Task 1.1

```
#TASK 1.1
def task1_1(filename): #find_most_common_ingredient
    ingredient_count = {}

    with open(filename, 'r') as file:
        lines = file.readlines()
        for line in lines:
            line = line.strip()
            if line and ',' in line:
                ingredient_name = line.split(',')[0]
                ingredient_count[ingredient_name] =
ingredient_count.get(ingredient_name, 0) + 1
    #file closed automatically
    most_common_ingredient = max(ingredient_count,
key=ingredient_count.get)
    occurrences = ingredient_count[most_common_ingredient]

    return most_common_ingredient, occurrences
```

Task 1.2

```
#TASK 1.2
def task1_2(arr): #insertion_sort
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
```

Task 1.3

```
#TASK 1.3
def task1_3():
    recipes = []
    #read recipes from file
    with open("RECIPES.txt", 'r') as file:
        for line in file:
            if ',' in line:
                parts = line.strip().split(',')
                ingredient_name = parts[0]
                quantity = float(parts[1])
                unit = parts[2]
                recipes.append((ingredient_name, quantity,
unit))
    #file closed automatically
    #The list recipes now contains the ingredient name,
quantity, unit
    insertion_sort(recipes) #sort list recipes
    #calculate total quantities for each ingredient
    ingredient_totals = {}
    for ingredient_name, quantity, unit in recipes:
        if ingredient_name in ingredient_totals:
            ingredient_totals[ingredient_name] =
[ingredient_totals[ingredient_name][0]+quantity, unit]
        else:
            ingredient_totals[ingredient_name] =
[quantity,unit]
    #output ingredient name, total quantity, unit
    print("\nTotal Quantities of Ingredients:")
    for ingredient_name, quantity_unit in
ingredient_totals.items():
        print(f"{ingredient_name}: {quantity_unit[0]}
{quantity_unit[1]}")
```

Task 2

```
import socket
hello=False

def process_input(input_text):
    input_text = input_text.lower()
    global hello
    if "hello" in input_text:
        if "hello" in input_text and hello==False:
            hello=True
            return "Hi, how are you?"
        else:
            return "Hello again, welcome back!"
    elif "thanks" in input_text or "thank you" in input_text:
        return "You are most welcome."
    elif "i " in input_text.lower() and " you" in
input_text.lower():
        parts = input_text.split()
        if parts.index('i') < parts.index('you'):
            indexA = parts.index('i')+1
            return f"You {parts[indexA]} me? I really
{parts[indexA]} you too."
        else:
            return "Sorry, I do not understand..."
    else:
        return "Sorry, I do not understand..."

def main():
    server_socket = socket.socket()
    server_socket.bind(("127.0.0.1", 12345))
    server_socket.listen()
    print("Chatterbot server is listening.\n")
    while True:
        client_socket, client_address =
server_socket.accept()
        while True:
            try:
                user_input =
client_socket.recv(1024).decode()
```

```
                if not user_input:
                    break
                if user_input.lower() == "exit\n":
                    break
                print("Client:",user_input.strip())
                response = process_input(user_input)
                print("Chatterbot:",response)
                print('\n')
                client_socket.send(response.encode()+b'\n')

            except:
                print("Client disconnected.")
                break
        break

    print("Server is shutting down.")
    client_socket.close()
    server_socket.close()

main()
```

Task 3.1

```
class Node:
    def __init__(self, data):
        self.data = data
        self.pointer = None

class LinkedList:
    def __init__(self):
        self.start = None

    def insert_last(self, current, data):
        if self.start is None:
            self.start = Node(data)
        else:
            if current.pointer == None:
                current.pointer = Node(data)
            else:
                self.insert_last(current.pointer, data)

    def display(self, num_nodes):
        current = self.start
        count = 0
        while current is not None and (num_nodes == 0 or
count < num_nodes):
            print(f"ID: {current.data[0]}, Name:
{current.data[1]}, Score: {current.data[2]}")
            current = current.pointer
            count += 1

def main():
    linked_list = LinkedList()
    with open("GAMERS.txt",'r') as file:
        for line in file:
            gamer_id, name, score = line.strip().split(',')
            linked_list.insert_last(linked_list.start,
(gamer_id, name, int(score)))
    #file closed automatically
    linked_list.display(8)

main()
```

Task 3.2

```
class HashTable:
    def __init__(self, size):
        self.size = size
        self.slots = [None] * size

    def hash_function(self, key):
        return int(key) % self.size

    def insert_record(self, key, data):
        index = self.hash_function(key)
        if self.slots[index] is None:
            self.slots[index] = LinkedList()

self.slots[index].insert_last(self.slots[index].start, data)

    def display_records(self, slot_number):
        if 0 <= slot_number < self.size:
            slot = self.slots[slot_number]
            if slot is not None:
                num_nodes_to_display = 0
                slot.display(num_nodes_to_display)
            else:
                print("Slot is empty.")
        else:
            print("Invalid slot number.")

#copy data from linked list to hash table
def linked_list_to_hash_table(linked_list, size):
    hash_table = HashTable(size)
    current = linked_list.start
    while current is not None:
        hash_table.insert_record(current.data[0],
current.data)
        current = current.pointer
    return hash_table

def main():
    linked_list = LinkedList()
    with open("GAMERS.txt",'r') as file:
        for line in file:
            gamer_id, name, score = line.strip().split(',')
            linked_list.insert_last(linked_list.start,
(gamer_id, name, int(score)))
    #file closed automatically
    linked_list.display(8)

    hash_table_size = 401
    hash_table = linked_list_to_hash_table(linked_list,
hash_table_size)
    # Display the records in slot number 14
    slot_number = 14
    print()
    print(f"Records in slot number {slot_number}:")
    hash_table.display_records(slot_number)

main()
```

Task 4.1

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def homepage():
    return render_template('homepage.html')

if __name__ == '__main__':
    app.run()
```

```
<head>
    <title>Fitness Club</title>
</head>
<body>
    <h1>Welcome to Fitness Club</h1>
        <h3>Member Details</h3>
        <h3>Fitness Statistics</h3>
        <h3>Add Fitness Record</h3>
</body>
```

Task 4.2

```
SELECT Name, Gender, Age, Weight, Height, Max(WorkoutDate)
FROM Member
LEFT OUTER JOIN FitnessRecord ON member.memberID =
FitnessRecord.MemberID
GROUP BY Name
ORDER BY Gender ASC, Name ASC;
```

```
@app.route('/member_details')
```

```
def member_details(): #display member details
    conn = sqlite3.connect('fitness.db')
    cursor = conn.cursor()

    query = """SELECT Name, Gender, Age, Weight, Height,
Max(WorkoutDate) FROM Member LEFT OUTER JOIN FitnessRecord ON
member.memberID = FitnessRecord.MemberID GROUP BY Name
ORDER BY Gender ASC, Name ASC; """
    cursor.execute(query)
    results = cursor.fetchall()
    conn.close()
    return render_template('member_details.html',
members=results)
```

```
<head>
    <title>Member Details</title>
</head>
<body>
    <h1>Member Details</h1>
    <table>
            <tr>
                <th>Name</th>
                <th>Gender</th>
                <th>Age</th>
                <th>Latest Weight</th>
                <th>Latest Height</th>
                <th>Workout Date</th>
            </tr>
            {% for member in members %}
            <tr>
                <td>{{ member[0] }}</td>
                <td>{{ member[1] }}</td>
                <td>{{ member[2] }}</td>
                <td>{{ member[3] }}</td>
                <td>{{ member[4] }}</td>
                <td>{{ member[5] }}</td>
            </tr>
            {% endfor %}
    </table>
</body>
```

Task 4.3

```
SELECT Gender, COUNT(*), ROUND(AVG(Age), 1),
ROUND(AVG(Weight), 1), ROUND(AVG(Height), 1) FROM Member
LEFT OUTER JOIN FitnessRecord ON Member.MemberID =
FitnessRecord.MemberID AND WorkoutDate = (SELECT
MAX(WorkoutDate) FROM FitnessRecord WHERE
FitnessRecord.MemberID = Member.MemberID)
GROUP BY Gender
```

```
@app.route('/statistics')
```

```
def statistics():
    conn = sqlite3.connect('fitness.db')
    cursor = conn.cursor()

    query = """SELECT Gender, COUNT(*), ROUND(AVG(Age), 1),
ROUND(AVG(Weight), 1), ROUND(AVG(Height), 1) FROM Member
LEFT OUTER JOIN FitnessRecord ON Member.MemberID =
FitnessRecord.MemberID AND WorkoutDate = (SELECT
MAX(WorkoutDate) FROM FitnessRecord WHERE
FitnessRecord.MemberID = Member.MemberID)
GROUP BY Gender"""
    cursor.execute(query)
    results = cursor.fetchall()
    return render_template('statistics.html',
statistics=results)
```

```
<head>
    <title>Fitness Statistics</title>
</head>
<body>
    <h1>Fitness Statistics</h1>
    <table>
            <tr>
                <th>Gender</th>
                <th>Total Members</th>
                <th>Avg Age</th>
                <th>Avg Weight</th>
                <th>Avg Height</th>
            </tr>
            {% for stat in statistics %}
            <tr>
                <td>{{ stat[0] }}</td>
                <td>{{ stat[1] }}</td>
                <td>{{ stat[2] }}</td>
                <td>{{ stat[3] }}</td>
                <td>{{ stat[4] }}</td>
            </tr>
            {% endfor %}
    </table>
</body>
```

Task 4.4

```
<head>
    <title>Add Fitness Record</title>
</head>
<body>
    <h1>Add Fitness Record</h1>
    <form action="/add_record" method="post">
        <label>Member ID:</label>
        <input type="text" name="member_id"><br>

        <label>Weight (kg):</label>
        <input type="number" name="weight"><br>

        <label>Height (cm):</label>
        <input type="number" name="height"><br>

        <label>Workout Date:</label>
        <input type="date" name="workout_date"><br>

        <input type="submit" value="Add Record">
    </form>
</body>
```

```
@app.route('/add_record', methods=['GET','POST'])
def add_record():
    if request.method == 'GET':
        return render_template('add_record.html')
    elif request.method == 'POST':
        member_id = request.form['member_id']
        weight = float(request.form['weight'])
        height = float(request.form['height'])
        workout_date = request.form['workout_date']

        conn = sqlite3.connect('fitness.db')
        cursor = conn.cursor()

        query = """
        INSERT INTO FitnessRecord (MemberID, Weight, Height,
WorkoutDate) VALUES (?, ?, ?, ?)
        """
        cursor.execute(query, (member_id, weight, height,
workout_date))

        conn.commit()
        conn.close()
        return redirect(url_for('homepage'))
```