List of Techniques Used:

- Use of additional libraries
- Database creation
- Inputting data into a database
- Deleting data from a database
- Data extraction from a database
- Two dimensional arrays
- For loops
- Exception handling
- Creation of an HTML webpage
- Form for user inputs
- Conversion of HTML user inputted information into Python variables

Use of Additional libraries

```
6   from flask import Flask, render_template, request
7   from pathlib import Path
8   import sqlite3
9   import random
```

The “flask” library was imported in order to easily create a more aesthetically pleasing, nicely formatted and user friendly interface.

“Path” from “pathlib” was used to check if a database file had already been created in order to prevent the “database already exists” error.

“Sqlite3” was imported for ease of database creation as well as inserting, manipulating and deleting data within the database.

The “random” library was implemented so that the schedule could be shuffled and therefore randomly generated, per the requirements of my client.

Database creation

In this program, two tables were created within one database. The first table stores the schedule name, times and tasks. The second table stores the start time and schedule name inputted by the user.

```
CURSOR.execute("""
    CREATE TABLE
        schedules (
            schedule_name NOT NULL,
            activity_name NOT NULL,
            time NOT NULL,
            clock_time NOT NULL,
            id INTEGER PRIMARY KEY,
            orders TEXT
        )
;""")
```

The above image shows the creation of the “schedules” table within the database. “NOT NULL” is used to indicate that the columns cannot be empty, and “TEXT” is used to indicate that the values within such a column must be a string. I used these constraints because it would be incredibly difficult to understand the schedule and what tasks need to be done, if “schedule_name” and “activity_name” are left empty. If “time” was allowed to be NULL, it would be impossible to generate a schedule, since it would be impossible to tell how long each task takes. “INTEGER” indicates that “id” must be an integer value, and “PRIMARY KEY” indicates that the “id” will be unique for each activity. It is necessary to call upon “id” rather than “activity_name”, because it is likely that my client can have multiple activities within different or even the same schedule with the same “schedule_name”. The program needs to ensuring that it withdraws the correct, specific activity from the database, as different entries with the same “schedule_name” may have different times.

<u>Inputting data into a database</u>

This program requires the user to input information through an HTML form. The following screenshot depicts the code that gets the user input from the HTML form and stores them in the variables “TASK” and “TIME”.

```
if request.form:
    TASK = request.form.get("task")
    TIME = request.form.get("time")
    if TASK != "" and TIME != "":
        addActivity(SCHEDULE_NAME, TASK, TIME)
        ALERT = f"Successfully added new activity to {SCHEDULE_NAME}!"
    else:
        ALERT = "Please fill in all fields."
return render_template("addTasks.html", alert=ALERT, schedule=SCHEDULE_NAME)
```

The inputted information is then inserted into the database through the “INSERT INTO” command. A “?” is used to input the variables to ensure proper data sanitization, and protect the data.

```
117     CONNECTION = sqlite3.connect(DB_NAME)
118     CURSOR = CONNECTION.cursor()
119     CURSOR.execute("""
120         INSERT INTO
121             info (
122                 schedule_name,
123                 start_time
124                 )
125         VALUES (
126             ?, ?
127         )
128     ;""", [SCHEDULE_NAME, START])
129     CONNECTION.commit()
130     CONNECTION.close()
```

A similar method is used when the user inputs the schedule's name and start time.

Deleting data from a database:

The user has a "Delete Schedule" button that opens a new page to delete the database, then redirects back to the "view_schedule" page. To delete the schedule, it is removed from the database, making it so that cannot retrieve it from the database.

```
87      CURSOR.execute("""
88          DELETE FROM
89              schedules
90          WHERE
91              schedule_name = ?
92      ;""", [SCHEDULE_NAME])
```

Data extraction from a database

When the user chooses to view their created schedule, data needs to be extracted from the database and displayed in an easily readable format.

```
152     CURSOR = CONNECTION.cursor()
153     SCHEDULE = CURSOR.execute("""
154         SELECT
155             *
156         FROM
157             schedules
158         WHERE
159             schedule_name = ?
160 
161     ;""", [SCHEDULE_NAME]).fetchall()
162     CONNECTION.close()
```

In the above screenshot, I used the ".fetchall()" command in order to extract data from the database. To make sure that I am only selecting data from the schedule that the user wants to view, I made sure to include the "WHERE" command to only select columns with a "schedule_name" matching that of what the user requests to view.

Two dimensional arrays

When ".fetchall()" is used in the above screenshot, a 2 dimensional array containing tuples is created. This array includes the schedule name, task, and time to complete said task, index, and clock time (e.g./ 8:30) in each tuple within the array. In the below screenshot, the clock time has yet to be calculated, so it shows up as "None".

```
[('Monday', 'Brush teeth', '5', 0, 2, None), ('Monday', 'Eat breakfast', '20', 0, 3, None),
```

For loops

A for loop is used in the HTML code to create a button for each schedule within the schedules database. A list of all schedule names are retrieved and displayed in alphabetical order for ease of finding the needed schedule. Each button is linked to the "/viewSchedule/" page where a randomized schedule is generated and displayed for the user.

```
30      {% for schedule in schedules %}
31      <td><a href="/get_info/{{schedule[0]}}" class="btn" role="button">View {{schedule[0]}}</a></td>
32      {% endfor %}
```

Exception handling

If a user tries to submit either HTML form with any blank information, an alert pops up letting the user know that they must fill in all fields. The user is unable to input non-integer values into areas that require integers due to the "type = number" field description in HTML; therefore, no exception handling is needed for these cases.

```
49          if TASK != "" and TIME != "":
50              addActivity(SCHEDULE_NAME, TASK, TIME)
51              ALERT = f"Successfully added new activity to {SCHEDULE_NAME}!"
52          else:
53              ALERT = "Please fill in all fields."
54      return render_template("addTasks.html", alert=ALERT, schedule=SCHEDULE_NAME)
```

(Word Count: 715)