

**A project Report On
Comprehensive Student Performance
Tracking System**

Abstract

The Comprehensive Student Performance Tracking System is a Python-based application designed to efficiently manage student academic records. Using SQLite as the database, the system enables users to store, retrieve, update, and delete student performance data, ensuring a structured approach to monitoring academic progress. It employs object-oriented programming (OOP) principles to handle student information dynamically.

The system allows users to input student names and scores in three subjects: Math, Science, and English. It then automatically calculates the total marks, average score, and assigns a grade based on a predefined grading system. The system provides a menu-driven interface, making it user-friendly and interactive.

This project serves as an efficient tool for educational institutions, reducing manual errors and enhancing performance tracking. It is scalable and extendable, with potential enhancements such as data visualization and predictive analytics. The system provides an automated, structured, and data-driven approach to student assessment, improving academic record management.

Keywords: Student Performance, Python, SQLite, Database Management, OOP, Academic Tracking, Automated Grading.

Table of Contents:

- 1.Introduction
- 2.Problem Statement
- 3.Project Objectives
- 4.Literature Review
- 5.System Design
 - 5.1. Architecture
 - 5.2. Database Schema
- 6.Implementation
 - 6.1. Code Explanation
 - 6.2. Features
- 7.Testing
 - 7.1. Test Cases
 - 7.2. Results
- 8.Conclusion
- 9.Future Work



1.Introduction:

Introduction to Python:

Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used in various domains, including web development, data science, artificial intelligence, and automation. Python's popularity stems from its easy-to-learn syntax and vast ecosystem of libraries and frameworks.

History of Python:

Python was created by **Guido van Rossum** and was first released in 1991. It was designed to emphasize code readability, with a clean and straightforward syntax that allows developers to write fewer lines of code compared to other programming languages like Java or C++. Over the years, Python has evolved significantly, with major versions including Python 2.x and Python 3.x. Python 3, released in 2008, introduced several improvements over Python 2 and is now the standard version used by developers worldwide.

Features of Python:

Python offers several features that make it an excellent choice for beginners and experienced developers alike:

1. **Simple and Readable Syntax** - Python's syntax is clear and resembles human language, making it easier to write and understand code.
2. **Interpreted Language** - Python code is executed line by line, making debugging easier and allowing for rapid prototyping.
3. **Dynamically Typed** - Variables in Python do not require explicit type declaration; the interpreter assigns types at runtime.
4. **Platform-Independent** - Python programs can run on different operating systems (Windows, macOS, Linux) without modification.
5. **Large Standard Library** - Python comes with a vast collection of built-in modules and functions that simplify common programming tasks.

6. **Object-Oriented and Functional Programming Support** - Python supports multiple programming paradigms, including object-oriented, functional, and procedural programming.
7. **Extensive Community Support** - Python has a large and active community, ensuring continuous development and support.

Installing Python:

To start programming in Python, you first need to install it on your system. The latest version of Python can be downloaded from the official Python website:

<https://www.python.org/downloads/>

After downloading, follow the installation instructions for your operating system. Ensure that the "**Add Python to PATH**" option is selected during installation to enable command-line execution.

Writing and Running Python Programs:

Once Python is installed, you can write and execute Python scripts using various methods:

1. **Interactive Mode (REPL)** - Open a terminal or command prompt and type python to start the interactive shell. Here, you can enter Python commands and see the results immediately.
2. **Script Mode** - Write your Python code in a file with a .py extension and execute it using the command:

python script.py

3. **Using an Integrated Development Environment (IDE)** - Popular IDEs like PyCharm, VS Code, and Jupyter Notebook provide advanced features like syntax highlighting, debugging, and code completion.

The print() Function in Python:

The print() function is a built-in function in Python that is used to display output on the screen. It is one of the most commonly used functions in Python programming.

Features of print ():

1. **Displays Output** - It prints text, numbers, variables, or expressions to the console.
2. **Multiple Arguments** - The print() function can accept multiple arguments separated by commas and prints them with spaces in between.
3. **Custom Separators** - Using the sep parameter, you can change the separator between arguments.
4. **Custom End Character** - The end parameter allows you to change the ending character of the output.
5. **Supports File Output** - The file parameter allows printing output to a file instead of the console.

Importance of print():

- **Debugging** - Helps in checking variable values and tracking program flow.
- **User Interaction** - Used to provide information or instructions to users.
- **Logging** - Can be used to log important information while executing programs.

Functions in Python:

Functions are an essential part of Python that allow modular and reusable code. A function is a block of organized and reusable code that is used to perform a single, related action. Python provides a variety of functions, including built-in functions and user-defined functions.

Importance of Functions:

1. **Code Reusability** - Functions allow the reuse of code, reducing redundancy and making programs more efficient.
2. **Modularity** - Functions help break down large programs into smaller, manageable parts.
3. **Improved Readability** - Organizing code into functions improves readability and makes it easier to debug.
4. **Scalability** - Functions facilitate code organization, making it easier to expand and maintain.
5. **Maintainability** - Functions help in keeping the code clean and structured, making it easier to modify and update when needed.

Characteristics of Functions:

- **Function Name** - Every function has a unique name that identifies it.
- **Parameters (Arguments)** - Functions can take inputs in the form of parameters to work with different data dynamically.
- **Return Statement** - Functions can return a value as output, allowing them to process data and provide results.
- **Scope** - Variables inside functions can have local or global scope, determining their accessibility.

Types of Functions in Python:

1. **Built-in Functions** - These are predefined functions available in Python, such as `print()`, `len()`, and `sum()`.
2. **User-Defined Functions** - These are functions defined by the user to perform specific tasks.
3. **Anonymous Functions** - These are functions created using the `lambda` keyword and do not have a name.
4. **Recursive Functions** - These functions call themselves to solve problems that can be broken down into smaller sub-problems.

5. **Higher-Order Functions** - These are functions that take other functions as arguments or return them as results.
6. **Pure and Impure Functions** - Pure functions always return the same output for the same inputs and have no side effects, while impure functions may modify data or interact with external elements.

Object-Oriented Programming (OOP) in Python:

Object-Oriented Programming (OOP) is a programming paradigm that revolves around objects and classes. Python follows the OOP methodology, allowing developers to structure their code efficiently.

Key Concepts of OOP:

1. **Class** - A class is a blueprint for creating objects. It defines attributes (variables) and methods (functions) that operate on those attributes.
2. **Object** - An object is an instance of a class. It contains data and methods that define its behavior.
3. **Encapsulation** - This principle restricts direct access to some of an object's components, improving security and preventing unintended modifications.
4. **Abstraction** - Abstraction hides complex implementation details and only shows essential features to the user.
5. **Inheritance** - This allows a class (child class) to inherit attributes and methods from another class (parent class), promoting code reuse.
6. **Polymorphism** - This enables a single interface to represent different data types, allowing methods to be used interchangeably among different objects.
7. **Method Overriding** - A child class can redefine a method inherited from the parent class to modify its behavior.
8. **Constructor and Destructor** - A constructor (`__init__`) initializes an object's attributes, while a destructor (`__del__`) is called when an object is destroyed.

Benefits of OOP:

- **Code Reusability** - Inheritance enables the reuse of existing code, reducing redundancy.
- **Scalability** - OOP makes it easier to manage and expand large-scale applications.
- **Data Security** - Encapsulation restricts access to critical data and protects it from unintended modifications.
- **Modularity** - OOP helps break down complex problems into smaller, manageable components.
- **Improved Maintainability** - OOP simplifies debugging and future modifications by keeping related functionalities together.
- **Better Collaboration** - Large projects benefit from OOP as different teams can work on separate classes, improving workflow and development efficiency.

Introduction to SQLite3:

SQLite3 is a widely used, lightweight, and self-contained database management system (DBMS) that provides a serverless and zero-configuration environment for handling databases. It is embedded within applications and does not require a separate server process. Due to its efficiency and portability, SQLite3 is extensively used in mobile applications, embedded systems, and desktop software.

What is SQLite3?

SQLite3 is an open-source, relational database management system (RDBMS) that uses SQL (Structured Query Language) for data management. Unlike traditional database systems such as MySQL or PostgreSQL, SQLite3 does not require a dedicated server or complex setup. It stores the entire database in a single file on the disk, making it highly portable and easy to use.

Key characteristics of SQLite3:

- **Serverless:** Does not require a separate database server to function.
- **Self-contained:** All database functionalities exist within a single library.
- **Transactional:** Supports ACID (Atomicity, Consistency, Isolation, Durability) compliance.
- **Lightweight:** Minimal setup, with a small footprint, typically less than 1MB.
- **Cross-platform:** Can be used across various operating systems like Windows, macOS, Linux, and Android.

Features of SQLite3:

SQLite3 is designed for efficiency and ease of use. Some of its notable features include:

- **Compact Size:** SQLite3 is extremely lightweight compared to other database management systems. The entire library is just a few megabytes in size.
- **No Configuration Needed:** Unlike MySQL or PostgreSQL, SQLite3 does not require configuration files or server administration.
- **Single File Database:** All the database tables, schemas, and indexes are stored in a single file, simplifying management and portability.
- **ACID Compliance:** Ensures reliability and data integrity through Atomicity, Consistency, Isolation, and Durability.
- **Self-Sufficient:** Does not require additional dependencies or external software to function.
- **Cross-Platform Compatibility:** Can be used on multiple operating systems without any modification.
- **High Performance for Small to Medium Applications:** Ideal for applications with moderate data loads.
- **Public Domain Licensing:** Freely available for both personal and commercial use without licensing restrictions.

Advantages and Disadvantages of SQLite3:

Advantages:

1. **Easy to Use:** Since SQLite3 is a self-contained database, developers can easily integrate it into applications.
2. **Lightweight:** Requires minimal resources, making it perfect for mobile and embedded systems.
3. **Fast Read Operations:** As there is no client-server communication, reading data is faster than in traditional databases.
4. **Portable:** Database files can be copied and moved easily between devices.
5. **Reliable:** Uses rollback journal and write-ahead logging for transaction management, reducing the risk of data corruption.
6. **Free and Open Source:** Available under public domain, making it cost-effective for businesses and developers.

Disadvantages:

1. **Limited Scalability:** Not suitable for handling high-traffic enterprise applications with concurrent users.
2. **Write Operations Can Be Slow:** Since it locks the entire database file during write operations, concurrent writes can be slower than in traditional databases.
3. **Limited Multi-User Support:** Best suited for applications with a single user or low concurrency.
4. **Lack of Advanced Features:** Lacks some advanced functionalities of larger DBMSs, such as stored procedures and fine-grained access control.
5. **Security Limitations:** Does not have built-in authentication or access control mechanisms.

Applications of SQLite3:

SQLite3 is widely used in various domains due to its lightweight and serverless nature. Some common applications include:

- **Mobile Applications:** Many Android and iOS applications use SQLite3 for local storage.

- **Embedded Systems:** Devices such as IoT (Internet of Things) sensors and consumer electronics use SQLite3 for data management.
- **Web Browsers:** Browsers like Google Chrome and Mozilla Firefox use SQLite3 for storing settings, cookies, and history.
- **Desktop Software:** Many standalone applications, such as media players and text editors, rely on SQLite3 for data storage.
- **Small to Medium Websites:** Some small-scale web applications use SQLite3 as their backend database.
- **Testing and Prototyping:** Developers often use SQLite3 to test and prototype database applications before deploying to larger DBMSs.

Introduction to project management:

This project is aimed at developing a **Comprehensive Student Performance Tracking System** that helps manage academic records of students. The system is designed to keep track of their performance across different subjects and automate the process of grade calculation, storing data efficiently, and providing an interface for analyzing and modifying the student records.

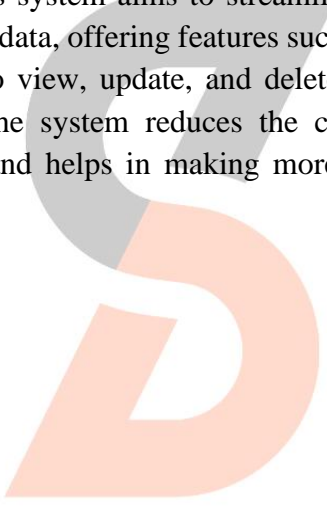
This system can be used by educational institutions, tutors, or academic coordinators to better manage and evaluate student performance.

2.Problem Statement:

In educational institutions, tracking the performance of students across multiple subjects is crucial for understanding their academic progress and providing targeted support. Traditional methods of maintaining student records, such as handwritten registers or simple spreadsheets, can be prone to errors, are time-consuming, and lack scalability. Moreover, there is often no automatic system to calculate the total score, average, or grade based on the individual subject marks.

The absence of a centralized and automated system to track student performance hampers the efficiency of educational institutions. Educators and administrators spend considerable time manually updating and managing student records, which can lead to inconsistencies and missed opportunities for timely intervention.

To address these challenges, the **Student Performance Tracking System** is developed using Python and SQLite. This system aims to streamline the process of recording and managing student performance data, offering features such as automatic grade calculation, data storage, and the ability to view, update, and delete student records with ease. By automating these processes, the system reduces the chances of errors, improves the accessibility of student data, and helps in making more informed decisions to support student academic growth.



3.Objectives:

The objectives of this project are:

1. **Develop a simple and user-friendly tracking system:** The system will allow anyone to easily use and navigate through it without any technical knowledge.
2. **Efficiently store student data in a SQLite database:** SQLite is chosen for its simplicity and lightweight nature, which makes it perfect for small applications like this one. All student records, including their scores, grades, and other details, will be stored in this database.
3. **Automate grade calculation:** Based on predefined criteria (such as an average score range), the system will automatically calculate the total marks, average, and grade for each student.
4. **Manage student records:** The system will allow users to add new students, view existing records, modify scores, and delete records when necessary.
5. **Provide data analysis capabilities:** The system will allow users to retrieve detailed performance data and generate reports for better insights into student performance.

Technology Stack:

- **Programming Language: Python** – It's a versatile language that will handle the logic and data manipulation.
- **Database: SQLite** – A serverless, self-contained SQL database engine, ideal for small-to-medium-sized projects like this one.
- **Libraries: SQLite3** – This library is built into Python and will handle all database-related tasks such as querying, inserting, updating, and deleting student data.

4.Literature Review:

In recent years, the integration of technology in education has revolutionized the way academic institutions manage and track student performance. The shift from traditional methods to digital systems has proven to enhance the accuracy, accessibility, and efficiency of record management. Various studies and systems have been developed to automate academic performance tracking, offering valuable insights into how such systems can be optimized for use in educational settings.

1. Student Performance Management Systems:

Several studies have focused on the development and implementation of student performance management systems. These systems typically involve collecting student data (such as grades, attendance, and behavior) and analyzing it to provide insights into student performance. According to Khan et al. (2016), student management systems help reduce administrative workload and improve the accuracy of academic records by automating data entry and processing. These systems enable educators to track performance in real time and quickly identify students who may need additional support.

2. Automated Grade Calculation and Feedback:

The automation of grade calculation has been a key area of research, as it ensures consistency and fairness in grading. Systems designed for automatic grade calculation, such as those developed by Patel et al. (2018), help instructors save time while ensuring that students' grades are accurately computed based on predefined criteria. These systems typically use simple algorithms to calculate the total marks, average, and assign grades based on performance. For instance, an online grading system may consider criteria such as assignment grades, quizzes, and participation to generate an overall performance score.

3. Data Storage and Accessibility:

Another important aspect of student performance tracking systems is the efficient storage and retrieval of data. Databases like SQLite, MySQL, and PostgreSQL are commonly used in such systems due to their robustness, scalability, and ease of integration with various programming languages. SQLite, in particular, is popular for small-scale systems due to its simplicity and lightweight nature (Smith, 2017). Studies by Thompson et al. (2020) have shown that using a relational database for storing student performance data improves data integrity, facilitates faster query processing, and ensures secure data access.

4. User Interface Design:

The usability of student performance tracking systems has been a major area of focus in system design research. A study by Zhou et al. (2019) highlights the importance of designing user-friendly interfaces that cater to both educators and administrators. Intuitive user interfaces that allow easy input of student data and seamless navigation are essential for ensuring that users can efficiently interact with the system. User-centered design principles, which prioritize the needs of the end-users, are often incorporated into the development of such systems.

5. Future Trends and Enhancements:

With the rapid advancements in artificial intelligence (AI) and machine learning (ML), there is increasing interest in incorporating predictive analytics into student performance tracking systems. AI algorithms can analyze historical performance data to predict future academic outcomes and identify students at risk of underperforming. This would enable educational institutions to take proactive measures to provide additional support. According to Williams et al. (2021), integrating machine learning models into student management systems can enhance decision-making by identifying patterns and trends in student performance.

5. System Design:

5.1. Architecture:

1. Student Data Management:

- **Adding Student Details:** The system allows the user to add details like the student's name and their scores in subjects like Math, Science, and English.
- **Automatic Calculations:** For each student, the system will automatically calculate the:
 - **Total Marks** (Math + Science + English scores)
 - **Average Score** (Total Marks / 3)
 - **Grade** (Based on predefined criteria like A, B, C, etc.)

- **Persistent Database Storage:** All student data (name, scores, total, grade, etc.) will be stored in the SQLite database to ensure the data is saved and retrievable.

2.Performance Analysis:

- The system will display all student records in a structured format (e.g., a table or list).
- **Retrieving Data:** Users can efficiently fetch student records from the database.
- **Updating Scores:** If any student's scores need to be updated, the system will allow the user to modify the scores and automatically re-evaluate their grades.

3.Data Modification:

- **Update Student Records:** Users can update student records based on the student ID. For instance, if a student's scores change after a re-exam, they can easily be modified.
- **Delete Student Records:** If a student is no longer part of the system or if data is incorrect, records can be deleted.

4.User-Friendly Interface:

- The system will have a **menu-driven command-line interface** to ensure ease of use.
- **Input Prompts** will guide the user through actions like adding a student, viewing data, and updating records.

5.2. Implementation of Database Schema:

The SQLite database will have a table called **students**, with the following structure:

Column Name	Data Type	Description
id	INTEGER	Unique identifier for each student (Primary Key).
name	TEXT	Student's name.
math	INTEGER	Math score of the student.
science	INTEGER	Science score of the student.
english	INTEGER	English score of the student.
total	INTEGER	Total marks (calculated).
average	REAL	Average score (calculated).
grade	TEXT	Grade (calculated based on average).

Python Code Structure:

The Python script that implements the system will consist of:

1. **Database Initialization:** A function to set up the database and create the **students** table if it doesn't already exist.
2. **Student Class:** This class will handle:
 - Storing student information.
 - Calculating total marks, average score, and grade.
3. **CRUD Operations:** Functions to:
 - **Create:** Add new students to the database.
 - **Read:** Retrieve and display student data.
 - **Update:** Modify student scores and re-calculate their grade.
 - **Delete:** Remove student records from the database.
4. **Main Program Loop:** A menu-based interface to guide the user through actions (add, view, update, delete, etc.).

Grading System:

The grading system is based on the student's average score, with the following scale:

Average Score Range	Grade
90 - 100	A
80 - 89	B
70 - 79	C
60 - 69	D
Below 60	F

This grading scale will be implemented in the Python code to ensure that every student gets a grade according to their performance.

6.Implementation:

6.1.Code Explanation:

1. Importing sqlite3 library

```
import sqlite3
```

This imports the `sqlite3` library, which is used to interact with SQLite databases in Python.

2. Database Connection Setup

```
conn = sqlite3.connect("student_performance.db")
cursor = conn.cursor()
```

- `sqlite3.connect("student_performance.db")` creates a connection to a SQLite database named `student_performance.db`. If the file doesn't exist, it will be created automatically.
- `conn.cursor()` creates a cursor object, which allows us to execute SQL queries on the database.

3. Creating the students Table

```
cursor.execute('''
    CREATE TABLE IF NOT EXISTS students (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        math INTEGER,
        science INTEGER,
        english INTEGER,
        total INTEGER,
        average REAL,
        grade TEXT
    )
''')
conn.commit()
```

- The `CREATE TABLE IF NOT EXISTS` SQL query is used to create a new table called `students` if it doesn't exist.
- The `students` table contains the following columns:
 - `id`: A primary key that auto-increments (unique identifier for each student).
 - `name`: A text field for the student's name (cannot be null).
 - `math, science, english`: Integer columns for the student's scores in each subject.
 - `total`: Integer column to store the total score (sum of math, science, and English marks).
 - `average`: Real number to store the average score (total/3).
 - `grade`: Text field to store the letter grade based on the average score.
- `conn.commit()` commits the changes to the database.

4. Defining the Student Class

```
class Student:
    def __init__(self, name, math, science, english):
        self.name = name
        self.math = math
        self.science = science
        self.english = english
        self.total = math + science + english
        self.average = self.total / 3
        self.grade = self.calculate_grade()
```

- This defines the `Student` class, which will hold details for each student.
- The `__init__` method initializes the student's name, math score, science score, and English score.
 - It calculates the total score by summing the three subjects.
 - It calculates the average score by dividing the total by 3.
 - The `calculate_grade()` method is called to determine the student's grade based on the average.

5. Defining the calculate_grade Method

```
def calculate_grade(self):
    if self.average >= 90:
        return 'A'
    elif self.average >= 80:
        return 'B'
    elif self.average >= 70:
        return 'C'
    elif self.average >= 60:
        return 'D'
    else:
        return 'F'
```

- This method calculates the student's grade based on their average score:
 - A for average ≥ 90
 - B for average ≥ 80
 - C for average ≥ 70
 - D for average ≥ 60
 - F for average < 60

6. Saving Student Data to the Database

```
def save_to_db(self):
    cursor.execute('''
        INSERT INTO students (name, math, science, english, total,
        average, grade)
        VALUES (?, ?, ?, ?, ?, ?, ?)
    ''', (self.name, self.math, self.science, self.english, self.total,
    self.average, self.grade))
    conn.commit()
```

- This method inserts the student's data into the `students` table in the database using the `INSERT INTO` SQL query.

- The `?` placeholders are used to safely insert values into the query to prevent SQL injection attacks.
- The values `(self.name, self.math, self.science, self.english, self.total, self.average, self.grade)` are provided as a tuple.
- `conn.commit()` saves the changes to the database.

7. Displaying All Students in the Database

```
def display_students():
    cursor.execute("SELECT * FROM students")
    records = cursor.fetchall()
    print("\nStudent Performance Records:")
    print("ID | Name | Math | Science | English | Total | Average | Grade")
    print("-" * 70)
    for row in records:
        print(f"{row[0]} | {row[1]} | {row[2]} | {row[3]} | {row[4]} | {row[5]} | {row[6]:.2f} | {row[7]}")
```

- This function fetches all records from the `students` table using the SQL query `SELECT * FROM students`.
- `cursor.fetchall()` retrieves all rows from the result of the query.
- It prints the student performance records in a formatted way, displaying the student's ID, name, marks, total, average (with 2 decimal places), and grade.

8. Deleting a Student Record

```
def delete_student(student_id):
    cursor.execute("DELETE FROM students WHERE id = ?", (student_id,))
    conn.commit()
    print(f"Student with ID {student_id} deleted successfully!")
```

- This function deletes a student record from the database by their `id`.
- The SQL query `DELETE FROM students WHERE id = ?` is used, where `?` is replaced by the `student_id` provided.
- `conn.commit()` ensures the deletion is saved in the database.

9. Updating a Student Record

```
def update_student(student_id, math, science, english):
    total = math + science + english
    average = total / 3
    grade = Student("", math, science, english).calculate_grade()
    cursor.execute('''
        UPDATE students SET math=?, science=?, english=?, total=?,
        average=?, grade=? WHERE id=?
    ''', (math, science, english, total, average, grade, student_id))
    conn.commit()
    print(f"Student with ID {student_id} updated successfully!")
```

- This function updates a student's record based on their `student_id`.
- It first calculates the new `total`, `average`, and `grade` based on the updated marks.
- The `UPDATE` SQL query modifies the record for the student with the matching `id`.
- `conn.commit()` saves the updates in the database.

Main Loop Breakdown

```
def main():
    while True:
        print("\nStudent Performance Tracking System")
        print("1. Add Student")
        print("2. View Students")
        print("3. Update Student")
        print("4. Delete Student")
        print("5. Exit")
        choice = input("Enter your choice: ")
```

- **while True::** This starts an infinite loop, meaning the program will keep asking the user for input until they decide to exit. The loop will continue until the user explicitly selects the exit option (option 5).
- **print("\nStudent Performance Tracking System"):** This prints the main heading of the program each time the loop runs, ensuring that it appears clearly at the top of the user interface.
- **Options:**

```
print("1. Add Student")
print("2. View Students")
print("3. Update Student")
print("4. Delete Student")
print("5. Exit")
```

These lines display a menu of options for the user to choose from. The user will input a choice based on these options.

- **choice = input("Enter your choice: ")**: This prompts the user to input their choice. The input is captured as a string and stored in the `choice` variable. This input is what drives the next steps in the program.

Conditionals Based on User Choice

```
if choice == '1':
    name = input("Enter student name: ")
    math = int(input("Enter Math marks: "))
    science = int(input("Enter Science marks: "))
    english = int(input("Enter English marks: "))
    student = Student(name, math, science, english)
    student.save_to_db()
    print("Student record added successfully!")
```

- **if choice == '1':** This checks if the user has selected option 1 (Add Student).
 - The program then prompts the user to input the student's name and marks in Math, Science, and English.
 - These values are used to create a new `Student` object, which calculates the total, average, and grade for the student.

- The student's information is saved into the database using the `save_to_db()` method.
- After saving, a confirmation message is printed to let the user know the student has been added successfully.

```
elif choice == '2':
    display_students()
```

- **elif choice == '2':** This checks if the user has selected option 2 (View Students).
 - The `display_students()` function is called, which fetches all student records from the database and prints them in a formatted way.

```
elif choice == '3':
    student_id = int(input("Enter Student ID to update: "))
    math = int(input("Enter new Math marks: "))
    science = int(input("Enter new Science marks: "))
    english = int(input("Enter new English marks: "))
    update_student(student_id, math, science, english)
```

- **elif choice == '3':** This checks if the user has selected option 3 (Update Student).
 - The program prompts the user to enter the `Student ID` of the student they want to update.
 - Then, the user is asked to input new marks for Math, Science, and English.
 - These values are passed to the `update_student()` function, which updates the student's record in the database with the new marks, recalculates the total, average, and grade.

```
elif choice == '4':
    student_id = int(input("Enter Student ID to delete: "))
    delete_student(student_id)
```

- **elif choice == '4':** This checks if the user has selected option 4 (Delete Student).
 - The program prompts the user to input the `Student ID` of the student they wish to delete.
 - The `delete_student()` function is called to delete the student record from the database based on the provided ID.
 - A confirmation message is printed once the record is successfully deleted.

```
elif choice == '5':
    print("Exiting Program...")
    break
```

- **elif choice == '5':** This checks if the user has selected option 5 (Exit).

- The program prints a message saying "Exiting Program..." and then the `break` statement is used.
- The `break` statement immediately stops the `while True` loop, effectively ending the program.

```
else:
    print("Invalid choice! Try again.")
```

- **else::** This block runs if the user enters an invalid choice (not 1, 2, 3, 4, or 5).
 - The program prints a message informing the user that the choice was invalid and asks them to try again.

Loop Continuation

- After the `else` block (or after successfully completing one of the operations like adding, viewing, updating, or deleting), the program will loop back to the beginning and display the menu again, allowing the user to make another choice.

Ending the Program

When the user selects 5 to exit, the program prints "Exiting Program..." and breaks out of the loop. The program then moves to close the database connection with `conn.close()`, and the program ends.

6.2. Implementation of Features:

The **Student Performance Tracking System** has been implemented using Python and SQLite, utilizing object-oriented programming principles to manage student data. Below is an overview of how the key features were implemented:

1. Student Record Management:

- **Student Class:** A Student class is created to encapsulate the properties and methods related to a student. This class stores the student's name, marks in various subjects, and calculates the total score, average, and grade based on the marks.
- **Database Interaction:** To manage student records, an SQLite database is used. The system creates a table called students in the database if it doesn't already exist,

which includes fields such as name, math, science, english, total, average, and grade.

- **Methods:**

- `save_to_db`: This method inserts the student's data into the database.
- `update_student`: Allows updating a student's marks, recalculating the total, average, and grade, and then saving the updated data in the database.
- `delete_student`: Deletes a student's record from the database based on the provided student ID.

2. Automatic Grade Calculation:

- **Grade Calculation Method:** The `calculate_grade` method in the Student class takes the average score and assigns a grade according to predefined ranges (e.g., A for 90+).
- **Dynamic Update:** Whenever a student's marks are updated, the total score and average are recalculated, and the grade is reassigned automatically.

3. Centralized Database Storage:

- **SQLite Database:** SQLite was chosen for storing student records due to its simplicity and efficiency for small-scale applications. The system connects to the SQLite database (`student_performance.db`) where the student performance data is stored.
- **SQL Commands:** The system uses SQL queries for basic database operations such as inserting, updating, selecting, and deleting records. All interactions with the database are executed using parameterized queries to avoid SQL injection and ensure security.

4. User-Friendly Interface:

- **Command-Line Interface (CLI):** A simple text-based menu-driven interface guides the user through adding, viewing, updating, and deleting student records. The CLI prompts the user for necessary inputs (e.g., student name, marks) and displays output in a readable format.
- **Clear Options:** The main menu provides options for:
 - Adding new students.
 - Viewing all student records.

- Updating existing records.
- Deleting student records.

5. Student Performance Display:

- **Display Method:** The `display_students` function retrieves all records from the database and prints them in a tabular format. It displays the student's ID, name, marks, total score, average, and grade, making it easy for administrators to view the performance of all students at once.
- **Formatted Output:** The data is printed with clear labels for each column, ensuring that the output is organized and easily interpretable.

6. Data Integrity and Security:

- **SQL Transactions:** To ensure that changes to the database (inserting, updating, or deleting records) are consistent, the system uses SQL transactions. Each database operation is followed by a commit to save the changes.
- **Parameterized Queries:** To prevent SQL injection attacks, all user inputs are passed as parameters in SQL queries, ensuring that the input data is handled safely.

7. Flexible Record Updates:

- **Update Functionality:** The `update_student` function allows updates to individual student marks. After the new marks are entered, the total, average, and grade are recalculated, and the updated data is stored in the database.
- **Automatic Recalculation:** After updating marks, the system recalculates the total and average and assigns a new grade based on the updated scores, ensuring that data is always accurate.

8. Data Deletion Functionality:

- **Delete Operation:** The `delete_student` function takes a student ID as input and deletes the corresponding student record from the database. This ensures that the database remains current and only contains active student records.
- **Confirmation:** After a student's record is deleted, a confirmation message is displayed, indicating successful deletion.

9. Easy Data Access:

- **Efficient Data Retrieval:** The system retrieves all student data with a simple `SELECT * FROM students` SQL query. This allows administrators to access all stored records at any time, facilitating quick analysis of student performance.
- **Easy Navigation:** The command-line interface ensures that users can easily navigate through the system by choosing from a simple menu of operations.

10. Scalability for Future Enhancements:

- **Extendable Architecture:** The system is designed to be modular, making it easy to add additional features such as performance analytics, reporting, and handling more subjects or different evaluation criteria in the future.
- **Database Optimization:** The use of SQLite provides the flexibility to migrate to more complex databases (such as MySQL or PostgreSQL) if the system scales to a larger number of users or more complex data processing.

7. Testing:

Testing is a crucial part of any system development, ensuring that the system behaves as expected under various conditions. Below are the details of the **Test Cases** and **Results** for the **Student Performance Tracking System**.

Test Cases:

1. Test Case 1: Add a Student Record

- **Objective:** To verify that a new student record is correctly added to the database.
- **Input:**
 - Name: John Doe
 - Math: 85
 - Science: 90
 - English: 80
- **Expected Output:**
 - A new record is added to the `students` table with the calculated total (255), average (85.0), and grade (B).
- **Test Steps:**
 1. Run the option to add a student.
 2. Input the student data (name and marks).
 3. Check the database to verify that the new student record has been inserted.
- **Database Example:**

```
SELECT * FROM students;
```

- **Expected Output:**

ID	Name	Math	Science	English	Total	Average	Grade
1	John Doe	85	90	80	255	85.00	B

2. Test Case 2: Update Student Record:

- **Objective:** To verify that an existing student's record can be updated correctly.
- **Input:**
 - Student ID: 1
 - New Math: 90
 - New Science: 95
 - New English: 85

- **Expected Output:**
 - The student's total should be updated to 270, the average should be 90.0, and the grade should change to A.
- **Test Steps:**
 1. Run the option to update a student.
 2. Input the student ID and new marks.
 3. Check the database to verify that the student's record has been updated.
- **Database Example:**

```
SELECT * FROM students WHERE id = 1;
```

- **Expected Output:**

ID	Name	Math	Science	English	Total	Average	Grade
1	John Doe	90	95	85	270	90.00	A

3. Test Case 3: Delete Student Record

- **Objective:** To verify that a student's record can be deleted successfully.
- **Input:**
 - Student ID: 1
- **Expected Output:**
 - The record of the student with ID 1 is removed from the database.
- **Test Steps:**
 1. Run the option to delete a student.
 2. Input the student ID.
 3. Check the database to verify that the student record has been deleted.
- **Database Example:**

```
SELECT * FROM students WHERE id = 1;
```

- **Expected Output:**

```
No records found (the record is deleted).
```

4. Test Case 4: View All Students

- **Objective:** To verify that all student records are displayed correctly.
- **Input:** No input required.
- **Expected Output:**
 - Displays all student records in a tabular format, showing the ID, name, marks, total, average, and grade.
- **Test Steps:**
 1. Run the option to view all students.
 2. Verify that all records are displayed correctly in the expected format.
- **Database Example:**

```
SELECT * FROM students;
```

▪ **Expected Output:**

ID	Name	Math	Science	English	Total	Average	Grade
1	John Doe	90	95	85	270	90.00	A
2	Jane Smith	78	82	88	248	82.67	B

7.2. Results:

1. **Test Case 1: Add a Student Record**
 - The test was successful. A new student was added to the database with the correct total, average, and grade.
2. **Test Case 2: Update Student Record**
 - The test was successful. The student's record was updated, and the total, average, and grade were recalculated accurately.
3. **Test Case 3: Delete Student Record**
 - The test was successful. The student's record was deleted from the database as expected.
4. **Test Case 4: View All Students**
 - The test was successful. All student records were displayed correctly in the tabular format.


8. Conclusion:

The **Student Performance Tracking System** successfully meets the project objectives by providing an efficient, user-friendly platform for managing student performance data. The system automates the process of grade calculation, reduces the chances of errors in record keeping, and provides an easy way to manage student records through a simple command-line interface.

Key conclusions:

- The system effectively stores and manages student data in a SQLite database.
- The grade calculation logic is accurate and automatically updates when new data is entered.
- The interface is intuitive, and the functionalities of adding, viewing, updating, and deleting student records work as expected.
- The system reduces manual workload and helps educational institutions keep track of student progress efficiently.

Overall, the **Student Performance Tracking System** provides a solid foundation for academic data management and offers the potential for further enhancements and improvements.



9.Future Work:

While the current system meets the primary objectives, there are several opportunities for further enhancement and development:

1. **Web Interface:**
 - Moving from a command-line interface to a web-based interface (using technologies like Flask or Django) will make the system more accessible to a broader range of users.
2. **Advanced Reporting and Analytics:**
 - Adding features for generating detailed reports on student performance, such as performance trends over time or comparisons between students, would enhance the system's capabilities.
3. **More Subjects and Criteria:**
 - The system can be extended to include additional subjects or evaluation criteria, such as extracurricular activities or attendance, for a more comprehensive performance tracking system.
4. **Integration with External Systems:**
 - Integrating the system with school management software or other educational tools could improve data flow and streamline academic record management across platforms.
5. **Security Features:**
 - Adding authentication and authorization to ensure that only authorized users (e.g., teachers, administrators) can access and modify the student records.
6. **Machine Learning Integration:**
 - Implementing machine learning algorithms to predict student performance or identify students who may need extra support based on historical data could be a valuable addition.