METHODIST COLLEGE | KUALA LUMPUR

Veritas Vincit Omnia

American Degree Transfer Program

Course: Programming in Python with Lab

Course Code: CSC1620

Assignment Number & Title: Project 1 – Student Grading System

Name of Lecturer: Mr. Civanesar

Name(s): Khaw Xiao Hui **Student ID Number(s):** 0030570

I/We declare that this is my/our own original work and any contributions made by others have been properly acknowledged and/or referenced.

Signature:

Date of Submission:

Table of Contents

Introduction

Overview of the project and its purpose.

The Student Grading System is a program designed that allows teachers to manage student records and calculate grades. The system allows users to add, view, search, update, delete, calculate and display grades for students based on their marks in multiple subjects. It also includes features such as validation, saving and loading records from a file. This project aims for efficient record management and grade calculation.

The primary goals of the project are:

- To create a user-friendly system for managing student records (add, view, search, update, delete) efficiently.
- To calculate the average marks for each student and assign grade based on the grading scale.
- To implement a command-line system using lists to store multiple student records and dictionaries to store individual student details.
- To save and load student records from a file, allowing records to be stored and retrieved across program execution.
- To ensure that user inputs are valid to avoid errors and ensure that the system runs smoothly.
- To practice and demonstrate proficiency in fundamental programming concepts.

Explanation of the fundamental programming concepts applied in the project.

The project applies several key programming concepts, including:

- **Control Structures:** Conditional statements and loops are used to handle user interactions, program flow and decision-making such as user can decide whether to search by ID or name and repeatedly ask for input until valid data is entered.
- **Functions:** Used to improve code reusability, readability and maintainability such as the operations for adding, viewing, updating, searching, deleting and calculating grades records are organized into separate functions.
- **Data Structures:** Dictionaries are used to store individual student information (such as name, ID and marks), while lists store multiple student records, with each record represented as a dictionary. This organized structure allows for easy access and manipulation of student data.

- **Variables and Data Types:** Used to store student data such as name, ID and marks.
- **File handling:** JSON is used as a format to store student records and its' functions are used to save and load data. This ensures that the records remain even after the program is closed.

System Design

UML Use Case Diagram

The following UML use case diagram illustrates the interactions between the user and the system:

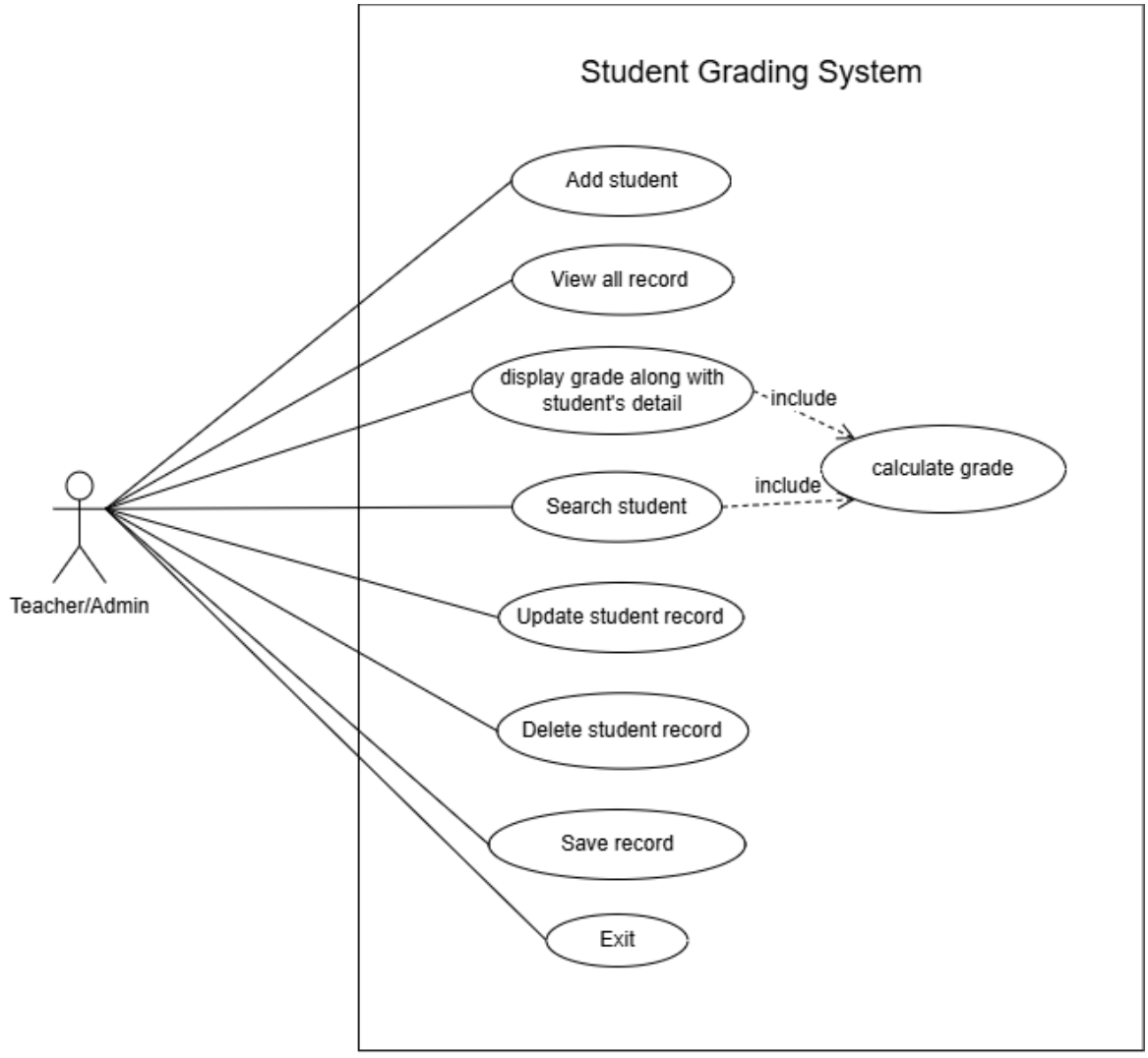

Figure 1: Use case diagram

Association relationship

The actor (teacher or admin) interacts with all the use cases, which means the admin can perform any operation related to the student records.

Include relationship

The "calculate grade" use case includes "display grade along with student's detail" and "search student" use case. Both actions happen every time the grades are calculated.

UML Activity Diagram

The activity diagram below outlines the process of running the program:

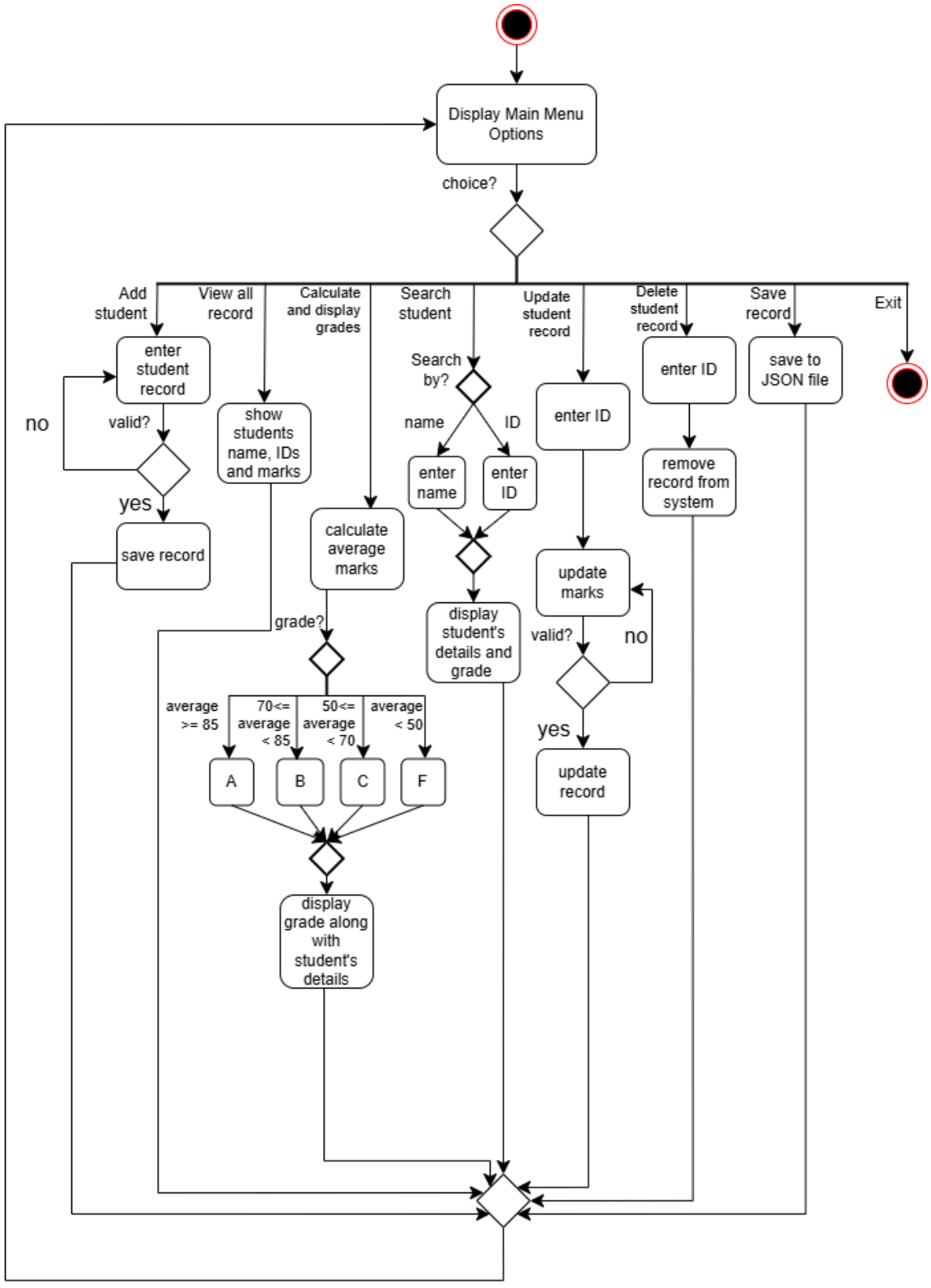

Figure 2: Activity diagram

1. **Start of the program**
   The diagram begins with the initial node (solid black circle), representing the starting point of the system.

2. **Display Main Menu Options**
   The user is given choices on what they want to do.
3. **Decision: Choices?**
   The diamond-shaped decision node determines the next step based on the user's input
   a. **Add Student**
      i. The user enters student details (name, ID, marks for multiple subjects).
      ii. The system checks if the data is valid (decision node).
      iii. If valid, the system saves the record. If not valid, it rejects the entry.
      iv. Returns to the main menu options.
   b. **View All Records**
      i. The system displays the list of all students with their name, IDs and marks.
      ii. Returns to the main menu options.
   c. **Calculate and Display Grades**
      i. The system calculates the average marks for each student.
      ii. Based on the average, it assigns grades:
         a. A: average >= 85
         b. B: 70 <= average < 85
         c. C: 50 <= average < 70
         d. F: average < 50
      iii. The system displays the grade along with student's details.
      iv. Returns to the main menu options.
   d. **Search Student**
      i. The user can search by name or ID.
      ii. The system retrieves and displays the students' details and grade.
      iii. Returns to the main menu options.
   e. **Update Student Record**
      i. The user enters a student's ID.
      ii. The user updates the student's marks.
      iii. The system checks if the data is valid (decision node).
      iv. If valid, the system updates the record. If not valid, it rejects the update.
      v. Returns to the main menu options.
   f. **Delete Student Record**
      i. The user enters a student's ID.
      ii. The system removes the record if it exists.
      iii. Returns to the main menu options.
   g. **Save record**
      i. The system saves all records into a JSON file.
      ii. Returns to the main menu options.
   h. **Exit**
      i. The user can choose to exit at any time.
      ii. The final node represents the successful termination of the game.

Implementation Details

Feature Implementation

Save Student Record

- The `save_record()` function saves the list of student records into a file named `student_record.json` using `json.dump()` to coverts student list into a JSON formatted string.
- The with statement ensures that the file is properly opened and closed after saving the data.
- Displays a confirmation message when the data is saved successfully.

Load Student Record

- The `load_record()` function loads the student record from the `student_record.json` using `json.load()`.
- If the file is not found (`FileNotFoundError`), an error message is displayed, and an empty students list is initialized.
- If the file exists, the student records are loaded into memory for future operations.

Add Student Record

- The system uses a series of while loops to prompt the user for valid inputs, ensuring the name and ID are not empty, and a valid number of subjects is provided.
- Marks for each subject are entered and validated to be integers between 0 and 100.
- After collecting all data, the student's record is created and appended to the students list.
- A success message is displayed after the student is added.

View Student Records

- If no student records are available, a message stating "No student record found" is displayed.
- If records are available, each student's details (name, ID, and marks) are printed out one by one.

Calculate and Display Grades

- The `calculate_grades()` function calculates the average marks of each student, assigns a grade based on the grading scale, and displays the student's details along with their grade.
- For each student, the marks are summed and divided by the number of subjects to calculate the average.
- The `calculate_grade()` function is called to determine the student's grade based on the average.
- The system prints the student's name, ID, average marks and grade.

Search Student Record

- The system prompts the user to choose between searching by name or ID.
- If searching by ID, the function matches the ID and displays the student's details (name, ID, marks, and grade).
- If searching by name, the function matches the name and displays the student's details.
- If the student is not found, a message is displayed indicating that no student was found with the given criteria.

Update Student Record

- The user enters a student ID and the system search for the student.
- If found, the system prompts the user to enter new marks for each subject.
- The new marks are validated and updated in the student's record.
- A success message is displayed after the marks are updated.

Delete Student Record

- The `delete_record()` function allows the user to delete a student record by their ID.
- The user enters the student ID and the system search for the student.
- If the student if found in the system, their record is removed from the students list.
- A success message is displayed confirming the deletion.
- If the student is not found, an error message is shown.

Main menu

- It displays a meu with options such as adding a student, viewing records, calculating grades, and more.
- Depending on the user's choice, it runs the corresponding function for each action.
- If the user selects option 8 (Exit), the program ends, and the student records are saved by calling the `save_record()` function.

Use of Data Structures and Functions

Dictionaries

Each student's details are stored in a dictionary. Each dictionary contains three key-value pairs: 'Name', 'ID', and 'Marks'. For example:

```
{
  "Name": "Kristine Hartman",
  "ID": "ABCD1234",
  "Marks": [80, 93, 72]
}
```

Lists

All student dictionaries are stored in a list called students. This list allows easy manipulation and random access to student's details. For example:

```
students = []
student_record = {
    "Name": name,
    "ID": id,
    "Marks": marks
}
students.append(student_record)
```

JSON data interchange format

JSON is used for saving data permanently as it allows the lists of dictionaries to be stored as a file and reloaded since lists and dictionaries are only temporary in memory. For example:

```
[
  {
    "Name": "Kristine Hartman",
    "ID": "ABCD1234",
```

```
    "Marks": [80, 93, 72]
  },
  {
    "Name": "Rex Rollins",
    "ID": "1T59N7LO",
    "Marks": [66, 100, 75]
  },
  {"Name": "Rosella Hutchinson",
    "ID": "KUNO1237",
    "Marks": [90, 89]},
  {"Name": "Keith Carroll",
    "ID": "keith123",
    "Marks": [99]},
  {"Name": "Shane A. Arrington",
    "ID": "shaneA88",
    "Marks": [30, 90, 70, 88]
  }
]
```


Functions and its role

- save_record(): Saves the list of student records into a file.
- load_record(): Loads the student record from a file.
- add_student(): Allows the user to add a new student's details (name, ID and marks for subjects) to the system.
- view_record(): Display the list of all student along with their details.
- calculate_grade(average): Calculate the grade based on average marks.
- calculate_grades(): Calculates the average marks of each student, call calculate_grade(average) then displays the student's details along with their grade.
- search_student(): Allows the user to search for a student by either their name of ID.
- update_record():Allows the user to update a student's marks for each subject.
- delete_record(): Allows the user to delete a student record by their ID.
- main(): Call load_record() at the start to load any existing data.

Challenges and Solutions

Challenges Faced

Handling JSON file Errors

If JSON file does not exist or was corrupted, the program would crash when trying to load records.

Input validation

User could enter empty names and ID, and invalid or non-numeric values for marks and subjects. This could lead to errors in processing student records.

Search function sensitivity

User could enter misspelled names. For instance, when user inputs “ali” instead of “Ali”, student was not found as the search function was case-sensitive, meaning searching for “ali” would not return “Ali”, leading to confusion.

Solutions Implemented

JSON file handling

Used try-except to handle missing files and displayed no record found when no previous records were found.

```
try:
    with open("students_record.json", "r") as file:
        students = json.load(file)
    print("Student record loaded from student_record.json")
except FileNotFoundError:
    print("No previous records found")
```

Input handling

Used `.strip()` to remove leading or trailing spaces and checked if the input was empty for name and ID. If so, prompted the user to re-enter a valid value. For marks and subjects, use `.isdigit()` to ensure numeric inputs before converting them to integers.

```
while True:
    id = (input("ID: ")).strip()
    if id == "":
        print("invalid ID")
    else:
        break
```

```
while True:
    mark = input("Mark: ")
    if mark.isdigit():
        mark = int(mark)
        if 0 <= mark <= 100:
            marks.append(mark)
            break
        else:
            print("Invalid mark")
    else:
        print("Invalid input")
```

Search function handling

Converted both input and stored names to lowercase by `using .lower()` for case-sensitive searching and added a flag (found = False) to indicate if the student was found, then prompted the user if student was not found.

```
found = False
if search_by == "id":
    search_id = input("Enter student ID: ").strip()
    for student in students:
        if search_id == student["ID"]:
            marks = student["Marks"]
            average = sum(marks) / len(marks)
            grade = calculate_grade(average)
            print(f"Name: {student['Name']}, ID:
{student['ID']}, Marks: {marks}, Grade: {grade}")
            found = True
            break

elif search_by == "name":
    search_name = input("Enter student name:
").strip().lower()
    for student in students:
        if search_name == student["Name"].lower():
            marks = student["Marks"]
            average = sum(marks) / len(marks)
            grade = calculate_grade(average)
            print(f"Name: {student['Name']}, ID:
{student['ID']}, Marks: {marks}, Grade: {grade}")
            found = True
            break

if not found:
    print("student not found")
```

Conclusion

The “Student Grading System” project successfully created a user-friendly and efficient platform for managing student records and grades. This system provides essential functionalities such as adding, viewing, searching, updating, deleting student records, and calculating along with displaying grades based on the marks entered for each student. Through this project, we reinforced fundamental programming concepts such as control structures, functions, data structures, and file handling.

Key Takeaways:

- Using lists and dictionaries helps manage student records easily and efficiently.
- Dividing the program into smaller functions makes the code cleaner, reusable and easier to maintain.
- Addressing potential issues like case-sensitive searches and file errors ensures the system to work smoothly.
- Saving student records in a JSON file ensures that data is stored and can be accessed later.

Challenges faced during development, such as handling JSON file errors, input validation and case-sensitive search issues were effectively addressed through careful error handling and some input techniques. In conclusion, this project not only serves as a functional tool for managing student data but also provides a solid foundation for further enhancements. Additional features such as advanced searching, reporting, or even a graphical user interface (GUI), could be incorporated in future versions to improve usability and expand the system’s capabilities. This project demonstrates a practical application of core programming concepts and provides a solid base for anyone interested in developing similar data-driven applications.

Appendix

Student record

```
[
  {
    "Name": "Kristine Hartman",
    "ID": "ABCD1234",
    "Marks": [80, 93, 72]
  },
  {
    "Name": "Rex Rollins",
    "ID": "1T59N7LO",
    "Marks": [66, 100, 75]
  },
  {"Name": "Rosella Hutchinson",
    "ID": "KUNO1237",
    "Marks": [90, 89]},
  {"Name": "Keith Carroll",
    "ID": "keith123",
    "Marks": [99]},
  {"Name": "Shane A. Arrington",
    "ID": "shaneA88",
    "Marks": [30, 90, 70, 88]
  }
]
```

Solution

```
import json

# Initialize an empty list to store student records
students = []

def save_record():
    with open("students_record.json", "w") as file:
        # Convert the student list to JSON and save it to a file
        json.dump(students, file)
    print("Student record saved to student_record.json")

def load_record():
    # Access the global student list
    global students
    try:
        with open("students_record.json", "r") as file:
            # load the student records from the file
            students = json.load(file)
        print("Student record loaded from student_record.json")
    except FileNotFoundError:
        print("No previous records found")
def add_student():
    while True:
        # .strip() removes leading and trailing whitespaces
        name = input("Name: ").strip()
        # Ensure name is not empty
        if name == "":
            print("invalid name")
        else:
            break
```

```
    while True:
        id = (input("ID: ")).strip()
        # Ensure id is not empty
        if id == "":
            print("invalid ID")
        else:
            break
    while True:
        subjects = input("How many subjects: ")
        # Ensure number of subjects is valid
        if subjects.isdigit():
            subjects = int(subjects)
            # Ensure number of subjects is positive
            if subjects > 0:
                break
        else:
            print("Invalid input")

    # List to store marks for each subject
    marks = []

    for num in range(subjects):
        while True:
            mark = input("Mark: ")
            # Ensure marks entered are valid
            if mark.isdigit():
                mark = int(mark)
                if 0 <= mark <= 100:
                    marks.append(mark)
                    break
                else:
                    print("Invalid mark")
            else:
                print("Invalid input")

    # Create a dictionary to store the student information
    student_record = {
        "Name": name,
        "ID": id,
        "Marks": marks
    }
    # Add the student record to the list
    students.append(student_record)
    print(f"Student {name} added")

def view_record():
    if not students:
        print("No student record found")
    else:
        for student in students:
            print(f"Name: {student['Name']}, ID: {student['ID']},
Marks: {student['Marks']}")

def calculate_grade(average):
    if average >= 85:
        return "A"
    elif 70 <= average < 85:
        return "B"
    elif 50 <= average < 70:
```

```
        return "C"
    else:
        return "F"
def calculate_grades():
    for student in students:
        name = student["Name"]
        id = student["ID"]
        marks = student["Marks"]
        average = sum(marks) / len(marks)

        # Determine the grade based on average
        grade = calculate_grade(average)

        print(f"Name: {name}, ID: {id}, Average: {average:.2f},
Grade: {grade}")

def search_student():
    while True:
        search_by = input("Search by id or name:
").strip().lower()
        if search_by not in ["id", "name"]:
            print("invalid input")
            continue
        break

    # Flag to check if student was found
    found = False
    if search_by == "id":
        search_id = input("Enter student ID: ").strip()
        for student in students:
            if search_id == student["ID"]:
                marks = student["Marks"]
                average = sum(marks) / len(marks)
                grade = calculate_grade(average)
                print(f"Name: {student['Name']}, ID:
{student['ID']}, Marks: {marks}, Grade: {grade}")
                found = True
                break

    elif search_by == "name":
        search_name = input("Enter student name:
").strip().lower()
        for student in students:
            # Case insensitive search
            if search_name == student["Name"].lower():
                marks = student["Marks"]
                average = sum(marks) / len(marks)
                grade = calculate_grade(average)
                print(f"Name: {student['Name']}, ID:
{student['ID']}, Marks: {marks}, Grade: {grade}")
                found = True
                break

    if not found:
        print("student not found")

def update_record():
    student_id = input("Enter student ID: ").strip()
    for student in students:
```

```
        if student_id == student["ID"]:
            # List to store updated marks
            new_marks = []
            for num in range(len(student["Marks"])):
                while True:
                    new_mark = input("Mark: ")
                    if new_mark.isdigit():
                        new_mark = int(new_mark)
                        # Ensure valid mark
                        if 0 <= new_mark <= 100:
                            new_marks.append(new_mark)
                            break
                        else:
                            print("Invalid mark")
                    else:
                        print("Invalid input")
            # Update the student's marks
            student["Marks"] = new_marks
            print(f"updated marks for
{student['Name']}({student['ID']}) to {student['Marks']}")
            return
    print("Student ID not found")

def delete_record():
    student_id = input("Enter student ID: ").strip()
    for student in students:
        if student_id == student["ID"]:
            # Remove the student from the list
            students.remove(student)
            print(f"deleted {student['Name']}({student['ID']})
record")
            return
    print("Student ID not found")
def main():
    # Load existing records if available
    load_record()

    while True:
        # Display the menu options
        # "\n" to enhance visibility
        print("\n1. Add student")
        print("2. View all record")
        print("3. Calculate and display grades")
        print("4. Search student")
        print("5. Update student record")
        print("6. Delete student record")
        print("7. Save record")
        print("8. Exit")

        choice = input("Enter your choice (e.g: 1): ")
        if choice == "1":
            add_student()
        elif choice == "2":
            view_record()
        elif choice == "3":
            calculate_grades()
        elif choice == "4":
            search_student()
        elif choice == "5":
```

```
            update_record()
        elif choice == "6":
            delete_record()
        elif choice == "7":
            save_record()
        elif choice == "8":
            print("Exiting program")
            break
        else:
            print("invalid choice")


main()
```