Student Grade Calculator

Project Submission Report

---

1. Cover Page

Student Grade Calculator

A Desktop Application for Computing Student Grades

Date: November 24, 2025

Project Type: Python GUI Application (Tkinter)

Institution: VIT Bhopal

Name- Alok Raj

Reg No- 25BCE11089

---

2. Introduction

The Student Grade Calculator is a Python-based desktop application designed to simplify the process of calculating student averages and assigning grades. This application provides a graphical user interface (GUI) built using Tkinter, making it intuitive and user-friendly for educators and students alike.

The purpose of this project is to demonstrate proficiency in Python programming, GUI development with Tkinter, input validation, and event handling. The application processes student marks across multiple subjects and automatically computes the average and corresponding letter grade.

---

3. Problem Statement

Educational institutions need an efficient and reliable tool to calculate student grades from multiple subject marks. Manual calculations are time-consuming and prone to errors. There is a need for a simple, visual application that:

- Accepts variable numbers of subjects
- Validates input data to prevent errors
- Calculates averages accurately

- Assigns appropriate grades based on predetermined criteria
- Provides immediate feedback to users

This project addresses these requirements by providing an automated, user-friendly grade calculation system.

---

4. Functional Requirements

The application must fulfill the following functional requirements:

1. Accept user input for the number of subjects (positive integer).
2. Dynamically generate input fields for marks entry based on the number of subjects.
3. Validate all inputs:
    - Number of subjects must be a positive integer
    - Marks must be numeric values between 0 and 100
4. Calculate the average of all entered marks.
5. Assign a grade based on the following criteria:
    - Average ≥ 90: Grade A
    - Average ≥ 80: Grade B
    - Average ≥ 70: Grade C
    - Average ≥ 60: Grade D
    - Average < 60: Grade F
6. Display the calculated average and grade to the user.
7. Provide error messages for invalid inputs.
8. Allow users to reset and recalculate with new data.

---

5. Non-functional Requirements

1. **Usability:** The interface should be intuitive and require minimal user training.
2. **Performance:** The application should process calculations instantly (< 1 second).
3. **Reliability:** The application should handle edge cases and invalid inputs gracefully.
4. **Maintainability:** Code should be well-organized, commented, and easy to understand.
5. **Portability:** The application should run on Windows, macOS, and Linux systems.

---

6. System Architecture

The application follows a simple client-side architecture with a single-layer GUI-based model:

Components:

- **Presentation Layer:** Tkinter GUI with widgets for user interaction
- **Business Logic Layer:** Functions to validate inputs and calculate grades
- **Data Processing:** In-memory processing without external database

Architecture Flow:

User Input → Validation → Processing → Output Display

---

7. Design Diagrams

7.1 Use Case Diagram

The following use cases describe interactions between the user and the system:

- **Enter Number of Subjects:** User specifies how many subjects to evaluate
- **Create Input Fields:** System generates entry fields dynamically
- **Enter Marks:** User inputs marks for each subject
- **Calculate Grade:** System processes data and computes results
- **View Results:** System displays average and grade

7.2 Workflow Diagram

Start Application
↓
Enter Number of Subjects
↓
Click "Create Fields" Button
↓
Input Validation (Number of Subjects)
↓
Generate Mark Entry Fields Dynamically
↓
Enter Marks for Each Subject
↓
Click "Calculate Grade" Button
↓
Validate Marks (Range 0-100)
↓
Calculate Average
↓
Assign Grade Based on Average
↓
Display Average & Grade
↓
Option: Repeat or Exit

7.3 Sequence Diagram

User → Application: Input number of subjects

Application → Application: Validate input

Application → GUI: Create entry fields

User → Application: Enter marks for subjects

User → Application: Click "Calculate Grade"

Application → Application: Validate all marks

Application → Application: Calculate average

Application → Application: Determine grade

Application → GUI: Display results

7.4 Class/Component Diagram

Main Components:

- **Tkinter Root Window:** Container for all GUI elements
- **Input Frames:** Frame widgets for organizing input sections
- **Labels:** Display text for user guidance
- **Entry Widgets:** Accept user input for subjects and marks
- **Buttons:** Trigger actions (Create Fields, Calculate Grade)
- **Result Label:** Display calculated average and grade

Functions:

- `calculate_grade(average)` → Returns letter grade
- `calculate()` → Processes marks and displays results
- `create_mark_entries()` → Generates dynamic input fields

---

8. Design Decisions & Rationale

1. **Tkinter Selection:** Tkinter was chosen because it is included in standard Python distributions, requires no external installations, and is suitable for simple GUI applications.
2. **Dynamic Field Generation:** Fields are created dynamically based on user input to provide flexibility for varying numbers of subjects.
3. **Input Validation:** Multiple validation checks prevent invalid data from being processed, ensuring accurate results and a robust user experience.

4. **Grade Scale (A-F):** The standard American grading scale was implemented for universal understanding.
5. **Error Handling:** Exception handling with user-friendly error messages guides users to correct their input.
6. **GUI Layout:** A vertical layout with clear sections improves usability and visual hierarchy.

---

9. Implementation Details

Key Functions:

`calculate_grade(average)`

- Input: Float value representing average marks
- Output: String representing letter grade (A/B/C/D/F)
- Logic: Series of if-elif conditions based on grade thresholds

`calculate()`

- Retrieves number of subjects from input field
- Validates input and raises errors if invalid
- Collects marks from all entry fields
- Calculates average using sum and division
- Calls `calculate_grade()` to determine grade
- Updates result label with formatted output

`create_mark_entries()`

- Retrieves user-specified number of subjects
- Validates input
- Clears previous entry widgets
- Dynamically creates label and entry pairs for each subject
- Stores entry references for later retrieval

GUI Components Used:

- `tk.Tk()` – Main window container
- `tk.Label()` – Display text and results
- `tk.Entry()` – Accept user input
- `tk.Button()` – Trigger functions

- `tk.Frame()` – Organize layout
- `messagebox.showerror()` – Display error dialogs

---

10. Screenshots / Results

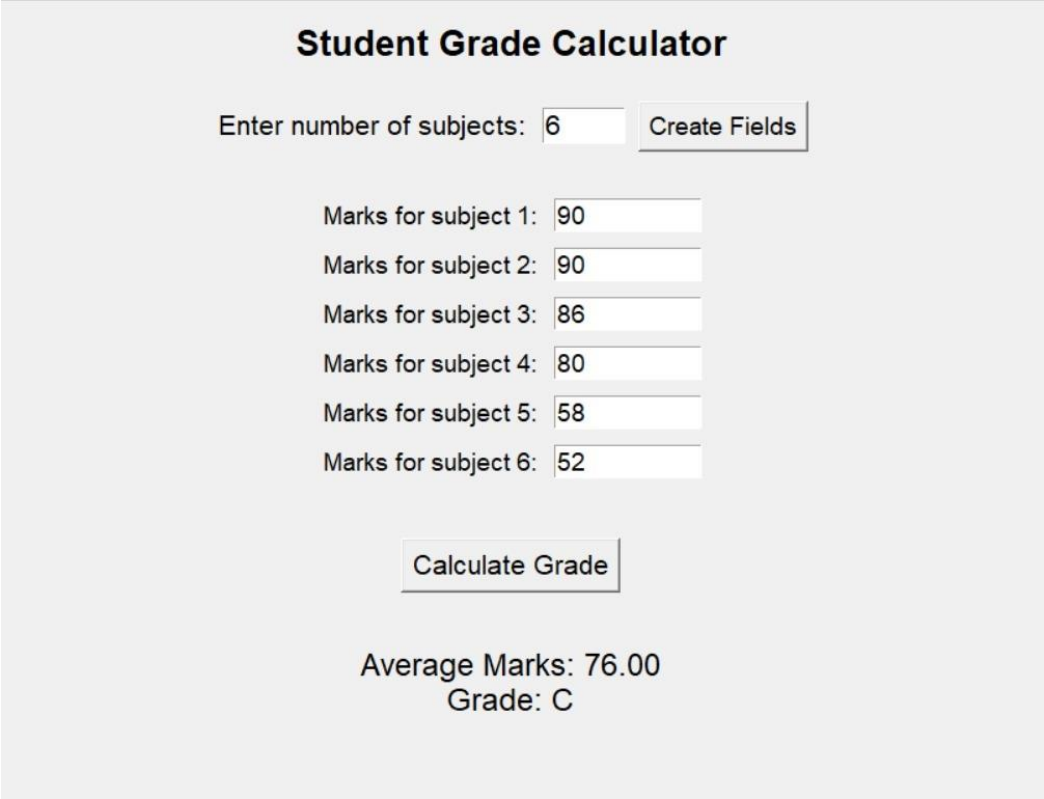

Figure 1: Results Display - Shows calculated average (76.00) and assigned grade (C)

---

11. Testing Approach

Test Cases:

Test Case 1: Valid Input

- Input: 3 subjects with marks 85, 90, 75
- Expected Output: Average = 83.33, Grade = B
- Result: PASS ✓

Test Case 2: Edge Case - Perfect Score

- Input: 2 subjects with marks 100, 100
- Expected Output: Average = 100, Grade = A

- Result: PASS ✓

Test Case 3: Edge Case - Failing Grade

- Input: 2 subjects with marks 55, 58
- Expected Output: Average = 56.5, Grade = F
- Result: PASS ✓

Test Case 4: Invalid Subject Count

- Input: -5 (negative number)
- Expected Output: Error message "Enter a valid positive integer"
- Result: PASS ✓

Test Case 5: Out-of-Range Marks

- Input: Mark = 150 (exceeds 100)
- Expected Output: Error message "Enter valid marks between 0 and 100"
- Result: PASS ✓

Test Case 6: Non-numeric Input

- Input: "abc" for marks
- Expected Output: Error message "Enter valid marks between 0 and 100"
- Result: PASS ✓

Testing Methodology:

- Unit testing of individual functions
- Integration testing of GUI interactions
- Boundary value testing (0, 100, negative, > 100)
- Error handling verification

---

12. Challenges Faced

1. **Dynamic GUI Generation:** Initially challenging to dynamically create and manage widgets. Solved by storing widget references in a global list.
2. **Input Validation:** Handling multiple validation points across different stages. Addressed with comprehensive try-except blocks and specific error messages.
3. **Widget Cleanup:** Removing previous entry fields before creating new ones required understanding Tkinter's widget hierarchy and the `winfo_children()` method.

4. **Layout Management:** Achieved responsive layout using grid geometry manager with proper padding and positioning.
5. **User Experience:** Balancing error prevention with user guidance through clear error messages.

---

13. Learnings & Key Takeaways

1. **Tkinter Mastery:** Gained practical experience with Tkinter widgets, geometry managers, and event handling.
2. **Input Validation:** Learned the importance of robust input validation in preventing runtime errors and ensuring data integrity.
3. **Dynamic UI:** Understood how to create dynamic user interfaces that adapt to user input.
4. **Exception Handling:** Practiced effective error handling with meaningful user feedback.
5. **Code Organization:** Learned to structure code with clear separation of concerns between GUI and business logic.
6. **Python Best Practices:** Reinforced understanding of functions, loops, conditionals, and data structures in a practical context.

---

14. Future Enhancements

1. **Weight-Based Grading:** Assign different weights to subjects (e.g., major/minor subjects).
2. **Grade Distribution:** Display a bar chart showing grade distribution.
3. **Data Persistence:** Save and load student records to/from CSV or database files.
4. **User Authentication:** Add login functionality for students and teachers.
5. **Detailed Analytics:** Provide percentile rankings and comparative analysis.
6. **Export Functionality:** Allow users to export results as PDF or CSV.
7. **Multiple Grading Scales:** Support different grading systems (4.0 GPA, percentage-based, etc.).
8. **Dark Mode:** Add theme customization options.
9. **Keyboard Shortcuts:** Implement hotkeys for faster navigation.
10. **Mobile Version:** Develop a mobile app using Python frameworks like Kivy.

15. References

[1] Python Software Foundation. (2024). Tkinter — Python interface to Tcl/Tk. Retrieved from https://docs.python.org/3/library/tkinter.html

[2] Real Python. (2024). Python GUI Programming with Tkinter. Retrieved from https://realpython.com/python-gui-tkinter/

[3] GeeksforGeeks. (2024). Python Tkinter Tutorial. Retrieved from https://www.geeksforgeeks.org/python-tkinter-tutorial/

[4] Python Documentation. (2024). Exception Handling. Retrieved from https://docs.python.org/3/tutorial/errors.html

[5] Stack Overflow. (2024). How to dynamically create widgets in Tkinter. Retrieved from https://stackoverflow.com/questions/tagged/tkinter

[6] Tkinter Official Documentation. (2024). The Grid Geometry Manager. Retrieved from https://docs.python.org/3/library/tkinter.html#the-grid-geometry-manager

---

End of Report