Techniques used for development:

- Additional libraries
- Creation of database
- Add data to database
- Search for specific data in database
- Data extraction
- Delete data from database
- 2 dimensional arrays
- Parsing a text file
- Recursion
- For loops

Additional Libraries

```
9  # LIBRARIES #
10 from flask import Flask, render_template, request, redirect
11 import pathlib
12 import sqlite3
13
```

The three libraries I used for this project were flask, pathlib, and sqlite3. Flask is a microframework for the frontend of the program. sqlite3 is a library that allows me to execute SQL commands from Python. pathlib is used to detect if a datafile already exists. The import function is a Python function that retrieves the libraries to be used in the program.

Reading a text file

```
244
245 def getData(FILENAME):
246     """
247     Gets data from the login csv file
248     :param FILENAME: csv file
249     :return: list
250     """
251
252     FILE = open(FILENAME)
253     TEXT_LIST = FILE.readlines()
254     FILE.close()
255
256     for i in range(len(TEXT_LIST)):
257         if TEXT_LIST[i][-1] == "\n":
258             TEXT_LIST[i] = TEXT_LIST[i][:-1]  # Removes the /n from the end of each line
259         TEXT_LIST[i] = TEXT_LIST[i].split(",")
260
261     TEXT_LIST = TEXT_LIST[1:]
262
263     return (TEXT_LIST)
264
```

The function getData() is made with a parameter of FILENAME. Firstly, the FILENAME is opened using the open function and stored in the FILE variable. The TEXT_LIST variable stores the information inside FILE using the readlines() function as a 2D array. The FILE is then closed to save space. The for loop traverses each line in TEXT_LIST and checks if there is a \n at the end, indicating a new line to be formed. If \n is present, it removes it from the line. Then, each line in TEXT_LIST is split by commas using the split() function. Lastly, the getData() function returns TEXT_LIST.

Creation of database

```
265
266 def createDatabase():
267     """
268     Creates a database for the members.
269     :return: none
270     """
271     global MEMBERS_DATABASE
272     CONNECTION = sqlite3.connect(MEMBERS_DATABASE)
273     CURSOR = CONNECTION.cursor()
274
275     CURSOR.execute("""
276     CREATE TABLE
277         members (
278             first_name TEXT NOT NULL,
279             last_name TEXT NOT NULL,
280             email TEXT NOT NULL,
281             age INT NOT NULL,
282             payment FLOAT NOT NULL,
283             start_date NOT NULL,
284             end_date NOT NULL
285         )
286     ;""")
287     CONNECTION.commit()
288     CONNECTION.close()
```

In the createDatabase() function, the global function is used to get the MEMBERS_DATABASE variable since that variable stores the database. Since the MEMBERS_DATABASE is a global variable, the global function is used so the createDatabase() function can use the MEMBERS_DATABASE variable. From there, the CONNECTION variable is made to connect to the database, and the CURSOR variable is made to make changes to the database. Then, the execute() function is applied to CURSOR to write SQL code in Python and update the database. Using CREATE TABLE, a table named members is created with 7 fields as shown above. Lastly, the changes are committed to the database using the commit() function from the sqlite3 library. Then, the close() function from the sqlite3 library is applied to CONNECTION to break the connection to the database.

Add data to database

```
290
291 def addData(LIST):
292     """
293     Adds data to the database
294     :param LIST: list
295     :return: None
296     """
297     global MEMBERS_DATABASE
298
299     CONNECTION = sqlite3.connect(MEMBERS_DATABASE)
300     CURSOR = CONNECTION.cursor()
301
302     CURSOR.execute("""
303         INSERT INTO
304             members
305         VALUES (
306             ?, ?, ?, ?, ?, ?, ?
307             )
308     ;""", LIST)
309
310     CONNECTION.commit()
311     CONNECTION.close()
```

Here, the function addData() is made with the parameter of LIST - an array that contains the information that needs to be entered into the database. Once again, MEMBERS_DATABASE is globaled in and CONNECTION and CURSOR are created. In CURSOR.execute(), the SQL command INSERT INTO is used to insert data into the table called members. The VALUES command is used to define the values that will go into each column. In this case, '?' are used as placeholders and the LIST parameter is added to the end. This will replace each '?' with the corresponding value found in the LIST array. This method is used as it is a more secure method of adding data and mitigates the risks of SQL injections.

Data Extraction

```
314 def getAllData():
315     """
316     fetches all the data from the database
317     :return: 2D array
318     """
319     global MEMBERS_DATABASE, TOTAL_MEMBERS, TOTAL_PAYMENTS
320     CONNECTION = sqlite3.connect(MEMBERS_DATABASE)
321     CURSOR = CONNECTION.cursor()
322 
323     MEMBER_DATA = CURSOR.execute("""
324         SELECT
325             *
326         FROM
327             members
328         ORDER BY
329             start_date
330     ;""").fetchall()
331 
332     CONNECTION.close()
333     return MEMBER_DATA
```

Here, the getAllData() function is made to retrieve all the data that is currently in the database. The MEMBERS_DATABASE variable is globaled once again, but this time, the TOTAL_MEMBERS and TOTAL_PAYMENTS variables are also globaled in. These two variables are global variables in the program. This time, the variable MEMBER_DATA stores the information CURSOR.execute() will return. Inside CURSOR.execute(), the SELECT command in SQL selects the columns that should be retrieved. The * means that all the data from all columns should be retrieved. The FROM command specifies to get all this data from the members table. The ORDER BY command organizes the retrieved data according to the start_date. The fetchall() function from sqlite3 fetches all this data and stores it in MEMBER_DATA as a 2D array. The CONNECTION is then closed. Notice how this function does not include CONNECTION.commit(). This is because this function is not making any changes to the database, it is only getting information from the database.

Search for data in database

```
154     # Fuzzy searches the database to see if name is there
155
156     USER_SEARCH = request.form.get("search")
157
158     SEARCH = CURSOR.execute(f"""
159             SELECT
160                 *
161             FROM
162                 members
163             WHERE
164                 first_name LIKE "%{USER_SEARCH}%"
165
166         ;""").fetchall()
167
```

In this function, the USER_SEARCH variable is created which stores the user's input from the search bar on the web app. The request.form.get() function is from the flask library which helps connect the front-end inputs and interactions of the user to the back-end logic and processing. Using the request.form.get() function from flask, the input of the user is retrieved and stored in the USER_SEARCH variable. In CURSOR.execute(), the SELECT and FROM commands are used once again from SQL. This time, the WHERE command is also used. This command serves to filter the data retrieved from the database. Above, the command states to select all the data using SELECT, from the members table using FROM, only where the first_name is similar to the USER_SEARCH using the WHERE command. The LIKE command is the command that fuzzy checks each row to see if the first_name is similar to the USER_SEARCH. The fetchall() function is used to get all the instances where the first_name in the members database is similar to the USER_SEARCH.

Delete from database:

```
198
199     CURSOR.execute(f"""
200         DELETE FROM
201             members
202         WHERE
203             email = '{MEMBER}'
204
205     ;""")
```

The code above deletes a member from the database. The MEMBER variable, although it isn't shown on the screenshot, stores the email address of the member. Since each member must have a unique email address, the email field is used to identify members. Once again using the execute() function from the sqlite3 library to make changes to the database, the SQL command DELETE FROM is used to specify what table the data should be deleted from. In the code above, we see that the DELETE FROM command states that the data should be deleted from the members table. The WHERE command is then used to specify what data should be deleted from the members table. In this case, the WHERE command states that if the email field in the members table is equal to the email address of the member stored in the MEMBER variable, then the row should be deleted. You may notice that this time, neither the fetchall() function, nor the commit() function is being used. This is because no new data is being added to the database and no data is being retrieved from the database.

For Loops

```
        ## Checks to see if correct username and password was entered
        for i in range(len(LOGIN_INFO)):
            if USER == LOGIN_INFO[i][0] and PASSWORD == LOGIN_INFO[i][1]:
                USER_NAME = LOGIN_INFO[i][0]
                USER_NAME = USER_NAME.split("_")
                # Determines the name of the user using the username entered
                USER_NAME = f"{USER_NAME[0].capitalize()} {USER_NAME[1].capitalize()}"
                return redirect("/home")
            else:
                ALERT = "Incorrect username or password."
```

This is one of the instances that a for-loop is being used in the program. The LOGIN_INFO variable is a 2D array that stores the usernames and passwords provided by the AGA president. The len() function runs the for loop for the length of LOGIN_INFO. The if-statement checks to see if the username, stored in USER, and password, stored is PASSWORD, match. If they do, the return function returns the user to the home screen using the redirect function from flask. If the username and password do not match, the ALERT variable is set to a string that is displayed to the user to inform them that they inputted the wrong username and password.

2 dimensional arrays

```
316     fetches all the data from the database
317     :return: 2D array
318     """
319     global MEMBERS_DATABASE, TOTAL_MEMBERS, TOTAL_PAYMENTS
320     CONNECTION = sqlite3.connect(MEMBERS_DATABASE)
321     CURSOR = CONNECTION.cursor()
322
323     MEMBER_DATA = CURSOR.execute("""
324         SELECT
325             *
326         FROM
327             members
328         ORDER BY
329             start_date
330     ;""").fetchall()
331
332     CONNECTION.close()
333     return MEMBER_DATA
```

This is the same code shown above that fetches data from the database. This is one example of the use of a 2 dimensional array in this program. The MEMBER_DATA variable is a 2 dimensional array that stores the information from the database. Each array in MEMBER_DATA stores the 7 values of each column. MEMBER_DATA is an array of all the rows in the database. In other words, it is an array of arrays, also known as a 2 dimensional array.

Recursion

```
61  def homePage():
62      """
63      Homepage of the web app
64      :return: html
65      """
66      global TOTAL_MEMBERS, TOTAL_PAYMENTS
67
68      ## Displays the total members and the total payments
69      return render_template("home.html", totalmembers=TOTAL_MEMBERS, totalpayments=TOTAL_PAYMENTS, user=USER_NAME)
70
```

This is a function for the home page of the web app. In this function, the global function is used to get access to the global variables TOTAL_MEMBERS, and TOTAL_PAYMENTS. The function then uses the return function of Python to return render_template(), a function from the flask that displays a template to the user. The render_template() returns “home.html” and also redefines the parameters totalmembers, totalpayments, and user to the variables TOTAL_MEMBERS, TOTAL_PAYMENTS, and USER_NAME respectively. This is a recursive function because the homepage() function returns itself when returning “home.html.”