人工智能与计算机学院

School of Artificial Intelligence and Computer Science Nantong University

Course Name	:	Algorithm Design and Analysis; Curriculum Design for Data Structure and Algorithms.
Report Title	:	Student Transcript Management System .
Lecturer	:	王皓晨 .
Project Date	:	2024-12-30 .
Student Name	:	Mahim Al Muntashir Billah .
Class	:	2021 .
Student No	:	2130130248 .

Contents

Introduction

The Student Transcript Management System is a full-fledged software application to facilitate management of student records. The system has the capability of adding new student records, display all the records, search for particular records and update or delete records. The records are saved in the CSV format to a file to prevent data loss between program runs. This paper presents the details of the program, the code for each of the modules and the overall experience learnt from the project.

Program Code

The following sections contain the code for each module, along with brief explanations of their functionality:

1. Main Menu Display

The **Main Menu Display** module will be created in order to offer the user a clear and easy to read menu so that he or she can choose and solve an option from a simply list of options available. Here's a breakdown of the code:

```
# Function to display the menu
def display_menu():
    print("****************************************")
    print("1. CREATE NEW")
    print("2. SHOW ALL")
    print("3. QUERY")
    print("0. SAVE AND QUIT")
    print("****************************************")
```

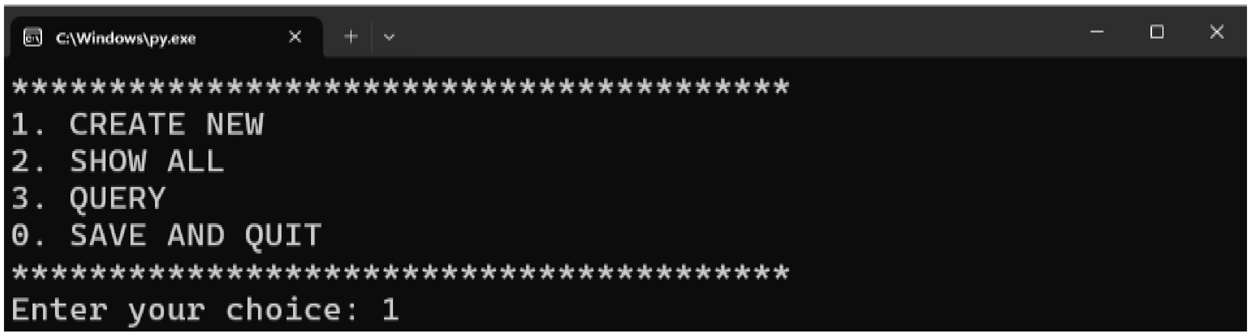
1. **Function Definition (`def display_menu():`)**:
    - This function will be used to display the menu and hence making it modular and reusable.
2. **Menu Structure**:
    - The `print()` statements are utilized to make smooth and clear options for users to select.
    - These options are set in the order that completely correspond to the number the user enters making the correspondence between the number and the action clear.
3. **Menu Options**:
    - **Option 1**: "CREATE NEW" - starts the feature that creates new records for the students.
    - **Option 2**: "SHOW ALL" - Displays all existing student records in a table.
    - **Option 3**: "QUERY" - Allows the user to search for a specific student by name.
    - **Option 0**: "SAVE AND QUIT" - Saves all data to a CSV file and exits the program.
4. **Separator (**)**:
    - It also helps to makes reading easier and add visual distinction to the menu, thus proves to be beneficial for users.

Purpose of the Function

- The function aims at serving as an interface where the user is directed to about how to move around within the program.

Execution Flow

- After displaying the menu, the program waits for user input (e.g., "Enter your choice:").
- Based on the user selection, the input runs the respective functionality on the processed input.



```
****************************************
1. CREATE NEW
2. SHOW ALL
3. QUERY
0. SAVE AND QUIT
****************************************
Enter your choice: 1
```

Output Example

* Data Loading

The **Data Loading** is a very important part of the program that has the function of loading pre-existing student data from a CSV file at the program run time. This makes ensured the record saved before is available for use, creating data persistence when the program is run again at a later time.

Code Analysis:

```python
# Load data from the CSV file
def load_data():
    try:
        with open("student.csv", "r") as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                row['Score'] = int(row['Score'])  # Convert Score back to integer
                data.append(row)
    except FileNotFoundError:
        print("No previous data found. Starting with an empty dataset.")
```

1. **Function Definition (`def load_data():`):**
   - It is a container for the set of operations for reading and manipulating data stored in a file.
   - This approaches means that components can be reused and separated in the architecture design.
2. **File Handling (`with open("student.csv", "r") as csvfile`):**
   - Opens the file `student.csv` in read mode (`"r"`).
   - The `with` statement guarantees the closing of the file after the operation even if an error encounter during operation.

3. **Reading the CSV File (`csv.DictReader`):**
    - `csv.DictReader(csvfile)` reads the CSV file and converts each row into a dictionary, where the column headers are the keys.
4. **Processing Each Row (`for row in reader`):**
    - Iterates through each row in the CSV file.
    - Converts the `Score` field from a string to an integer using `int(row['Score'])`.
    - Appends the processed row (as a dictionary) to the `data` list, which acts as the program's in-memory database.
5. **Exception Handling (`except FileNotFoundError`):**
    - If the file `student.csv` is not found, a `FileNotFoundError` is raised.
    - The program handles this error gracefully by displaying a message (`"No previous data found. Starting with an empty dataset."`) and initializing an empty dataset.

Purpose of the Function

- Checks through every record saved to ensure that all student records which/that had been saved are intact to allow the program to manipulate them when the program starts.
- Copes with cases when the data file is failed to be recognized and lets the program work properly.

Execution Flow

1. When the program starts, `load_data()` is called.
2. If `student.csv` exists:
    - Reads the file and populates the `data` list with student records.
3. If `student.csv` does not exist:
    - Prints an error message and initializes `data` as an empty list.

2. Create New Record

The **Create New Records** module enables creation of new records of students by providing details of the student. Here's a detailed breakdown of the code:

```
# Function to create a new student
def create_new():
    while True:
        student = {
            "Major": input("Major: "),
            "ID": input("ID: "),
            "Name": input("Name: "),
            "Gender": input("Gender: "),
            "Subject": input("Subject: "),
            "Score": int(input("Score: ")),
        }
        data.append(student)
        print("Student info successfully created.")
        cont = input("Would you hope to continue to create a new one? (Y/N): 
").strip().upper()
```

```
        if cont != 'Y':
            break
```

1. **Function Definition (`def create_new():`)**:
   - Responsible for implementation of operations of capturing new student details and entering a new record.
2. **Infinite Loop (`while True`)**:
   - Guarantees that the user is able to add multiple records of students consecutively with the need of having to re-do the process.
   - In this case, the loop comes to an end only if the user does not want to go on with the conversation actively.
3. **Input Collection**:
   - Prompts the user to input details for each field of the student record:
     - **Major**: The department or field of study.
     - **ID**: A unique identifier for the student.
     - **Name**: The student's name.
     - **Gender**: The student's gender.
     - **Subject**: The course or subject being recorded.
     - **Score**: The student's score for the subject, converted to an integer using `int()`.
   - These inputs are collected into a dictionary called `student`.
4. **Appending the Record**:
   - The newly created `student` dictionary is appended to the `data` list, which acts as the program's in-memory database.
5. **Confirmation Message**:
   - After adding the record, the program prints a message: `"Student info successfully created."`, confirming the operation's success.
6. **Continuation Prompt**:
   - The user is asked if they wish to add another record (`"Would you hope to continue to create a new one? (Y/N): "`).
   - Additional spaces are removed, and to maintain a relatively uniform data type, all the input strings are converted to upper case letters.
   - If the user enters anything other than `'Y'` the loop terminates and the function is over.

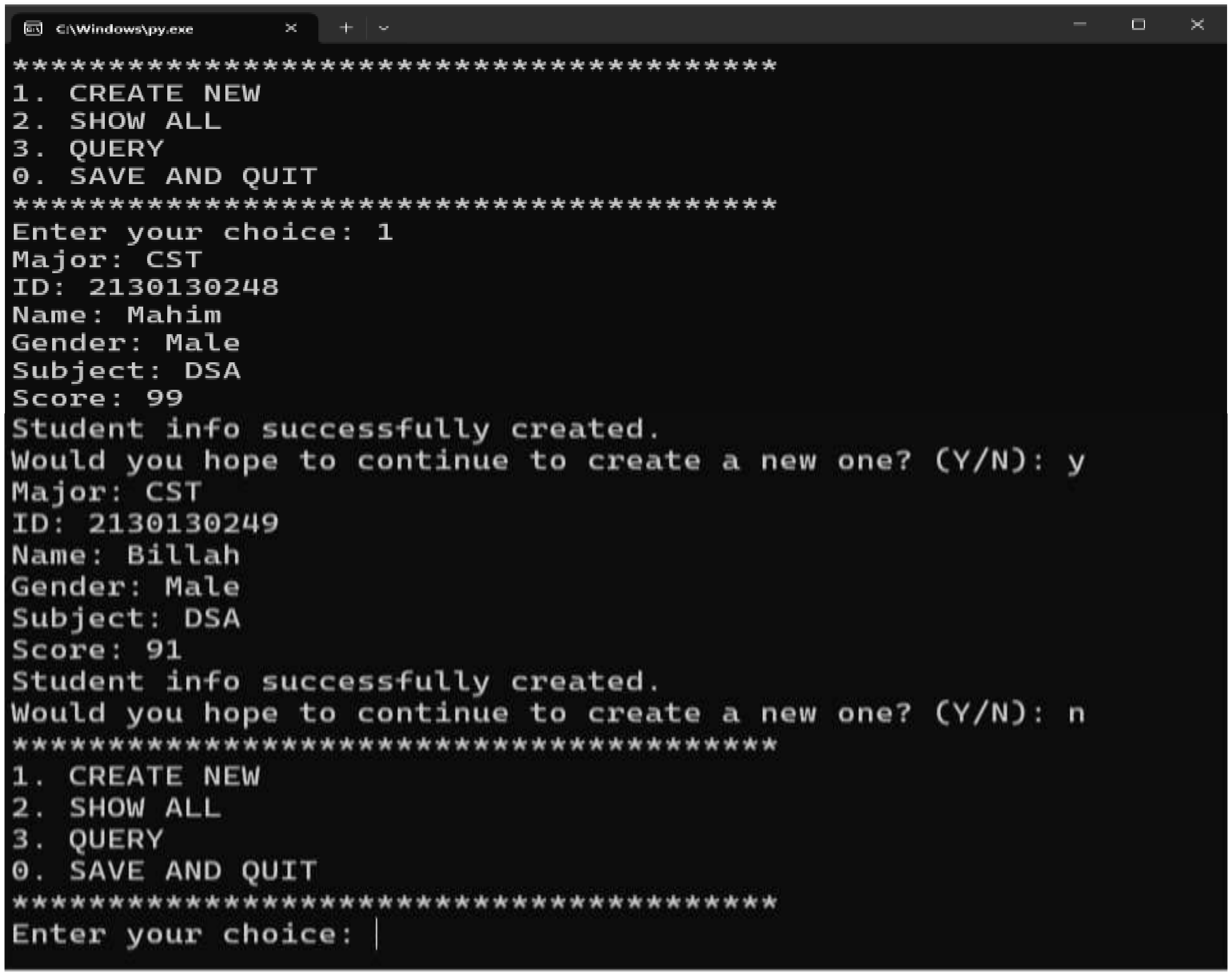
Purpose of the Function

- It is the module that forms the basis of the data entry of the program capturing student records as and when the users input this data.

Execution Flow

1. The function asks the user to enter students' data.
2. A new record is formed and appended as a new entry to the `data` list.
3. Nuances: At the end of the process the user is asked if they want to add records again.
4. This continues until and unless the user decides to exit the process.

Output Example

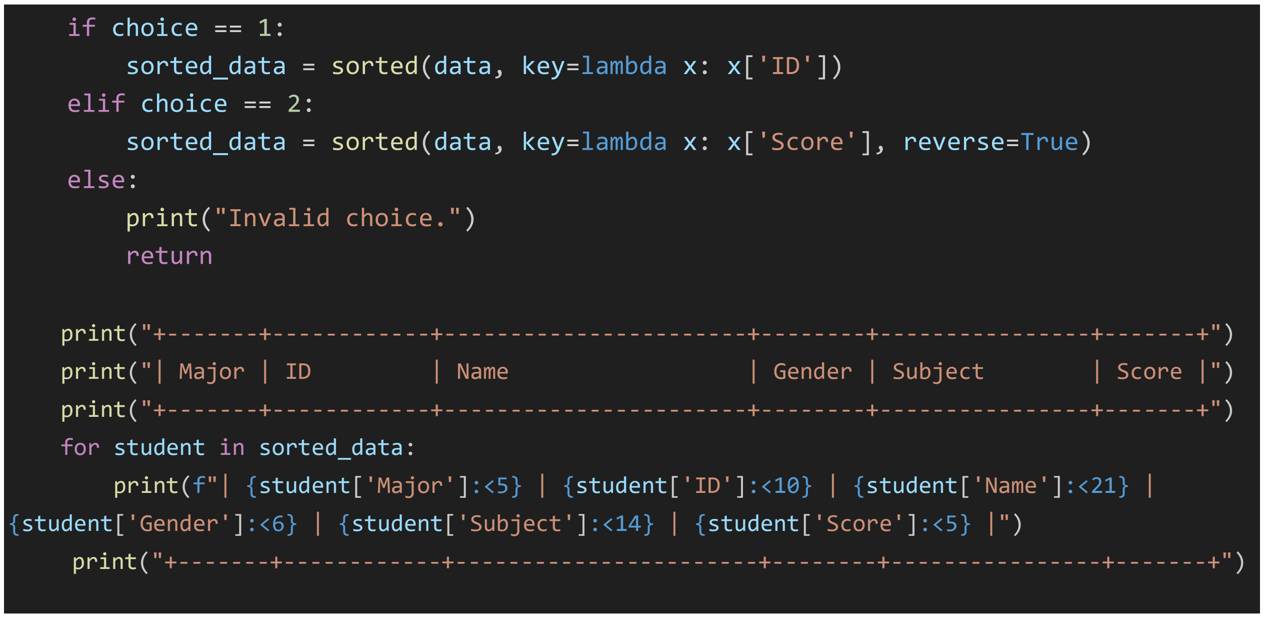


```
****************************************
1. CREATE NEW
2. SHOW ALL
3. QUERY
0. SAVE AND QUIT
****************************************
Enter your choice: 1
Major: CST
ID: 2130130248
Name: Mahim
Gender: Male
Subject: DSA
Score: 99
Student info successfully created.
Would you hope to continue to create a new one? (Y/N): y
Major: CST
ID: 2130130249
Name: Billah
Gender: Male
Subject: DSA
Score: 91
Student info successfully created.
Would you hope to continue to create a new one? (Y/N): n
****************************************
1. CREATE NEW
2. SHOW ALL
3. QUERY
0. SAVE AND QUIT
****************************************
Enter your choice:
```

3. Show All Records

The **Show All Records** module presents all the students records in a tabular form arranged by either ID or Score as chosen by the user. Here's a detailed breakdown of the code:

```python
# Function to show all students
def show_all():
    if not data:
        print("No students found.")
        return
    print("Please choose the mode for display:")
    print("1. Sort by ID")
    print("2. Sort by Score")
    choice = int(input("Choice: "))
```



```
    if choice == 1:
        sorted_data = sorted(data, key=lambda x: x['ID'])
    elif choice == 2:
        sorted_data = sorted(data, key=lambda x: x['Score'], reverse=True)
    else:
        print("Invalid choice.")
        return

    print("+-------+------------+-----------------------+--------+----------------+-------+")
    print("| Major | ID         | Name                  | Gender | Subject        | Score |")
    print("+-------+------------+-----------------------+--------+----------------+-------+")
    for student in sorted_data:
        print(f"| {student['Major']:<5} | {student['ID']:<10} | {student['Name']:<21} |
{student['Gender']:<6} | {student['Subject']:<14} | {student['Score']:<5} |")
    print("+-------+------------+-----------------------+--------+----------------+-------+")
```

1. **Function Definition (`def show_all():`)**:
   - It is a function that forms the basis for displaying all the records of students.
2. **Check for Empty Data (`if not data:`)**:
   - If having no students in the `data` list, the function displays the message (`"No students found."`). and stops there, which will not lead to sorting or displaying errors.
3. **Display Sorting Options**:
   - Prompts the user to select how the records should be sorted:
     - **Option 1**: Sort by ID.
     - **Option 2**: Sort by Score.
4. **User Input Handling (`choice = int(input("Choice: "))`)**:
   - The user's choice determines the sorting criteria:
     - If `choice == 1`, the records are sorted by `ID` in ascending order using `sorted(data, key=lambda x: x['ID'])`.
     - If `choice == 2`, the records are sorted by `Score` in descending order using `sorted(data, key=lambda x: x['Score'], reverse=True)`.
   - If an invalid choice is entered, the function prints `"Invalid choice."` and exits.
5. **Formatted Table Display**:
   - The records have been sorted logically and the size of columns are uniform across the table.
   - Each row in the table represents a student record, showing:
     - **Major**: The student's field of study.
     - **ID**: The student's unique identifier.
     - **Name**: The student's full name.
     - **Gender**: The student's gender.
     - **Subject**: The subject or course name.
     - **Score**: The student's score in the subject.
6. **Loop to Display Records (`for student in sorted_data`)**:

    - Iterates through the sorted list (`sorted_data`) and prints each record in the formatted table.

Purpose of the Function

- Maybe as a means to make it easy for the user to retrieve all records in the system in any order the user would desire.
- Improves data presentation and retrievability.

Sorting Algorithms and Time Complexity

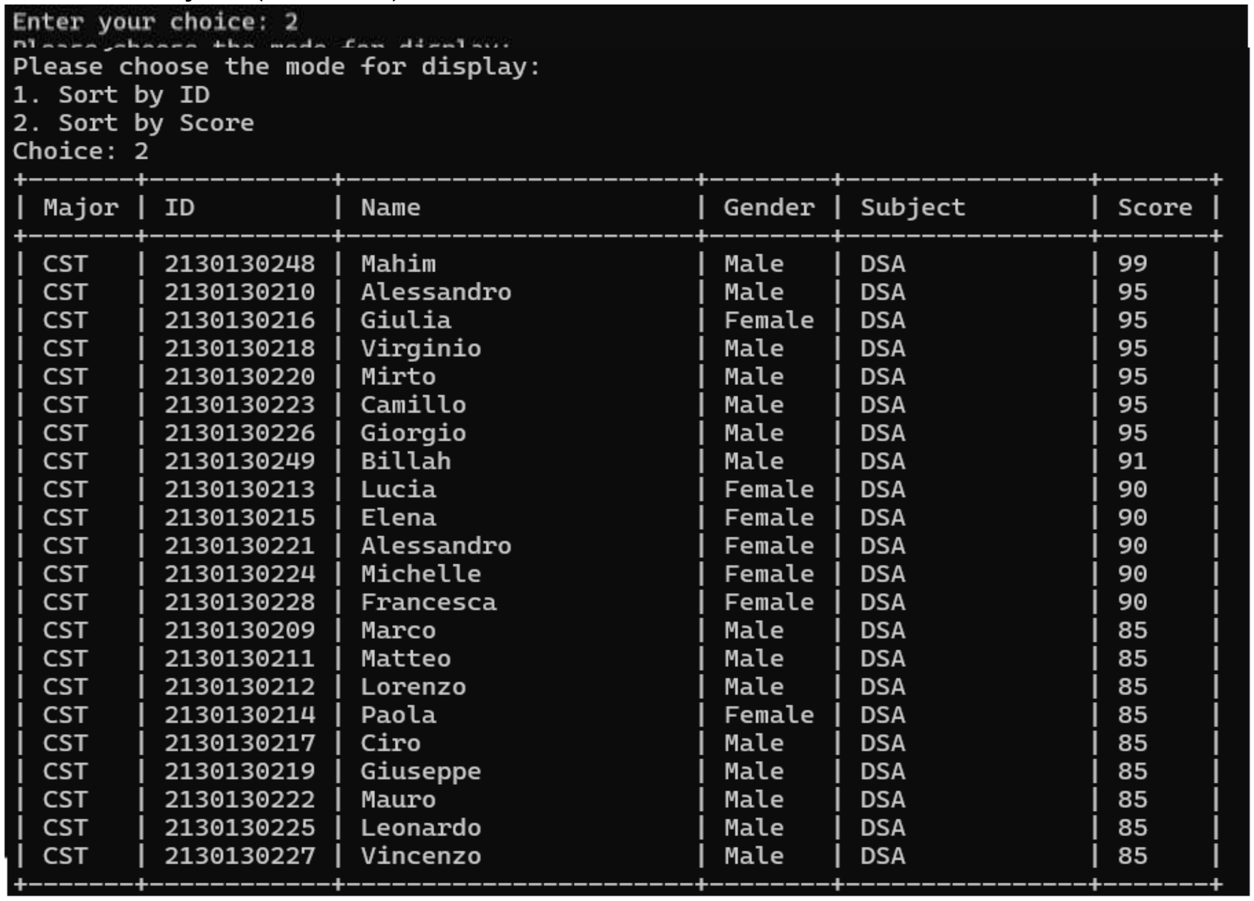
- **Sorting by ID**:
    - Uses Python's built-in `sorted()` function with a lambda function as the key.
    - **Time Complexity**: O(nlogn), where n is the number of records.
- **Sorting by Score**:
    - Similar to sorting by ID but sorts in descending order (`reverse=True`).
    - **Time Complexity**: O(nlogn).

Execution Flow

1. The function is called, and it checks if there are any student records in the `data` list.
2. If records exist, the user is prompted to choose the sorting method.
3. The records are sorted and displayed in a formatted table.
4. If the user enters an invalid choice, an error message is shown, and the function exits.

Output Examples

1. **Sort by ID (Choice: 1)**:



```
Enter your choice: 2
Please choose the mode for display:
1. Sort by ID
2. Sort by Score
Choice: 2
+-------+------------+----------------------+--------+----------------+-------+
| Major | ID         | Name                 | Gender | Subject        | Score |
+-------+------------+----------------------+--------+----------------+-------+
| CST   | 2130130248 | Mahim                | Male   | DSA            | 99    |
| CST   | 2130130210 | Alessandro           | Male   | DSA            | 95    |
| CST   | 2130130216 | Giulia               | Female | DSA            | 95    |
| CST   | 2130130218 | Virginio             | Male   | DSA            | 95    |
| CST   | 2130130220 | Mirto                | Male   | DSA            | 95    |
| CST   | 2130130223 | Camillo              | Male   | DSA            | 95    |
| CST   | 2130130226 | Giorgio              | Male   | DSA            | 95    |
| CST   | 2130130249 | Billah               | Male   | DSA            | 91    |
| CST   | 2130130213 | Lucia                | Female | DSA            | 90    |
| CST   | 2130130215 | Elena                | Female | DSA            | 90    |
| CST   | 2130130221 | Alessandro           | Female | DSA            | 90    |
| CST   | 2130130224 | Michelle             | Female | DSA            | 90    |
| CST   | 2130130228 | Francesca            | Female | DSA            | 90    |
| CST   | 2130130209 | Marco                | Male   | DSA            | 85    |
| CST   | 2130130211 | Matteo               | Male   | DSA            | 85    |
| CST   | 2130130212 | Lorenzo              | Male   | DSA            | 85    |
| CST   | 2130130214 | Paola                | Female | DSA            | 85    |
| CST   | 2130130217 | Ciro                 | Male   | DSA            | 85    |
| CST   | 2130130219 | Giuseppe             | Male   | DSA            | 85    |
| CST   | 2130130222 | Mauro                | Male   | DSA            | 85    |
| CST   | 2130130225 | Leonardo             | Male   | DSA            | 85    |
| CST   | 2130130227 | Vincenzo             | Male   | DSA            | 85    |
+-------+------------+----------------------+--------+----------------+-------+
```

2. **Sort by Score (Choice: 2)**:

4. Query Records

The **Query Records** module enables users to search for a specific student by name, display the matching record(s), and perform operations such as modifying or deleting the student information:

```python
# Function to query a student
def query():
    name = input("Please input the student name for query: ")
    found_students = [s for s in data if s['Name'].lower() == name.lower()]

    if not found_students:
        print("No student found with that name.")
        return

    print("+-------+------------+----------------------+--------+----------------+-------+")
    print("| Major | ID         | Name                 | Gender | Subject        | Score |")
    print("+-------+------------+----------------------+--------+----------------+-------+")
    for student in found_students:
        print(f"| {student['Major']:<5} | {student['ID']:<10} | {student['Name']:<21} |
{student['Gender']:<6} | {student['Subject']:<14} | {student['Score']:<5} |")
    print("+-------+------------+----------------------+--------+----------------+-------+")

    print("Please choose the operation on this student:")
    print("1. Modify")
    print("2. Delete")
    print("0. Back to the menu")
    choice = int(input("Choice: "))

    if choice == 1:
        modify(found_students[0])
    elif choice == 2:
        delete(found_students[0])
```

1. **Function Definition (`def query():`)**:
    - The function provides an interface for the generic navigation and for the modification/deletion of an individual student record.
2. **User Input (`name = input(...)`)**:
    - Asks the user the name of the student about whom he/she wants information.
    - the text is converted to lower case for a case insensitive search.
3. **Search for the Student (`found_students`)**:
    - Uses a list comprehension to filter records in the `data` list where the `Name` matches the input (case-insensitive).
    - The result is stored in the `found_students` list.
4. **Check for No Results (`if not found_students`)**:
    - If no matching records are found, the function prints `"No student found with that name."` and exits.
5. **Display Matching Records**:
    - When such records are identified at the database, the function presents the values in a tabular form.
    - The table has **Major**, **ID**, **Name**, **Gender**, **Subject**, and **Score** as the column variables.
6. **Prompt for Further Action**:
    - The user is prompted to choose an operation on the queried student:
        - **1. Modify**: Calls the `modify()` function to update the student's details.
        - **2. Delete**: Calls the `delete()` function to remove the student from the `data` list.
        - **0. Back to the menu**: Exits the function without performing any action.
7. **Function Calls for Actions**:
    - If the user chooses **Modify (1)**, the first matching record is passed to the `modify()` function.
    - If the user chooses **Delete (2)**, the first matching record is passed to the `delete()` function.

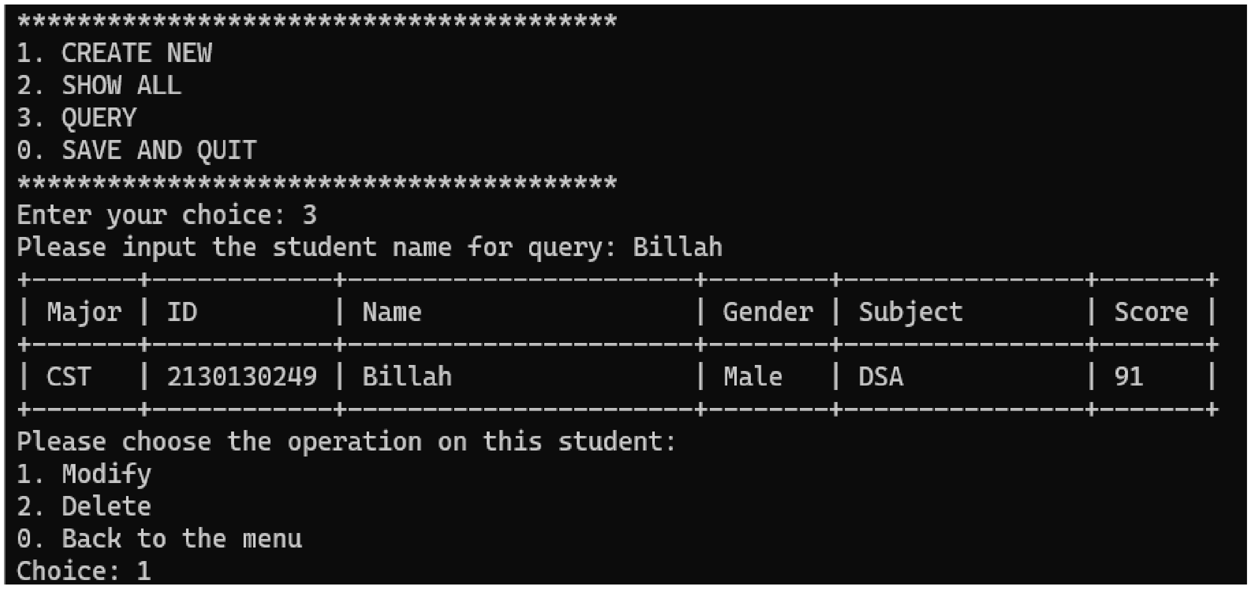
Purpose of the Function

- Makes it possible to quickly search records by name.
- Users can use it to edit or delete records immediately after using the search functionality.
- improves the operation of the program by providing logical manipulation of a subset of records.

Execution Flow

1. The user will then type the name of the student he would like to search.
2. Records, which correlate with entered values, are sought.
3. If there are no matches returned then a message is shown, and the function is terminated.
4. If matches are found, they consist of records given in a table form.
5. This means that the user will have the choice of either editing or even deleting that record.

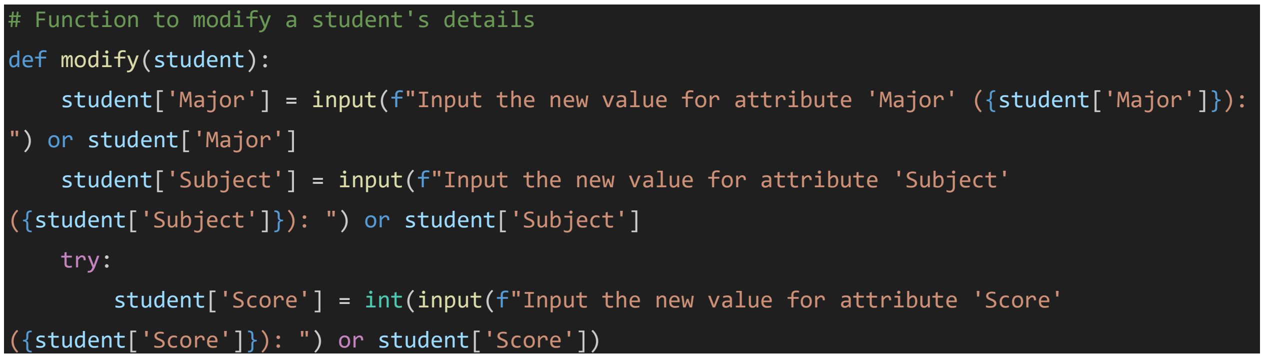
Example Outputs



```
****************************************
1. CREATE NEW
2. SHOW ALL
3. QUERY
0. SAVE AND QUIT
****************************************
Enter your choice: 3
Please input the student name for query: Billah
+-------+------------+-----------------------+--------+----------------+-------+
| Major | ID         | Name                  | Gender | Subject        | Score |
+-------+------------+-----------------------+--------+----------------+-------+
| CST   | 2130130249 | Billah                | Male   | DSA            | 91    |
+-------+------------+-----------------------+--------+----------------+-------+
Please choose the operation on this student:
1. Modify
2. Delete
0. Back to the menu
Choice: 1
```

5. Modify Records

The **Modify Records** module grants users an opportunity to alter precise features of a particular student record. Here's how the code was executed:



```python
# Function to modify a student's details
def modify(student):
    student['Major'] = input(f"Input the new value for attribute 'Major' ({student['Major']}): ") or student['Major']
    student['Subject'] = input(f"Input the new value for attribute 'Subject' ({student['Subject']}): ") or student['Subject']
    try:
        student['Score'] = int(input(f"Input the new value for attribute 'Score' ({student['Score']}): ") or student['Score'])
    except ValueError:
        print("Invalid score, keeping the old value.")
    print("Modification is successful!")

    # Print the modified student details in table format
    print("+-------+------------+-----------------------+--------+----------------+-------+")
    print("| Major | ID         | Name                  | Gender | Subject        | Score |")
    print("+-------+------------+-----------------------+--------+----------------+-------+")
    print(f"| {student['Major']:<5} | {student['ID']:<10} | {student['Name']:<21} |
{student['Gender']:<6} | {student['Subject']:<14} | {student['Score']:<5} |")
    print("+-------+------------+-----------------------+--------+----------------+-------+")
```

1. **Function Definition (`def modify(student):`)**:
   - This function is designed to update specific attributes of a given student record.
2. **Updating the `Major` Field**:
   - Prompts the user to input a new value for the `Major` attribute.
   - If the user presses Enter without typing a new value, the current value is retained using the `or student['Major']` statement.
3. **Updating the `Subject` Field**:
   - Similar to the `Major` field, prompts the user for a new value.
   - Retains the current value if no input is provided.
4. **Updating the `Score` Field**:
   - Prompts the user to input a new score.
   - Converts the input to an integer using `int()`.
   - If the input is invalid (e.g., not a number), a `ValueError` is raised, and the program prints `"Invalid score, keeping the old value."`.
5. **Confirmation Message**:
   - After updating the attributes, the function prints `"Modification is successful!"` to confirm the changes.
6. **Displaying the Updated Record**:
   - The record of the modified student now appears in another table format for greatest visibility.
   - The table includes columns for **Major**, **ID**, **Name**, **Gender**, **Subject**, and **Score**.

Purpose of the Function

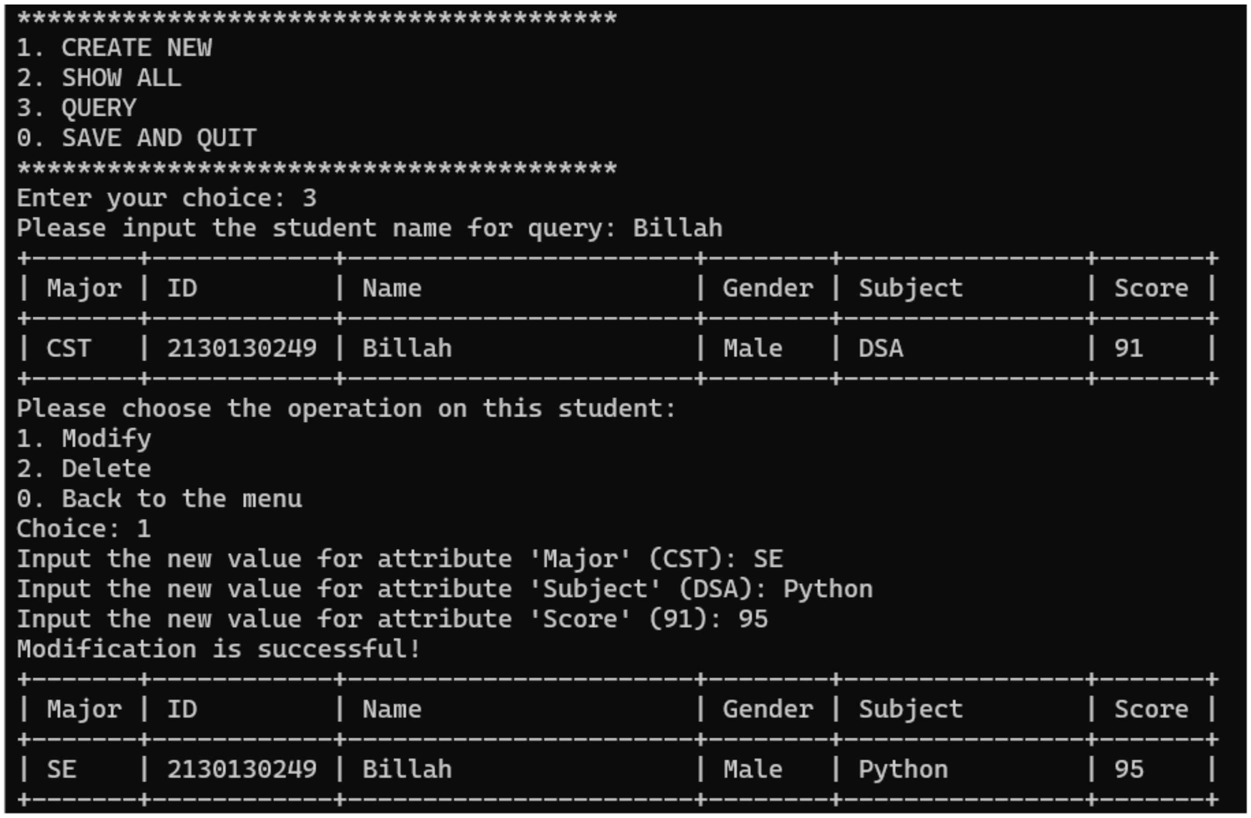
- Offers an efficient method of updating information in students' records to reduce the chances of making an error.
- Makes sure that the user can get feedback of the changes made as soon as she or he made it.

Execution Flow

1. The function is called with a common student record that is to be updated.
2. The user is prompted to input new values for the `Major`, `Subject`, and `Score` attributes.

3. In case of valid inputs these attributes are set.
4. Details of the modified record are provided in the next section of the program.

Example Outputs



```
****************************************
1. CREATE NEW
2. SHOW ALL
3. QUERY
0. SAVE AND QUIT
****************************************
Enter your choice: 3
Please input the student name for query: Billah
+-------+------------+-----------------------+--------+----------------+-------+
| Major | ID         | Name                  | Gender | Subject        | Score |
+-------+------------+-----------------------+--------+----------------+-------+
| CST   | 2130130249 | Billah                | Male   | DSA            | 91    |
+-------+------------+-----------------------+--------+----------------+-------+
Please choose the operation on this student:
1. Modify
2. Delete
0. Back to the menu
Choice: 1
Input the new value for attribute 'Major' (CST): SE
Input the new value for attribute 'Subject' (DSA): Python
Input the new value for attribute 'Score' (91): 95
Modification is successful!
+-------+------------+-----------------------+--------+----------------+-------+
| Major | ID         | Name                  | Gender | Subject        | Score |
+-------+------------+-----------------------+--------+----------------+-------+
| SE    | 2130130249 | Billah                | Male   | Python         | 95    |
+-------+------------+-----------------------+--------+----------------+-------+
```

6. Delete Records

The **Delete Records** module enables a user to permanently delete a record of a particular student and other records are displayed. Here's a detailed breakdown:

```
# Function to delete a student
def delete(student):
    data.remove(student)
    print("Student record deleted successfully.")
    # Display all remaining students in table format
    if data:
        print("+-------+------------+-----------------------+--------+----------------+-------+")
        print("| Major | ID         | Name                  | Gender | Subject        | Score |")
        print("+-------+------------+-----------------------+--------+----------------+-------+")
        for student in data:
            print(f"| {student['Major']:<5} | {student['ID']:<10} | {student['Name']:<21} |
{student['Gender']:<6} | {student['Subject']:<14} | {student['Score']:<5} |")
        print("+-------+------------+-----------------------+--------+----------------+-------+")
    else:
        print("No students remain in the database.")
```

Step-by-Step Explanation

1. **Function Definition (`def delete(student):`)**:
   - This function implements the actual deletion of a record as well as update of the database.
2. **Removing the Student Record (`data.remove(student)`)**:
   - Removes the specified `student` dictionary from the `data` list, which serves as the in-memory database.
3. **Confirmation Message**:
   - After the record is successfully removed, the function prints `"Student record deleted successfully."`.
4. **Display Remaining Records**:
   - If the `data` list is not empty, the function displays all remaining student records in a tabular format.
   - Each row in the table includes **Major**, **ID**, **Name**, **Gender**, **Subject**, and **Score**.
5. **Empty Database Check (`if data:`)**:
   - If the `data` list is empty after deletion, the function prints `"No students remain in the database."` to notify the user.

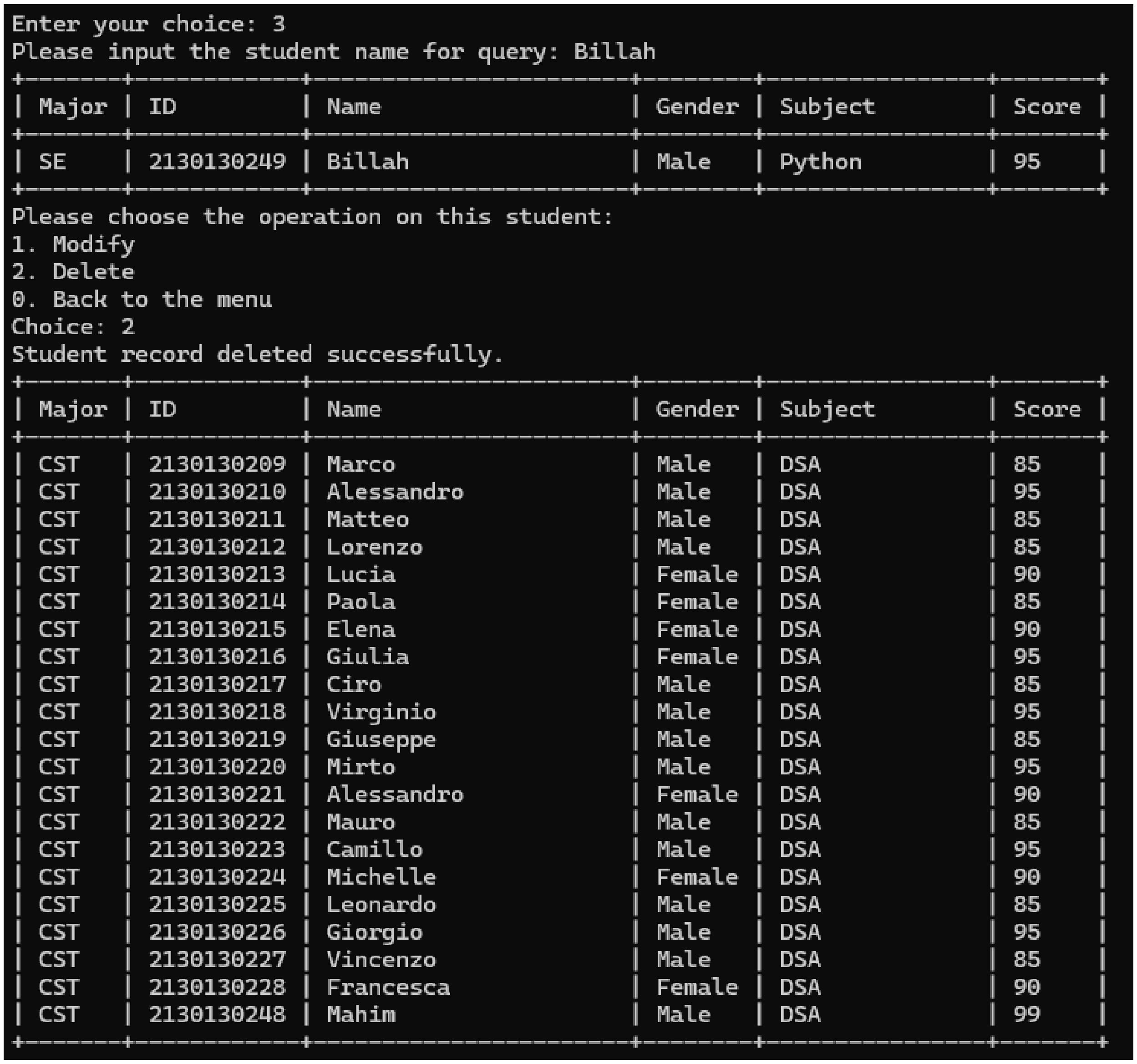
Purpose of the Function

- It also affords an easy and efficient means for purging out inaccurate or illegitimate records.
- Makes sure that a user is able to see the updated database list right after they have deleted a record.

Execution Flow

1. The function takes the `student` record to be deleted as parameter.
2. It means that the record is deleted from the `data` list available.
3. A confirmation message is received.
4. The rest of records are shown in about a formatted table.
5. No records message: In case no records are left in the customer account, the program shows a message is displayed indicating an empty database.

Example Outputs



```
Enter your choice: 3
Please input the student name for query: Billah
+-------+------------+-----------------------+--------+----------------+-------+
| Major | ID         | Name                  | Gender | Subject        | Score |
+-------+------------+-----------------------+--------+----------------+-------+
| SE    | 2130130249 | Billah                | Male   | Python         | 95    |
+-------+------------+-----------------------+--------+----------------+-------+
Please choose the operation on this student:
1. Modify
2. Delete
0. Back to the menu
Choice: 2
Student record deleted successfully.
+-------+------------+-----------------------+--------+----------------+-------+
| Major | ID         | Name                  | Gender | Subject        | Score |
+-------+------------+-----------------------+--------+----------------+-------+
| CST   | 2130130209 | Marco                 | Male   | DSA            | 85    |
| CST   | 2130130210 | Alessandro            | Male   | DSA            | 95    |
| CST   | 2130130211 | Matteo                | Male   | DSA            | 85    |
| CST   | 2130130212 | Lorenzo               | Male   | DSA            | 85    |
| CST   | 2130130213 | Lucia                 | Female | DSA            | 90    |
| CST   | 2130130214 | Paola                 | Female | DSA            | 85    |
| CST   | 2130130215 | Elena                 | Female | DSA            | 90    |
| CST   | 2130130216 | Giulia                | Female | DSA            | 95    |
| CST   | 2130130217 | Ciro                  | Male   | DSA            | 85    |
| CST   | 2130130218 | Virginio              | Male   | DSA            | 95    |
| CST   | 2130130219 | Giuseppe              | Male   | DSA            | 85    |
| CST   | 2130130220 | Mirto                 | Male   | DSA            | 95    |
| CST   | 2130130221 | Alessandro            | Female | DSA            | 90    |
| CST   | 2130130222 | Mauro                 | Male   | DSA            | 85    |
| CST   | 2130130223 | Camillo               | Male   | DSA            | 95    |
| CST   | 2130130224 | Michelle              | Female | DSA            | 90    |
| CST   | 2130130225 | Leonardo              | Male   | DSA            | 85    |
| CST   | 2130130226 | Giorgio               | Male   | DSA            | 95    |
| CST   | 2130130227 | Vincenzo              | Male   | DSA            | 85    |
| CST   | 2130130228 | Francesca             | Female | DSA            | 90    |
| CST   | 2130130248 | Mahim                 | Male   | DSA            | 99    |
+-------+------------+-----------------------+--------+----------------+-------+
```

7. Save and Quit

The **Save and Quit** module is to ensure that all record of all the students gets saved into CSV before exiting the program. Here's a detailed breakdown of the code:

```python
# Function to save data to a CSV file
def save_and_quit():
    with open("student.csv", "w", newline="") as csvfile:
        fieldnames = ["Major", "ID", "Name", "Gender", "Subject", "Score"]
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(data)
    print("File successfully saved. Welcome next use!")
```

1. **Function Definition (`def save_and_quit():`)**:
    - This function contains the saving of structures all records to a CSV file and termination of the program logic.
2. **Opening the CSV File (`with open(...) as csvfile`)**:
    - The `open()` function opens a file named `student.csv` in write mode (`"w"`).
    - The `newline=""` argument prevents the addition of extra blank lines in the file.
    - The `with` statement ensures that the file is properly closed after writing, even if an error occurs.
3. **Field Names (`fieldnames`)**:
    - Defines the column headers for the CSV file: **Major**, **ID**, **Name**, **Gender**, **Subject**, and **Score**.
4. **CSV Writer (`csv.DictWriter`)**:
    - Initializes a `DictWriter` object, which allows writing dictionaries (each student record) to the CSV file.
5. **Writing the Header (`writer.writeheader()`)**:
    - Writes the column headers to the CSV file as the first row.
6. **Writing the Data (`writer.writerows(data)`)**:
    - Writes all student records (stored in the `data` list) to the CSV file.
7. **Confirmation Message**:
    - Prints `"File successfully saved. Welcome next use!"` to confirm that the data has been saved successfully.

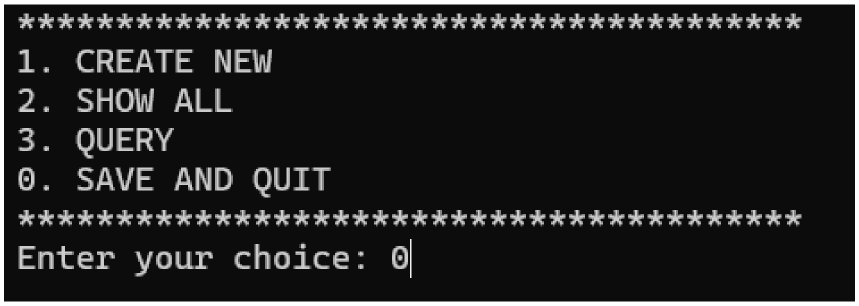
Purpose of the Function

- Makes sure all student records are saved in a CSV file for future purpose.
- Protects against accidental loss of data by storing the current state of the data list before the program's completion.

Execution Flow

1. When the user selects the **Save and Quit** option (menu option `0`), this function is called.
2. The function prints all of the records in the data list as the contents of the `student.csv` file.
3. Then the application sends a confirmation message to the user.
4. Depending on the function, the program stops running after the function call.

Output Example

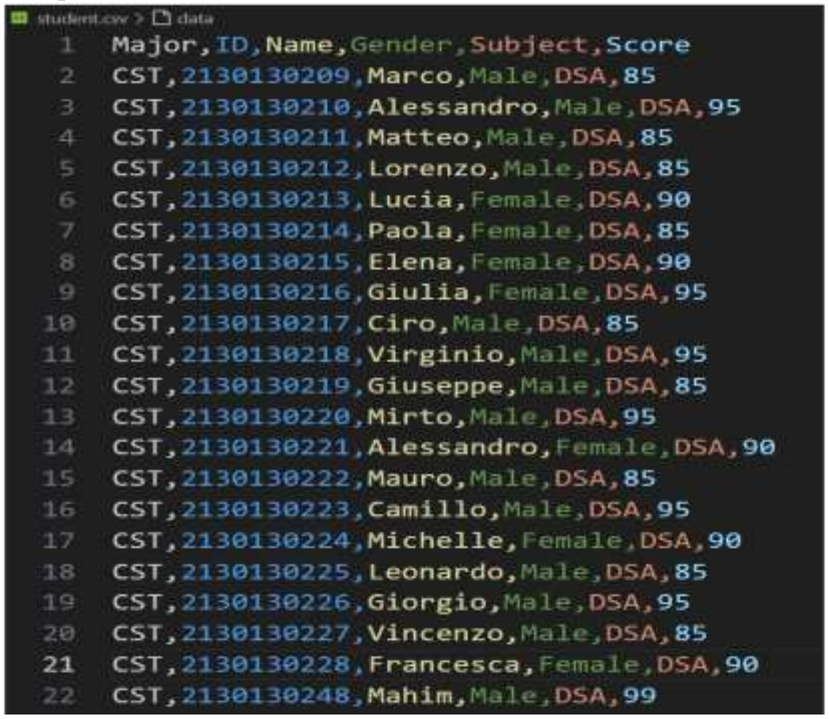


```
****************************************
1. CREATE NEW
2. SHOW ALL
3. QUERY
0. SAVE AND QUIT
****************************************
Enter your choice: 0
```

8. CSV File Creation

This file contains the student information imported from **student.csv** and is created using Python's **Save and Quit** module to organize and format all the students' data consistently. The example given down below depicts a well formatted CSV file with six fields, namely `Major`, `ID`, `Name`, `Gender`, `Subject`, and `Score`.

1. **Automated Creation**:
   - It is produced by the application **Save and Quit** function to produce the file for student evaluation.
   - People don't have to create or edit the file through Excel or some other tool by their own efforts.
2. **Structure**:
   - The file includes six columns:
     - **Major**: Represents the student's field of study (e.g., CST).
     - **ID**: A unique identifier assigned to each student (e.g., 2030130209).
     - **Name**: The full name of the student (e.g., Marco).
     - **Gender**: The student's gender (e.g., Male or Female).
     - **Subject**: The subject or course the student is enrolled in (e.g., Intro-Com-Sci).
     - **Score**: The student's score in the subject (e.g., 85).
3. **Data Entry**:
   - Data for each row corresponds to a student record that was created, modified, or updated during the program's execution.
4. **CSV Writer in Python**:
   - The program uses the `csv.DictWriter` module to write the data to the CSV file.
   - The `fieldnames` attribute ensures that the file headers match the structure of the data.
5. **Content Example**:
   - Example rows from `student.csv`:



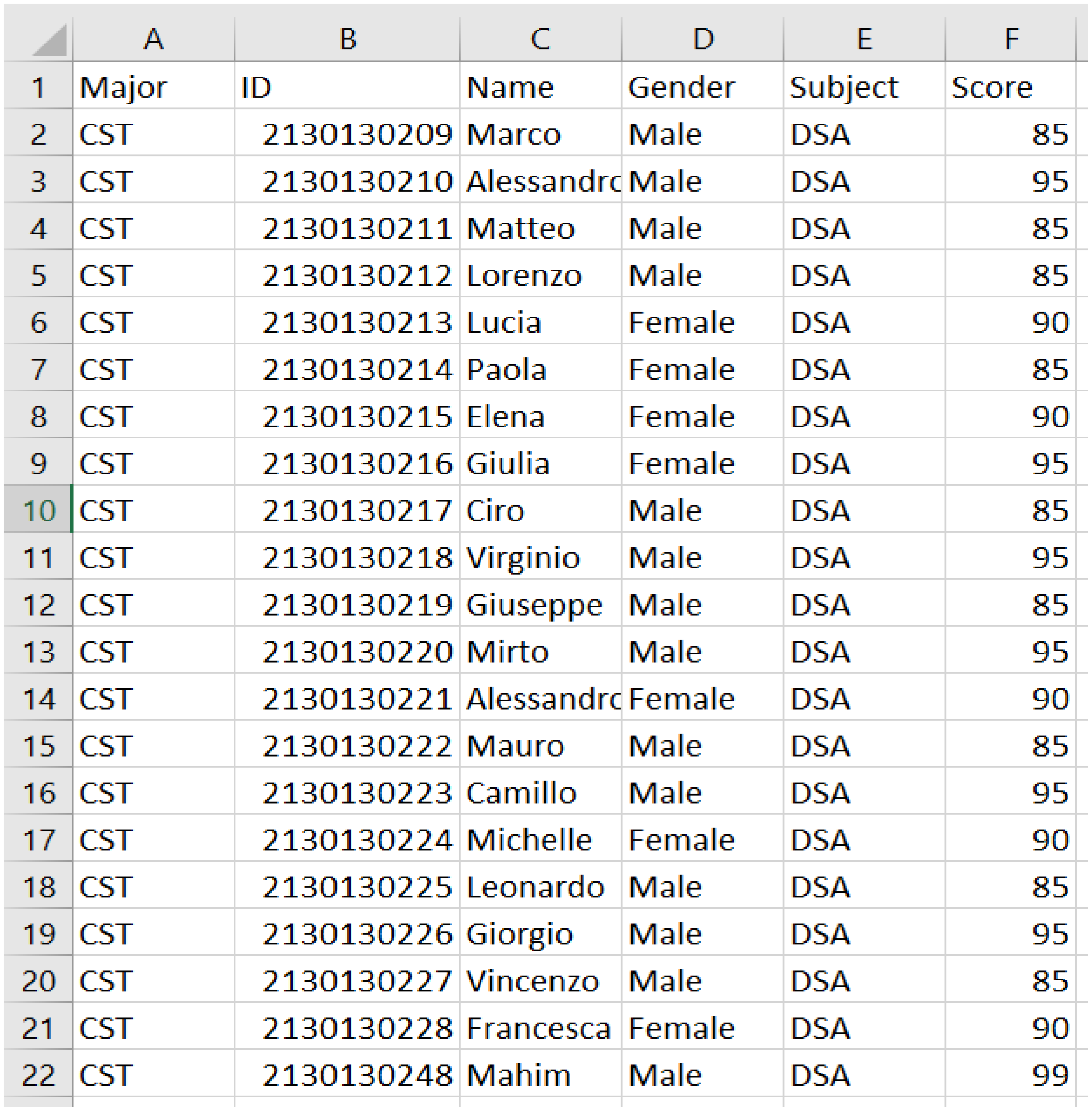
```
Major,ID,Name,Gender,Subject,Score
CST,2130130209,Marco,Male,DSA,85
CST,2130130210,Alessandro,Male,DSA,95
CST,2130130211,Matteo,Male,DSA,85
CST,2130130212,Lorenzo,Male,DSA,85
CST,2130130213,Lucia,Female,DSA,90
CST,2130130214,Paola,Female,DSA,85
CST,2130130215,Elena,Female,DSA,90
CST,2130130216,Giulia,Female,DSA,95
CST,2130130217,Ciro,Male,DSA,85
CST,2130130218,Virginio,Male,DSA,95
CST,2130130219,Giuseppe,Male,DSA,85
CST,2130130220,Mirto,Male,DSA,95
CST,2130130221,Alessandro,Female,DSA,90
CST,2130130222,Mauro,Male,DSA,85
CST,2130130223,Camillo,Male,DSA,95
CST,2130130224,Michelle,Female,DSA,90
CST,2130130225,Leonardo,Male,DSA,85
CST,2130130226,Giorgio,Male,DSA,95
CST,2130130227,Vincenzo,Male,DSA,85
CST,2130130228,Francesca,Female,DSA,90
CST,2130130248,Mahim,Male,DSA,99
```

Advantages of Python-Generated CSV

- **Automation**:
  - Eliminates labor-intensiveness and improve accuracy when it comes to formulation and management of student records.
- **Consistency**:
  - It enhances the formats and structures regardless the number of records.
- **Portability**:
  - The CSV file can be easily sent to any other individual, it can be opened directly in Excel or other MS office tools if required.

Use Case

- It also stores the student data in the form of a file for data input exercise performed throughout the course of the program's session, the file is helpful as a database of records.
- It is second and can be replenished by program when run again, in other words, memory it is used as back up and can be refilled once program is reopened.



	A	B	C	D	E	F
1	Major	ID	Name	Gender	Subject	Score
2	CST	2130130209	Marco	Male	DSA	85
3	CST	2130130210	Alessandrc	Male	DSA	95
4	CST	2130130211	Matteo	Male	DSA	85
5	CST	2130130212	Lorenzo	Male	DSA	85
6	CST	2130130213	Lucia	Female	DSA	90
7	CST	2130130214	Paola	Female	DSA	85
8	CST	2130130215	Elena	Female	DSA	90
9	CST	2130130216	Giulia	Female	DSA	95
10	CST	2130130217	Ciro	Male	DSA	85
11	CST	2130130218	Virginio	Male	DSA	95
12	CST	2130130219	Giuseppe	Male	DSA	85
13	CST	2130130220	Mirto	Male	DSA	95
14	CST	2130130221	Alessandrc	Female	DSA	90
15	CST	2130130222	Mauro	Male	DSA	85
16	CST	2130130223	Camillo	Male	DSA	95
17	CST	2130130224	Michelle	Female	DSA	90
18	CST	2130130225	Leonardo	Male	DSA	85
19	CST	2130130226	Giorgio	Male	DSA	95
20	CST	2130130227	Vincenzo	Male	DSA	85
21	CST	2130130228	Francesca	Female	DSA	90
22	CST	2130130248	Mahim	Male	DSA	99

What Have You Obtained Through This Experience?

1. Programming Skills

This project has helped me to improve greatly my programming skills especially in the language Python. I also got to practice in handling complicated data structures and collections including dictionaries and lists that are useful in data arrangement. Also, there was an opportunity to work with files, especially to work with csv files using the CSV module to read and write data. From these tasks, I learned to write clean code, modular code, and reusable code.

2. Algorithm Design and Analysis

One of the most important aspects of the project was the possibility of sorting the results by the ID and Score. It made it possible to focus on the topic of algorithm design. Thus, by implementing sorting algorithms I was able to compare and contrast the time and performance of different approaches. This is because such considerations form the backbone of performance enhancement in software and especially in those that deal with large data sets.

3. Problem-Solving and Debugging

In the course of the development, I faced some problems which demanded innovation to come up with the best solutions. For example, tasks such as generating and formatting the table output and preserving data used in a program session across its future runs were not easy to achieve. Solving these issues enhanced my debugging abilities and helped me to learn how to solve problems logically. This incident emphasized the need for testing and the role which testing plays in the development of software.

4. Data Management Best Practices

The project gave a chance to develop an effective system for data management. I got to understand how to arrange data in the best way possible and also to maintain its stability during insertion, modification and removal. Additionally, file management and error checking and prevention when working with persistent storage became evident to avoid losing or corrupting data.

5. User Interface and Experience Design

Despite the fact that the project was a command line application it stressed the importance of interfaces for users. Clear instructions, proper handling of invalid input and proper presentation of data in tabulated format were some of the considerations. All these make for a good user experience and can as well be used in other complex GUIs.

6. Time Management and Project Planning

Finishing the project by a certain time was a good experience in terms of time organization. Thus, dividing the whole project into tasks and ranking them in terms of their complexity as well as their interconnections made it possible to advance steadily. This therefore can be used in large projects and group assignments in order to achieve the set goals within the specified time.

7. Real-World Application Development

This system allowed the class to see what true software development is like in the real world. I acquired knowledge on developing a useful item that can be employed in academic or organizational environments. This approach helps to close the gap between theoretical tasks and real-life software engineering, providing me with a set of practicable tools for industry tasks.

8. Continuous Learning and Adaptability

From this project, it was clear that the process of software development is never stopping and that changes need to be made continuously. When learning about Python libraries, fixing problems that appeared out of the blue, or improving the code, I always had to be ready to learn how to do it better. This way of thinking is important if one is to remain creditable in the ever-advancing world of technology.

9. Collaboration and Communication

Although this was an individual task, the documentation procedure reflected some aspects of a group work. Some of the key communication skills that are useful in team environments include commenting the code, ensuring that code is arranged in a logical manner and submitting a very detailed report. Such practices make it easy for other people to continue with the work without having to unravel what has been done.

Conclusion

The Student Transcript Management System can be used for practical purposes to control student records. For this project, I was not only able to refine my technical knowledge but also to gain some experience in software design and development. It can also be noted that the practical experience gained in the process of this project will be useful in the future.

Also, I learned the value of consistency in code commenting and following the rules of software development. In order to achieve good reliability and high user-friendliness of the application, I used modular functions, clear user prompts, and proper error handling. This has also added to the understanding that planning and successive enhancements are critical to the success of a project.

The project also featured the issues of practical relevance as one of the project outcomes. In the course of designing the system I was able to picture how similar tools could be used in academic institutions to improve the management of students' records. This practical approach helped me realize that software is a way of solving problems, which made me look at the theoretical aspect of the software.

Also, the problems faced and solved during the development process has made me even more confident especially as a programmer. I also learnt that I am capable of overcoming technical difficulties and meeting set deadlines when it comes to problem solving. These attributes will surely be useful in future work and studies.

In general, the experience gained during this project has been immense, from the technical aspects to other skills including, time management, critical thinking, and interpersonal skills. The process presented in this paper will remain valuable for future research and practice and will help me to improve my contribution to the field of software engineering.