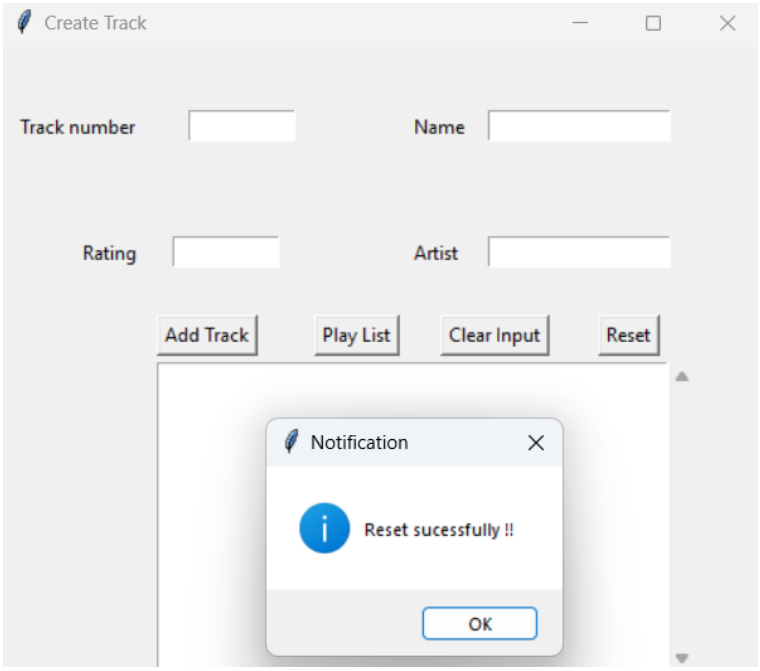


Figure 20: After reset

# 3.3. Update Track

## 3.3.1. Interface

The update interface is simply designed including:

- Text box allowing to enter track number
- Text box allowing to enter new rating
- Corresponding labels
- Text area displaying the library of available tracks similar to previous interfaces

The illustrated interface will be:

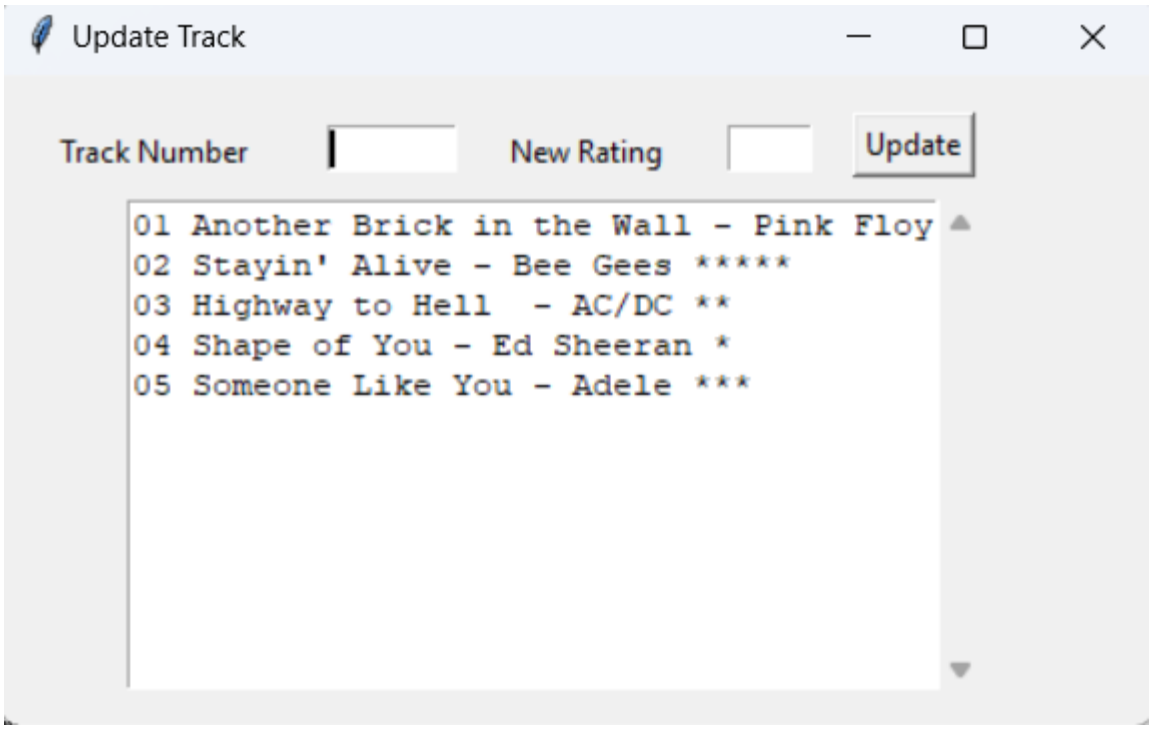


Figure 21: Update track interface

## 3.3.2. Function

For the update track function, define a function named update_rating. This function will check the input track number to see if it exists in the library. If it exists, it will update the new rating for that track. Or report an error if it is not found. After updating, the library will be reloaded and the new rating will be displayed.

The update_rating function is defined as follows:

```
def update_rating(self):
    track_number = self.txt_track_number.get().strip()
    new_rating = self.txt_new_rating.get().strip()

    if not track_number or not new_rating:
        messagebox.showerror( title: "Error",  message: "Please fill in both fields.")
        return

    try:
        new_rating = int(new_rating)
        if new_rating < 1 or new_rating > 5:
            messagebox.showerror( title: "Error",  message: "Rating should be between 1 and 5.")
            return
    except ValueError:
        messagebox.showerror( title: "Error",  message: "Rating should be an integer.")
        return
    if track_number not in lib.library:
        messagebox.showerror( title: "Error",  message: f"Track {track_number} not found.")
        return

    play_count = lib.get_play_count(track_number)

    lib.set_rating(track_number, new_rating)

    track_name = lib.get_name(track_number)
    artist = lib.get_artist(track_number)
    track_details = f"Track Name: {track_name}\nArtist: {artist}\nNew Rating: {new_rating}\nPlay Count: {
    self.list_tracks_clicked()
    messagebox.showinfo( title: "Track Updated", track_details)
```

Figure 22: Function update_rating

And we got the following result. Figure 23 is before update, with track number 05 having rating of 3. Update it to 5 and the result will then be shown in figure 24.


Figure 23: Before update

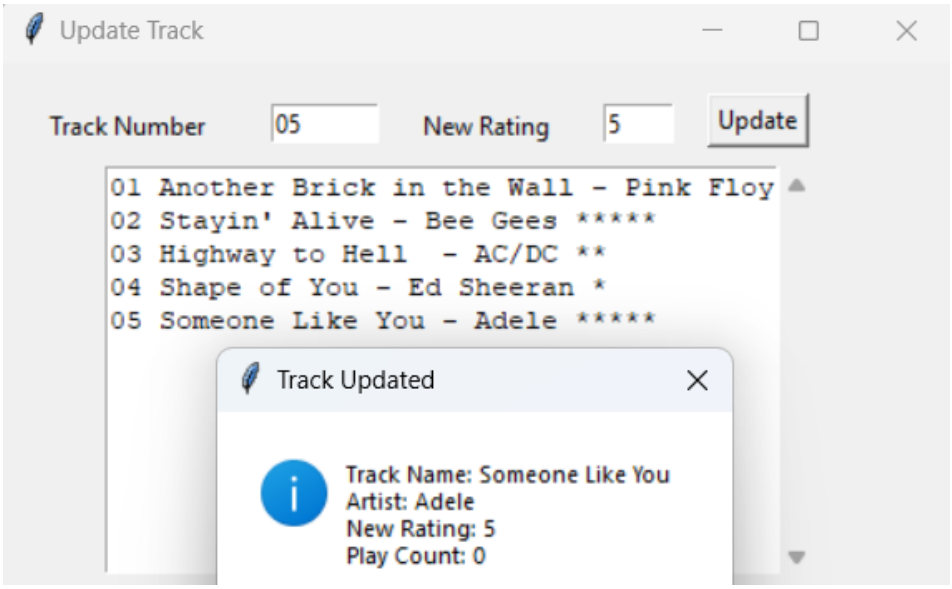


Figure 24: After update

Data errors are also checked when the user enters a track number that does not exist and not valid

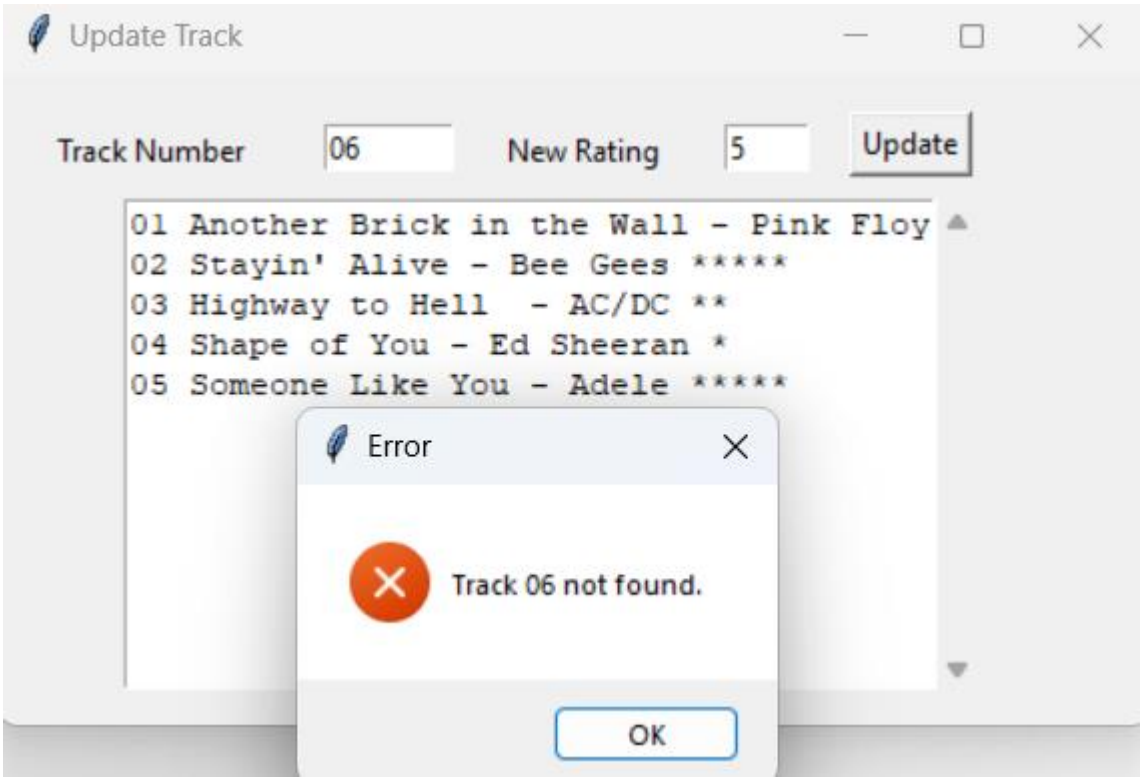


Figure 25: In case track number does not exist

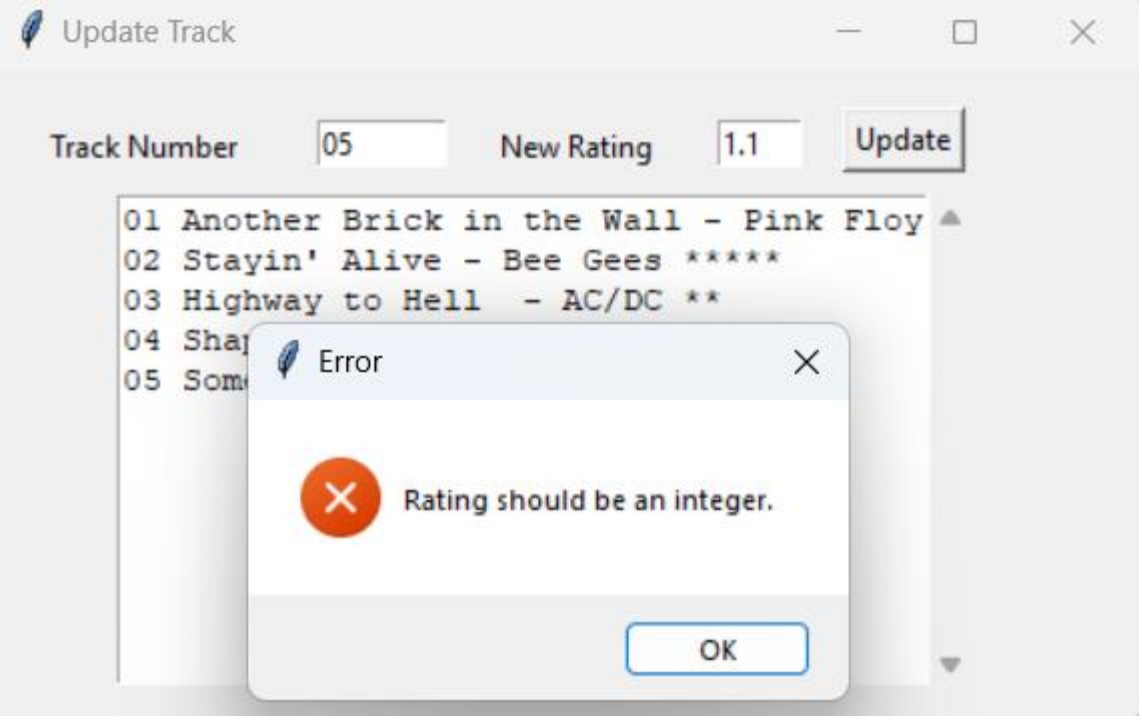


Figure 26: In case new rating is not integer

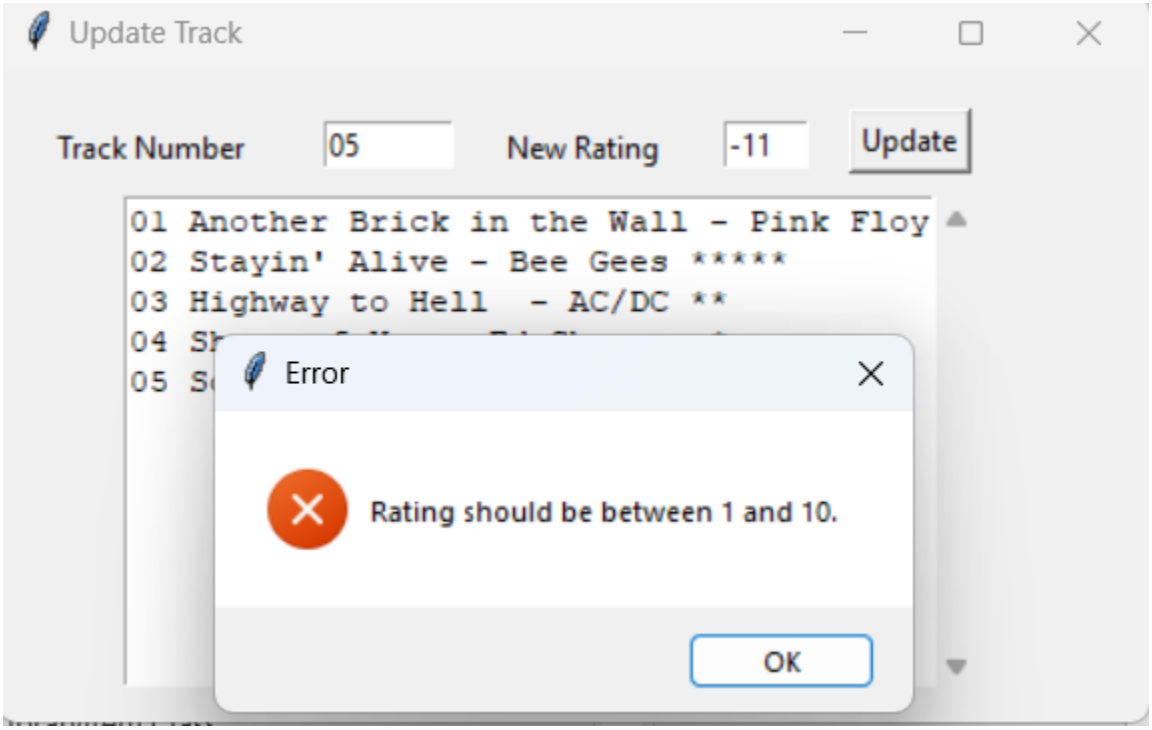


Figure 27: In case new rating < 0 or new rating > 10

## 3.4. Get Top Tracks From API

### 3.4.1. Interface

Instead of having to enter data manually, this software provides a utility to help users get the top tracks from artists provided through a combobox. Lastfm is a popular music listening application worldwide, and it provides api to interact with data

First, a combobox will be created. And users can select the artist data from this combobox. Then, when pressing the get button, the top 10 tracks of that artist will be loaded into the current Library.

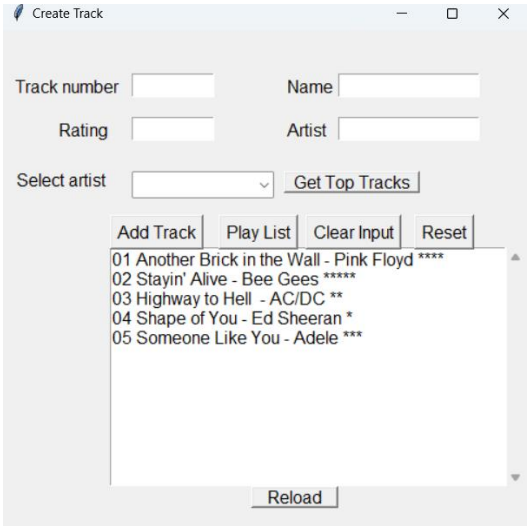


Figure 28: Screen of create track after update.

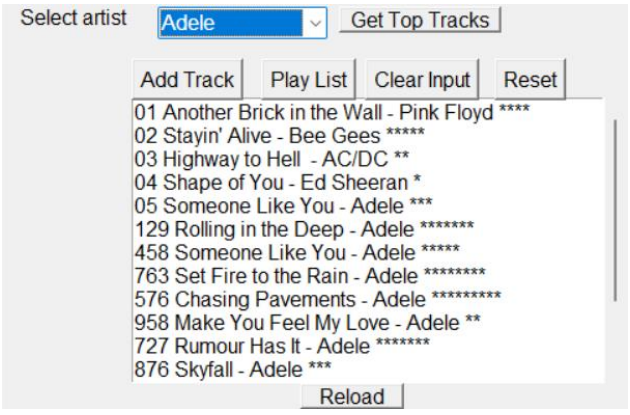


Figure 29: Result top tracks of Adele

## 3.4.2. Function

To do the above, define a function to get data from Lastfm API as follows the get data function is defined as follows:

```
def fetch_top_tracks(artist_name):
    url = f"http://ws.audioscrobbler.com/2.0/?method=artist.gettoptracks&artist={artist_name}&api_key={api_key}&format=json"
    response = requests.get(url)
    top_tracks_data = response.json()

    track_list = []
    top_tracks = top_tracks_data['toptracks']['track'][:10]

    if 'toptracks' in top_tracks_data and 'track' in top_tracks_data['toptracks']:
        unique_ids = random.sample(range(100, 1000), len(top_tracks_data['toptracks']['track']))

        for index, track in enumerate(top_tracks):
            track_number = unique_ids[index]
            track_name = track['name']
            track_artist = artist_name
            rating = generate_random_rating()

            track_list.append({
                "track_number": track_number,
                "track_name": track_name,
                "track_artist": track_artist,
                "rating": rating,
            })
    return track_list
```

Figure 30: The fetch_top_track function returns a track list of songs by artist name.

Once done, at the GUI of create track; just get the selected artis from the comboBox passed in and attach it to the current library. The data will be automatically retrieved and displayed on the interface.

```
def get_top_tracks(self):
    selected_artist = self.comboBox.get()
    if selected_artist:
        top_track = fetch_top_tracks(self.comboBox.get())
        for track in top_track:
            track_number = str(track['track_number'])
            lib.library[track_number] = LibraryItem(track['track_name'], track['track_artist'],
                                                    track['rating'])

        track_lists = lib.list_all()
        set_text(self.list_txt, track_lists)
    else:
        messagebox.showerror( title: "Warning", message: "Please select artis first !")
```

Figure 31: Get top tracks function interacts with interface

# 3.5. Listen Track Online

## 3.5.1. Interface

The program also updates an option to listen to music online for users. At the initial track view screen, when the user enters a valid track number and successfully searches; the Listen Online button will be clickable. A browser will open and take you straight to the website with that song, and all you have to do is press the play button to enjoy.

The interface after the update will be:

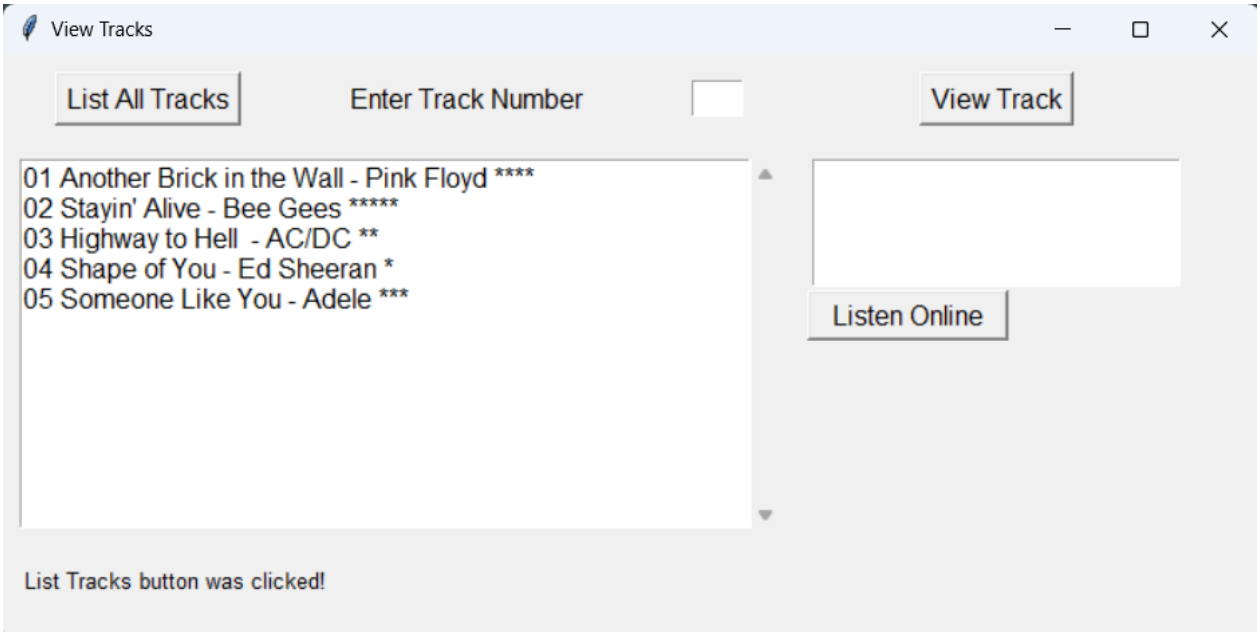


Figure 32: Interface if view tracks after update

### 3.5.2. Function

With the above function, you will need to get the track name from the textbox. Then create a url in the form https://www.last.fm/music/{track_name} to lead to the website. The rest is just to open the browser with the webrowser library

```
def view_track_in_lastfm(self):
    content = self.track_txt.get( index1: "1.0", tk.END)
    lines = content.strip().split("\n")
    track_name = lines[0] if len(lines) > 0 else None
    if track_name:
        print(track_name)
        url = f"https://www.last.fm/music/{track_name}"
        webbrowser.open(url)
```

Figure 33: Function view track in lastfm

And the final result

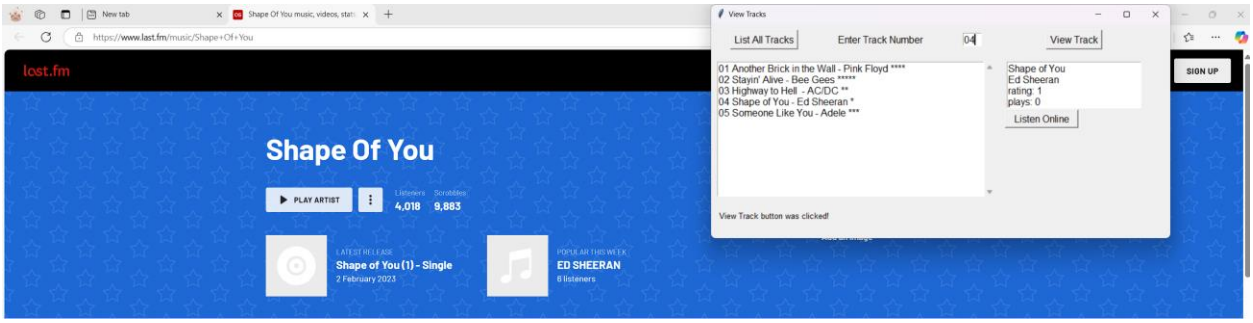


Figure 34: Listen track online

# 4. Testing

## 4.1. Pytest

In this test suite, we are performing unit tests on the LibraryItem class, which represents a music track with properties like name, artist, rating, and play_count. This class also includes methods to return information about the track and the number of stars based on its rating.

The tests are written using the pytest framework, which is a popular tool for writing and running Python tests. In this suite, we validate the following functionalities:

- Initialization of the object: We verify that the LibraryItem class correctly initializes an object with the given name, artist, and rating, and that the play_count starts at zero.
- Info Method: We test the info() method that provides a formatted string of the track's details, including its name, artist, and rating in the form of stars.

- Stars Method: The stars() method returns a string representation of the track's rating using stars. We test different rating values, including zero and the maximum rating.
- Play Count: We test the play_count functionality by checking its initial value and verifying that it can be incremented correctly.

By running these tests, we ensure that the LibraryItem class behaves as expected, providing reliable results and making it easier to maintain or extend in the future.

```python
import pytest
from library_item import LibraryItem

def test_initialization():
    item = LibraryItem( name: "Song A",  artist: "Artist A",  rating: 3)
    assert item.name == "Song A"
    assert item.artist == "Artist A"
    assert item.rating == 3
    assert item.play_count == 0

def test_initialization_with_default_rating():
    item = LibraryItem( name: "Song B",  artist: "Artist B")
    assert item.name == "Song B"
    assert item.artist == "Artist B"
    assert item.rating == 0
    assert item.play_count == 0
```

Figure 35: Using pytest for testing LibraryItem

And after running the program, the results were extremely positive.

```
============================= test session starts ==============================
collecting ... collected 9 items

library_test.py::test_initialization PASSED                              [ 11%]
library_test.py::test_initialization_with_default_rating PASSED          [ 22%]
library_test.py::test_info PASSED                                        [ 33%]
library_test.py::test_stars PASSED                                       [ 44%]
library_test.py::test_stars_with_zero_rating PASSED                      [ 55%]
library_test.py::test_stars_with_full_rating PASSED                      [ 66%]
library_test.py::test_play_count_initially_zero PASSED                   [ 77%]
library_test.py::test_increment_play_count PASSED                        [ 88%]
library_test.py::test_set_rating PASSED                                  [100%]

============================== 9 passed in 0.61s ===============================
```

Figure 36: Result of test suite

# 4.2. GUI Test

## *4.2.1. View Track*

Test case for screen view track.

**Table 1: Test case of view track**

| Test Case | Sample Input | Sample Action | Expected Output | Actual Output |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Test 1: View Tracks button (valid track number) | Track number: 1 (valid) | Enter a track number (e.g., 1) and click "View Track" | Display the track details in the track text area (e.g., name, artist, rating, play count). | Track details are displayed correctly (e.g., "Track 1 details"). |
| Test 2: View Tracks button (invalid track number) | Track number: 999 (invalid) | Enter a track number (e.g., 999) and click "View Track" | Display "Track 999 not found" in the track text area. | "Track 999 not found" is displayed. |
| Test 3: Listen Online button (with valid track) | Track number: 1 | Enter a valid track number (e.g., 1), click "View Track", and click "Listen Online" | Open a web browser with the Last.fm page for the track (e.g., https://www.last.fm/music/Track1). | Browser opens with the Last.fm page for the track. |
| Test 4: Listen Online button (without track name) | Track number: 999 (invalid) | Enter an invalid track number (e.g., 999), click "View Track", and click "Listen Online" | Disable button “Listen Online” | Disable button “Listen Online” |
| Test 5: Status Label after View Track | Track number: 1 | Enter a valid track number (e.g., 1), click "View Track" | Status label shows "View Track button was clicked!" | Status label shows the correct message. |
| Test 6: Status Label after List All Tracks | N/A | Click "List All Tracks" | Status label shows "List Tracks button was clicked!" | Status label shows the correct message. |

### *4.2.2. Create Track*

Test cases of create track screen

**Table 2: Test cass of create track**

| Test Case | Sample Input | Sample Action | Expected Output | Actual Output |
|---|---|---|---|---|
| Test 1: Add Track (valid input) | Track Number: 1, Name: "Song 1", Artist: "Artist 1", Rating: 5 | Enter the details and click "Add Track" | The track is added to the library, and the track list is updated. No error message shown. | Track is added and list is updated. |
| Test 2: Add Track (missing input) | Track Number: 1, Name: "", Artist: "Artist 1", Rating: 5 | Enter incomplete details and click "Add Track" | Show a warning message: "Please fill in all fields!" | Warning message is displayed. |
| Test 3: Add Track (invalid rating) | Track Number: 1, Name: "Song 1", Artist: "Artist 1", Rating: "abc" | Enter invalid rating and click "Add Track" | Show a warning message: "Rating must be a number!" | Warning message is displayed. |
| Test 4: Add Track (duplicate track number) | Track Number: 1, Name: "Song 2", Artist: "Artist 2", Rating: 4 | Try to add a track with an existing track number (e.g., 1) | Show an error message: "Track number '1' already exists!" | Error message is displayed. |
| Test 5: Clear Input | N/A | Click "Clear Input" | All input fields (Track Number, Name, Artist, Rating) are cleared. | Input fields are cleared. |
| Test 6: Play List (valid tracks) | N/A | Click "Play List" | Show a message box displaying the play count for each track and the updated play count after increment. | Message box with play count is displayed. |
| Test 7: Play List (empty library) | N/A | Click "Play List" when no tracks are added | Show an empty message box or a message indicating no tracks are available. | Message box is displayed, indicating empty library. |

| | | | | |
|---|---|---|---|---|
| Test 8: Reset Library | N/A | Click "Reset" | Library is cleared, track list is emptied, and a success message ("Reset successfully!") is displayed. | Library is reset, and message is displayed. |
| Test 9: Reload Library | N/A | Click "Reload" | The track list is reloaded from the library and displayed in the text area. | Track list is reloaded and displayed. |
| Test 10: Get Top Tracks (no artist selected) | N/A | Click "Get Top Tracks" without selecting an artist | Show an error message: "Please select artist first!" | Error message is displayed. |
| Test 11: Get Top Tracks (valid artist) | Select "Bruno Mars" | Select an artist (e.g., "Bruno Mars") and click "Get Top Tracks" | The top tracks for the selected artist are added to the library and the track list is updated. | Top tracks are displayed for the selected artist. |
| Test 12: Get Top Tracks (empty list) | N/A | Select an artist and click "Get Top Tracks" when no tracks exist for that artist | Show a message box with no tracks found or an empty list in the track list area. | No tracks are displayed or empty message. |

### *4.2.3. Update Track*

Test case of update track screen

**Table 3: Test cases of update track screen**

| **Test Case** | **Sample Input** | **Sample Action** | **Expected Output** | **Actual Output** |
|---|---|---|---|---|
| Test 1: Update Rating (valid input) | Track Number: "1", New Rating: "8" | Enter valid track number and rating, then click "Update" | The track rating is updated, and a message box with track details is displayed. | Rating is updated and message is displayed. |

| | | | | |
|---|---|---|---|---|
| Test 2: Update Rating (missing input) | Track Number: "", New Rating: "8" | Leave one or both fields empty and click "Update" | Show an error message: "Please fill in both fields." | Error message is displayed. |
| Test 3: Update Rating (invalid rating) | Track Number: "1", New Rating: "abc" | Enter invalid rating (e.g., "abc") and click "Update" | Show an error message: "Rating should be an integer." | Error message is displayed. |
| Test 4: Update Rating (out of range) | Track Number: "1", New Rating: "15" | Enter a rating out of range (e.g., "15") and click "Update" | Show an error message: "Rating should be between 1 and 10." | Error message is displayed. |
| Test 5: Update Rating (track not found) | Track Number: "999", New Rating: "8" | Enter a track number that doesn't exist and click "Update" | Show an error message: "Track 999 not found." | Error message is displayed. |
| Test 6: Update Rating (track exists) | Track Number: "1", New Rating: "7" | Enter an existing track number and rating, then click "Update" | The track rating is updated, and the track list is refreshed. A message box is displayed with updated details. | Track rating is updated and message is displayed. |
| Test 7: List Tracks | N/A | Click "List Tracks" button | The track list is displayed in the text area. | Track list is displayed correctly. |
| Test 8: Empty Library (update rating) | Track Number: "1", New Rating: "8" | Try to update rating when the library is empty | Show an error message: "Track 1 not found." | Error message is displayed (library is empty). |
| Test 9: Successful Rating Update Display | Track Number: "2", New Rating: "9" | Enter track number and new rating and click "Update" | After the update, the track details including track name, artist, new rating, and play count should be shown. | Track details are shown in the info message box. |

# 5. Conclusion

The simple music listening application developed in this project successfully integrates essential functionalities for managing and interacting with a track library. By utilizing Tkinter for the graphical user interface (GUI), the application allows users to perform a range of actions such as viewing all tracks, adding new tracks, updating track ratings, and playing a playlist. The key features, including track management, user input validation, and dynamic updates, ensure a seamless user experience.

In particular, the application showcases a robust implementation of the following functions:

- Track Listing: Users can view the list of all available tracks.
- Track Addition: New tracks can be added to the library with details such as track number, name, rating, and artist.
- Track Rating Update: Users can update the rating of a track, with built-in validation to ensure that inputs are correct and within the required range.
- Play List: The "Play List" functionality tracks play counts and updates them in the library.

Additionally, the application employs error handling mechanisms, such as warnings for missing fields or invalid inputs (non-numeric ratings, out-of-range values, etc.), ensuring that users are guided through proper usage. The interface also includes a combobox for selecting artists, which aids in fetching top tracks from external sources.

Overall, the application fulfills its primary objective of providing a straightforward and user-friendly interface for managing music tracks. While it serves as a basic music library and player, there is potential for future enhancements, such as integrating additional music sources, improving the GUI design, and incorporating features like track search and advanced playlist management.

This project has demonstrated key software engineering practices, including modular code structure, input validation, and interactive GUI development, making it a good foundation for further development in the realm of music applications.