Project Setup, Imports & Helper Functions

AiCE Software Engineering Foundations — Python Programming (Advanced) — Capstone

Every concept from this entire program has led to this project. The AiCE Student Management System is not an exercise — it is a real, working, command-line application that a teacher in Lagos, a school administrator in Nairobi, or a training coordinator in Accra could genuinely install and use to manage student records. Build it carefully. Type every line yourself. Test every feature. This is the centerpiece of your Python portfolio.

1. Project Overview

The AiCE Student Management System is a command-line application with five core features:

Feature	What It Does
Add Student	Collect name, age, and Math/English/Science scores. Validate every input. Save to JSON.
View All Students	Load all records and display them in a formatted, colour-coded table with averages, grades, and status.
Search Student	Find a student by name (case-insensitive) and display their full profile.
Delete Student	Remove a student by name, with a confirmation message.
View Class Summary	Display total students, pass/fail counts, class average, and highest/lowest averages.

All data is stored in a JSON file, students_data.json. The program loads from this file at the start of every operation and saves back to it after every change — meaning data persists between sessions. Close the program, reopen it, and every student is still there.

This one application draws on every concept covered across this entire program:

json — for persistent data storage and retrieval.

os — for checking whether the data file exists before reading.

colorama — for colour-coded terminal output.

tabulate — for professional, aligned table display.

Exception handling — for robust input validation that prevents crashes.

Functions — each feature is a separate, focused function with a single responsibility.

List comprehensions — for filtering and processing student data.

2. Project Setup

Step 1 — Create a Dedicated Repository

This project deserves its own GitHub repository, separate from your chapter exercises:

Repository name: aice-student-management-system

Description: AiCE Student Management System — Python Culminating Project

Visibility: Public

Clone it and navigate into it:

```
git clone
https://YOUR_TOKEN@github.com/yourusername/aice-student-manage
ment-system.git
cd aice-student-management-system
```

Step 2 — Confirm Dependencies Are Installed

```
pip3 install colorama tabulate
```

Even if these were installed in an earlier unit, running this again is harmless — pip will simply report they are already satisfied.

Step 3 — Create the Project File

```
vim student_system.py
```

Press i to enter Insert Mode. You are ready to build the system, section by section.

3. Section 1 — Imports and Initialisation

Every Python file begins with its imports. Type the following at the very top of student_system.py:

```
import json
import os
from colorama import Fore, Style
from tabulate import tabulate
import colorama

colorama.init()

FILENAME = "students_data.json"
```

Line by line:

import json — Python's built-in JSON module for reading and writing data.

import os — for os.path.exists(), which checks whether the data file is present before reading.

from colorama import Fore, Style — brings the colour and style constants into scope without the colorama. prefix.

from tabulate import tabulate — brings the tabulate function directly into scope.

colorama.init() — initialises colorama for cross-platform compatibility.

FILENAME — a module-level constant holding the data file name. Using a constant means you only need to change the filename in one place if it ever changes.

4. Section 2 — Helper Functions

Helper functions are small, focused utilities used by multiple feature functions. Define them next, beneath the imports:

```
# ——— Helper Functions ————————————————————————————

def load_students():
    """Load student records from the JSON file."""
    if os.path.exists(FILENAME):
        with open(FILENAME, "r") as file:
            return json.load(file)
    return []

def save_students(students):
    """Save student records to the JSON file."""
    with open(FILENAME, "w") as file:
        json.dump(students, file, indent=4)

def get_average(scores):
    """Calculate and return the average of a list of
scores."""
    return round(sum(scores) / len(scores), 2)

def get_grade(average):
    """Return the letter grade corresponding to the average
score."""
    if average >= 70: return "A"
    elif average >= 60: return "B"
    elif average >= 50: return "C"
    elif average >= 40: return "D"
    else: return "F"

def get_status(average):
    """Return PASS or FAIL based on the average score."""
    return "PASS" if average >= 50 else "FAIL"

def print_header(title):
    """Print a cyan-coloured section header."""
    print(Fore.CYAN + "=" * 50)
    print(f" {title}")
    print("=" * 50 + Style.RESET_ALL)

def print_success(message):
```

```
    """Print a green success message."""
    print(Fore.GREEN + message + Style.RESET_ALL)

def print_error(message):
    """Print a red error message."""
    print(Fore.RED + message + Style.RESET_ALL)
```

These eight helper functions handle the repetitive tasks every feature needs. load_students() and save_students() are the most critical — they are the bridge between the in-memory Python data and the JSON file on disk. get_average(), get_grade(), and get_status() compute performance metrics. print_header(), print_success(), and print_error() handle consistent, coloured output.

> **One Change, One Place**
> *By defining these once and using them everywhere, any change to how grading works — or how errors display — requires modifying only one function. This is separation of concerns applied at the function level, and it is the single most important habit this project will reinforce.*

What's Next

With imports, initialisation, and all eight helper functions in place, student_system.py is ready for its five core features. The next lesson builds add_student(), view_all_students(), search_student(), delete_student(), and view_summary() — one at a time, on top of the foundation just laid.