The Main Menu, Complete System, Testing & Deployment

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

This closing lesson builds the main menu and application entry point, walks through a complete testing sequence that exercises every feature, makes explicit the design principles behind every decision in the system, and covers pushing the finished project to GitHub with a professional README. This closes the entire AiCE Software Engineering Foundations Python series.

1. The Main Menu and Application Entry Point

The final two pieces are show_menu(), which displays the menu, and main(), which runs the application loop:

```
# ——— Main Menu ————————————————————————————————————————

def show_menu():
    """Display the main application menu."""
    print_header("AiCE STUDENT MANAGEMENT SYSTEM")
    print(" 1. Add Student")
    print(" 2. View All Students")
    print(" 3. Search Student")
    print(" 4. Delete Student")
    print(" 5. View Class Summary")
    print(" 6. Exit")
    print(Fore.CYAN + "=" * 50 + Style.RESET_ALL)

def main():
    """Main application loop — runs until the user exits."""
    while True:
        show_menu()
        choice = input("\nEnter your choice (1-6): ").strip()

        if choice == "1":
```

```
            add_student()
        elif choice == "2":
            view_all_students()
        elif choice == "3":
            search_student()
        elif choice == "4":
            delete_student()
        elif choice == "5":
            view_summary()
        elif choice == "6":
            print_success("Thank you for using the AiCE
Student Management System. Goodbye!")
            break
        else:
            print_error("Invalid choice. Please enter a number
between 1 and 6.")

        input("\nPress Enter to continue...")

# ——— Entry Point
————————————————————————————————————————

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

main() contains a while True loop — the program runs continuously, showing the menu and waiting for input, until the user selects option 6 and break exits the loop. This is the standard pattern behind every menu-driven application.

> **Two Small Details That Matter**
>
> *The if __name__ == '__main__': guard means main() only runs when the file is executed directly, not when imported as a module — allowing this code to be reused elsewhere if needed. And input('\nPress Enter to continue...') at the end of each loop gives the user time to read output before the menu reappears; without it, the menu would flash back immediately and could obscure what they needed to read.*

2. Running and Testing the System

Once the entire file is typed, save and quit, then run the system:

```
python3 student_system.py
```

The main menu appears in cyan. Work through this complete test sequence to verify every feature:

Test 1 — Add Students

Add at least four students with different score ranges: one with all scores above 70 (grade A, PASS), one in the 50–69 range (grade B or C, PASS), one below 50 (grade F, FAIL). Then deliberately try an empty name, a score of 150, and a duplicate name — confirm each produces the correct error.

Test 2 — View All Students

Confirm the table displays every added student with correct averages, grades, and colour-coded status (green PASS, red FAIL).

Test 3 — Search Student

Search for a student you added — confirm their full profile appears. Search for a name that does not exist — confirm the error message appears.

Test 4 — Delete Student

Delete one student, confirm the success message, then view all students again to confirm they're gone.

Test 5 — View Class Summary

Verify total students, passed, failed, class average, and highest/lowest averages all match what you entered.

Test 6 — Data Persistence

Exit the program, then run it again. View all students — every one is still there. This is the core promise of the system: data is never lost between sessions.

Inspect the raw data file directly to confirm your records:

```
cat students_data.json
```

You'll see every student record stored as neatly formatted JSON — a file readable by any text editor, any programming language, and transferable to any other system.

3. Understanding the Design Decisions

Every decision in this system reflects a principle learned across this entire program:

Principle	Where It Shows Up
Separation of concerns	Every function has exactly one responsibility — load_students() only loads, add_student() only adds
Input validation before storage	Nothing is saved until every input passes validation, cheapest checks first
Guard clauses over deep nesting	Each validation failure returns immediately, keeping the main logic unindented
Load-modify-save pattern	Every write operation loads the full dataset, modifies it in memory, then saves it back
The while True menu loop	The program serves the user until they explicitly choose to exit
Case-insensitive name matching	'Amara Diallo' and 'amara diallo' are always treated as the same student

4. Pushing to GitHub

Before pushing, generate a requirements.txt so anyone who clones the repository knows exactly what to install:

```
pip3 freeze > requirements.txt
```

Then create a README.md — the first thing anyone sees when they visit the repository:

````
# AiCE Student Management System

A complete command-line student management application built
with Python.

Developed as the culminating project of the AiCE Software
Engineering
Foundations course.

## Features
- Add students with validated name, age, and subject scores
- View all students in a formatted, colour-coded table
- Search for a student by name
- Delete a student from the system
- View class summary statistics

## Requirements
Install dependencies:
```
pip3 install -r requirements.txt
```

## Run
```
python3 student_system.py
```

## Built With
- Python 3
- colorama (terminal colours)
- tabulate (formatted tables)
- JSON (data persistence)

## Author
Built by [Your Name] as part of the AiCE Software Engineering
Foundations programme.
````

Save, then push everything:

```
git add .
git commit -m "Complete AiCE Student Management System — Python culminating project"
git push
```

Visit your GitHub repository in the browser. You'll see student_system.py, students_data.json, requirements.txt, and README.md — a complete, professional project repository.

> **Your Commit History Is Evidence**
>
> *"The code on your GitHub is evidence. It says: I showed up, I learned, and I built something. Every commit in this repository represents a decision you made, a line you typed, and a concept you applied. That history belongs to you."*
>
> *— Adesina Olagunju, Founder, AiCE*

Closing: You Have Completed the Python Series

Three books. Every unit in this program. One complete application. You started with a single print statement. You end with a working system that manages real data, validates real input, stores records between sessions, and presents information professionally in a coloured, formatted interface.

Along the way, this program covered:

The Developer Environment — terminal, Vim, Git and GitHub.

Python Beginner — variables, data types, strings, numbers, user input, conditionals, loops, lists, dictionaries.

Python Intermediate — functions, scope, modules, file handling, exception handling, list comprehensions, tuples, sets.

Python Advanced — OOP, inheritance, external libraries with pip, JSON data storage, and this capstone.

But the most important thing learned here cannot be found in any single lesson. It is the habit of sitting down, opening a terminal, and building something — consistent, patient, incremental. That habit is what separates developers who keep growing from those who plateau.

You Have Earned It

"You have earned the right to call yourself a Python developer. Not because someone gave it to you — because you built it. Now build something that matters."

— Adesina Olagunju, Founder, AiCE