Project Documentation

Student Performance Analyzer (Console-Based)

Project Overview

This project is a **console-based Student Performance Analyzer** developed in Python. The purpose of this system is to help instructors **manage student academic records**, **calculate final results**, and **generate performance reports** in an organized and automated way.

The system works using **Object-Oriented Programming (OOP)** concepts and stores all student data in a **CSV file**, ensuring data persistence between program runs.

Project Objectives

The main objectives of this project are:

- To manage student records efficiently.
- To convert numerical scores into letter grades.
- To generate useful summary statistics for the class.
- To save and load student data using a CSV file.

Core Features of the System

The system allows the instructor to:

- Add new student records
- Remove existing student records
- Search for a student using their ID
- Update student information and marks Each student record contains:

1. Student ID
2. Student Name
3. Midterm Exam Score
4. Final Exam Score
5. Assignment Score

Final Score Calculation

The final score of each student is calculated using the following formula:

final_score = (0.4 × midterm) + (0.5 × final_exam) + (0.1 × assignment_score)

This formula gives more weight to the **final exam**, followed by the **midterm**, and then the **assignment**.

Letter Grade Assignment

After calculating the final numerical score, the system converts it into a **letter grade** using conditional statements.

Example grading logic:

- A → Excellent performance
- B → Good performance
- C → Average performance
- D → Below average
- F → Fail

(This grading is implemented using `if-elif-else` conditions.)

Summary Statistics Generation

The system generates useful class-level statistics, including:

- **Highest final score** in the class
- **Lowest final score** in the class
- **Class average score**
- **Grade distribution**, showing the number of students in each letter grade These statistics help instructors quickly understand overall class performance.

CSV File Handling (Data Persistence)

- All student records are stored in a **CSV file**
- Data is **automatically loaded** when the program starts
- Any changes (add, update, delete) are **saved back to the CSV file** This ensures that student data is **not lost** when the program is closed.

Programming Concepts Used

This project uses the following Python concepts:

- Object-Oriented Programming (Classes & Objects)
- File Handling using CSV
- Conditional Statements
- Loops
- Functions and Modular Design
- Exception Handling (for safe input and file operations)

Conclusion

The Student Performance Analyzer is a **simple, modular, and efficient system** designed to manage academic data in a structured way.
It demonstrates strong use of **Python fundamentals**, **OOP principles**, and **real-world problem solving**, making it suitable for academic assignments and beginner-level software projects.

Imported Modules in the Code

```
import csv
```

The `csv` module is used to **work with CSV (Comma-Separated Values) files**.
It allows the program to **read student records from a CSV file** and **write updated records back to the file**.
This module helps in storing student data in a structured and readable format.

```
import os
```

The `os` module is used to **interact with the operating system**.
In this project, it is mainly used to **check whether the CSV file exists or not** before reading or writing data.
This helps prevent file-related errors and ensures safe file handling.

Purpose in the Project

- `csv` → Handles saving and loading student records
- `os` → Manages file existence and system-level checks These modules make the

program **reliable and persistent**.

```
import csv


import os
```

Constants & Setup

Variables Used

```
file_name = "students.csv"
```

This variable stores the **name of the CSV file** where all student records are saved. It is used to **read existing data** and **write updated data** to the same file.

`std_list = []`

This is a **global list** used to store all `Student` objects.
Each element in this list represents **one student record** during program execution.

```
file_name = "students.csv"
#Global list to store Student objects
std_list = []
```

Task 4 — Object-Oriented Programming (5 Marks)

Class Used in the Code

Class Name: `Student`

The `Student` class represents **one student record** in the system.
It stores student information such as ID, name, exam scores, final score, and grade.
This class is used to **create student objects** and perform operations related to a single student.

Functions (Methods) in the `Student` Class

`__init__(self, id, name, mid_term, final_term, assignment)`

This function is the **constructor** of the class.
It is used to **initialize** a student object with basic details and scores when the object is created. It assigns values to attributes like student ID, name, marks, final score, and grade.

`compute_final(self)`

This function is used to **calculate the final numerical score** of a student.
It applies a fixed formula using midterm, final exam, and assignment marks. The calculated result is stored in the `final_score` attribute.

`compute_grade(self)`

This function is used to **convert the final numerical score into a letter grade**.

It checks the `final_score` using conditions and assigns grades such as A, B, C, D, or F. The result is stored in the `grade` attribute.

`to_dict(self)`

This function is used to **convert the student object into a dictionary format**.
It prepares student data so it can be **easily saved to a CSV file**. Each key-value pair represents one field of the student record.

Summary

- **Class**: `Student` → Represents a single student

- **Functions**:
  - `__init__` → Create and initialize a student
  - `compute_final` → Calculate final score
  - `compute_grade` → Assign letter grade
  - `to_dict` → Prepare data for CSV storage

This class makes the program **modular, organized, and easy to manage**.

```python
class Student:
    # Initialize attributes as per Task 4 instructions.
    def __init__(self,id,name,mid_term,final_term,assignment):
        self.id = int(id)
        self.name = name
        self.mid_term = float(mid_term)
        self.final_term = float(final_term)
        self.assignment = float(assignment)
        self.final_score = 0.0
        self.grade = ""

    def compute_final(self):
        #Compute final score using formula:
        self.final_score = (self.mid_term * 0.3) + (self.final_term  * 0.4) + (self.assignment * 0.3)

    def compute_grade(self):
        #Determine Letter grade based on conditions:
        #A: 85-100, B: 70-84, C: 55-69, D: 40-54, F: <40
        if  85 <= self.final_score <= 100:
            self.grade = "A"
        elif 70 <= self.final_score < 85:
            self.grade = "B"
        elif 50 <= self.final_score < 70:
            self.grade = "C"
        elif 40 <= self.final_score < 55:
            self.grade = "D"
        else:
            self.grade = "F"
```

```python
    def to_dict(self):
        #Return dictionary representation for CSV storage.
        return {
            "id" : self.id,
            "name" : self.name,
            "mid_term" : self.mid_term,
            "final_term" : self.final_term,
            "assingment" : self.assignment,
            "final_score" : self.final_score,
            "grade" : self.grade
        }
```

Task 3 — File Handling (4 Marks)

Function: `load_data()`

This function is used to **load student records from the CSV file** into the program.
It first clears the existing student list to avoid duplicate data.
Then it reads each row from the CSV file and **creates a `Student` object** from that data.
For every student, it calculates the final score and grade before storing the object in the list.
If the file does not exist, the program safely starts with an empty student list.

```python
def load_data():

    global std_list
    std_list.clear()

    try:
        with open(file_name, mode = 'r') as file:
            reader = csv.DictReader(file)
            for row in reader:
                #create object from csv data
                student = Student(
                    id         =  row['id'] ,
                    name       =  row['name'] ,
                    mid_term   =  row['mid_term'] ,
                    final_term =  row["final_term"] ,
                    assignment =  row["assingment"]
                )
                student.compute_final()
                student.compute_grade()
                std_list.append(student)
        print(f"Data loaded successfully from {file_name}.")
    except FileNotFoundError:
        print("File not found. Starting with an empty student list.")
    except Exception as e:
        print(f"An error occurred while loading data: {e}")
```

Function: `save_data()`

This function is used to **save all student records into the CSV file**.
It opens the file in write mode and defines the column names for the data.
Each `Student` object in the list is converted into a dictionary and written as a row in the file. If any error occurs during saving, an error message is displayed to the user.

```
def save_data():
    try:
        with open(file_name,mode='w',newline='') as file:
            fieldnames = ["id","name","mid_term","final_term","assingment","final_score","grade"]
            writer = csv.DictWriter(file, fieldnames=fieldnames)
            for student in std_list:
                writer.writerow(student.to_dict())
        print("Data Saved Successfully")
    except Exception as e:
        print(f"An error occurred while saving data: {e}")
```

Task 2 — Functions for Core Operations (5 Marks)

Function: `add_std()`

This function is used to **add a new student record** to the system.
It takes student details as input and first checks that the student ID is unique.
After receiving valid marks, it creates a new `Student` object and calculates the final score and grade.
The new student is then added to the global student list.
If invalid input is entered, the function displays an error message.

```python
def add_std():
    print("/n --- Add New Student ---")
    try:
        s_id = int(input("Enter Student ID :"))
        # Check for unique ID
        for s in std_list:
            if s.id == s_id:
                print("Student ID already exists. Please use a unique ID.")
                return
        name = input("Enter Student Name :")
        mid_term = float(input("Enter Mid Term Score (0-30):"))
        final_term = float(input("Enter Final Term Score (0-50):"))
        assingment = float(input("Enter Assignment Score (0-20):"))

        # Calculate final score and grade
        new_std = Student(s_id, name , mid_term, final_term, assingment)
        new_std.compute_final() #Calculate final score
        new_std.compute_grade() #Determine grade.

        std_list.append(new_std)

        print(f"Student {name}  added successfully!")
    except ValueError:
        print("Invalid input. Please enter the correct id and score.")
```

Function: `remove_std()`

This function is used to **remove a student record** from the system using the student ID.
It searches the global student list for the given ID.
If the student is found, the record is removed, and a success message is displayed.
If the ID does not exist, the function notifies the user.
It also handles invalid input by showing an error message.

```python
def remove_std():
    print("\n --- Remove Student ---")
    try:
        s_id = int(input("Enter Student ID to remove: "))
        found = False
        for s in std_list:
            if s.id == s_id:
                std_list.remove(s)
                print(f"Student with ID {s_id} removed successfully.")
                found = True
                break
            if not found:
                print(f"Student with  ID  {s_id} not found.")
        pass
    except ValueError:
        print("Invalid input. Please enter a valid Student ID.")
```

📝 Function: `search_student()`

This function is used to **search for a student record** using the student ID.
It looks through the global student list and, if found, **displays all details** of the student including scores, final score, and grade.
If the student ID does not exist, it shows a "not found" message.
It also handles invalid input to prevent errors.

```
def search_student():
    print("\n --- Search Student ---")
    try:
        s_id = int(input("Enter Student ID to search: "))
        found = False
        for s in std_list:
            if s.id == s_id:
                print("/n Student Found:")
                print(f"ID : {s.id} | Name : {s.name}")
                print(f"Score of Midterm : {s.mid_term}")
                print(f"Score of Finalterm : {s.final_term}")
                print(f"Score of Assignment : {s.assignment}")
                print(f"Final Score : {s.final_score:.2f}")
                print(f"Grade : {s.grade}")
                found = True
                break
        if not found:
            print(f"Student with ID {s_id} not found.")
    except ValueError:
        print("Invalid input. Please enter a valid Student ID.")
```

Function: `update_student()`

This function is used to **update an existing student record**.
It first searches the global student list using the student ID.
If the student is found, the instructor can **update the name, midterm, final, and assignment scores**.
Pressing Enter keeps the previous value.
After updating, the function **recalculates the final score and grade**.
If the student ID is not found or input is invalid, appropriate messages are displayed.

```python
def update_student():
    print("/n --- Update Student ---")
    try:
        s_id =int(input("Enter Student ID to update:"))
        student = None
        for s in std_list:
            if s.id == s_id:
                student = s
                break

        if student :
            print(f"Updating record for {student.name}.Press Enter to keep value.")

            name = input(f"Enter new name for {student.name}. SO : /n")
            if name: student.name = name

            m = input(f"Enter new Midterm score for {student.name}. SO : /n")
            if m: student.mid_term = float(m)

            f = input(f"Enter new Finalterm score for {student.name}. SO : /n")
            if f: student.final_term = float(f)

            a = input(f"Enter new Assignment score for {student.name}. SO : /n")
            if a: student.assignment = float(a)

            #Re compute after update
            student.compute_final()
            student.compute_grade()
            print("Student updated and grades re-calculated")
        else:
            print(f"Student with ID {s_id} not found.")
    except ValueError :
        print("Invalid input. Please enter a valid Student ID.")
```

Function: `show_all_students()`

This function is used to **display all student records** in a table format.
It shows the student ID, name, final score, and grade for every student in the list.
If there are no records, it displays a message indicating that the list is empty. This function provides a **quick overview of the entire class performance**.

```python
def show_all_students():
    print("/n --- All Student Records ---")
    if not  std_list:
        print("No records found.")
        return

    print(f"{'ID':<10} {'Name':<15}{'Final Score':<20} {'Grade':<10}")
    print("-"*45)

    for s in std_list:
        print(f"{s.id:<10} {s.name:<15}{s.final_score:<20} {s.grade:<10}")
```

Task 5 — Reporting & Statistics (6 Marks)

Function: `generate_report()`

This function is used to **generate a class performance report**.
It calculates and displays the following statistics:

- **Highest and Lowest Final Score** – Shows the top and bottom performers.
- **Class Average** – Calculates the average final score of all students.
- **Grade Distribution** – Counts how many students received each letter grade (A, B, C, D, F).
- **Sorted List by Final Score** – Orders students from highest to lowest final score for quick ranking.

If there are no student records, it displays a message stating that no data is available.

```
# Calculate and display
# a. Highest and lowest score
# b. Class Average
# c. Grade Distribution
# d. Sorted list by final score


def generate_report():
    print("\n --- Class Performance Report ---")
    if not std_list:
        print("No data avaliable to generate statistics.")
        return

    # Extract Scores
    scores = [s.final_score for s in std_list]

    # a. Highest and lowest score
    highest = max(scores)
    lowest = min(scores)

    # b. Class Average
    average  = sum(scores) / len(scores)

    # c. Grade Distribution
    grades = [s.grade for s in std_list]
    grade_counts = {"A":0 , "B": 0 , "C":0 , "D":0 ,"F":0}
    for g in grades:
        if g in grade_counts:
            grade_counts[g] += 1

    # d. Sorted list (Descending order)
    sorted_students  = sorted(std_list,key = lambda x: x.final_score , reverse=True)
```

Function: `main()`

The `main()` function is the **entry point of the program** and controls the overall flow.

- It **loads existing student data** from the CSV file using `load_data()`.
- Displays a **menu of options** to the instructor:

1. Add Student
2. Remove Student
3. Update Student
4. Search Student
5. Show All Students
6. Generate Report (Statistics)
7. Exit

- Uses a **while loop** to repeatedly take user input and call the corresponding functions based on the choice.
- On exit, it **saves all data** using `save_data()` and terminates the program.
- Ensures the system runs continuously until the instructor chooses to exit.

This function ties together all modules and makes the program **interactive and user-friendly**.

```python
def main():
    load_data()

    print("============================================================")
    print("=====      Student Performance Analyzer       =====")
    print("============================================================\n")
    print("1. Add Student")
    print("2. Remove Student")
    print("3. Update Student")
    print("4. Search Student")
    print("5. Show All Students")
    print("6. Generate Report (Stats)")
    print("7. Exit")


    while True:
        choice = input("\nEnter your choice (0-7)")
        if choice == '1':
            add_std()
        elif choice == '2':
            remove_std()
        elif choice == '3':
            update_student()
        elif choice == '4':
            search_student()
        elif choice  == '5':
            show_all_students()
        elif choice == '6':
            generate_report()
        elif choice == '7':
            save_data()
            print("Exiting System. Good bye!")
            break
        else:
            print("Invalid Choice. Please try again")
```

```python
if __name__ == "__main__":
    main()
```

```
Data loaded successfully from students.csv.
============================================================
=====      Student Performance Analyzer       =====
============================================================

1. Add Student
2. Remove Student
3. Update Student
4. Search Student
5. Show All Students
6. Generate Report (Stats)
7. Exit
```