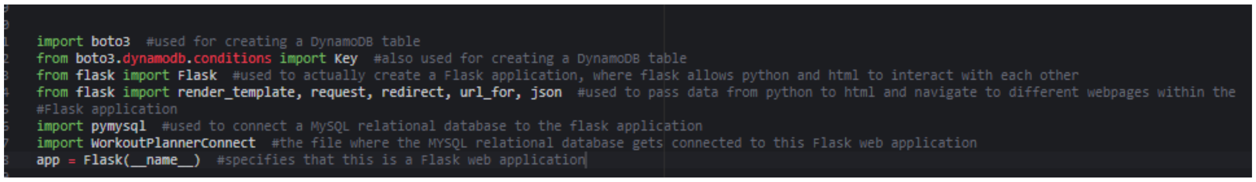
Creating a Web Application Using Flask

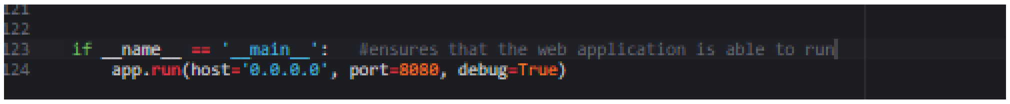
1. Preparation

Before the creation of the application, I imported several different things. The things imported are portrayed in Figure A (image is from a python file titled WorkoutPlannerFlask.py). Additionally, there were two global variables that were initialized, which is portrayed in Figure D.

Figure A



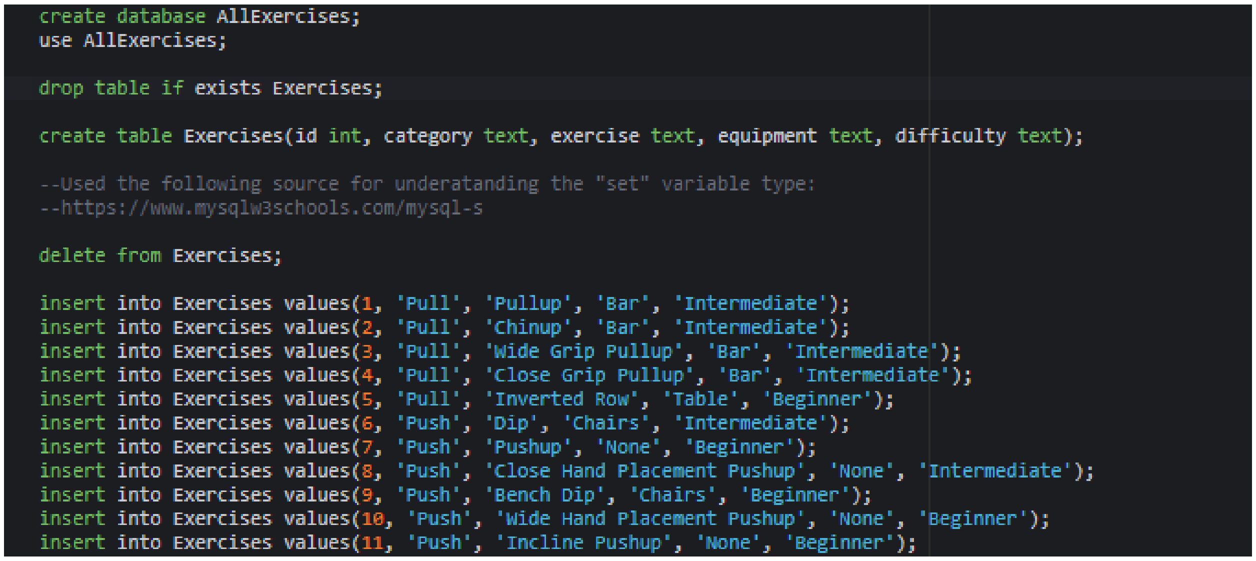
```python
import boto3  #used for creating a DynamoDB table
from boto3.dynamodb.conditions import Key  #also used for creating a DynamoDB table
from flask import Flask  #used to actually create a Flask application, where flask allows python and html to interact with each other
from flask import render_template, request, redirect, url_for, json  #used to pass data from python to html and navigate to different webpages within the
#Flask application
import pymysql  #used to connect a MySQL relational database to the flask application
import WorkoutPlannerConnect  #the file where the MYSQL relational database gets connected to this Flask web application
app = Flask(__name__)  #specifies that this is a Flask web application
```



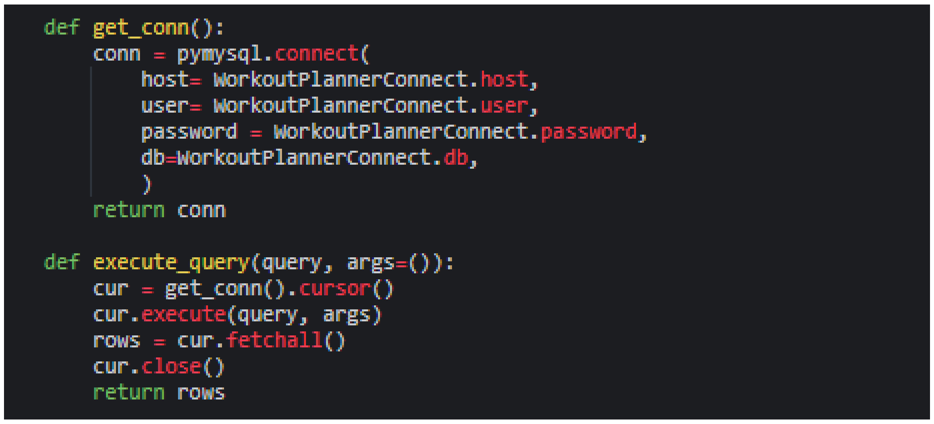
```python
if __name__ == '__main__':    #ensures that the web application is able to run
    app.run(host='0.0.0.0', port=8080, debug=True)
```

This image shows all the things that were imported. The comments explain what these imported things are used for in the Flask web application. As shown in the first image, one of the imported things is “pymysql”, which allows a MySQL relational database to be connected to python. Figure B shows the MySQL database created to be used for this application (image is from file titled WorkoutPlanner.sql). Meanwhile, Figure C shows the DynamoDB non-relational database created to be used for this application (as specified by the imports with the keyword “boto3”). Finally, it’s important to note that the code portrayed in the second image must be written at the very end of the WorkoutPlannerFlask.py file after everything else is written, in order for the Flask web application to work.

Figure B



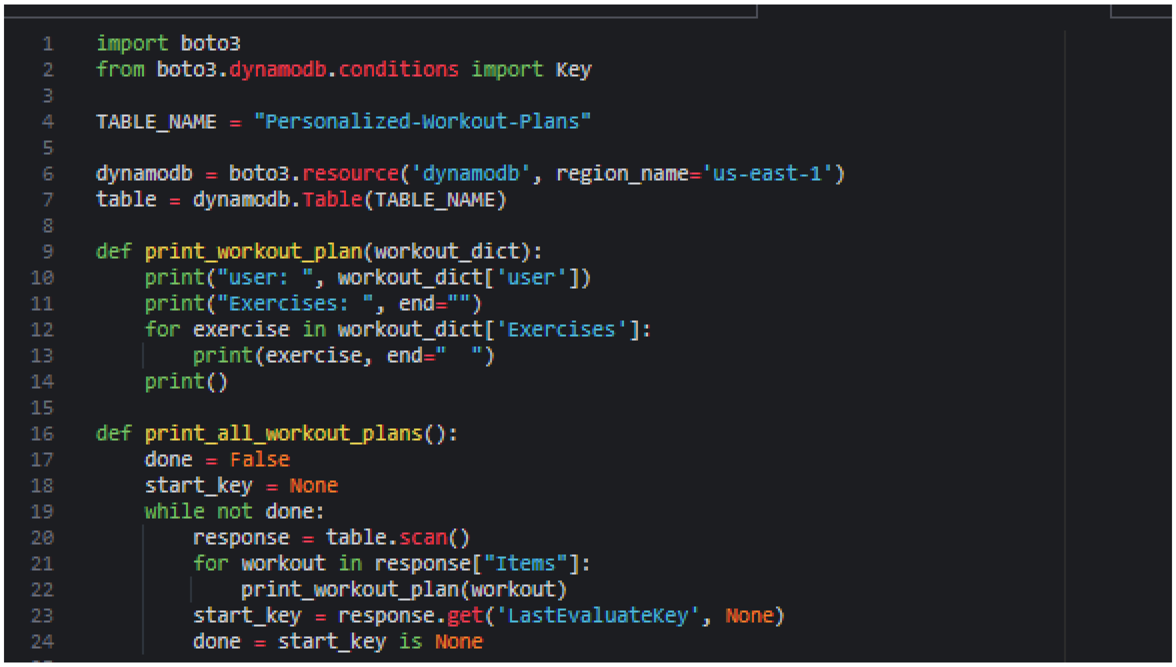
```sql
create database AllExercises;
use AllExercises;

drop table if exists Exercises;

create table Exercises(id int, category text, exercise text, equipment text, difficulty text);

--Used the following source for underatanding the "set" variable type:
--https://www.mysqlw3schools.com/mysql-s

delete from Exercises;

insert into Exercises values(1, 'Pull', 'Pullup', 'Bar', 'Intermediate');
insert into Exercises values(2, 'Pull', 'Chinup', 'Bar', 'Intermediate');
insert into Exercises values(3, 'Pull', 'Wide Grip Pullup', 'Bar', 'Intermediate');
insert into Exercises values(4, 'Pull', 'Close Grip Pullup', 'Bar', 'Intermediate');
insert into Exercises values(5, 'Pull', 'Inverted Row', 'Table', 'Beginner');
insert into Exercises values(6, 'Push', 'Dip', 'Chairs', 'Intermediate');
insert into Exercises values(7, 'Push', 'Pushup', 'None', 'Beginner');
insert into Exercises values(8, 'Push', 'Close Hand Placement Pushup', 'None', 'Intermediate');
insert into Exercises values(9, 'Push', 'Bench Dip', 'Chairs', 'Beginner');
insert into Exercises values(10, 'Push', 'Wide Hand Placement Pushup', 'None', 'Beginner');
insert into Exercises values(11, 'Push', 'Incline Pushup', 'None', 'Beginner');
```



```python
def get_conn():
    conn = pymysql.connect(
        host= WorkoutPlannerConnect.host,
        user= WorkoutPlannerConnect.user,
        password = WorkoutPlannerConnect.password,
        db=WorkoutPlannerConnect.db,
        )
    return conn

def execute_query(query, args=()):
    cur = get_conn().cursor()
    cur.execute(query, args)
    rows = cur.fetchall()
    cur.close()
    return rows
```

The first image from Figure B shows that the database created to be used for this Flask web application is called AllExercises. The database AllExercises contains a single table called Exercises, which includes the following: a unique identifier (id), the group of muscles for which the exercise targets (category), the name of the actual exercise (exercise), which equipment is needed to perform this exercise (equipment), and the fitness level for which the exercise is suitable for (difficulty). Meanwhile, the second image from Figure B shows how that MySQL database became connected to python in the file WorkoutPlannerFlask.py. Note that the WorkoutPlannerConnect.py file was the file actually used to connect the MySQL database AllExercises to Flask.

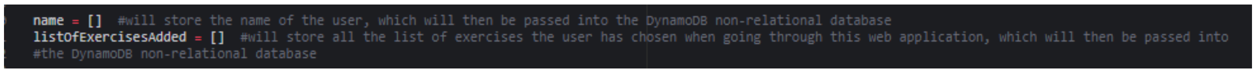
Figure C



```python
import boto3
from boto3.dynamodb.conditions import Key

TABLE_NAME = "Personalized-Workout-Plans"

dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
table = dynamodb.Table(TABLE_NAME)

def print_workout_plan(workout_dict):
    print("user: ", workout_dict['user'])
    print("Exercises: ", end="")
    for exercise in workout_dict['Exercises']:
        print(exercise, end="  ")
    print()

def print_all_workout_plans():
    done = False
    start_key = None
    while not done:
        response = table.scan()
        for workout in response["Items"]:
            print_workout_plan(workout)
        start_key = response.get('LastEvaluateKey', None)
        done = start_key is None
```

This image shows that the DynamoDB non-relational database, or table, created is called Personalized-Workout-Plans. This database, or table, stores the name of the user as well as all the list of exercises that they would have chosen when going through this web application. This image is from the file titled WorkoutPlannerDynamoDB.py.

Figure D



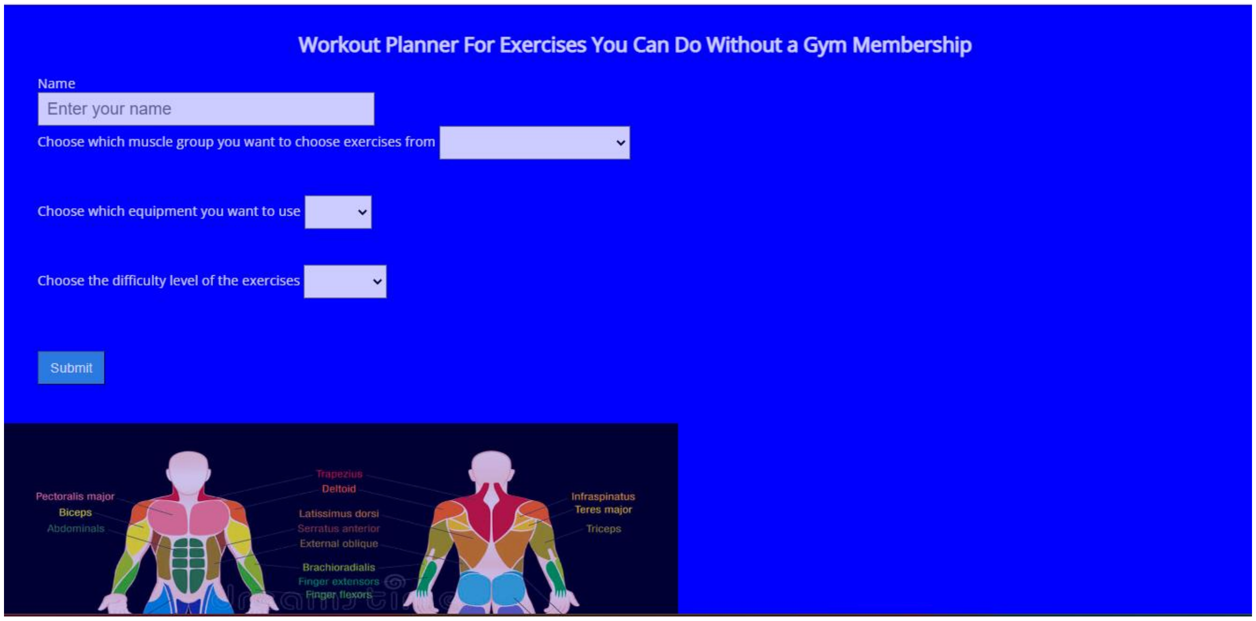
```python
name = []  #will store the name of the user, which will then be passed into the DynamoDB non-relational database
listOfExercisesAdded = []  #will store all the list of exercises the user has chosen when going through this web application, which will then be passed into
#the DynamoDB non-relational database
```

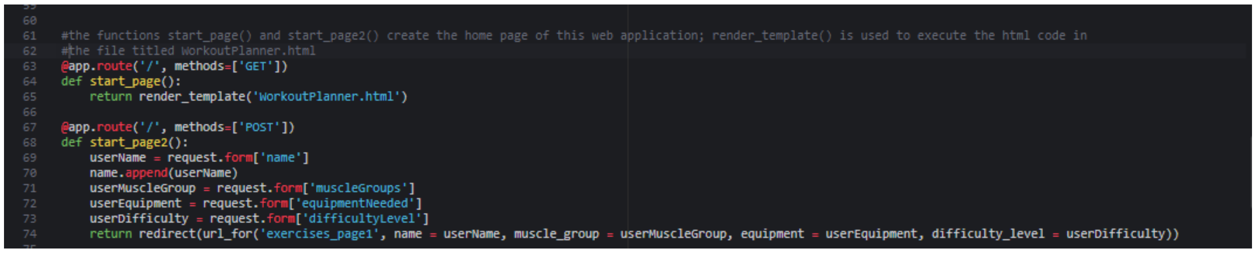
This image (from WorkoutPlannerFlask.py) shows the two global variables that were initialized, with comments explaining what is their use.

2. Creating Home Page

The goal of the home page was to take in and store the name of the user, as well as have user select what group of muscles they want to focus on, what kind of equipment they have available, and their fitness level. Figure A portrays how the home page looks like, as well as the functions within the Flask Python file WorkoutPlannerFlask.py that were used to create the home page.

Figure A

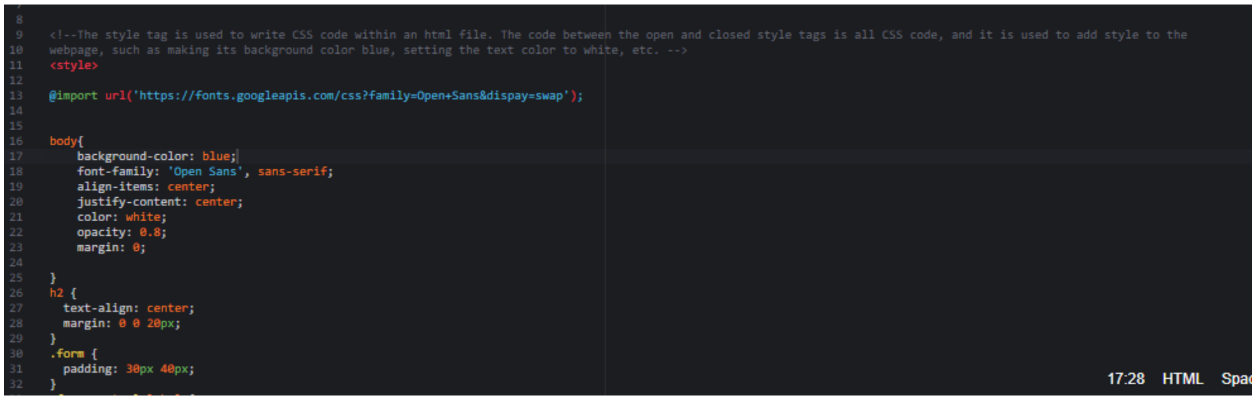

Workout Planner For Exercises You Can Do Without a Gym Membership
Name
Enter your name
Choose which muscle group you want to choose exercises from
Choose which equipment you want to use
Choose the difficulty level of the exercises
Submit
Pectoralis major
Biceps
Abdominals
Trapezius
Deltoid
Latissimus dorsi
Serratus anterior
External oblique
Brachioradialis
Finger extensors
Finger flexors
Infraspinatus
Teres major
Triceps



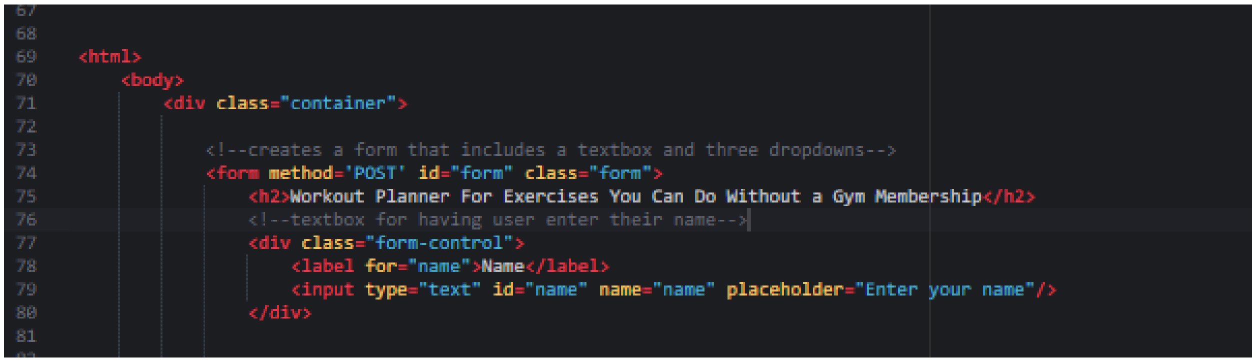
```
#the functions start_page() and start_page2() create the home page of this web application; render_template() is used to execute the html code in
#the file titled WorkoutPlanner.html
@app.route('/', methods=['GET'])
def start_page():
    return render_template('WorkoutPlanner.html')

@app.route('/', methods=['POST'])
def start_page2():
    userName = request.form['name']
    name.append(userName)
    userMuscleGroup = request.form['muscleGroups']
    userEquipment = request.form['equipmentNeeded']
    userDifficulty = request.form['difficultyLevel']
    return redirect(url_for('exercises_page1', name = userName, muscle_group = userMuscleGroup, equipment = userEquipment, difficulty_level = userDifficulty))
```

As described by the comments in the second image, the functions start_page() and start_page2() rely on the html file WorkoutPlanner.html, where the actual design of the home page takes place. Figure B shows some of the code used for that file.

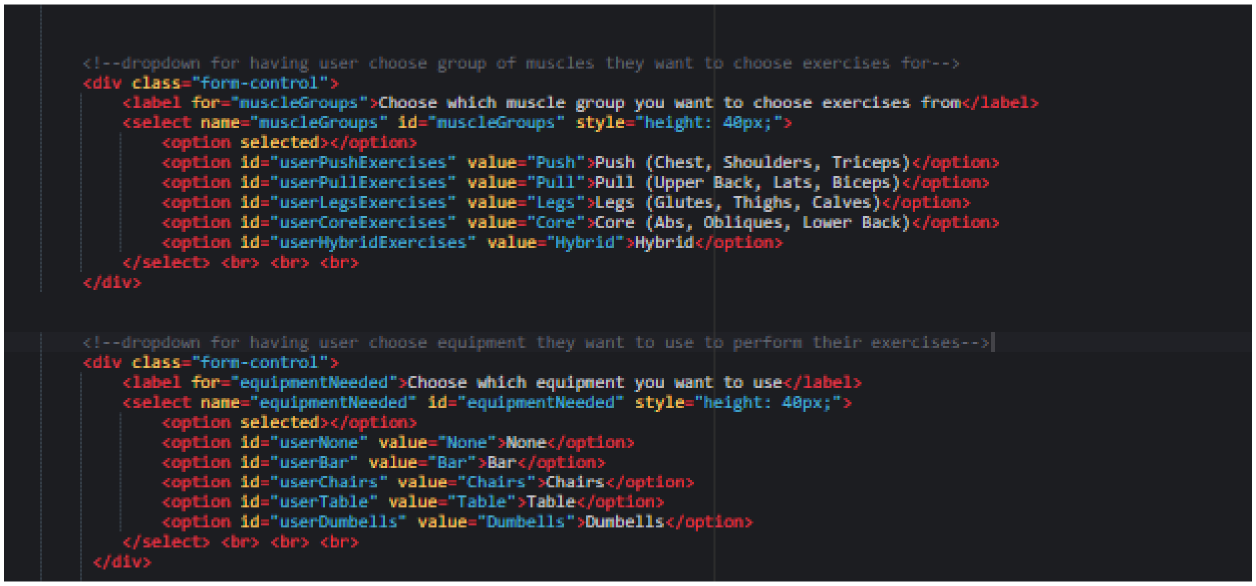
Figure B



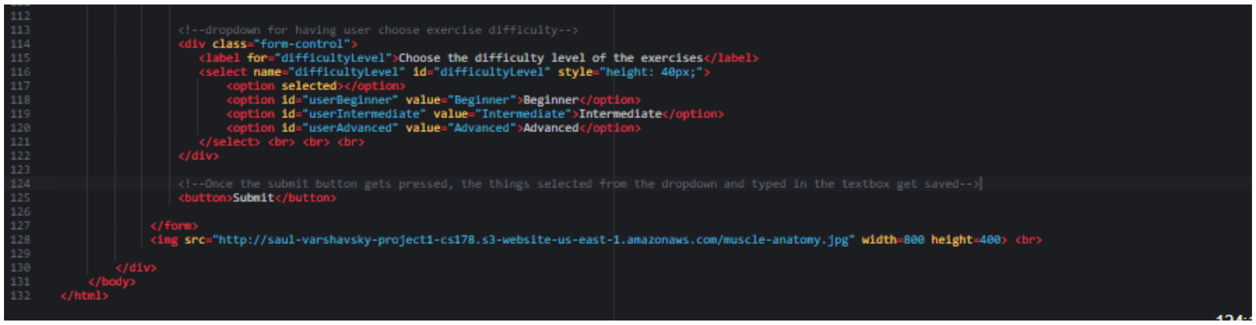
```
<!--The style tag is used to write CSS code within an html file. The code between the open and closed style tags is all CSS code, and it is used to add style to the
webpage, such as making its background color blue, setting the text color to white, etc. -->
<style>

@import url('https://fonts.googleapis.com/css?family=Open+Sans&dispay=swap');


body{
    background-color: blue;
    font-family: 'Open Sans', sans-serif;
    align-items: center;
    justify-content: center;
    color: white;
    opacity: 0.8;
    margin: 0;

}
h2 {
  text-align: center;
  margin: 0 0 20px;
}
.form {
  padding: 30px 40px;
}
```



```html
<html>
    <body>
        <div class="container">

            <!--creates a form that includes a textbox and three dropdowns-->
            <form method='POST' id="form" class="form">
                <h2>Workout Planner For Exercises You Can Do Without a Gym Membership</h2>
                <!--textbox for having user enter their name-->
                <div class="form-control">
                    <label for="name">Name</label>
                    <input type="text" id="name" name="name" placeholder="Enter your name"/>
                </div>

```



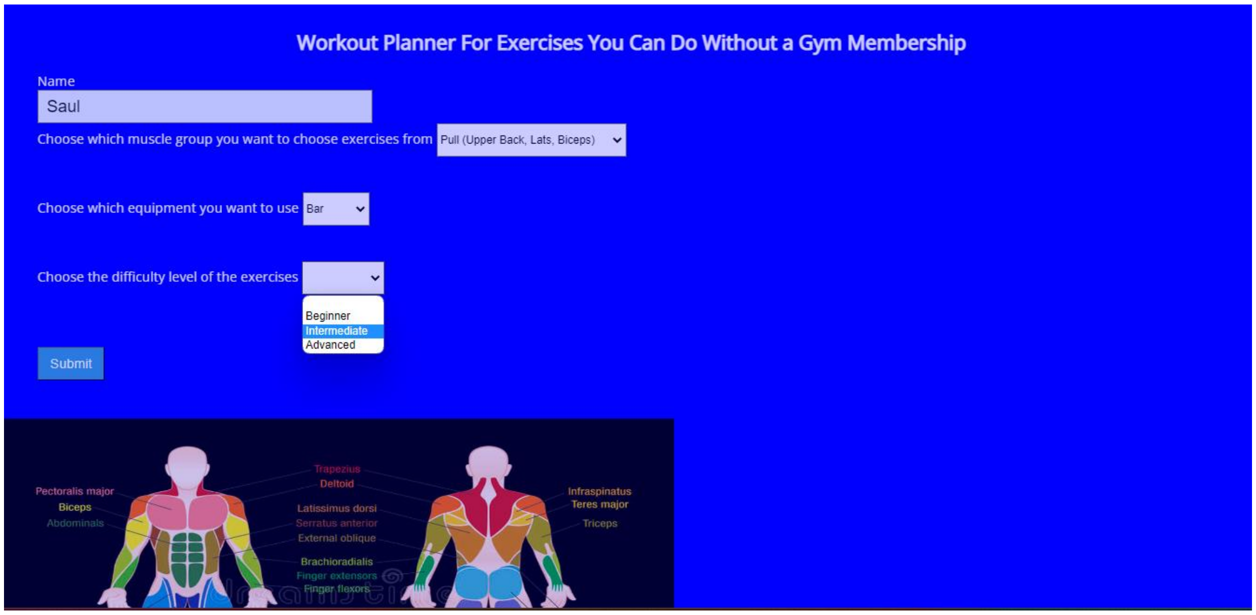
```html
<!--dropdown for having user choose group of muscles they want to choose exercises for-->
<div class="form-control">
    <label for="muscleGroups">Choose which muscle group you want to choose exercises from</label>
    <select name="muscleGroups" id="muscleGroups" style="height: 40px;">
        <option selected></option>
        <option id="userPushExercises" value="Push">Push (Chest, Shoulders, Triceps)</option>
        <option id="userPullExercises" value="Pull">Pull (Upper Back, Lats, Biceps)</option>
        <option id="userLegsExercises" value="Legs">Legs (Glutes, Thighs, Calves)</option>
        <option id="userCoreExercises" value="Core">Core (Abs, Obliques, Lower Back)</option>
        <option id="userHybridExercises" value="Hybrid">Hybrid</option>
    </select> <br> <br> <br>
</div>


<!--dropdown for having user choose equipment they want to use to perform their exercises-->
<div class="form-control">
    <label for="equipmentNeeded">Choose which equipment you want to use</label>
    <select name="equipmentNeeded" id="equipmentNeeded" style="height: 40px;">
        <option selected></option>
        <option id="userNone" value="None">None</option>
        <option id="userBar" value="Bar">Bar</option>
        <option id="userChairs" value="Chairs">Chairs</option>
        <option id="userTable" value="Table">Table</option>
        <option id="userDumbells" value="Dumbells">Dumbells</option>
    </select> <br> <br> <br>
 </div>
```



```html

                <!--dropdown for having user choose exercise difficulty-->
                <div class="form-control">
                    <label for="difficultyLevel">Choose the difficulty level of the exercises</label>
                    <select name="difficultyLevel" id="difficultyLevel" style="height: 40px;">
                        <option selected></option>
                        <option id="userBeginner" value="Beginner">Beginner</option>
                        <option id="userIntermediate" value="Intermediate">Intermediate</option>
                        <option id="userAdvanced" value="Advanced">Advanced</option>
                    </select> <br> <br> <br>
                </div>

                <!--Once the submit button gets pressed, the things selected from the dropdown and typed in the textbox get saved-->
                <button>Submit</button>

            </form>
            <img src="http://saul-varshavsky-project1-cs178.s3-website-us-east-1.amazonaws.com/muscle-anatomy.jpg" width=800 height=400> <br>

        </div>
    </body>
</html>
```

The first image specifically shows how CSS code is used inside this HTML file to add to the style of the home page (very similar CSS code was also used in other HTML files for the creation of this web application), such as making its background color blue, setting the text color to white, etc. Meanwhile, the second image creates a form that contains a textbox (noted specifically in the second image) as well as three dropdowns. The third and fourth images portray the dropdowns created. Once the Submit button gets pressed (portrayed in the fourth image of Figure B), that data then gets sent to the function start_page2() in flask, since methods=['POST'] was specified (refer to the second image in Figure A). In the start_page2() function, the lines stating "request.form[nameOfTextboxOrDropdown]" are used to retrieve the specific values from the html form (i.e. group of muscles, equipment, difficulty, etc.) and store them into local variables in *python (refer to the second image in Figure A). From there, the user leaves the home page and heads over to a new webpage that displays the exercises filtered based on their responses from the home page. Figure C shows the transition between the home page and the webpage that displays the exercises filtered based on a given user's responses.*

Figure C



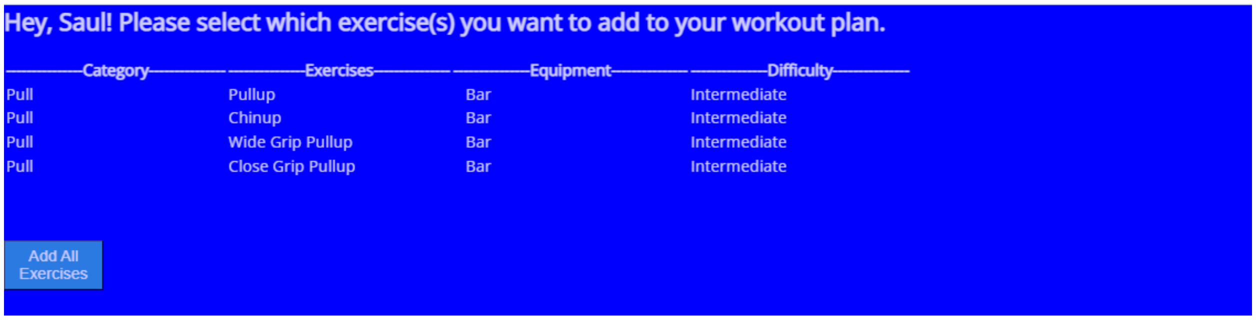
Workout Planner For Exercises You Can Do Without a Gym Membership

Name: Saul

Choose which muscle group you want to choose exercises from: Pull (Upper Back, Lats, Biceps)

Choose which equipment you want to use: Bar

Choose the difficulty level of the exercises: Beginner / Intermediate / Advanced



Hey, Saul! Please select which exercise(s) you want to add to your workout plan.

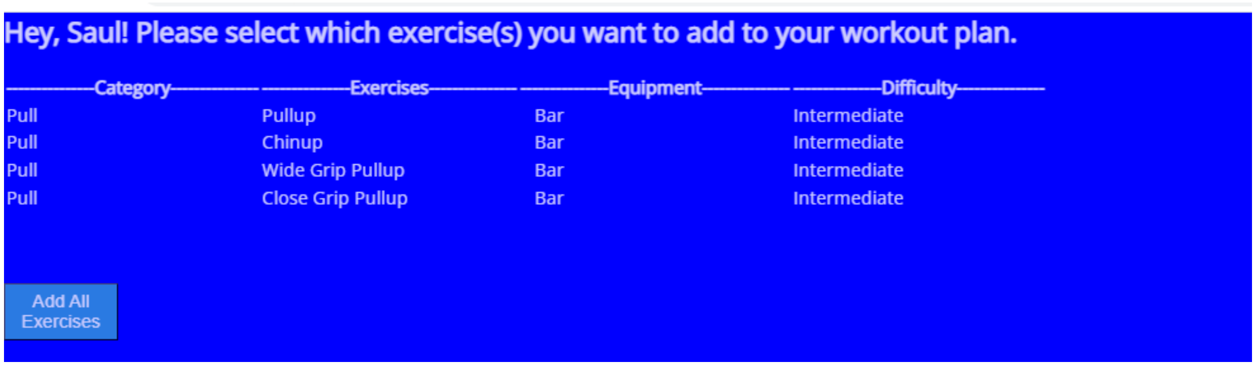
Category	Exercises	Equipment	Difficulty
Pull	Pullup	Bar	Intermediate
Pull	Chinup	Bar	Intermediate
Pull	Wide Grip Pullup	Bar	Intermediate
Pull	Close Grip Pullup	Bar	Intermediate

As shown in Figure C, after I click on the submit button when typing in my name and selecting the appropriate values from the dropdowns, I get directed to a webpage displaying the exercises that align with my responses from the home page.

3. Creating Exercises Page

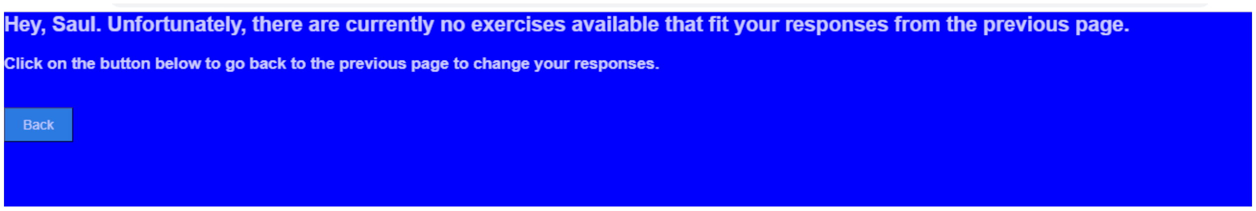
The goal of the exercises page was to display the appropriate exercises for the user to choose from based on their responses from the home page. Figure A portrays how the exercises page looks like, as well as the function within the Flask Python file WorkoutPlannerFlask.py that was used to create the exercises page. Similar to the home page, the design of the exercises page was done in the HTML files WorkoutPlanner_2.html or WorkoutPlanner_3.html, depending on *whether or not at least one exercise was filtered from the MySQL database based on the user's responses (refer to Figure B for the HTML files).*

Figure A



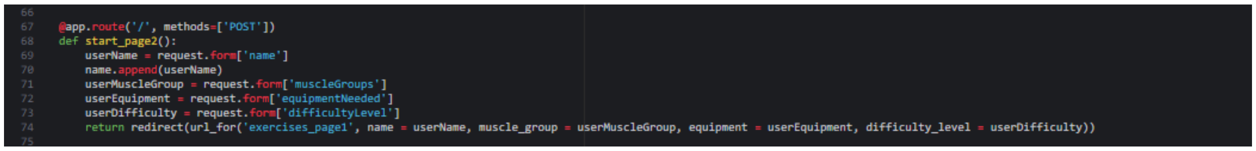
Hey, Saul! Please select which exercise(s) you want to add to your workout plan.

------------Category------------	------------Exercises------------	------------Equipment------------	------------Difficulty------------
Pull	Pullup	Bar	Intermediate
Pull	Chinup	Bar	Intermediate
Pull	Wide Grip Pullup	Bar	Intermediate
Pull	Close Grip Pullup	Bar	Intermediate

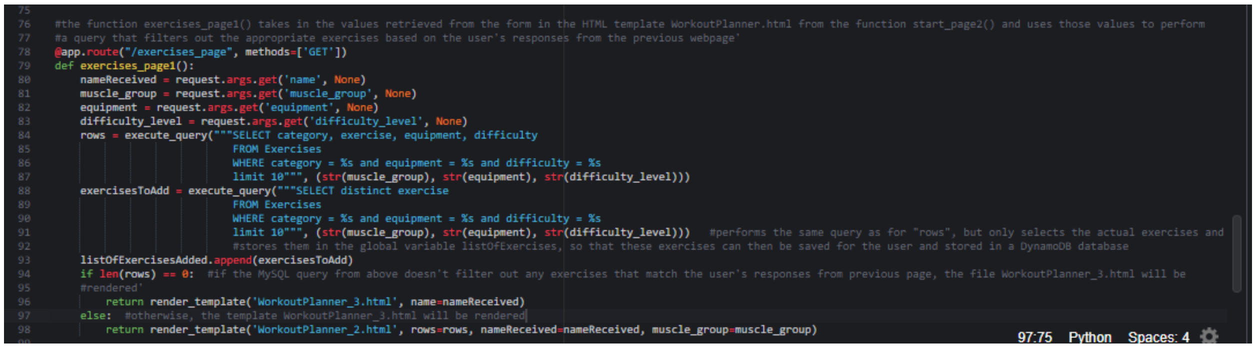


Hey, Saul. Unfortunately, there are currently no exercises available that fit your responses from the previous page.

Click on the button below to go back to the previous page to change your responses.



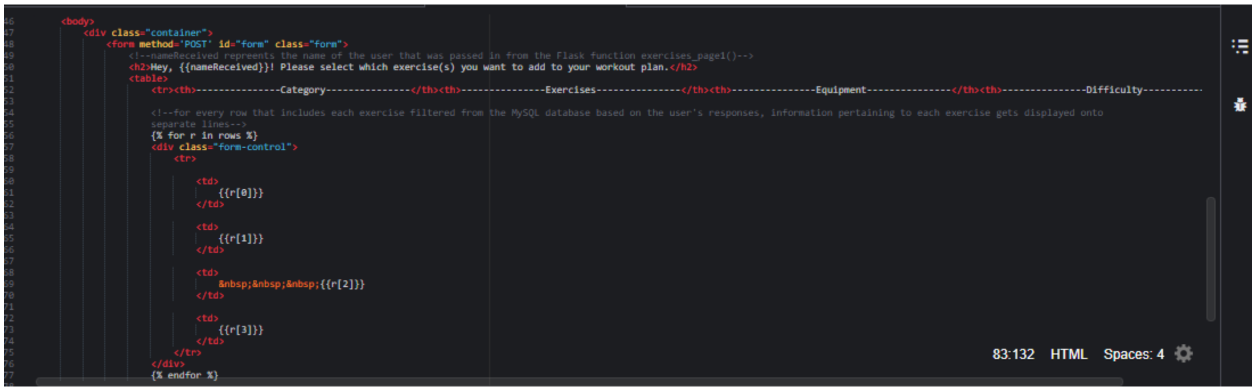
```python
@app.route('/', methods=['POST'])
def start_page2():
    userName = request.form['name']
    name.append(userName)
    userMuscleGroup = request.form['muscleGroups']
    userEquipment = request.form['equipmentNeeded']
    userDifficulty = request.form['difficultyLevel']
    return redirect(url_for('exercises_page1', name = userName, muscle_group = userMuscleGroup, equipment = userEquipment, difficulty_level = userDifficulty))
```



```python
#the function exercises_page1() takes in the values retrieved from the form in the HTML template WorkoutPlanner.html from the function start_page2() and uses those values to perform
#a query that filters out the appropriate exercises based on the user's responses from the previous webpage'
@app.route("/exercises_page", methods=['GET'])
def exercises_page1():
    nameReceived = request.args.get('name', None)
    muscle_group = request.args.get('muscle_group', None)
    equipment = request.args.get('equipment', None)
    difficulty_level = request.args.get('difficulty_level', None)
    rows = execute_query("""SELECT category, exercise, equipment, difficulty
                            FROM Exercises
                            WHERE category = %s and equipment = %s and difficulty = %s
                            limit 10""", (str(muscle_group), str(equipment), str(difficulty_level)))
    exercisesToAdd = execute_query("""SELECT distinct exercise
                            FROM Exercises
                            WHERE category = %s and equipment = %s and difficulty = %s
                            limit 10""", (str(muscle_group), str(equipment), str(difficulty_level)))  #performs the same query as for "rows", but only selects the actual exercises and
                            #stores them in the global variable listOfExercises, so that these exercises can then be saved for the user and stored in a DynamoDB database
    listOfExercisesAdded.append(exercisesToAdd)
    if len(rows) == 0:  #if the MySQL query from above doesn't filter out any exercises that match the user's responses from previous page, the file WorkoutPlanner_3.html will be
    #rendered'
        return render_template('WorkoutPlanner_3.html', name=nameReceived)
    else:  #otherwise, the template WorkoutPlanner_3.html will be rendered
        return render_template('WorkoutPlanner_2.html', rows=rows, nameReceived=nameReceived, muscle_group=muscle_group)
```

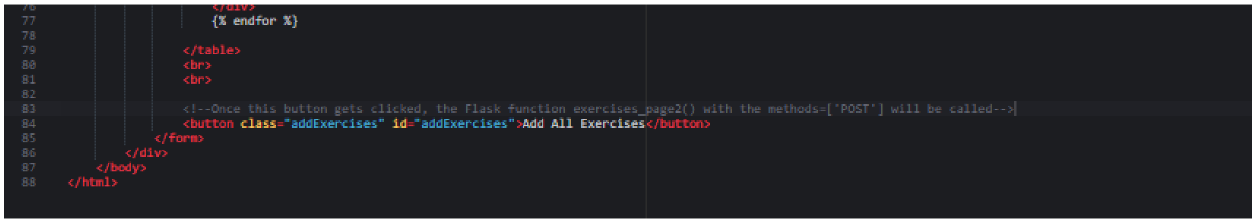
As shown in the second image of Figure A, the user's responses to the textbox and dropdowns from the home page get retrieved (refer to the third image) and then sent to the function exercises_page1() portrayed in the fourth image. From there, these values are used to filter out the appropriate exercises from the MySQL database based on the user's responses from the home page. However, it's important to note that if the user provides responses that don't contain any exercises filtered based on these responses from the MySQL database (refer to the second image), then they will be led to the webpage portrayed in the second image, where clicking the button "Back" takes them back to the home page.

Figure B

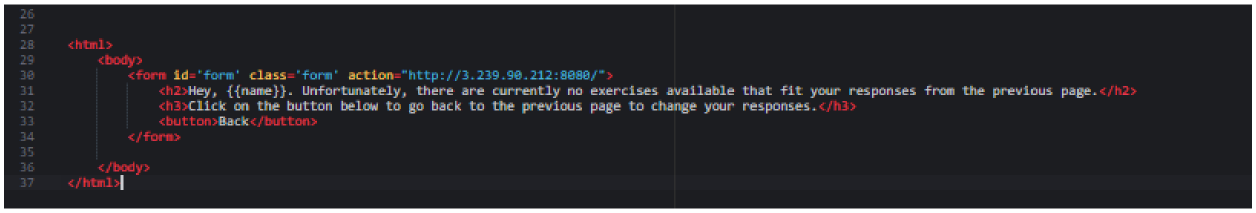


```
<body>
    <div class="container">
        <form method='POST' id="form" class="form">
            <!--nameReceived repreents the name of the user that was passed in from the Flask function exercises_page1()-->
            <h2>Hey, {{nameReceived}}! Please select which exercise(s) you want to add to your workout plan.</h2>
            <table>
                <tr><th>---------------Category---------------</th><th>---------------Exercises---------------</th><th>---------------Equipment---------------</th><th>---------------Difficulty----------

                <!--for every row that includes each exercise filtered from the MySQL database based on the user's responses, information pertaining to each exercise gets displayed onto
                separate lines-->
                {% for r in rows %}
                <div class="form-control">
                    <tr>

                        <td>
                            {{r[0]}}
                        </td>

                        <td>
                            {{r[1]}}
                        </td>

                        <td>
                            &nbsp;&nbsp;&nbsp;{{r[2]}}
                        </td>

                        <td>
                            {{r[3]}}
                        </td>
                    </tr>
                </div>
                {% endfor %}
```

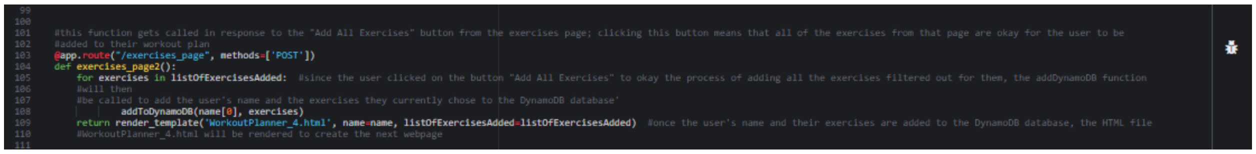
83:132 HTML Spaces: 4



```
                </div>
                {% endfor %}

            </table>
            <br>
            <br>

            <!--Once this button gets clicked, the Flask function exercises_page2() with the methods=['POST'] will be called-->
            <button class="addExercises" id="addExercises">Add All Exercises</button>
        </form>
    </div>
</body>
</html>
```



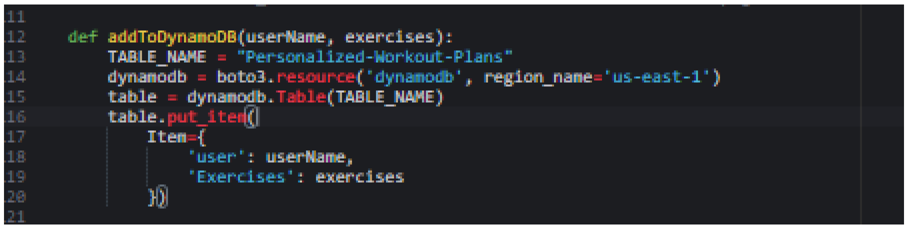
```
<html>
    <body>
        <form id='form' class='form' action="http://3.239.90.212:8080/">
            <h2>Hey, {{name}}. Unfortunately, there are currently no exercises available that fit your responses from the previous page.</h2>
            <h3>Click on the button below to go back to the previous page to change your responses.</h3>
            <button>Back</button>
        </form>

    </body>
</html>
```



```
#this function gets called in response to the "Add All Exercises" button from the exercises page; clicking this button means that all of the exercises from that page are okay for the user to be
#added to their workout plan
@app.route("/exercises_page", methods=['POST'])
def exercises_page2():
    for exercises in listOfExercisesAdded:  #since the user clicked on the button "Add All Exercises" to okay the process of adding all the exercises filtered out for them, the addDynamoDB function
    #will then
    #be called to add the user's name and the exercises they currently chose to the DynamoDB database'
            addToDynamoDB(name[0], exercises)
    return render_template('WorkoutPlanner_4.html', name=name, listOfExercisesAdded=listOfExercisesAdded)  #once the user's name and their exercises are added to the DynamoDB database, the HTML file
    #WorkoutPlanner_4.html will be rendered to create the next webpage
```

The first and second images show the HTML from the file WorkoutPlanner_2.html. Once the button "Add All Exercises" (refer to the first image from Figure A) gets pressed, the function exercises_page2() from Flask will be called, since it is specified to have methods=['POST'] (refer to the fourth image). Meanwhile, the third image shows the HTML file WorkoutPlanner_3.html. Once the button Back gets pressed, the user navigates to the home page, which is marked by the url http://3.239.90.212:8080/. From there, as shown in the fourth image, all the exercises the user was okay with adding from the exercises page will be added to the DynamoDB database along with their name (refer to Figure C). Once the information is added to the DynamoDB database, the HTML file WorkoutPlanner_4.html will be rendered, which deals with the design of the next webpage.

Figure C



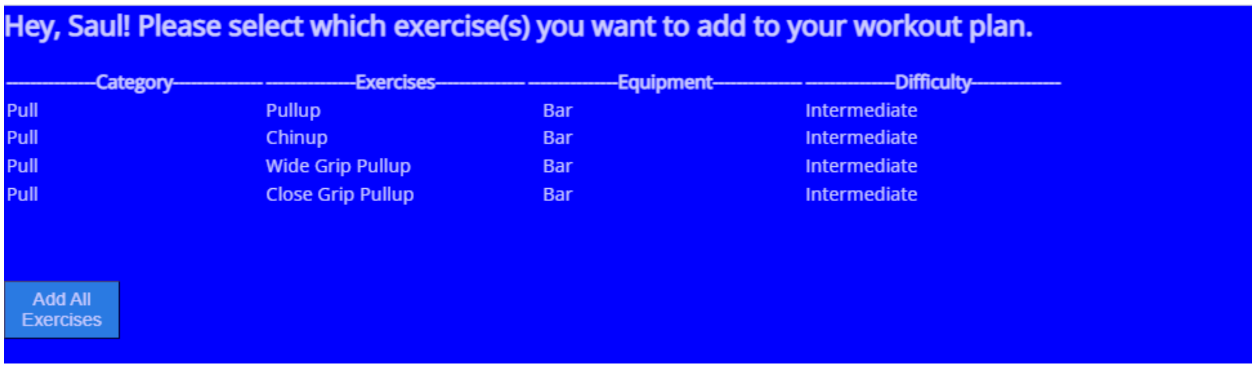
```python
def addToDynamoDB(userName, exercises):
    TABLE_NAME = "Personalized-Workout-Plans"
    dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
    table = dynamodb.Table(TABLE_NAME)
    table.put_item(
        Item={
            'user': userName,
            'Exercises': exercises
        })
```

This image shows the function addToDynamoDB(), which adds the user's name and the exercises selected from the exercises page into the DynamoDB database.

4. Displaying User's Workout Plan

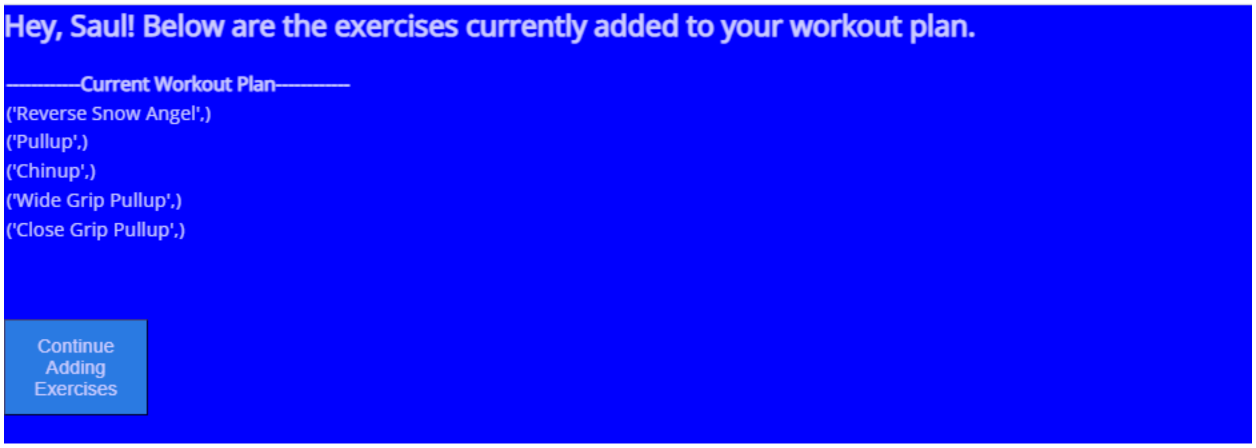
As mentioned in the previous section, once the user's name and exercises selected get added to the DynamoDB database, the HTML file WorkoutPlanner_4.html gets rendered, which uses HTML to create the design of the last webpage. The last webpage simply displays all the exercises the user has currently selected (refer to Figure A for how the last webpage looks like as well as the file WorkoutPlanner_4.html used to create that webpage).

Figure A



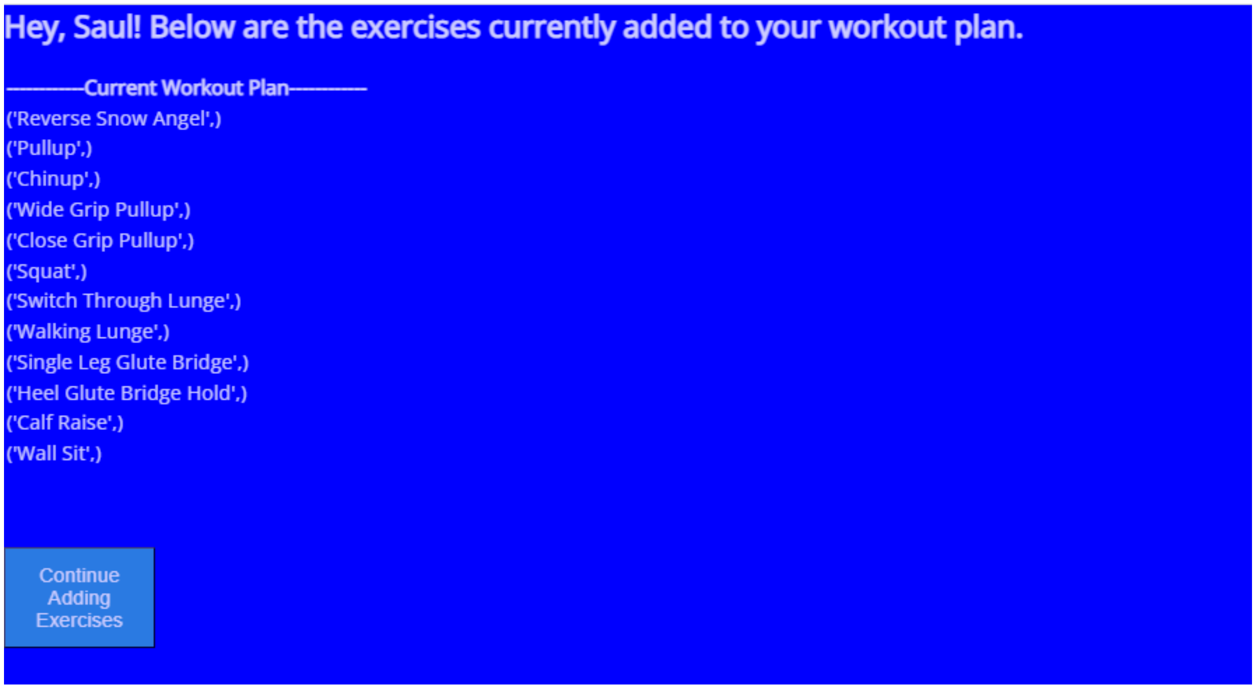
Hey, Saul! Please select which exercise(s) you want to add to your workout plan.

------------Category------------	------------Exercises------------	------------Equipment------------	------------Difficulty------------
Pull	Pullup	Bar	Intermediate
Pull	Chinup	Bar	Intermediate
Pull	Wide Grip Pullup	Bar	Intermediate
Pull	Close Grip Pullup	Bar	Intermediate



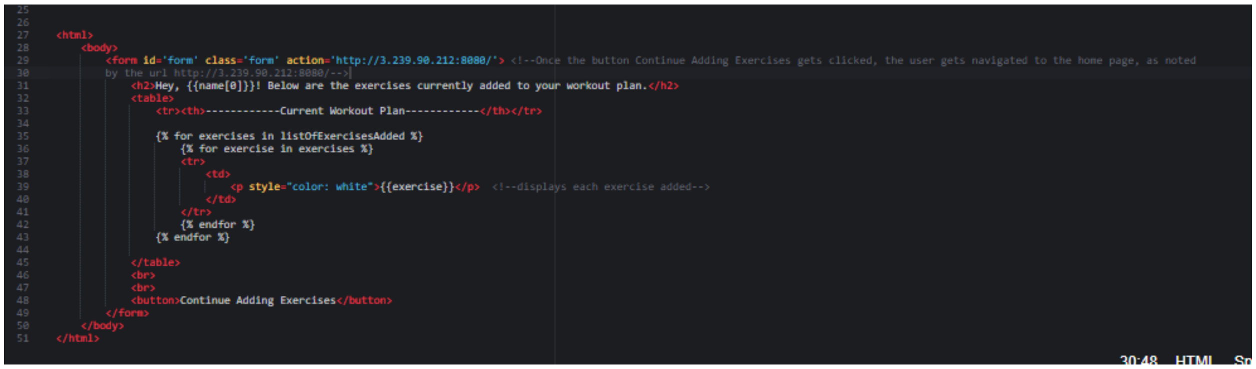
Hey, Saul! Below are the exercises currently added to your workout plan.

```
----------Current Workout Plan----------
('Reverse Snow Angel',)
('Pullup',)
('Chinup',)
('Wide Grip Pullup',)
('Close Grip Pullup',)
```



Hey, Saul! Below are the exercises currently added to your workout plan.

------------Current Workout Plan------------
('Reverse Snow Angel',)
('Pullup',)
('Chinup',)
('Wide Grip Pullup',)
('Close Grip Pullup',)
('Squat',)
('Switch Through Lunge',)
('Walking Lunge',)
('Single Leg Glute Bridge',)
('Heel Glute Bridge Hold',)
('Calf Raise',)
('Wall Sit',)



```


<html>
    <body>
        <form id='form' class='form' action='http://3.239.90.212:8080/'> <!--Once the button Continue Adding Exercises gets clicked, the user gets navigated to the home page, as noted
        by the url http://3.239.90.212:8080/-->
            <h2>Hey, {{name[0]}}! Below are the exercises currently added to your workout plan.</h2>
            <table>
                <tr><th>------------Current Workout Plan------------</th></tr>

                {% for exercises in listOfExercisesAdded %}
                    {% for exercise in exercises %}
                    <tr>
                        <td>
                            <p style="color: white">{{exercise}}</p>  <!--displays each exercise added-->
                        </td>
                    </tr>
                    {% endfor %}
                {% endfor %}

            </table>
            <br>
            <br>
            <button>Continue Adding Exercises</button>
        </form>
    </body>
</html>
```

The first image shows the exercises that are filtered based on a given user's responses from the home page. Once the button "Add All Exercises" gets pressed, the user then gets navigated to the webpage portrayed in the second image, which displays the exercises the user has currently selected. Note that the exercises a given user selects get saved. If the same user were to continue adding new exercises, I would still have those exercises saved from before (refer to the third image).

5. Sources Used

https://youtu.be/PMQUYTrwLjg

https://youtu.be/PMQUYTrwLjg

https://youtu.be/OWaQWpVd95k

https://stackoverflow.com/questions/2125509/how-do-i-set-the-size-of-an-html-text-box https://www.w3schools.com/csS/css_text.asp

https://stackoverflow.com/questions/13931571/how-can-change-width-of-dropdown-list

https://youtu.be/mwG4_qAa4AU

https://youtu.be/CCLWsiCJ3KI

https://stackoverflow.com/questions/9907460/how-to-set-the-width-of-the-website

https://youtu.be/Cg40KWtmHPY

https://www.scaler.com/topics/how-to-make-a-button-link-to-another-page-in-html/

https://stackoverflow.com/questions/3410198/how-to-assign-a-block-of-html-code-to-a-javascript-variable

https://stackoverflow.com/questions/3206344/passing-html-to-template-using-flask-jinja2

https://stackoverflow.com/questions/12655155/jinja2-for-loop-with-conditions#:~:text=You%20can%20combine%20for%20-loops%20with%20if%20-conditionals,else%20%25%7D%20no%20true%20items%20%7B%25%20endfor%20%25%7D

https://stackoverflow.com/questions/3727045/set-variable-in-jinja

https://stackoverflow.com/questions/13897001/how-to-simulate-while-loop-in-jinja2

https://youtu.be/R4i6k-yEdH8

https://www.w3schools.com/cssref/css_colors.php

https://stackoverflow.com/questions/27611216/how-to-pass-a-variable-between-flask-pages

https://www.geeksforgeeks.org/retrieving-html-from-data-using-flask/

https://stackoverflow.com/questions/42572173/how-can-i-add-a-default-text-in-my-html-input-text

https://www.w3schools.com/tags/tag_label.asp

https://stackoverflow.com/questions/37259740/passing-variables-from-flask-to-javascript

https://stackoverflow.com/questions/351409/how-to-append-something-to-an-array

https://youtu.be/7YyimN9QlPs

https://stackoverflow.com/questions/16601741/checkbox-value-in-array-javascript

https://documentation.bloomreach.com/engagement/docs/datastructures