

## Project Overview

Build a student gradebook management system that demonstrates sufficient understanding of all concepts covered so far.

## Project Requirements

### Part 1: Data Structures

Create a module called `student_data.py` that defines:

#### 1. Student Class with:

- Constructor accepting name, student\_id, and major
- Attributes: name, student\_id, major, grades (empty dictionary initially)
- Method `add_grade(course, score)` - adds a course grade (0-100)
- Method `get_gpa()` - calculates GPA using 4.0 scale:
  - A (90-100): 4.0
  - B (80-89): 3.0
  - C (70-79): 2.0
  - D (60-69): 1.0
  - F (0-59): 0.0
- Method `get_letter_grade(score)` - converts numeric score to letter
- Method `__str__()` - returns formatted student information
- Method `get_standing()` - returns "Good Standing" if GPA  $\geq 2.0$ , else "Academic Probation"

#### 2. GraduateStudent Class (inherits from Student) with:

- Additional attribute: thesis\_topic
- Override `get_gpa()` to weight graduate courses differently (A: 4.0, B: 3.5, C: 2.5, D: 1.0, F: 0.0)
- Method `set_thesis_topic(topic)` - sets the thesis topic

### Part 2: Gradebook Manager

Create `gradebook.py` with a Gradebook class:

### 1. Attributes:

- `students`: dictionary mapping `student_id` to `Student` objects
- `course_list`: list of available courses

### 2. Methods:

- `add_student(student)` - adds a student with exception handling for duplicate IDs
- `remove_student(student_id)` - removes a student with exception handling
- `find_student(student_id)` - uses bisection search on sorted student IDs to find a student - EXTRA CREDIT
- `calculate_class_average()` - computes average GPA of all students
- `get_students_by_standing()` - returns dictionary with "Good Standing" and "Probation" lists
- `generate_statistics()` - returns a tuple with (total\_students, average\_gpa, highest\_gpa, lowest\_gpa)

## Part 3: Main Program

Create `main.py` that provides an interactive menu system:

```
STUDENT GRADEBOOK SYSTEM
=====
1. Add Student
2. Add Graduate Student
3. Record Grade
4. View Student
5. Remove Student
6. View Top Students
7. View Class Statistics
8. View Students by Standing
9. Search for Student (Bisection Search) - EXTRA CREDIT
10. Export Data
11. Import Data
12. Grade Distribution Analysis
13. Exit
```

### Requirements for `main.py`:

1. **Menu Loop:** Use a while loop with proper control flow
2. **Input Validation:** Use try-except blocks for all user inputs

3. **String Operations:** Format all output nicely using f-strings
4. **List/Dictionary Operations:**
  - Store sample courses in a list
  - Use dictionary for quick student lookup
5. **File I/O:** Implement save/load functionality with error handling
6. **Algorithm:** Implement a bisection search to find students by ID - EXTRA CREDIT
7. **Grade Distribution Analysis:**
  - Show count of A's, B's, C's, D's, F's across all students
  - Use a loop to iterate through all grades
  - Display as a formatted table

## Part 4: Testing & Demonstration

Include a function `demo_system()` that:

1. Creates 5 sample students (mix of Student and GraduateStudent)
2. Adds grades for each student in 3-4 courses
3. Demonstrates all major functionality
4. Uses list comprehensions to generate sample data
5. Shows proper exception handling

## Tips for Success

1. **Start with classes:** Get Student and GraduateStudent working first
2. **Test incrementally:** Test each method as you write it
3. **Handle errors gracefully:** Every user input should have validation
4. **Keep functions small:** Each function should do one thing well
5. **Comment your code:** Especially the complex algorithms
6. **Use descriptive names:** Variable names should be self-documenting

## Submission Guidelines

- Three files: `student_data.py`, `gradebook.py`, `main.py`
- All code should run without errors
- Include `demo_system()` function showing all features