|  |
| --- |
| **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')) |