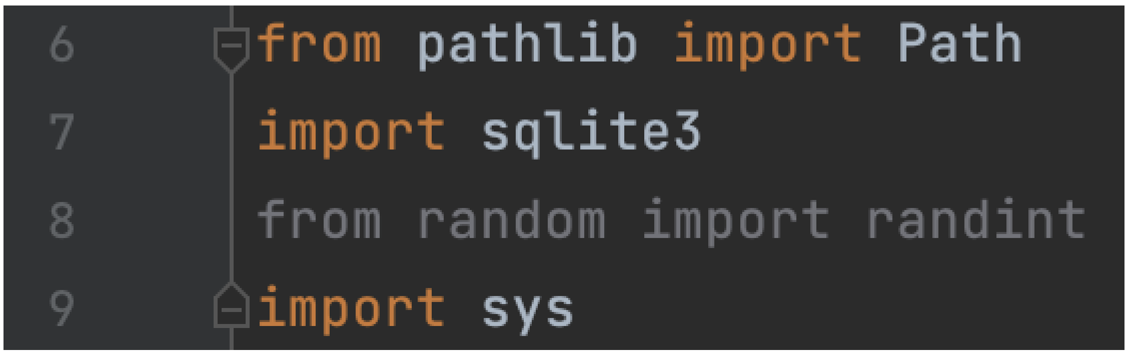
LIST OF TECHNIQUES USED

- Additional libraries
- Parsing a CSV file
- 2D arrays
- Recursive functions
- Database creation
- Inputting data into a database
- Data extraction from a database
- Exception handling
- While & For loops
- Simple & complex selection

Use of Additional Libraries



```python
from pathlib import Path
import sqlite3
from random import randint
import sys
```

"Pathib" is used in this program to detect whether the database file required for the program to function properly was already created. This helps avoid the SQLite3 error, "table already exists."

The "sqlite3" library was imported so databases could be created and handled in the Python program.

The "randint" function from the "random" library was used in the '6. generate playlist' and '7. have the program guess your favourite artist or genre' options from the menu.

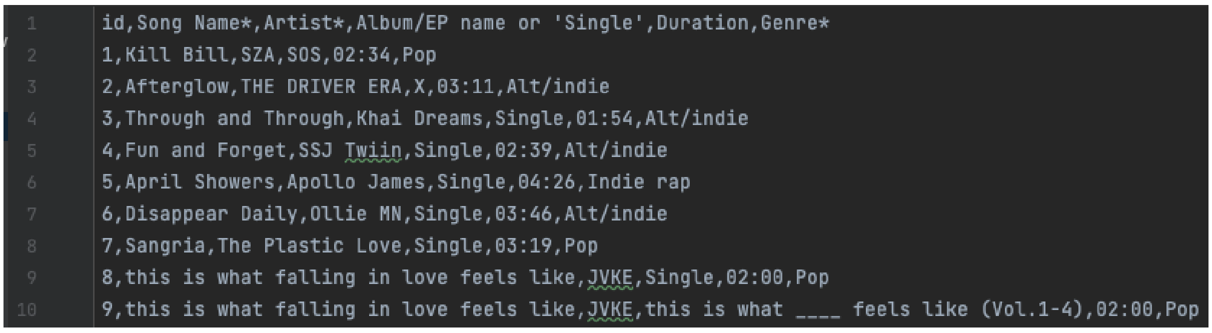
In the 'generate a playlist' option, randint is used to generate a random playlist length if the user chooses to do so. Randint is also used to select random song IDs from the database to fill the playlist or to select guesses of the user's favourite artists or genres.

"Sys" was imported so the program could exit using "sys.exit()" regardless of the Python version the user uses as not all Python versions have exit() built in.

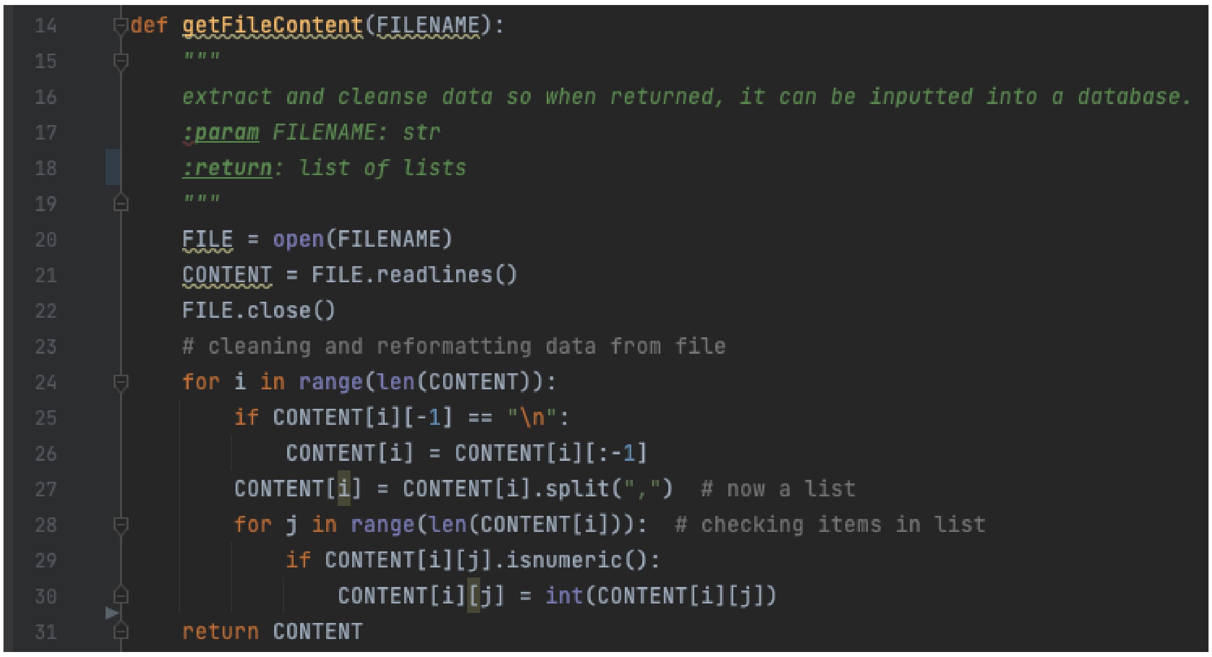
Parsing a CSV file

The spreadsheet provided by the client was converted to a CSV file so the program could extract its data and input it into a database.

This is the function used to extract the lines of text from the CSV file (as shown).



```
id,Song Name*,Artist*,Album/EP name or 'Single',Duration,Genre*
1,Kill Bill,SZA,SOS,02:34,Pop
2,Afterglow,THE DRIVER ERA,X,03:11,Alt/indie
3,Through and Through,Khai Dreams,Single,01:54,Alt/indie
4,Fun and Forget,SSJ Twiin,Single,02:39,Alt/indie
5,April Showers,Apollo James,Single,04:26,Indie rap
6,Disappear Daily,Ollie MN,Single,03:46,Alt/indie
7,Sangria,The Plastic Love,Single,03:19,Pop
8,this is what falling in love feels like,JVKE,Single,02:00,Pop
9,this is what falling in love feels like,JVKE,this is what ____ feels like (Vol.1-4),02:00,Pop
```



```python
def getFileContent(FILENAME):
    """
    extract and cleanse data so when returned, it can be inputted into a database.
    :param FILENAME: str
    :return: list of lists
    """
    FILE = open(FILENAME)
    CONTENT = FILE.readlines()
    FILE.close()
    # cleaning and reformatting data from file
    for i in range(len(CONTENT)):
        if CONTENT[i][-1] == "\n":
            CONTENT[i] = CONTENT[i][:-1]
        CONTENT[i] = CONTENT[i].split(",")  # now a list
        for j in range(len(CONTENT[i])):  # checking items in list
            if CONTENT[i][j].isnumeric():
                CONTENT[i][j] = int(CONTENT[i][j])
    return CONTENT
```

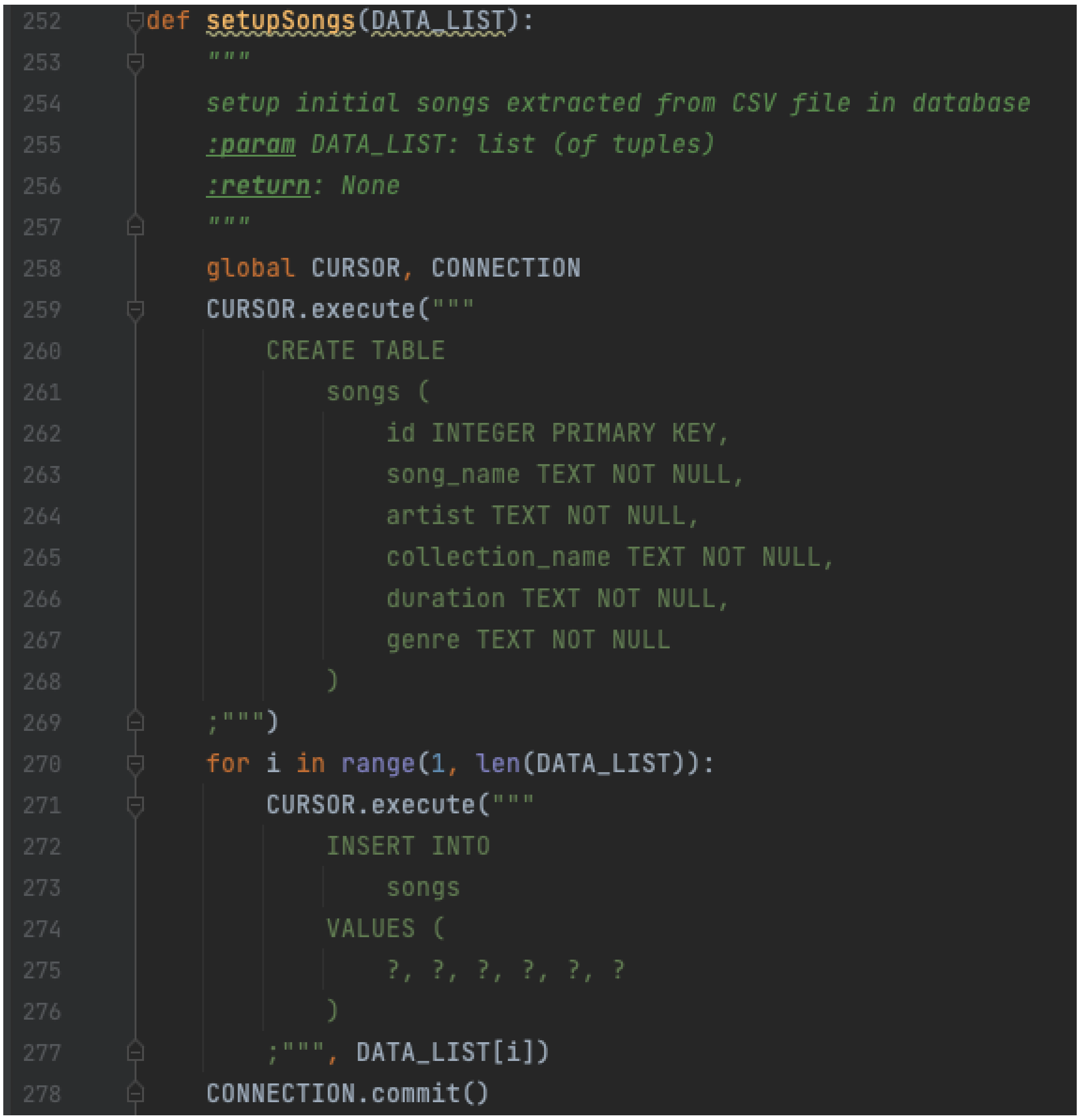
The file is opened and its content extracted using ".readLines()" rather than using ".read()" or ".readLine()".

The ".readLines()" function converts every line of text to a list item. In contrast, the ".read()" function returns the file content as a long string, which would require more manipulation (split by "\n" and "," versus only split by "," in .readLines()). As well, ".readLine()" is not used since it only takes the first line of data, or only one song's information, from the file.

Data cleansing in this function includes removing the "\n" at the end of all but the last line. Cleansing also includes changing all number-only strings into integers, which also ensures that data matches their constraints in the database (specifically song ID being an integer). After extracting, data-cleansing, and closing the file, the function returns a **2D array**.

Database creation

One database was created for this program, and it only contains one table (with the same name). This table stores all information about songs, including each song's name, artist, its collection name or whether it's a single, its duration, and its genre.

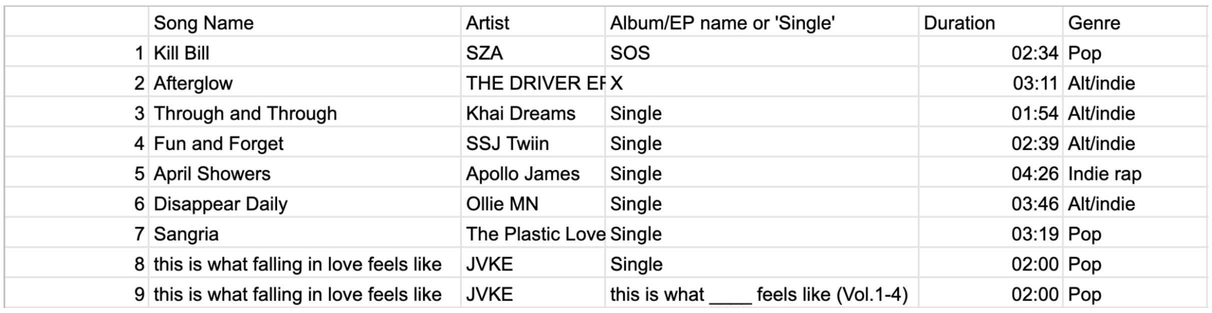


```python
def setupSongs(DATA_LIST):
    """
    setup initial songs extracted from CSV file in database
    :param DATA_LIST: list (of tuples)
    :return: None
    """
    global CURSOR, CONNECTION
    CURSOR.execute("""
        CREATE TABLE
            songs (
                id INTEGER PRIMARY KEY,
                song_name TEXT NOT NULL,
                artist TEXT NOT NULL,
                collection_name TEXT NOT NULL,
                duration TEXT NOT NULL,
                genre TEXT NOT NULL
            )
    ;""")
    for i in range(1, len(DATA_LIST)):
        CURSOR.execute("""
            INSERT INTO
                songs
            VALUES (
                ?, ?, ?, ?, ?, ?
            )
        ;""", DATA_LIST[i])
    CONNECTION.commit()
```

This is the function that creates the table and thus, the database. The song ID is the primary key of the table. None of the other columns of data collected can be used as the primary key: songs can have the same name (whether they're from the same artist or not), artists can write multiple songs, multiple singles can be added, albums often consist of many songs, songs completely independent of each other can have the same duration, and songs can have the same genre. This makes the inclusion of an "id" column for songs appropriate as it ensures all songs are recognized as unique (and properly counts the total number of songs in the database since each song in the database has a unique, sequential number associated with it because of this column).

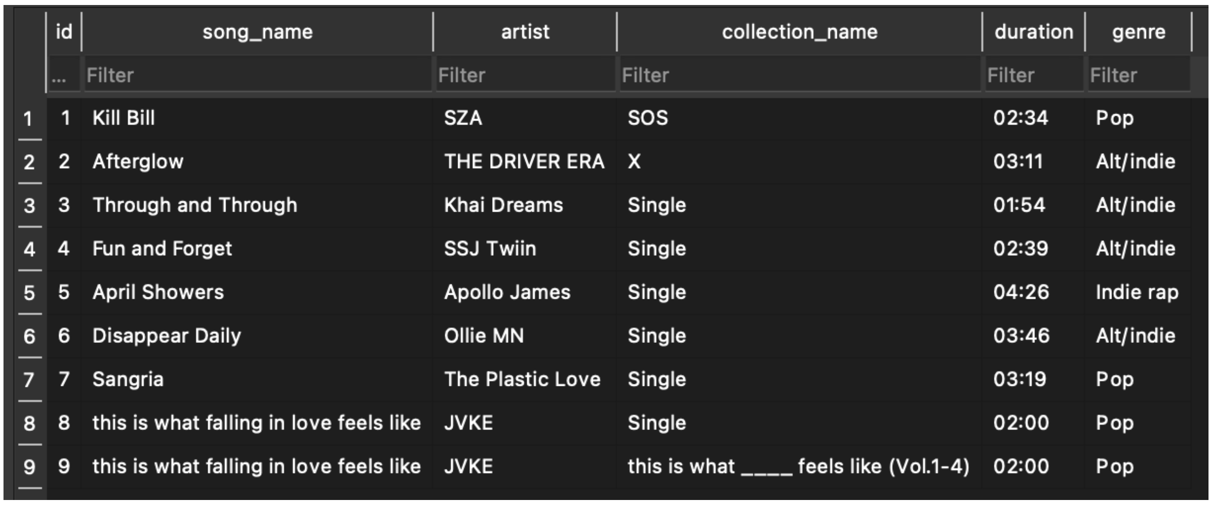
The “NOT NULL” constraints are in place to correspond to the fact that no song can lack those pieces of information and to comply with database normalization. Duration is saved as text in the database as entering this information in a more familiar format (‘XX:XX’) is more efficient for the user and acceptable since this data won’t be used in mathematical calculations in this program.

Immediately after its extraction from the CSV file, the data is inputted into the newly created database. As well, notice the “?” used in the INSERT INTO SQL query. This method of sanitizing and then inserting data ensures data validation in the database.



	Song Name	Artist	Album/EP name or 'Single'	Duration	Genre
1	Kill Bill	SZA	SOS	02:34	Pop
2	Afterglow	THE DRIVER EF	X	03:11	Alt/indie
3	Through and Through	Khai Dreams	Single	01:54	Alt/indie
4	Fun and Forget	SSJ Twiin	Single	02:39	Alt/indie
5	April Showers	Apollo James	Single	04:26	Indie rap
6	Disappear Daily	Ollie MN	Single	03:46	Alt/indie
7	Sangria	The Plastic Love	Single	03:19	Pop
8	this is what falling in love feels like	JVKE	Single	02:00	Pop
9	this is what falling in love feels like	JVKE	this is what ____ feels like (Vol.1-4)	02:00	Pop

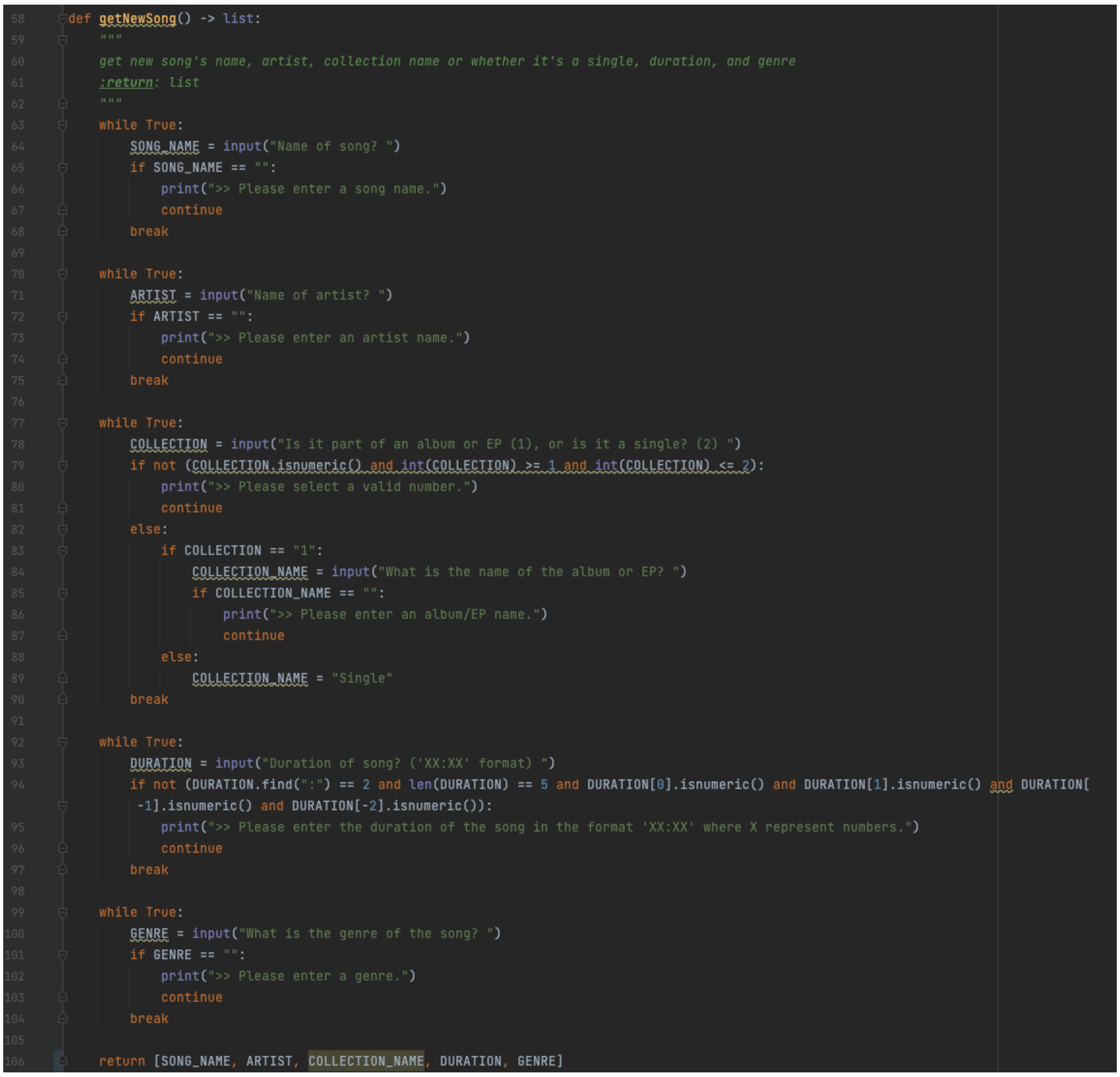
This is the spreadsheet file provided by the client, which was converted into a CSV file, extracted by the program, and added to the database (which is seen in the image below).



	id	song_name	artist	collection_name	duration	genre
	...	Filter	Filter	Filter	Filter	Filter
1	1	Kill Bill	SZA	SOS	02:34	Pop
2	2	Afterglow	THE DRIVER ERA	X	03:11	Alt/indie
3	3	Through and Through	Khai Dreams	Single	01:54	Alt/indie
4	4	Fun and Forget	SSJ Twiin	Single	02:39	Alt/indie
5	5	April Showers	Apollo James	Single	04:26	Indie rap
6	6	Disappear Daily	Ollie MN	Single	03:46	Alt/indie
7	7	Sangria	The Plastic Love	Single	03:19	Pop
8	8	this is what falling in love feels like	JVKE	Single	02:00	Pop
9	9	this is what falling in love feels like	JVKE	this is what ____ feels like (Vol.1-4)	02:00	Pop

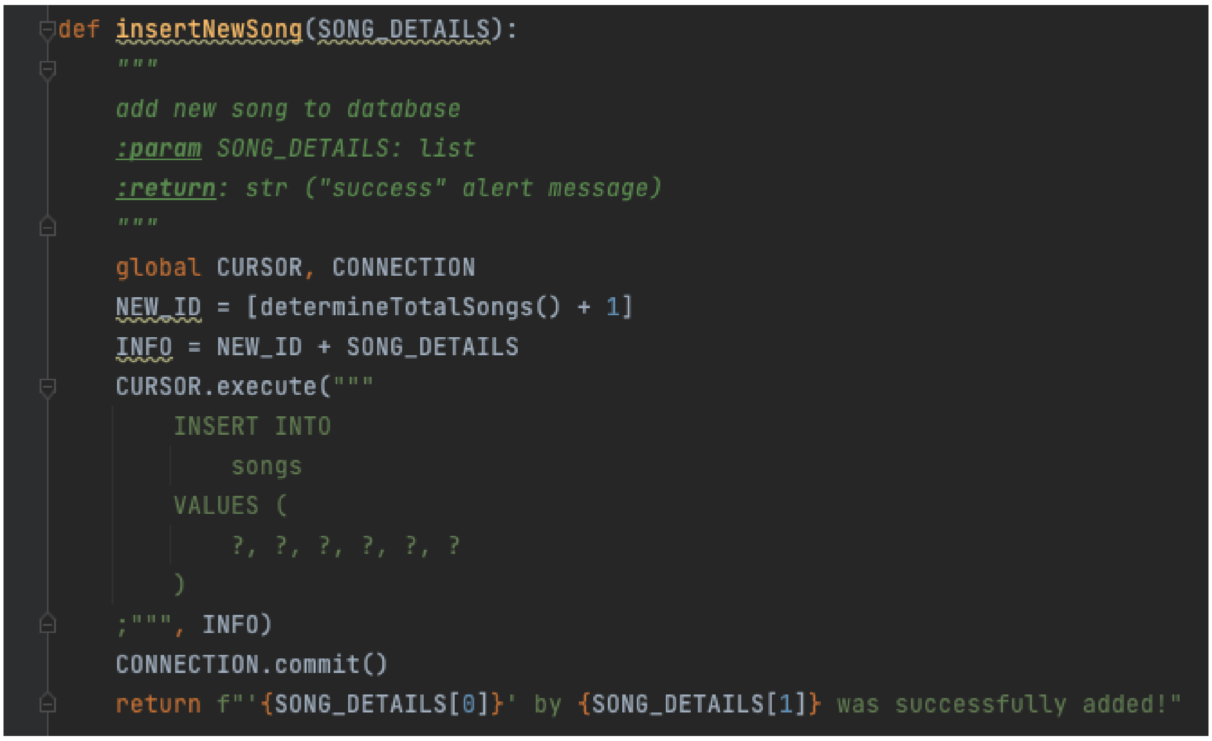
Inputting data into the database

Often, the user’s input adds to or updates information in the database. Here is one such example. The user has selected the option to add a new song to the database. This is the subroutine used to obtain the new song’s information from the user.

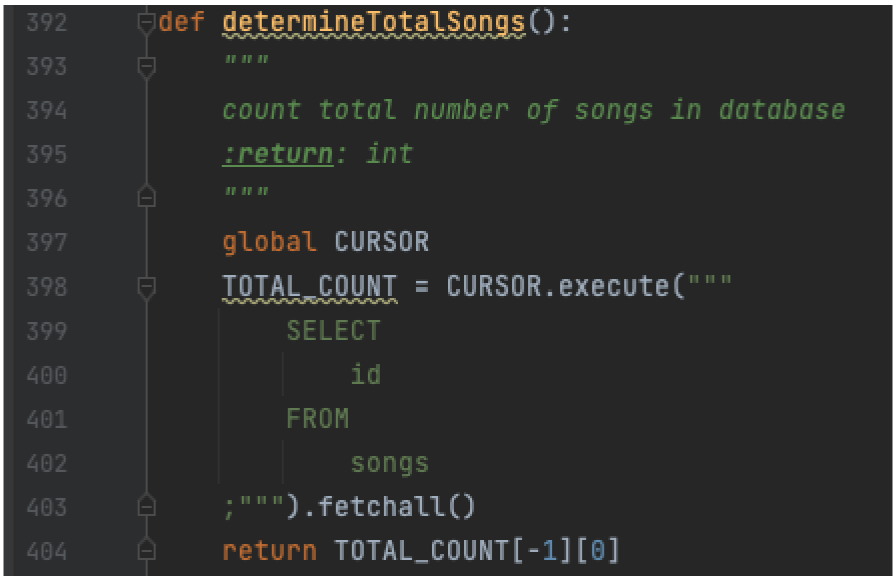


```python
def getNewSong() -> list:
    """
    get new song's name, artist, collection name or whether it's a single, duration, and genre
    :return: list
    """
    while True:
        SONG_NAME = input("Name of song? ")
        if SONG_NAME == "":
            print(">> Please enter a song name.")
            continue
        break

    while True:
        ARTIST = input("Name of artist? ")
        if ARTIST == "":
            print(">> Please enter an artist name.")
            continue
        break

    while True:
        COLLECTION = input("Is it part of an album or EP (1), or is it a single? (2) ")
        if not (COLLECTION.isnumeric() and int(COLLECTION) >= 1 and int(COLLECTION) <= 2):
            print(">> Please select a valid number.")
            continue
        else:
            if COLLECTION == "1":
                COLLECTION_NAME = input("What is the name of the album or EP? ")
                if COLLECTION_NAME == "":
                    print(">> Please enter an album/EP name.")
                    continue
            else:
                COLLECTION_NAME = "Single"
        break

    while True:
        DURATION = input("Duration of song? ('XX:XX' format) ")
        if not (DURATION.find(":") == 2 and len(DURATION) == 5 and DURATION[0].isnumeric() and DURATION[1].isnumeric() and DURATION[
         -1].isnumeric() and DURATION[-2].isnumeric()):
            print(">> Please enter the duration of the song in the format 'XX:XX' where X represent numbers.")
            continue
        break

    while True:
        GENRE = input("What is the genre of the song? ")
        if GENRE == "":
            print(">> Please enter a genre.")
            continue
        break

    return [SONG_NAME, ARTIST, COLLECTION_NAME, DURATION, GENRE]
```

To ensure that the “NOT NULL” constraint on every column of data in the database is met, every piece of user input is individually checked for blankness. Individually checking with multiple **while loops** rather than one large loop makes entering data easier and more efficient for the user since making an error in one field won’t require the user to re-enter data for all fields before. **Complex selection** is also employed in this function to determine whether to ask the user for an album/EP name after they’ve indicated whether the song is a single or not since it would be useless to ask the user the album/EP name after they’ve indicated it’s a single.



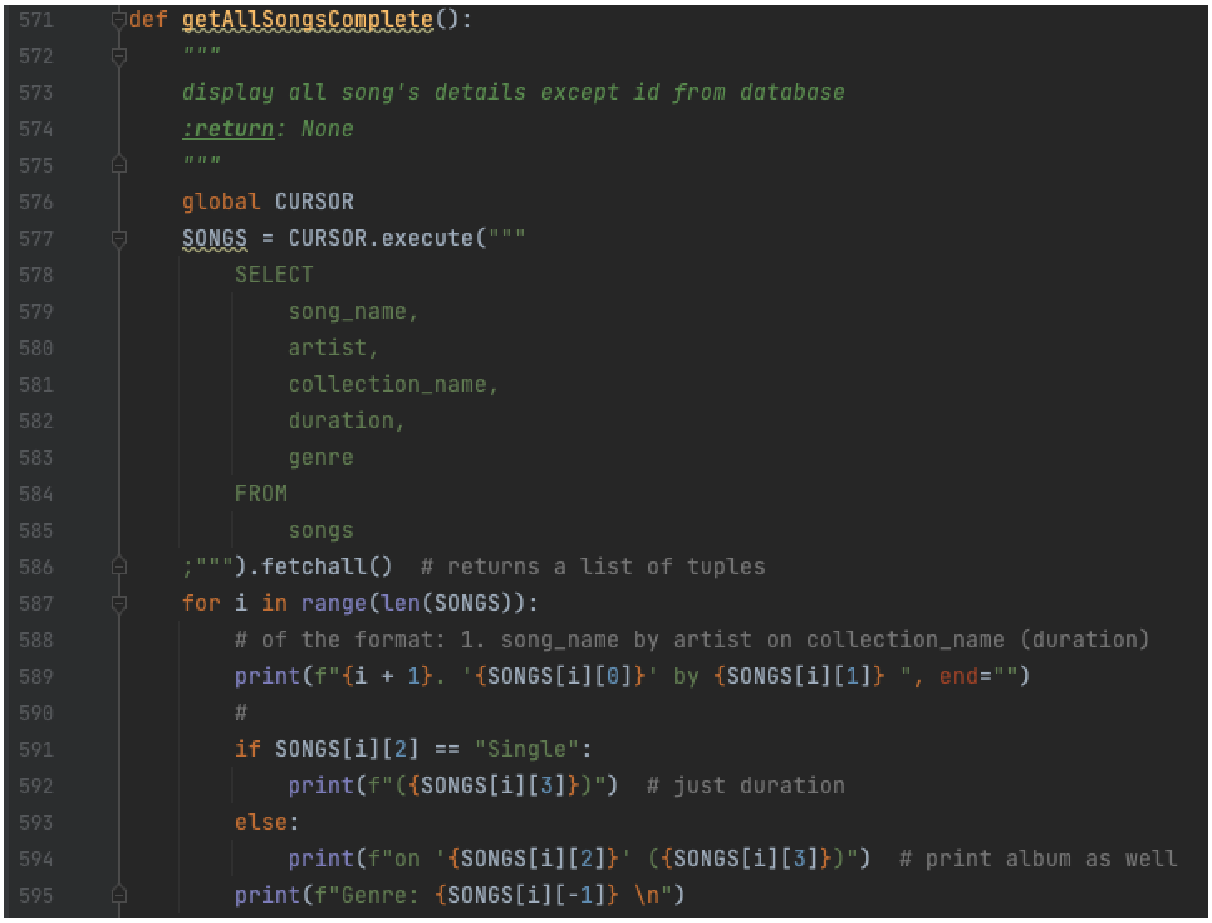
```python
def insertNewSong(SONG_DETAILS):
    """
    add new song to database
    :param SONG_DETAILS: list
    :return: str ("success" alert message)
    """
    global CURSOR, CONNECTION
    NEW_ID = [determineTotalSongs() + 1]
    INFO = NEW_ID + SONG_DETAILS
    CURSOR.execute("""
        INSERT INTO
            songs
        VALUES (
            ?, ?, ?, ?, ?, ?
        )
    ;""", INFO)
    CONNECTION.commit()
    return f"'{SONG_DETAILS[0]}' by {SONG_DETAILS[1]} was successfully added!"
```



```python
def determineTotalSongs():
    """
    count total number of songs in database
    :return: int
    """
    global CURSOR
    TOTAL_COUNT = CURSOR.execute("""
        SELECT
            id
        FROM
            songs
    ;""").fetchall()
    return TOTAL_COUNT[-1][0]
```

The program also ensures efficiency by automatically attaching an ID to the new song. It does so by calculating the total number of songs in the database and then adding one onto that integer. It then converts that value into a list so that it can be concatenated with the list of the new song information just received as user input.

Data extraction from the database



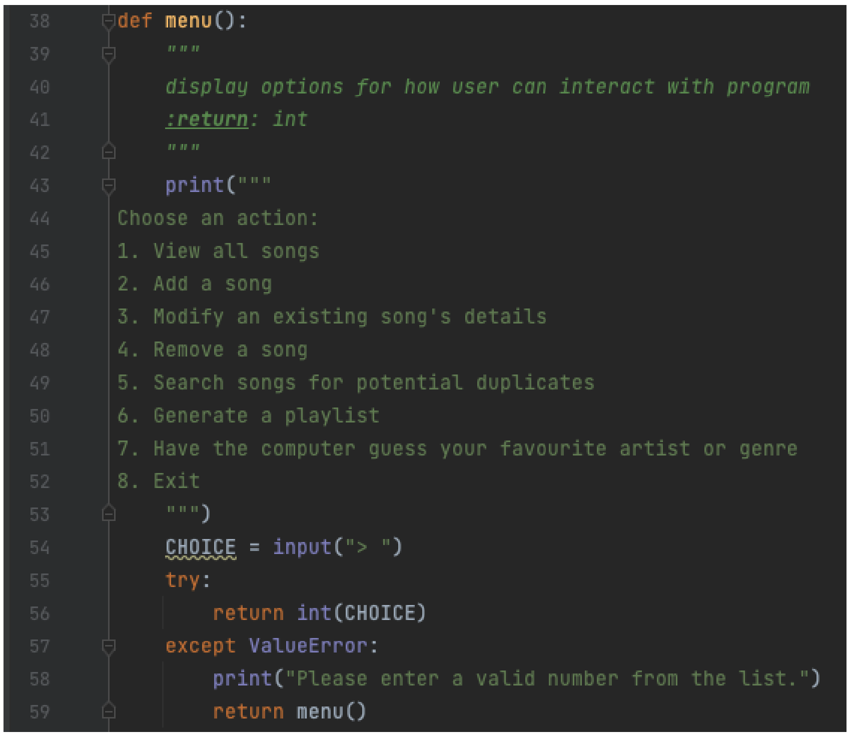
```python
def getAllSongsComplete():
    """
    display all song's details except id from database
    :return: None
    """
    global CURSOR
    SONGS = CURSOR.execute("""
        SELECT
            song_name,
            artist,
            collection_name,
            duration,
            genre
        FROM
            songs
    ;""").fetchall()  # returns a list of tuples
    for i in range(len(SONGS)):
        # of the format: 1. song_name by artist on collection_name (duration)
        print(f"{i + 1}. '{SONGS[i][0]}' by {SONGS[i][1]} ", end="")
        #
        if SONGS[i][2] == "Single":
            print(f"({SONGS[i][3]})")  # just duration
        else:
            print(f"on '{SONGS[i][2]}' ({SONGS[i][3]})")  # print album as well
        print(f"Genre: {SONGS[i][-1]} \n")
```

In order to display all songs to the user if they select the first menu option ('view all songs'), song information must be retrieved. The ".fetchall()" SQLite3 command returns a **2D array** consisting of every row's song name, artist, collection name, duration, and genre.

The individual contents of this **2D array** are then displayed using a **for loop**. **Simple selection** (a singular if--else) is also employed to format the collected information: if a song is part of an album or EP, the album/EP name is printed; otherwise, no information about the collection is displayed.

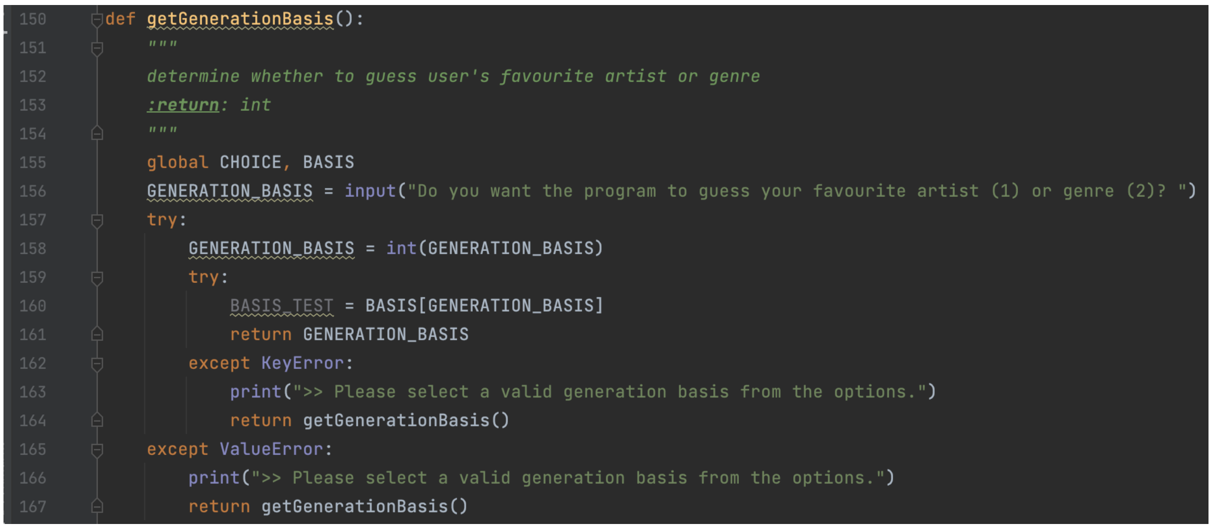
Recursive Functions & Exception Handling

Recursion and exception handling are used to ensure that the user is giving the program the expected input.



```
38  def menu():
39      """
40      display options for how user can interact with program
41      :return: int
42      """
43      print("""
44  Choose an action:
45  1. View all songs
46  2. Add a song
47  3. Modify an existing song's details
48  4. Remove a song
49  5. Search songs for potential duplicates
50  6. Generate a playlist
51  7. Have the computer guess your favourite artist or genre
52  8. Exit
53      """)
54      CHOICE = input("> ")
55      try:
56          return int(CHOICE)
57      except ValueError:
58          print("Please enter a valid number from the list.")
59          return menu()
```

In the menu, if the user enters a non-numeric character, they are asked to enter a number since line 56 will raise the *ValueError* the try/except is handling, and the menu() will return itself. The **while loop** used in the main program code ensures that all numbers entered into this function are ones from the list, for if a number not within 1-8 is entered, the menu() function is run again.



```
150 def getGenerationBasis():
151     """
152     determine whether to guess user's favourite artist or genre
153     :return: int
154     """
155     global CHOICE, BASIS
156     GENERATION_BASIS = input("Do you want the program to guess your favourite artist (1) or genre (2)? ")
157     try:
158         GENERATION_BASIS = int(GENERATION_BASIS)
159         try:
160             BASIS_TEST = BASIS[GENERATION_BASIS]
161             return GENERATION_BASIS
162         except KeyError:
163             print(">> Please select a valid generation basis from the options.")
164             return getGenerationBasis()
165     except ValueError:
166         print(">> Please select a valid generation basis from the options.")
167         return getGenerationBasis()
```

Similarly, in this subroutine, to ensure the user enters the number 1 or 2, two try/except blocks are used. The outer try/except (line 157 and 165) ensures that the user enters a number; the inner one (line 159 and 162) ensures that the user entered either 1 or 2 as those are the only keys in the BASIS dictionary. If the user’s input raises either error being handled, the subroutine returns itself so the user can fix their input.

Word Count: 1198