

J276/03 – Programming Task – TASK 1

Micah Waring 11S - 7714

Contents

1	Design/Analysis	1
1.1	Background	1
1.2	Success Criteria	1
1.3	Login/Registration Program Flow (Flowchart)	2
1.4	Main Program Algorithm (Pseudocode)	3
1.5	User Interface	5
1.6	Variables	6
1.7	Testing	7
2	Development	8
2.1	Setup	9
2.2	The Welcome form	11
2.3	Registering	11
2.4	Logging in	12
2.5	The Home form	12
2.6	The Game	12
3	Evaluation	14
3.1	SQLite Database	14
3.2	The Songs	14
3.3	Additional Comments	15
4	References	15

1 Design/Analysis

1.1 Background

This objective of this task is to create a music game in which the user attempt to guess the name of a song given the first letter of each word in the song and the artist name.

The brief is as follows:

Noel is creating a music quiz game.

The game stores a list of song names and their artist (e.g. the band or solo artist name). The player needs to try and guess the song name. The game is played as follows:

- A random song name and artist are chosen.
- The artist and the first letter of each word in the song title are displayed.
- The user has two chances to guess the name of the song.
- If the user guesses the answer correctly the first time, they score 3 points. If the user guesses the answer correctly the second time they score 1 point. The game repeats.
- The game ends when a player guesses the song name incorrectly the second time.

Only authorised players are allowed to play the game.

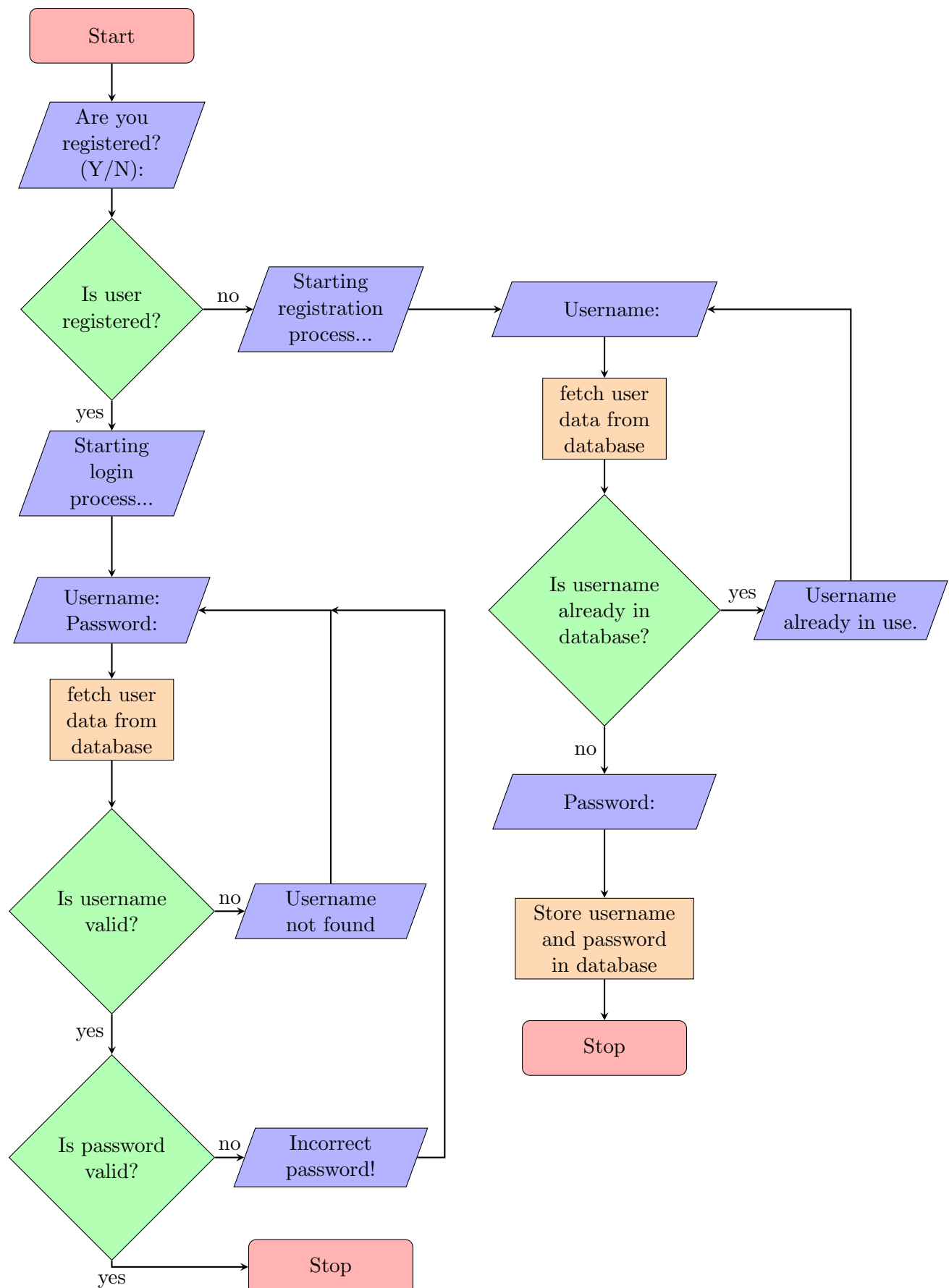
Where appropriate, input from the user should be validated.

After spending time looking at the different options for programming languages for this project, I have decided to use a python console application, mainly due to my familiarity with the language but also because the code is simply in a single .py file, as opposed to a whole project directory.

1.2 Success Criteria

1. Create an accounts system that will have:
 - (a) A predefined list of users,
 - (b) Functionality to login users.
 - (c) Additional functionality to register new users.
 - (d) Store all data in an SQLite3 local database file.
2. Add a list of songs:
 - (a) Stored in a CSV file,
 - (b) Contains song information for the top 50 in the charts for every month from Jan 2016 to March 2019.
3. Create the main game functionality that will:
 - (a) Give the user the first character of each word in the song and the artists,
 - (b) Give the user 2 guesses to obtain the correct answer,
 - (c) Give the user a score based on how many guesses they took,
 - (d) End the game if the user uses up both guesses,
 - (e) Store user score in SQLite3 local database file, in the leaderboard and the best score in the users table.

1.3 Login/Registration Program Flow (Flowchart)



1.4 Main Program Algorithm (Pseudocode)

```

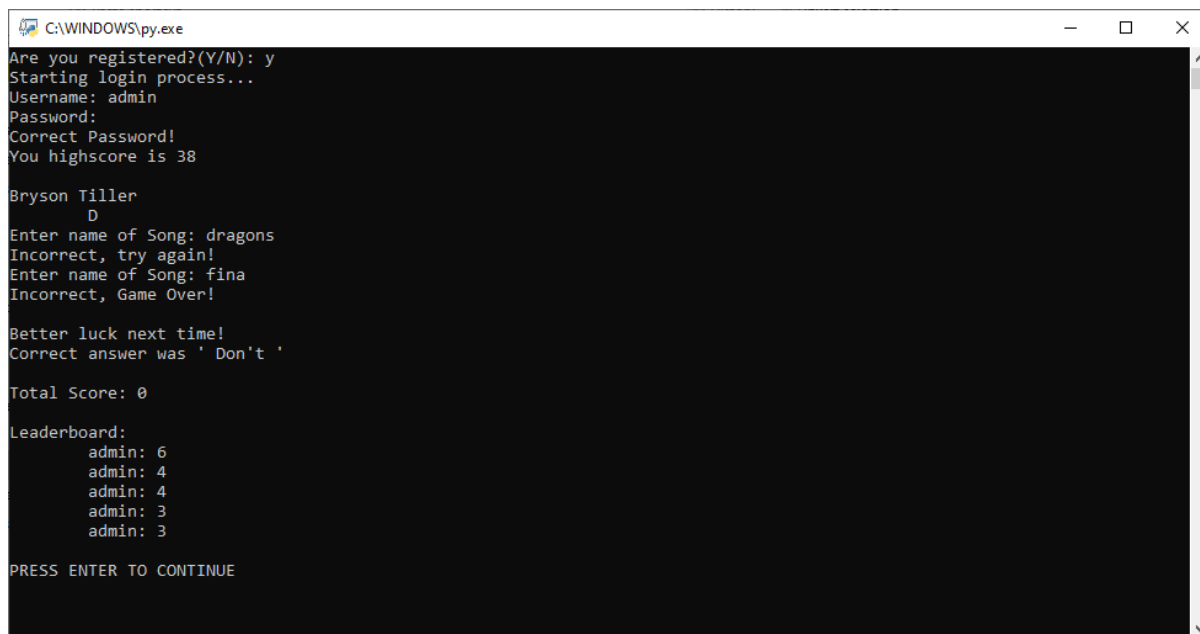
1 song_List="chart2000-songmonth-0-3-0050.csv"
2 data = READ song_List into LIST of DICTIONARIES
3 length = data.length - 1
4
5 SQLite_Cursor = DB cursor object for "users.sqlite3"
6 tuple fromDB = ()
7
8 PROCEDURE onExit()
9     close SQLite connection
10 ENDPROCEDURE
11 add onExit to program close event
12
13 registered = ""
14
15 WHILE registered != "y" AND registered != "n"
16     registered = input("Are you registered?(Y/N): ").lower()
17 ENDWHILE
18
19 IF registered == "y" THEN
20     print("Starting login process...")
21     correct_Login = false
22     WHILE NOT correct_Login
23         password = getpass.getpass()
24         fromDB = select first line from 'users' with given username
25         IF fromDB == None THEN
26             print("Username not found")
27             continue
28         ENDIF
29         IF NOT passhash.verify_password(fromDB[2], password) THEN
30             print("Incorrect Password")
31             continue
32         ENDIF
33         print("Correct Password!")
34         correct_Login = True
35     END WHILE
36 ELSE
37     print("Starting registration process...")
38     valid_Registration = false
39     WHILE NOT valid_Registration
40         username = input("Username: ")
41         fromDB = select users from DB with given username
42         IF fromDB != None THEN
43             print("Username already in use.")
44             continue
45         ENDIF
46         password = getpass.getpass()
47         insert new user data
48         correct_Login = true
49     ENDWHILE
50 ENDIF
51
52 bestScore=bestScore column value from logged in user's row from SQLite DB
53 print("You  highscore  is %s\n" % bestScore)
54 score=0
55 playing=True
56
57 WHILE playing
58
59     chosen_Data=data[randomInt(0, length)]
60     songName=chosen_Data['song']
61     output=chosen_Data['artist'] + "\n\t" + First letter of every word in

```

```
        songName with tab delimiter
62
63     print(output)
64
65     FOR i=1 TO 2
66         user_Input=input("Enter name of Song: ")
67         IF user_Input == chosen_Data['Song'] THEN
68             toadd = (-2*i)+5
69             score+=toadd
70             print("+ ",toadd)
71             break
72         ELSE
73             IF i == 2 THEN
74                 print("Incorrect , Game Over!\n\nBetter luck next time!")
75                 print("Correct answer was      ", chosen_Data["song"],"      \n")
76                 playing = False
77             ELSE
78                 print("Incorrect , try again!")
79             ENDIF
80         ENDIF
81     NEXT i
82     print("Total Score: " + str(score) + "\n")
83
84 END WHILE
85
86 Insert score data for user into SQLite DB
87 leaderboard=Get score leaderboard from SQLite DB
88 leaderboard=sorts leaderboard by score, best first
89
90 top_five=top 5 list of '<username>: score' from leaderboard
91 print("Leaderboard: ")
92
93 FOR EACH lScore AS string IN top_five
94     print("\t", lScore)
95 NEXT lScore
96
97 IF score > bestScore THEN
98     Update DB to contain user's new bestScore
99 ENDIF
100
101 Pause console
```

1.5 User Interface

For the user interface, I will be using a generic python console application. The reason for this is that, in my opinion, a GUI in python wouldn't be worth it for what this program is, as it functions perfectly well without over-complicating it. That being said, a GUI could be implemented to further improve the application.



```
C:\WINDOWS\py.exe
Are you registered?(Y/N): y
Starting login process...
Username: admin
Password:
Correct Password!
You highscore is 38

Bryson Tiller
D
Enter name of Song: dragons
Incorrect, try again!
Enter name of Song: fina
Incorrect, Game Over!

Better luck next time!
Correct answer was ' Don't '

Total Score: 0

Leaderboard:
    admin: 6
    admin: 4
    admin: 4
    admin: 3
    admin: 3

PRESS ENTER TO CONTINUE
```

Figure 1: Example of user interface in the python console.

1.6 Variables

Variables	Type	Purpose
song_List	Constant String	Relative path of csv file containing song data.
data	Constant List of orderedDictionaries	Data read in from the csv file.
length	Constant Integer	Last accessible key of data (length of data - 1)
SQLite_Connection	Constant SQLite3 connection object	Stores connection to the database file.
SQLite_Cursor	Constant SQLite3 cursor object	used to execute queries to the database.
fromDB	Tuple	Temporary storage for data queried from the database.
registered	String	stores y/n input from the user to determine if they're registered
username	String	stores username input from user
password	String	stores password input from user
bestScore	Integer	Retrieved user's best score from SQL database
score	Integer	Stores user's game score
playing	Boolean	Controls while loop for main game logic
chosen_Data	Dictionary (JSON object with only string identifiers)	Random index from main data list (random song)
output	String	String to be outputted to the console containing 'question' information
user_Input	String	Answer taken from user
toadd	Integer	Amount of score to add based on how many guesses the user took
leaderboard	Python list (of Tuples)	Stores list of leaderboard entries from SQLite DB
top_five	Python list (of Strings)	Top 5 entries from leaderboard formatted to be printable
lScore String (iterator) <i>five</i>	String (iterator) Used to iterate over top_five	Used to iterate over top lScore
tmp	String	Only used as to maintain good practice in storing the result of an 'input' method

1.7 Testing

Test Number	What to Test	Action	Expected Result
1	Prerequisites	Indicating if registered	"Starting login process..." + start of login process OR "Starting registration process..." + start of registration process, depending on user input.
2	Login	Invalid username entered	"Username not found" + back to start of login process.
3		Valid username but incorrect password	"Incorrect Password" + back to start of login process.
4		Valid username and password	"Correct Password!" + proceeds into user's welcome page and main game.
5	Registration	Entered already taken username	"Username already in use." + back to start of registration process.
6		Username not taken	proceeds into user's welcome page and main game.
7	Quiz/Main Game	Start of each question	Print out of Artist and first letter of each word in song name, correctly and with decent formatting.
8		Enter incorrect song name	"Incorrect, try again!" + either gives another guess OR "Better luck next time!", correct answer, highscore and leaderboard.
9		Enter correct song name	" +3" OR " +1", depending on number of guesses + Total score.
10	Leaderboard etc.	End game	Prints out: total score, leaderboard of top 5 in DB and "PRESS ENTER TO CONTINUE".

2 Development

```

1 from csv import DictReader
2 from random import randint
3 from re import sub
4 from pyfiglet import Figlet
5 import passhash, getpass, sqlite3, atexit
6
7 f = Figlet(font='standard')
8 print(f.renderText('Welcome'))
9 song_List = "chart2000-songmonth-0-3-0050.csv"
10 data = list(DictReader(open(song_List))) #Reads in csv file to a list of
    OrderedDict objects
11 length = len(data) - 1
12
13 SQLite_Connection = sqlite3.connect("users.sqlite3") #Connects to SQLite
    Database
14 SQLite_Cursor = SQLite_Connection.cursor() #Creates cursor object for executing
    queries
15 fromDB = () #initialises empty tuple to read future SQLite queries into
16 def onExit():
17     SQLite_Connection.close()
18 atexit.register(onExit)
19 #Initialises function that closes DB connection and adds to event that fires on
    program close
20 registered = ""
21 while registered != "y" and registered != "n":
22     registered = input("Are you registered?(Y/N): ").lower()
23 registered = ord(registered)//121
24 #Gets user input until it is either 'y' or 'n', then converts that char to 1
    and 0 respectively
25 if registered:
26     print("Starting login process...")
27     login_Valid = False
28     while not login_Valid:
29         username = input("Username: ") #Reads username into a tuple of length 1 (
    for ease of query execution)
30         password = getpass.getpass() #Uses getpass module to read in password
    securely
31         SQLite_Cursor.execute('SELECT * FROM users WHERE username=?', (username,))
    #executes query to retrieve user data connected to specified username
32         fromDB = SQLite_Cursor.fetchone() #Fetches first line from query output
33         if fromDB == None: #If the username isn't found in the DB
34             print("Username not found")
35         elif not passhash.verify_password(fromDB[2], password): #If the given
    password hashed doesn't equal the hash in the DB
36             print("Incorrect Password")
37         else:
38             print("Correct Password!")
39             login_Valid = True
40     else:
41         print("Starting registration process...")
42         registration_Valid = False
43         while not registration_Valid:
44             username = input("Username: ")
45             SQLite_Cursor.execute('SELECT username FROM users WHERE username=?', (
    username,)) #executes query to retrieve all usernames with given name
46             if SQLite_Cursor.fetchone() != None: #If the username's already in use
47                 print("Username already in use.")
48             else:
49                 password = getpass.getpass()
50                 SQLite_Cursor.execute('INSERT INTO users(username, passHash) VALUES
    (?, ?)', (username, passhash.hash_password(password))) #Inserts a new line

```

```

    into the DB with new Username and Password
51     SQLite_Connection.commit() #Commits change to the DB
52     registration_Valid = True
53 print("\n" + f.renderText(username))
54 SQLite_Cursor.execute('SELECT bestScore FROM users WHERE username=?', (username
    ,)) #Gets the users bestScore from the DB
55 bestScore = SQLite_Cursor.fetchone()[0]
56 print("You highscore is %s\n" % bestScore)
57 score = 0
58 playing = True
59 while playing:
60     chosen_Data = data[randint(0, length)] #Gets a random song from the songList
61     output = chosen_Data["artist"] + "\n\t" + "\t".join([word[0] for word in
        chosen_Data["song"].split()]) #Creates an output string containing the song
        Artist andthe first letter of every word in the song name
62     print(output)
63     for i in range(1, 3):
64         user_Input = input("Enter name of Song: ")
65         if sub(r'[^w\s]', '', user_Input.lower()) == sub(r'[^w\s]', '', chosen_Data
            ["song"].lower()): #If the guess is correct (+string cleanup using regex)
66             toadd = 3 if i == 1 else 1 #calculates score increase based on how
                many guesses
67             score += toadd
68             print("+", toadd)
69             break
70         else:
71             if i == 2: #If this was the second incorrect guess
72                 print("Incorrect, Game Over!\n\nBetter luck next time!")
73                 print("Correct answer was '%s'\n" % chosen_Data["song"])
74                 playing = False #breaks out of playing loop
75             else:
76                 print("Incorrect, try again!")
77     print("Total Score: " + str(score) + "\n")
78
79
80 SQLite_Cursor.execute("INSERT INTO scores (username, score) VALUES (?, ?)", (
    username, score)) #execute query adding the new score to the score table
81 SQLite_Connection.commit()
82 SQLite_Cursor.execute('SELECT username, score FROM scores') #Gets score
    leaderboard
83 scores = SQLite_Cursor.fetchall()
84 leaderboard = list(reversed(sorted(scores, key=lambda score: score[1]))) #Sorts
    retrieved leaderboard, reverses it (best first) and converts returned
    iterable object to list
85 top_five = [" ".join(x) for x in [[str(y) for y in x] for x in leaderboard
    [0:5]]] #Creates list of strings containing '<username>: <score>' for the
    top 5 score in leaderboard
86 print("Leaderboard:")
87 for lScore in top_five:
88     print("\t" + lScore)
89 if score > bestScore: #If the user broke their personal best
90     SQLite_Cursor.execute("UPDATE users SET bestScore=? WHERE username=?", (
        score, username[0])) #update users personal best in DB
91     SQLite_Connection.commit()
92 input("\nPRESS ENTER TO CONTINUE")

```

2.1 Setup

- Possibly the most important part of the setup for the program is the SQLite database. This is contained in a .sqlite3 file separate from the main python script, containing 3 tables:
 - 'sqlite_sequence' - holds metadata for the other 2 tables, created by SQLite.

- 'users' - contains the uniqueID, username, bestscore and password hash for each user.
- 'scores' - contains the uniqueID, username and value for each score achieved by any user.

A connection to the DB file is created through the SQLite3 python module - chosen as it's free and easy to use, with a portable SQLite3 database file browser for debugging. This connection is used throughout the program to execute queries to the tables.

```
SQLite_Connection = sqlite3.connect("users.sqlite3")
SQLite_Cursor = SQLite_Connection.cursor()
```

As well as closing itself, the program must ensure that the connection is closed when the program exits. Although not strictly speaking necessary, it is good practice and helps to reduce the chance of the database file getting corrupted.

For this, I used the atexit module, enabling the program to define a function and then register it to an onExit event.

```
def onExit():
    SQLite_Connection.close()
atexit.register(onExit)
```

- Also a vital part of the program, the song data must be read in before the game starts. For this, I chose to use a website called chart2000.com, and their data file containing information for the top 50 songs of each month from Jan 2000 to Mar 2019. However, I quickly realised that, in order to make the game easier and also reduce the file size, I should cut down the amount of songs, leaving just the songs from Jan 2016 onwards - just short of 2000 entries.

month	position	artist	song	score	us	uk	de	fr	ca	au
Jan-16	1	Adele	Hello	2568.706	1	3	1	1	1	2
Jan-16	2	Justin Biel	Sorry	2412.344	1	2	6	4	1	2
Jan-16	3	Justin Biel	Love Your	2219.517	3	1	3	9	3	1
Jan-16	4	Shawn McStitches		1876.021	6	1	2	86	10	4
Jan-16	5	Justin Biel	What Do Y	1669.61	4	3	-	21	3	8
Jan-16	6	Drake	Hotline Bl	1635.34	3	16	-	18	4	18
Jan-16	7	The Week	The Hills	1317.11	6	18	-	14	6	17
Jan-16	8	Selena Go	Same Old	1202.464	5	99	-	26	6	40
Jan-16	9	Alessia Ca	Here	1196.433	6	28	-	37	19	12
Jan-16	10	The Week	Can't Feel	1075.022	15	19	-	42	8	24
Jan-16	11	Coldplay	Adventur	1039.526	39	8	-	2	21	32
Jan-16	12	Ellie Goul	On My Mii	1033.61	13	33	-	180	13	36
Jan-16	13	Fetty Wap		679	1028.72	11	32	-	15	43
Jan-16	14	The Week	In The Nig	1018.649	12	48	-	127	12	13
Jan-16	15	Twenty O	Stressed C	979.645	4	86	-	40	5	-
Jan-16	16	Meghan T	Like I'm G	976.037	9	-	-	-	8	-
Jan-16	17	Major Laz	Lean On	965.663	17	38	-	25	9	38
Jan-16	18	Elle King	Ex's & Oh	918.22	12	15	-	-	23	5
Jan-16	19	One Direc	Perfect	880.652	26	19	-	-	15	15
Jan-16	20	Drake & F	Jumpman	875.016	12	95	-	-	-	-
Jan-16	21	R City & A	Locked Av	869.773	33	28	-	57	15	40
Jan-16	22	Taylor Swi	Wildest D	799.818	18	97	-	183	10	32
Jan-16	23	Travis Sco	Antidote	790.785	16	-	-	-	46	-
Jan-16	24	The Chain	Reperc	790.38	13	-	-	130	0	37

Although there was the option of reading the CSV file in line by line, python contains the functionality to do this simply and easily in a single line through the use of the csv.DictReader method.

```
data = list(DictReader(open(song_List)))
```

Although this returns a list of OrderedDict Objects, so the program knows which order the columns come in, this is just as a result of the dictreader method and its functionality isn't exploited in my program.

2.2 The Welcome form

- First, I decided on welcoming the user in with some ASCII art. After briefly considering manually typing it out, I quickly went to use a module - pyfiglet, a python port of the original command-line program 'Figlet', written in C. This lets me choose from a selection of fonts and output to the command-line with the use of a single method.



- Next, I had to ask if the user is registered. This was done quite simply, through a 'y/n' input in the console.

```
Are you registered? (Y/N) : |
```

In order to clean up the if statement later on in the code, I converted the 'y/n' user input to a 1 or 0, using some char-code magic.

```
registered = ord(registered)//121
```

2.3 Registering

- The first issue encountered in the registration is with duplicate users. In order to counter this, after the user has entered their username, the program queries the SQLite DB to check for any existing entries in the table with the same username.

```
SQLite_Cursor.execute('SELECT username FROM users WHERE username=?', (
username,))
if SQLite_Cursor.fetchone() != None:
    print("Username already in use.")
```

- As a minor security measure, the getpass module is used. This hides the password as it is entered, so long as it is run in the python console, not IDLE.
- Again focusing on security, passwords are hashed and salted before entering the SQLite DB.

For the salt, a random byte array (obtained using the os.urandom method) is hashed using sha256. The result of this is a 64 byte long sha256 hash.

```
>>> hashlib.sha256(os.urandom(60)).hexdigest().encode('ascii')
b'ec7f6ec6e933e0337bff2228868aa57b8d6dcbaddabf16209557c28a5a68a54'
```

Using this as the salt and the sha512 algorithm, the program then uses the hashlib.pbkdf2_hmac method to produce a hash over 10,000 iterations. The reason I chose this over just a simple sha512 hash of the password is its low susceptibility to brute force attacks.

```
pwdhash = hashlib.pbkdf2_hmac('sha512', password.encode('utf-8'), salt,
100000)
pwdhash = binascii.hexlify(pwdhash)
```

To store this in the SQLiteDB, I concatenated the salt and hash together and decoded it to ascii.

```
(salt + pwdhash).decode('ascii')
```

2.4 Logging in

- As with registration, the getpass module is used to maintain password confidentiality.
- A minor check is used by the program to ensure that the username is valid. This is done by querying the database for the given username. If it isn't valid, then the user is sent to the beginning of the login process again.

```
SQLite_Cursor.execute('SELECT username FROM users WHERE username=?', (
username,))
if SQLite_Cursor.fetchone() != None:
    print("Username already in use.")
```

- The entered password is checked against the hash in the SQLite DB using the same pbkdf2_hmac method used in registration. The hash is retrieved from the database, split into hash and salt and then the hash is compared with the one produced using the password just entered.

```
def verify_password(stored_password, provided_password):
    salt = stored_password[:64].encode('ascii')
    stored_password = stored_password[64:]
    pdhash = hashlib.pbkdf2_hmac('sha512', provided_password.encode('
utf-8'), salt, 100000)
    pdhash = binascii.hexlify(pdhash).decode('ascii')
    return pdhash == stored_password
```

2.5 The Home form

- Once the user has correctly logged in, they are welcomed with an ascii-art of their username, again using the pyfiglet module, and their high-score - retrieved from the database.



You highscore is 38

2.6 The Game

- The game utilises simple logic, operating within a while loop, iterating until Game Over, and within each cycle a for loop of range(1, 3) is used to keep track of guesses over each song.

```

Shawn Mendes
      I      M      B
Enter name of Song: in my blood
+ 3
Total Score: 3

Cardi B & Bruno Mars
      P      M
Enter name of Song: please me
+ 3
Total Score: 6

Imagine Dragons
      W      I      T
Enter name of Song: fdfsdfd
Incorrect, try again!
Enter name of Song: whatever it takes
+ 1
Total Score: 7

```

- As seen in the image above, 1 guess taken gives 3 points, while 2 guesses taken gives only 1 point. Each score obtained is shown clearly to the user on a newline along with the new total score for the game.

```

toadd = 3 if i == 1 else 1
score += toadd
print('+ ', toadd)

```

- One issue encountered here is the need to clean up the strings, so that capital letters, numbers, hyphens etc. don't cause correct answers to fail, e.g. the program will recognise an input of *'Im so sorry'* so be correct for a song name of *'I'm so sorry'*. In order to solve this issue, I used the python module re (short for regex), more specifically the sub method. This replaces sub-strings within a string based on a regex pattern. The program uses this to only leave white-space, letters & numbers in the given answer and actual answer before comparing them. In addition, the string.lower() function is used to remove caps-lock sensitivity.

```

if sub(r'^\w\s', '', user_Input.lower()) == sub(r'^\w\s', '',
chosen_Data["song"].lower()):

```

Taking apart the regex pattern:

- `[^]`: Start of a negated set (Matches any character not in the set),
- `\w`: Matches any word (alphanumeric or underscore) character,
- `\s`: Matches any white-space character.
- `]`: End of the negated set.

As a result, any character that isn't white-space or a word character is removed from the string, so they aren't compared. This helps to remove confusion created by hyphens, dollar signs etc.

- After game over, the user's score is then inserted into the scores table and, if the score is better than the user's bestScore, the users row in the users table is updated to contain the new data.
- The program then outputs a leaderboard of the top 5 scores taken from the scores table.

```
Leaderboard:
    a: 7
  admin: 6
  admin: 4
  admin: 4
    a: 3
```

```
SQLite_Cursor.execute('SELECT username, score FROM scores')
scores = SQLite_Cursor.fetchall()
leaderboard = list(reversed(sorted(scores, key=lambda s: s[1])))
top_five = [": ".join(x) for x in [[str(y) for y in x] for x in
leaderboard[0:5]]]
print("Leaderboard:")
for lScore in top_five:
    print("\t" + lScore)
```

The leaderboard variable here stores a ordered leaderboard of the scores tables, using the sorted function with a lambda expression to sort the list of tuples based on the second value in each tuple. Because this returns a list in the reversed order, the reversed function is used and, as it returns an iterator, not a list, the list function is included as well. Some formatting is used here with tabs just to make the leaderboard clearer.

- The last problem faced in the program was with it exiting immediatley after having finished the game, so an input statement is used at the end just to make sure the user has read through the leaderboard etc. before closing the program.

```
input("\nPRESS ENTER TO CONTINUE")
```

3 Evaluation

3.1 SQLite Database

- One main improvement that I think could be taken is in relation to the SQLite DB and how all the user and score information is stored. At the moment, all data is stored in single .sqlite3 file in the same directory as the main python script. This leaves it vulnerable to users just editing the file to add fake users and scores.
- A solution to this would be to host an SQL database in the cloud, using a system like MySQL with phpmyadmin for maintenance. This would mean that users have no direct access to the database, as authentication would be required to query the database.

3.2 The Songs

- At the moment, the songs used, as explained above, are taken from the top of the charts over the last few years. This means that familiarity with both the pop genre and songs over a long period of time are required to do well in the game.

- To resolve this, one option could be to use songs from the users Spotify playlist, retrieved using Spotify's REST API through a python module, e.g. urllib or httpplib to send POST and GET requests. Additionally, login to the Spotify account is done purely by Spotify, so not only could that be used to connect to the Spotify account but that could also be used to identify the user for the game.

3.3 Additional Comments

- Something else, although not within the scope, that could be added is functionality to listen to a snippet from the song to help users after 1 guess. This would require some more complex code, with extra modules to support playing music, as well as internet access to stream the music (because pre-downloading all the music files on the client side would be ridiculous in terms of storage).
- In general, the program code isn't that organised, with limited use of functions and only one external file used. Of course, for something with this simple general structure, nothing overly complicated is needed but perhaps some functions and classes could help clean up the code and make it less intimidating to others looking at it.
- Lastly, as mentioned in section 1.5, a GUI could be used to make the program more user friendly and just prettier in general. However, the time required to add this functionality would be too much for something not in the scope of the project.

4 References

- <https://pypi.org/project/pyfiglet/0.7/>
- <http://www.figlet.org/examples.html>
- <https://regexr.com/>
- <https://docs.python.org/3/library/hashlib.html>
- <https://docs.python.org/3/library/sqlite3.html>
- <https://stackoverflow.com/questions/3850261/doing-something-before-program-exit>
- <https://stackoverflow.com/a/35340988>
- Song and Artist details taken from <https://chart2000.com> under fair use, using 'chart2000-songmonth-0-3-0050' dataset.