Building the Five Core Features

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

With imports, initialisation, and helper functions in place, this lesson builds all five feature functions — the actual behavior of the Student Management System. Each function has exactly one responsibility, and each builds directly on the helpers from the previous lesson.

1. Feature 1 — Add Student

add_student() collects a student's details, validates every input, checks for duplicates, and saves the new record:

```
# ——— Feature Functions ————————————————————————————

def add_student():
    """Collect and save a new student record with full
validation."""
    print_header("ADD NEW STUDENT")

    # Validate name
    name = input("Enter student name: ").strip()
    if name == "":
        print_error("Name cannot be empty.")
        return

    # Check for duplicates
    students = load_students()
    for student in students:
        if student["name"].lower() == name.lower():
            print_error(f"A student named '{name}' already
exists.")
            return

    # Validate age
    try:
```

```
        age = int(input("Enter student age: "))
        if age <= 0 or age > 100:
            print_error("Please enter a valid age between 1
and 100.")
            return
    except ValueError:
        print_error("Invalid age. Please enter a whole
number.")
        return

    # Validate scores
    try:
        math = float(input("Enter Math score (0-100)   : "))
        english = float(input("Enter English score (0-100):
"))
        science = float(input("Enter Science score (0-100):
"))
        for score in [math, english, science]:
            if score < 0 or score > 100:
                print_error("All scores must be between 0 and
100.")
                return
    except ValueError:
        print_error("Invalid score. Please enter a number.")
        return

    # Build the student record
    student = {
        "name": name.title(),
        "age" : age,
        "scores": {
            "Math"   : math,
            "English": english,
            "Science": science
        }
    }

    # Save
    students.append(student)
```

```
    save_students(students)
    print_success(f"Student '{name.title()}' added
successfully.")
```

The validation sequence is deliberate: name first (quickest to check), then duplicates (requires loading the full dataset), then age, then scores. If any validation fails, return immediately — the guard clause pattern from an earlier unit. Nothing is saved unless every validation passes.

> **Case-Insensitive Duplicate Checking**
> *The duplicate check — student['name'].lower() == name.lower() — means 'Amara Diallo' and 'amara diallo' are treated as the same student, preventing accidental duplicates caused only by differences in capitalisation.*

2. Feature 2 — View All Students

```
def view_all_students():
    """Display all students in a formatted, colour-coded
table."""
    print_header("ALL STUDENTS")
    students = load_students()

    if len(students) == 0:
        print_error("No students found. Please add a student
first.")
        return

    table_data = []
    for student in students:
        scores = list(student["scores"].values())
        average = get_average(scores)
        grade = get_grade(average)
        status = get_status(average)

        # Apply colour to status
        status_display = (
            Fore.GREEN + status + Style.RESET_ALL
```

```
            if status == "PASS"
            else Fore.RED + status + Style.RESET_ALL
        )

        table_data.append([
            student["name"], student["age"],
            student["scores"]["Math"],
student["scores"]["English"],
            student["scores"]["Science"], average, grade,
status_display
        ])

    headers = ["Name", "Age", "Math", "English", "Science",
"Average", "Grade", "Status"]
    print(tabulate(table_data, headers=headers,
tablefmt="fancy_grid"))
    print(f"\nTotal students: {len(students)}")
```

The function builds a list of rows, one per student, with the Status column colour-coded using colorama. tabulate's fancy_grid format produces a polished, rounded-corner grid that looks professional directly in the terminal.

3. Feature 3 — Search Student

```
def search_student():
    """Search for a student by name and display their full
profile."""
    print_header("SEARCH STUDENT")
    search_name = input("Enter student name to search:
").strip()

    if search_name == "":
        print_error("Please enter a name to search.")
        return

    students = load_students()
    found = False

    for student in students:
```

```
        if student["name"].lower() == search_name.lower():
            scores = list(student["scores"].values())
            average = get_average(scores)
            grade = get_grade(average)
            status = get_status(average)

            print(Fore.CYAN + "\nStudent Found:" +
Style.RESET_ALL)
            print("-" * 30)
            print(f"Name    : {student['name']}")
            print(f"Age     : {student['age']}")
            print(f"Math    : {student['scores']['Math']}")
            print(f"English : {student['scores']['English']}")
            print(f"Science : {student['scores']['Science']}")
            print(f"Average : {average}")
            print(f"Grade   : {grade}")

            if status == "PASS":
                print(Fore.GREEN + f"Status  : {status}" +
Style.RESET_ALL)
            else:
                print(Fore.RED + f"Status  : {status}" +
Style.RESET_ALL)

            print("-" * 30)
            found = True
            break

    if not found:
        print_error(f"No student found with the name
'{search_name}'.")
```

4. Feature 4 — Delete Student

```
def delete_student():
    """Remove a student from the system by name."""
    print_header("DELETE STUDENT")
    search_name = input("Enter student name to delete:
").strip()
```

```
    if search_name == "":
        print_error("Please enter a name.")
        return

    students = load_students()
    original_count = len(students)

    # Filter out the student to delete
    students = [s for s in students if s["name"].lower() !=
search_name.lower()]

    if len(students) == original_count:
        print_error(f"No student found with the name
'{search_name}'.")
    else:
        save_students(students)
        print_success(f"Student '{search_name.title()}'
deleted successfully.")
```

The delete function uses a list comprehension to filter out the target student, then compares the new list length to the original — if they match, nothing was removed and an error is shown; if the new list is shorter, the deletion succeeded. This is a clean, elegant deletion pattern with no separate search step or index tracking needed.

5. Feature 5 — View Class Summary

```
def view_summary():
    """Display overall class statistics."""
    print_header("CLASS SUMMARY")
    students = load_students()

    if len(students) == 0:
        print_error("No students found. Please add students
first.")
        return

    all_averages = []
```

```
    passed = 0
    failed = 0

    for student in students:
        scores = list(student["scores"].values())
        average = get_average(scores)
        all_averages.append(average)
        if average >= 50:
            passed += 1
        else:
            failed += 1

    class_average = round(sum(all_averages) /
len(all_averages), 2)
    highest = max(all_averages)
    lowest = min(all_averages)

    summary_data = [
        ["Total Students", len(students)],
        ["Passed", Fore.GREEN + str(passed) +
Style.RESET_ALL],
        ["Failed", Fore.RED + str(failed) + Style.RESET_ALL],
        ["Class Average", class_average],
        ["Highest Average", Fore.GREEN + str(highest) +
Style.RESET_ALL],
        ["Lowest Average", Fore.RED + str(lowest) +
Style.RESET_ALL],
    ]

    print(tabulate(summary_data, headers=["Metric", "Value"],
tablefmt="simple"))
    print("-" * 30)
```

What's Next

All five feature functions are now complete: add, view, search, delete, and summarise. The next lesson builds the main menu and the application's entry point, assembles the complete file, walks through a full testing sequence, and covers pushing the finished project to its dedicated GitHub repository with a proper README.