## **Project: Student Performance Tracker**

### **Objective**:

Develop a **Student Performance Tracker** using Python, which allows a teacher to input student scores, track performance across subjects, and calculate statistics such as averages and feedback on passing or failing. This project reinforces concepts in Object-Oriented Programming (OOP), loops, conditionals, and data structures (lists and dictionaries).

## **Project Steps & Guidelines**:

### **Step 1: Design the Project**

* **Goal**: Sketch out the structure of the project. What are the key actions you want to perform with the tracker? (e.g., add students, store their grades, calculate averages).
* **Think about**: What kind of data will you be working with? (student names, subjects, grades).

### **Step 2: Class Design**

* **Objective**: Create a basic structure with classes to represent both a **Student** and a **Performance Tracker**.

#### **Tasks**:

1. **Student Class**:
   * **Attributes**:
     + name: The student’s name (string).
     + scores: A list of integers representing scores in subjects (e.g., math, science, English).
   * **Methods**:
     + calculate\_average(): This method calculates the average of the student’s scores.
     + is\_passing(): This method determines if the student is passing all subjects (e.g., a passing score might be 40 for each subject).
2. **PerformanceTracker Class**:
   * **Attributes**:
     + students: A dictionary where the keys are student names and the values are Student objects.
   * **Methods**:
     + add\_student(): This method adds a student and their scores to the system.
     + calculate\_class\_average(): This method calculates the overall average score for the entire class.
     + display\_student\_performance(): This method prints out the performance (average score, pass/fail status) of each student.

### **Step 3: Input Handling**

* **Objective**: Allow users (teachers) to input student names and their scores. Use a loop to keep asking for input until the teacher decides to stop.

#### **Tasks**:

1. **Write Input Prompts**:
   * Ask the user to input a student’s name.
   * Prompt for grades in 3 subjects (e.g., math, science, English). Ensure these are integers.
2. **Handle Invalid Input**:
   * Use try-except blocks to handle cases where the user accidentally enters something other than a number for scores. Give them feedback if they make an error and ask again.
3. **Store Data**:
   * Once the student’s information is entered, store it in a dictionary (using the student’s name as the key and a list of scores as the value).

### **Step 4: Data Processing**

* **Objective**: After inputting data, calculate useful metrics such as:
  + The average score for each student.
  + Whether a student is passing or failing based on their grades.
  + The overall class average.

#### **Tasks**:

1. **Student Average**:
   * For each student, calculate the average of their subject scores using a method in the Student class.
2. **Pass/Fail Check**:
   * Check if any score is below the passing threshold (e.g., 40). If so, mark the student as needing improvement.
3. **Class Average**:
   * Once all students have been added, calculate the class average using the PerformanceTracker class.

### **Step 5: Display Output**

* **Objective**: Display the results of the calculations, including:
  + Each student’s name, average score, and pass/fail status.
  + The class average.

#### **Tasks**:

1. **Format the Output**:
   * Ensure the output is clean and easy to read. Use f-strings to display the student’s name, their average score, and whether they are passing.
2. **Class Average**:
   * At the end of the program, display the class’s overall average score.

### **Step 6: Error Handling and Edge Cases**

* **Objective**: Make the system robust to handle common input errors.

#### **Tasks**:

1. **Handle Invalid Input**:
   * If the user enters non-numeric input for grades, prompt them to try again.
2. **Empty Inputs**:
   * Handle cases where no students or scores are entered, and ensure the system does not crash.

## **Detailed Student Guidelines**:

### **Step-by-Step Breakdown**:

### 1. **Create the Student Class**

In this step, you will create a class that represents each student. Each student has a name and scores in multiple subjects. The class will also include methods to calculate the average grade and check if the student is passing.

#### **To Do**:

* Define a class Student.
* Add two attributes: name and scores.
* Write a method calculate\_average() to calculate the student’s average score.
* Write a method is\_passing() to check if the student has passed all subjects.

### 2. **Create the PerformanceTracker Class**

This class will manage multiple students and provide methods to calculate the class average and display student performance.

#### **To Do**:

* Define a class PerformanceTracker.
* Add an attribute students, which is a dictionary where the keys are student names and the values are Student objects.
* Write a method add\_student() to add new students to the tracker.
* Write a method calculate\_class\_average() to calculate the average score across all students.
* Write a method display\_student\_performance() to print each student’s performance.

### 3. **Handle User Input**

You will allow users (teachers) to input student names and scores for three subjects. You’ll also handle invalid input, such as non-numeric grades, using a try-except block.

#### **To Do**:

* Write a loop that continuously asks the teacher to enter student data.
* Ask for the student’s name and their scores for three subjects.
* Use try-except to handle invalid input and give feedback to the user.

### 4. **Calculate Averages and Display Performance**

After the students’ data has been entered, you will calculate and display each student’s performance and the class average.

#### **To Do**:

* For each student, call the method calculate\_average() to get their average score.
* Use is\_passing() to check if the student is passing or needs improvement.
* Display this information to the user in a clear format.

## **Submission Process using GitHub**

### **Step 1: Create a GitHub Account (If Needed)**

If your students don’t already have a GitHub account: 1. Go to [GitHub](https://github.com/) and sign up for an account. 2. Verify the account via email and log in.

### **Step 2: Create a New Repository**

1. After logging in, click the **+** icon in the top-right corner and select **New repository**.
2. Name the repository, e.g., Student\_Performance\_Tracker.
3. Add a brief description (e.g., “Python project to manage student performance and track grades”).
4. Keep the repository **Public** (unless you’d prefer it to be private).
5. Click the **Create repository** button.

### **Step 3: Add the Project Files**

#### **Option A: If using Colab**

1. After completing the project on Colab:
   * Go to **File** → **Download** → **Download .ipynb**.
   * This will download the notebook in .ipynb format.
2. Upload the .ipynb file to your GitHub repository:
   * In the repository, click **Add file** → **Upload files**.
   * Drag and drop the .ipynb file or select it using the **Choose your files** button.
   * Add a **commit message** (e.g., “Initial commit with Colab notebook”).
   * Click **Commit changes**.

#### **Option B: If using Jupyter Notebook (Locally)**

1. After completing the project in Jupyter Notebook:
   * Save the file by going to **File** → **Save As** → Save it in the .ipynb format.
2. Upload the .ipynb file to your GitHub repository by following the same steps:
   * Click **Add file** → **Upload files**.
   * Upload your notebook file.
   * Add a **commit message** and click **Commit changes**.

#### **Option C: If using Plain Python (.py files)**

1. If you’re writing the code in a .py file:
   * Ensure the file is saved on your local machine.
2. Upload the .py file to your GitHub repository:
   * Click **Add file** → **Upload files**.
   * Upload the .py file (e.g., student\_performance\_tracker.py).
   * Add a **commit message** (e.g., “Initial commit with Python file”).
   * Click **Commit changes**.

### **Step 4: Add a README File**

1. In the GitHub repository, click **Add file** → **Create new file**.
2. Name the file README.md. This file will contain information about the project.
3. In the text editor, write a simple overview of the project:

* # Student Performance Tracker  
    
  This project is a Python-based system to manage and track student grades using Object-Oriented Programming concepts. It calculates averages, checks if students are passing, and provides performance feedback.  
    
  ## How to Use:  
    
  - Clone the repository to your local machine or open the `.ipynb` file on Colab.  
  - Add student names and their scores for different subjects.  
  - The program will calculate the class average and individual student performance.  
    
  ## Files:  
  - `student\_performance\_tracker.ipynb`: Jupyter/Colab notebook for the project.  
  - `student\_performance\_tracker.py`: Python script for the project.  
    
  ## Requirements:  
  - Python 3.x  
  - If using Jupyter or Colab, simply run the cells in sequence.  
    
  ## Installation:  
  1. Clone the repository using `git clone https://github.com/your\_username/Student\_Performance\_Tracker.git`  
  2. Open the notebook in Jupyter or run the Python script directly.

### **Step 5: Submit the GitHub Link**

1. After uploading all files to the repository, go to the repository’s main page.
2. Copy the URL from your browser’s address bar (e.g., https://github.com/your\_username/Student\_Performance\_Tracker).
3. Submit this URL in **Google Classroom**.

## **Guidelines for Submission**:

1. **Naming Conventions**:
   * The GitHub repository should have a descriptive name (e.g., Student\_Performance\_Tracker).
   * Use clear and consistent file names, such as student\_performance\_tracker.ipynb or student\_performance\_tracker.py.
2. **README File**:
   * Ensure the README.md file clearly explains how to use and run the project.
   * Include installation instructions and any dependencies.
3. **GitHub Commit Messages**:
   * Encourage students to write meaningful commit messages (e.g., “Added student class”, “Fixed input validation”).
4. **Use of .ipynb or .py**:
   * Students can submit in either .ipynb (Colab or Jupyter Notebook) or .py format.
   * Make sure all necessary files are in the repository before submission.
5. **Final Step**:
   * Submit the **GitHub repository URL** on Google Classroom.

### **Additional Tips for Students**:

* If you’re working on **Colab**, you can directly connect it to GitHub by going to **File** → **Save a copy to GitHub** and following the prompts.
* If you need to update the repository with new changes, repeat the process of uploading the file and committing the changes.

### **Evaluation Criteria**:

1. **Proper GitHub Setup**:
   * Does the repository contain all necessary files?
   * Is the README file informative and clear?
2. **Code Quality and Functionality**:
   * Does the code meet all requirements (e.g., adding students, calculating performance)?
   * Is the code well-structured and modular (using functions and classes)?
3. **Use of GitHub**:
   * Did the student make appropriate and meaningful commits?
   * Is the repository organized clearly?
4. **Functionality**:
   * Does the project meet all the requirements?
   * Can you add, update, and track students and their grades?
5. **Code Quality**:
   * Is the code clean and well-organized?
   * Have you used appropriate variable and method names?
6. **Application of OOP**:
   * Have you properly used classes, attributes, and methods to organize your code?
   * Are the class methods responsible for handling their specific functionality?
7. **Error Handling**:
   * Have you accounted for possible input errors (e.g., non-numeric grades)?
   * Does the system handle edge cases gracefully?