A MINI PROJECT REPORT
ON
STUDENT GRADE CHECKER

submitted in partial fulfillment of requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

S.V.S.N. TEJA SWARUP	**(229Y1A05G8)**
T. HARSHA VARDHAN	**(229Y1A05H5)**
T. AMRUTHA	**(229Y1A05H6)**
V. NAGA DIVYA	**(229Y1A05I8)**
K. SUNIL KUMAR	**(239Y5A0507)**

under the Esteemed Supervision of

Smt. O.V. Sowmya, MTech., (Ph.D).

Assistant Professor.,

Dept of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

K.S.R.M. COLLEGE OF ENGINEERING

(UGC-Autonomous)

Approved by AICTE , New Delhi and affiliated to JNTUA, Anantapuramu

Accredited by NAAC with A+Grade &B.tech (EEE,ECE,CSE,CE and ME) Programs by NBA

Kadapa, Andhra Pradesh, India– 516 003

2024-2025

K.S.R.M. COLLEGE OF ENGINEERING

(UGC-Autonomous)
Approved by AICTE , New Delhi and affiliated to JNTUA, Anantapuramu
Accredited by NAAC with A+Grade &B.tech (EEE,ECE,CSE,CE and ME) Programs by NBA
Kadapa, Andhra Pradesh, India– 516 003

VISION

To evolve as center of repute for providing quality academic programs amalgamated with creative learning and research excellence to produce graduates with leadership qualities, ethical and human values to serve the nation.

MISSION

M1: To provide high quality education with enriched curriculum blended with impactful teaching-learning practices.

M2: To promote research, entrepreneurship and innovation through industry collaborations.

M3: To produce highly competent professional leaders for contributing to Socio-economic development of region and the nation.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION

To evolve as a recognized center of excellence in the area of Computer Science and Engineering and other related inter-disciplinary fields.

MISSION

M1: To produce competent and industry ready professionals through well balanced curriculum and innovative pedagogy.

M2: To provide conducive environment for research by establishing centre of excellence and industry collaborations.

M3: To install leadership qualities, ethical values among students through various co-curricular and extracurricular activities.

B.Tech. (Computer Science And Engineering)

PROGRAM EDUCATIONAL OBJECTIVES

B. Tech-Computer Science and Engineering Program Objectives.

A graduate of the K.S.R.M.C.E, C.S.E should have a successful career in CSE or a related field, and within three to five years, should

PEO1: To excel in their career as competent software engineer in IT and allied organizations.

PEO2: To pursue higher education and to demonstrate research temper for providing solutions to engineering problems.

PEO3: To contribute for the societal development by exhibiting leadership, through professional, social and ethical values.

PROGRAM OUTCOMES

PO1 - Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2 - Problem Analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3 - Design/Development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 - Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 - Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6 - The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7 - Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8 - Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.

PO9 - Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 - Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11 - Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12 - Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

PSOs are the statements that describes what the graduates of a specific engineering program should be able to:

PSO1-Professional Skills: The ability to understand, analyse and develop the computer programs in areas related to algorithms, system software, multimedia, web design, big data analysis and networking for efficient design of computer-based systems of varying complexity.

PSO2-Problem Solving Skills: The ability to apply standard practices and strategies in software project development using open – ended programming environments to deliver a quality product or business success

PSO3 – Successful career and Entrepreneurship: The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and Zest for higher studies

COURSE OUTCOMES

CO 1: Design algorithms using appropriate design techniques (divide and conquer, greedy, dynamic programming, etc.,).

CO 2: Implement variety of algorithms such as sorting, searching, graph related, etc., in a high level language.

CO 3: Analyze and compare the performance of algorithms using language features.

CO-PO MAPPING

CourseOutcome	Program Outcomes												Program Specific Outcomes		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12	PSO1	PSO2	PSO3
CO1	3	2	3		2	1			2	2			3	2	2
CO2	2	2	3		3					2	2		3	3	2
CO3		3		3	3				2		2	2	2	3	2

K.S.R.M. COLLEGE OF ENGINEERING

(UGC-Autonomous)
Approved by AICTE , New Delhi and affiliated to JNTUA, Anantapuramu
Accredited by NAAC with A+Grade &B.tech (EEE,ECE,CSE,CE and ME) Programs by NBA
Kadapa, Andhra Pradesh, India– 516 003

CERTIFICATE

This is to certify that the Design and Analysis of Algorithms Mini project entitled

STUDENT GRADE CHECKER
is the bonafide work done & submitted by

S.V.S.N. TEJA SWARUP	**(229Y1A05G8)**
T. HARSHA VARDHAN	**(229Y1A05H5)**
T. AMRUTHA	**(229Y1A05H6)**
V. NAGA DIVYA	**(229Y1A05I8)**
K. SUNIL KUMAR	**(239Y5A0507)**

in the Department of Computer Science and Engineering,

K.S.R.M.C.E, Kadapa and is submitted to **Jawaharlal Nehru Technological University Anantapur,** in partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering during 2022-2026.

Supervisor	**Internal Examiner**	**Head of the Department**
Smt. O.V. Sowmya		**A. RamPrakash Reddy**
M.Tech,(Ph.D).		**M.Tech,(Ph.D).**
Assistant Professor		**Assistant Professor & HOD**
Dept of CSE.		**Dept of CSE.**

DECLARATION

We hereby declare that this Mini Project report titled "**STUDENT GRADE CHECKER** " is a genuine project work carried out by us, in B. Tech (**Computer Science and Engineering**) degree course of Jawaharlal Nehru Technological University Anantapur and has not been submitted to any other course or University for the award of any degree by us.

Signature of the Student

S.V.S.N. TEJA SWARUP

T. HARSHA VARDHAN

T. AMRUTHA

V. NAGA DIVYA

K. SUNIL KUMAR

ACKNOWLEDGEMENTS

An endeavor over a long period can be successful only with the advice and supports of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We are extremely thankful to our beloved Managing Director **Dr. K. Chandra Obul Reddy** Garu who took keen interest and encouraged us in every effort throughout this course. We are deeply indebted to the supervisor, **Smt. O.V. Sowmya**, M.Tech.,(Ph.D)., Assistant Professor, Department of Computer Science and Engineering for valuable guidance, constructive criticism and keen interest evinced throughout the course of our DAA project work. We are really fortunate to associate ourselves with such an advising and helping guide in every possible way, at all stages, for the successful completion of this work.

We express our deep sense of gratitude to **A.RamPrakash Reddy**, M.Tech., (Ph.D)., Assistant Professor and Head of Department of Computer Science and Engineering forhis valuable guidance and constant encouragement given to us during this DAA projectand the course. We take this opportunity to express our deep gratitude and appreciation toall those who encouraged us for successfully completion of this Community Service Project work. We wish to express our sincere to gratitude to **Dr. T. Nageswara Prasad**, M.Tech., Ph.D. Vice Principal of K.S.R.M.C.E, Kadapa and **Dr. V.S.S. Murthy**, M.Tech., Ph.D. Principal of K.S.R.M.C.E, Kadapa and for their consistent help and encouragement to complete the DAA project work. We are pleasedto express our heart full thanks to our faculty in Department of CSE of KSRMCE for their moral support and good wishes. Finally,we have a notation to express our sincere thanks to friends and all those who guided, inspired and helped us in the completion of our project work.

S.V.S.N. TEJA SWARUP (229Y1A05G8)

T. HARSHA VARDHAN (229Y1A05H5)

T. AMRUTHA (229Y1A05H6)

V. NAGA DIVYA (229Y1A05I8)

K. SUNIL KUMAR (239Y5A0507)

INDEX

List Of Figures

ABSTRACT

The Student Grade Checker is a user-friendly software solution designed to manage, calculate, and store academic records of students. The system allows for the input and calculation of important academic metrics such as SGPA (Semester Grade Point Average) and CGPA (Cumulative Grade Point Average) based on marks obtained in various subjects across multiple semesters. Additionally, the application supports the storage and retrieval of student data in a text file for future reference. An admin interface allows authorized users to view and delete records, ensuring smooth and efficient management of student information. The system uses a graphical user interface (GUI) developed with Tkinter, providing an intuitive and interactive experience for both students and administrators. The solution aims to streamline the process of managing student academic data while ensuring the accuracy and security of records.

The system also includes features such as automatic grade calculation based on predefined thresholds for marks, making it easier for students and administrators to track academic performance without manual intervention. It supports flexible handling of data across multiple semesters, allowing students to update their records as they progress through their academic journey. The integration of file management ensures that all student data is persistently stored, easily retrievable, and secure, while also allowing for quick data updates by administrators. By providing a simple yet powerful tool for managing academic records, the system contributes to reducing administrative workload and enhances the overall efficiency of academic data handling in educational institutions.

1. INTRODUCTION

The **Student Grade Checker** proves to be a comprehensive and efficient tool for managing, calculating, and storing student academic records. By integrating functionalities such as SGPA and CGPA calculation, grade assignment, and secure data management, the system significantly enhances the way student data is handled in educational institutions. The system's design ensures that both students and administrators can easily interact with the application through a clean, user-friendly graphical interface built using Python's Tkinter library. This interface facilitates seamless input of student details, such as marks and semester information, and automatically processes them to calculate essential academic metrics.

A standout feature of the system is its ability to calculate SGPA and CGPA based on predefined grade point thresholds, which reduces the complexity and errors often associated with manual calculations. By offering an automated method of calculating and displaying grades and academic performance, the system provides students with an accurate reflection of their academic journey. Additionally, the grades are determined based on a simple set of criteria, making it easy to adjust and scale the grading system if needed, depending on institutional requirements.

Another key component of the system is the file management functionality. The system stores student records in a text file, ensuring data persistence and enabling the retrieval of records whenever necessary. The ability to store multiple semesters of data in a well-organized manner allows students to keep track of their progress over time. The search feature, based on roll numbers, allows students and administrators to easily find specific records. Furthermore, the inclusion of an admin interface with a secure login mechanism adds a layer of protection, ensuring that only authorized personnel can make changes to the records. Admins can view, update, and delete records, maintaining data integrity and ensuring that outdated or incorrect information is properly handled.

The system’s ability to store, manage, and retrieve student data effectively minimizes the administrative workload and significantly reduces the possibility of errors in record keeping.

With the option to delete outdated or erroneous data and update student information when necessary, the system offers flexibility and scalability, which are essential in educational institutions where student data is continuously updated. The automated grade and CGPA calculations ensure that the system is not only accurate but also highly efficient in providing the required academic metrics.

In addition, the software is designed to be adaptable to various academic structures. Whether handling a small number of students or large batches, the system can easily accommodate more data by simply extending the records. This makes it a versatile solution that can grow with the needs of an institution. The simplicity of the Tkinter interface ensures that both students and administrators can interact with the system without requiring advanced technical skills.

Overall, the **Student Grade Checker** delivers a streamlined, secure, and accurate method for managing student records. The combination of grade calculation automation, secure file management, and an easy-to-use interface makes it an indispensable tool for educational institutions. It not only improves efficiency and accuracy but also significantly reduces the burden of manual data handling. With the added benefit of an admin interface for secure data management, the system ensures that student records remain up-to-date and protected. By modernizing the way student academic data is processed and stored, this system contributes to a more organized, transparent, and effective academic environment.

2. MODULES

The **Main Module** of the above code is responsible for initializing the entire application by creating the main window and starting the Tkinter event loop. This module serves as the entry point for the application, bringing together the different modules (UI, file management, admin interface, etc.) and managing the flow of execution.

2.1.Main Module

Here's the explanation and code for the **Main Module**:

- The **main module** initializes the root Tkinter window and instantiates the `StudentDataApp` class, which is the core of the application.
- It then enters the Tkinter event loop (`root.mainloop()`), where it waits for user interactions such as button clicks or other actions.
- The app is structured in such a way that various components like the UI, admin interface, and data management are handled by separate methods inside the `StudentDataApp` class, but everything is triggered from this central module.

2.2. User Interface (UI) Module:

This module is responsible for creating and managing the graphical interface that users interact with. It handles the presentation of the application and provides input/output dialogs for the user.

- **Key Components**:
    - Main Application Window**:** This is the primary window of the app where buttons, labels, and inputs are displayed.
    - Buttons**:** Buttons for different functionalities like calculating student data, viewing previous data, admin login, etc.
    - Dialogs**:** Simple dialogs for receiving user input (name, roll number, marks, etc.) and displaying message boxes (error, info, etc.).

- **Main Elements**:
  - tk.Label: Displays labels (e.g., "Student Data Management").
  - tk.Button: Creates buttons for different actions like "Calculate Student Data", "Check Previous Data", etc.
  - tk.simpledialog: Used to ask the user for inputs (e.g., name, roll number, number of semesters).
  - tk.messagebox: Used to show error, info, or result messages.
- **Example Functionality**:
  - Calculate Student Data**:** Collects data for calculating SGPA, CGPA, and grades.
  - Check Previous Data: Allows users to search for and view previously stored student data.
  - Admin Login**:** Provides an interface for entering the admin password and accessing administrative functions.

2.3. Admin Interface Module:

This module handles the functionalities available to the administrator of the system. It is accessible after a successful admin login and includes actions like viewing and deleting student data.

- **Key Components**:
  - Admin Login**:** Secured login with a password to ensure only authorized access.
  - View All Data**:** Allows the admin to view all student records stored in the file.
  - Delete Student Data**:** Provides the ability to delete student records based on the roll number.
- **Example Functionality**:
  - Admin Login**:** Verifies the entered password (e.g., "KSRMADMIN").
  - View All Student Data: Displays all the saved student data records.
  - Delete Student Data**:** Deletes a student record based on the provided roll number.

2.4. File Management Module:

This module handles the reading and writing of student data from and to a file (in this case, a text file). It allows the application to store student records persistently and load them when needed.

- **Key Components**:
    - Save Data**:** This function writes the student data to a file in a structured format (e.g., CSV).
    - Load Data**:** This function reads the data from the file and returns it in a usable format (list of dictionaries).
    - File Existence Check**:** Ensures that the file exists before attempting to read data.
- **Example Functionality**:
    - Save Data**:** Appends the student information (name, roll number, SGPA, CGPA, etc.) to the `student_data.txt` file.
    - Load Data**:** Reads the file to retrieve saved student data for display or searching.

2.5. Data Calculation Module:

This module is responsible for the calculation of SGPA, CGPA, and grades based on the input marks.

- **Key Components**:
    - SGPA Calculation**:** Based on the marks of individual subjects in a semester, this function calculates the SGPA (Semester Grade Point Average).
    - CGPA Calculation**:** This function averages the SGPA values across all semesters to calculate the CGPA (Cumulative Grade Point Average).
    - Grade Assignment**:** Assigns a grade letter (S, A, B, C, etc.) based on the marks obtained in each subject.
- **Example Functionality**:
    - SGPA Calculation**:** Converts marks into grade points and calculates SGPA.
    - CGPA Calculation**:** Averages the SGPA values from all semesters to compute the CGPA.

- Grade Calculation: Based on a given mark, the function assigns the appropriate grade letter.

2.6. Sorting Algorithms:

Usage: A quick sort algorithm is implemented (‘quick_sort(data,key)’) to sort student records based on a specified key (e.g., roll number or total marks).

Functionality: Sorting algorithms are essential for organizing data, making it easier to view and manage records, especially in the admin interface where all student data can be viewed.

2.7. Divide and Conquer:

Definition:

Divide and conquer is an algorithm design paradigm that involves breaking a problem into smaller, more manageable subproblems, solving each subproblem independently, and then combining the solutions to solve the original problem.

Quick Sort Function:

The quick_sort function exemplifies the divide-and-conquer approach. Here's how it works:

Divide: The function selects a pivot element from the list. It then partitions the list into three sublists:

- Elements less than the pivot.
- Elements equal to the pivot.
- Elements greater than the pivot.

Conquer: The function recursively applies the same sorting logic to the left and right sublists.

Combine: Finally, it concatenates the sorted left sublist, the middle sub list, and the sorted right sublist to produce a fully sorted list.

Benefits: The divide-and-conquer method allows for efficient sorting of data. The average time complexity of quick sort is (O(n \log n)), making it suitable for large datasets.

2.8. Greedy Algorithms:

Definition:

Greedy algorithms build up a solution piece by piece, always choosing the next piece that offers the most immediate benefit (i.e., the locally optimal choice). They do not reconsider their choices, which can lead to a globally optimal solution in some cases.

Grading System: The get_grade function assigns grades based on marks using a series of conditional statements. This can be seen as a greedy approach: Each mark is evaluated independently to assign the highest possible grade point. For example, if a student scores 85, they immediately receive a grade of 'A' without considering other factors.

Benefits: Greedy algorithms are typically easier to implement and can be very efficient. However, they do not always yield the optimal solution for all problems, so their application needs to be carefully evaluated.

2.9. Dynamic Programming:

Definition:

Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems and storing the results of these subproblems to avoid redundant calculations. This approach is particularly useful for optimization problems.

Application in Code:

While dynamic programming is not explicitly used in the provided code, it is worth noting how it could be applied: If the application required more complex calculations (e.g., determining the best possible SGPA or CGPA across multiple semesters with constraints), dynamic programming could be used to store intermediate results and make the process more efficient.

2.10. Time and Space Complexity:

2.10.1 Time Complexity:

The overall time complexity of the application can be summarized as:

- O(S * T) for calculate_student_data (where S is the number of semesters and T is the average number of subjects).
- O(N) for check_previous_data, view_all_data, and delete_student_data.
- The most significant term will dominate, so if S and T are relatively small compared to N, the overall complexity can be approximated as O(N).

2.10.2 Space Complexity:

The overall space complexity can be summarized as:

- O(S * T) for calculate_student_data (for storing marks and grades).
- O(N) for check_previous_data, view_all_data, and delete_student_data.
- The dominant term will depend on the values of S, T, and N, but in general, it can be approximated as O(N) if N is significantly larger than S * T.

In conclusion, the overall time complexity is O(N) (or O(S * T) if S and T are large), and the overall space complexity is O(N) (or O(S * T) if storing marks and grades is significant).

3. System Requirements

Hardware Requirements:

The following hardware specifications are recommended to run the Student Grade Checker application effectively:

- ✓ Processor: Intel Core i3 or equivalent
- ✓ RAM: Minimum 4 GB (8 GB recommended)
- ✓ Storage: At least 100 MB of free disk space for application files and data storage
- ✓ Display: 1024 x 768 resolution or higher
- ✓ Input Device: Keyboard and Mouse

Software Requirements:

The application is developed using Python and requires the following software components:

- ✓ Operating System: Windows 10 or later, macOS Mojave (10.14) or later, Linux (Ubuntu 18.04 or later)
- ✓ Python Version: Python 3.6 or later
- ✓ Required Libraries:
  tkinter (for GUI development; included with standard Python installations)
  os (for file handling; included with standard Python installations)
- ✓ TEditor/IDE: Any text editor or Integrated Development Environment (IDE) that supports Python (e.g., PyCharm, Visual Studio Code, or IDLE)

Additional Requirements:

- ✓ File System Access: The application requires permission to read from and write to the local file system to store and retrieve student data.
- ✓ Internet Connection: Not required for the basic functionality of the application, but may be needed for downloading Python and other necessary packages.

4.SOURCE CODE

```
import os
import tkinter as tk
from tkinter import messagebox, simpledialog

# Function to calculate SGPA
def calculate_sgpa(marks, total_subjects):
    grade_points = []
    for mark in marks:
        if mark >= 90:
            grade_points.append(10)
        elif mark >= 80:
            grade_points.append(9)
        elif mark >= 70:
            grade_points.append(8)
        elif mark >= 60:
            grade_points.append(7)
        elif mark >= 50:
            grade_points.append(6)
        elif mark >= 40:
            grade_points.append(5)
        else:
            grade_points.append(0)

    total_grade_points = sum(grade_points)
    sgpa = total_grade_points / total_subjects if total_subjects > 0 else 0
    return sum(marks), sgpa

# Function to calculate CGPA
def calculate_cgpa(sgpas):
    total_sgpa = sum(sgpas)
    cgpa = total_sgpa / len(sgpas) if sgpas else 0
```

```
    return cgpa

# Function to get grade based on marks
def get_grade(mark):
    if mark >= 90:
        return 'S'
    elif mark >= 80:
        return 'A'
    elif mark >= 70:
        return 'B'
    elif mark >= 60:
        return 'C'
    elif mark >= 50:
        return 'D'
    elif mark >= 35:
        return 'E'
    else:
        return 'F'

# Function to save data to a text file
def save_data(data, filename='student_data.txt'):
    with open(filename, 'a') as text_file:
        text_file.write(','.join(map(str, data.values())) + '\n')

# Function to load previous data from a text file
def load_data(filename='student_data.txt'):
    if not os.path.exists(filename):
        return []

    with open(filename, 'r') as text_file:
        data = []
        for line in text_file:
```

```
            fields = line.strip().split(',')
            if len(fields) < 7:  # Ensure there are enough fields
                continue  # Skip this line if it doesn't have enough fields
            try:
                data.append({
                    'name': fields[0],
                    'roll_number': fields[1],
                    'branch': fields[2],
                    'total_marks': float(fields[3]),
                    'sgpa': fields[4].strip('[]'),  # Store SGPA as a string representation of a list
                    'cgpa': float(fields[5]),
                    'grades': fields[6].split(';')
                })
            except ValueError:
                continue  # Skip this line if there's a ValueError
    return data

# Quick Sort function (sorts in descending order)
def quick_sort(data, key):
    if len(data) <= 1:
        return data
    pivot = data[len(data) // 2][key]
    left = [x for x in data if x[key] > pivot]  # Change to '>' for descending order
    middle = [x for x in data if x[key] == pivot]
    right = [x for x in data if x[key] < pivot]  # Change to '<' for descending order
    return quick_sort(left, key) + middle + quick_sort(right, key)

# Linear Search function
def linear_search(data, roll_number):
    for entry in data:
        if entry['roll_number'] == roll_number:
            return entry
```

```
    return None

class StudentDataApp:
    def __init__(self, master):
        self.master = master
        master.title("Student Grade Checker")

        # Create UI components
        self.label = tk.Label(master, text="Student Grade Checker")
        self.label.pack()

        self.calculate_button = tk.Button(master, text="Calculate Student Data", command=self.calculate_student_data)
        self.calculate_button.pack()

        self.check_button = tk.Button(master, text="Check Previous Data", command=self.check_previous_data)
        self.check_button.pack()

        self.admin_button = tk.Button(master, text="Admin Login", command=self.admin_login)
        self.admin_button.pack()

        self.exit_button = tk.Button(master, text="Exit", command=master.quit)
        self.exit_button.pack()

    def calculate_student_data(self):
        name = simpledialog.askstring("Input", "Enter student's name:")
        roll_number = simpledialog.askstring("Input", "Enter roll number (10 characters):")
        branch = simpledialog.askstring("Input", "Enter branch (CSE, ECE, ME, EEE, AIML, CE):").upper()
```

```
    # Validate roll number length
    if len(roll_number) != 10:
        messagebox.showerror("Error", "Roll number must be exactly 10 characters long.")
        return

    # Validate branch
    valid_branches = ["CSE", "ECE", "ME", "EEE", "AIML", "CE"]
    if branch not in valid_branches:
        messagebox.showerror("Error", "Invalid branch. Please enter a valid branch.")
        return

    # Input number of semesters
    total_semesters = simpledialog.askinteger("Input", "Enter the number of semesters:")
    if total_semesters <= 0:
        messagebox.showerror("Error", "Number of semesters must be a positive integer.")
        return

    all_sgpas = []
    for semester in range(total_semesters):
        total_subjects = simpledialog.askinteger("Input", f"Enter the number of subjects for semester {semester + 1}:")
        if total_subjects <= 0:
            messagebox.showerror("Error", "Number of subjects must be a positive integer.")
            return

        marks = []
        grades = []

        for i in range(total_subjects):
            mark = simpledialog.askfloat("Input", f"Enter marks for subject {i + 1} (0-100):")
            if mark is not None and 0 <= mark <= 100:
                marks.append(mark)
```

```
                grades.append(get_grade(mark))
            else:
                messagebox.showerror("Error", "Marks must be between 0 and 100.")
                return

        total_marks, sgpa = calculate_sgpa(marks, total_subjects)
        all_sgpas.append(sgpa)

    cgpa = calculate_cgpa(all_sgpas)

    student_data = {
        'name': name,
        'roll_number': roll_number,
        'branch': branch,
        'total_marks': sum(marks),
        'sgpa': str(all_sgpas),  # Store SGPA as a string representation of a list
        'cgpa': cgpa,
        'grades': ';'.join(grades)
    }

    save_data(student_data)

    messagebox.showinfo("Results", f"Student Name: {name}\nRoll Number: {roll_number}\nBranch: {branch}\nTotal Marks: {sum(marks):.2f}\nSGPA: {all_sgpas}\nCGPA: {cgpa:.2f}")

  def check_previous_data(self):
    previous_data = load_data()
    if not previous_data:
        messagebox.showinfo("Info", "No previous data found.")
        return
```

```
        search_roll_number = simpledialog.askstring("Input", "Enter roll number to search for your data:")
        found_entry = linear_search(previous_data, search_roll_number)

        if found_entry:
            messagebox.showinfo("Your Record", f"Student Name: {found_entry['name']}\nRoll Number: {found_entry['roll_number']}\nBranch: {found_entry['branch']}\nTotal Marks: {found_entry['total_marks']:.2f}\nSGPA: {found_entry['sgpa']}\nCGPA: {found_entry['cgpa']:.2f}")
        else:
            messagebox.showinfo("Info", "No student found with that roll number.")

    def admin_login(self):
        password = simpledialog.askstring("Admin Login", "Enter admin password:")
        if password == "KSRMADMIN":
            self.admin_interface()
        else:
            messagebox.showerror("Error", "Incorrect password.")

    def admin_interface(self):
        admin_window = tk.Toplevel(self.master)
        admin_window.title("Admin Interface")

        self.view_data_button = tk.Button(admin_window, text="View All Student Data", command=self.view_all_data)
        self.view_data_button.pack()

        self.delete_data_button = tk.Button(admin_window, text="Delete Student Data", command=self.delete_student_data)
        self.delete_data_button.pack()
```

```
        self.exit_admin_button = tk.Button(admin_window, text="Exit Admin", command=admin_window.destroy)
        self.exit_admin_button.pack()

    def view_all_data(self):
        all_data = load_data()
        if not all_data:
            messagebox.showinfo("Info", "No student data available.")
            return

        sorted_data = quick_sort(all_data, 'total_marks')  # Sort data by total marks
        results = ""
        for entry in sorted_data:
            results += f"Student Name: {entry['name']}\nRoll Number: {entry['roll_number']}\nBranch: {entry['branch']}\nTotal Marks: {entry['total_marks']:.2f}\nSGPA: {entry['sgpa']}\nCGPA: {entry['cgpa']:.2f}\n\n"
        messagebox.showinfo("All Student Records", results)

    def delete_student_data(self):
        roll_number = simpledialog.askstring("Delete Student Data", "Enter roll number of the student to delete:")
        all_data = load_data()
        updated_data = [entry for entry in all_data if entry['roll_number'] != roll_number]

        if len(updated_data) < len(all_data):
            with open('student_data.txt', 'w') as text_file:
                for entry in updated_data:
                    text_file.write(','.join(map(str, entry.values())) + '\n')
            messagebox.showinfo("Success", "Student data deleted successfully.")
        else:
            messagebox.showinfo("Info", "No student found with that roll number.")
```

```
def main():
    root = tk.Tk()
    app = StudentDataApp(root)
    root.mainloop()

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

5. INPUT AND OUTPUT SCREENS

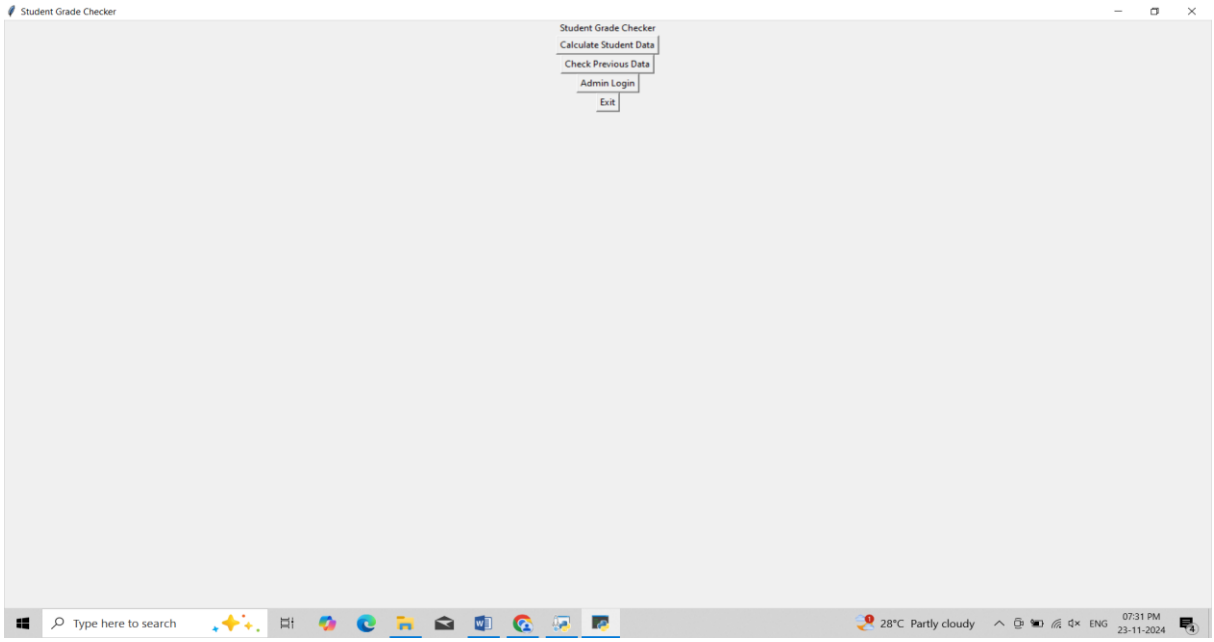

Fig 5.1.: Prompt output page of the above written code

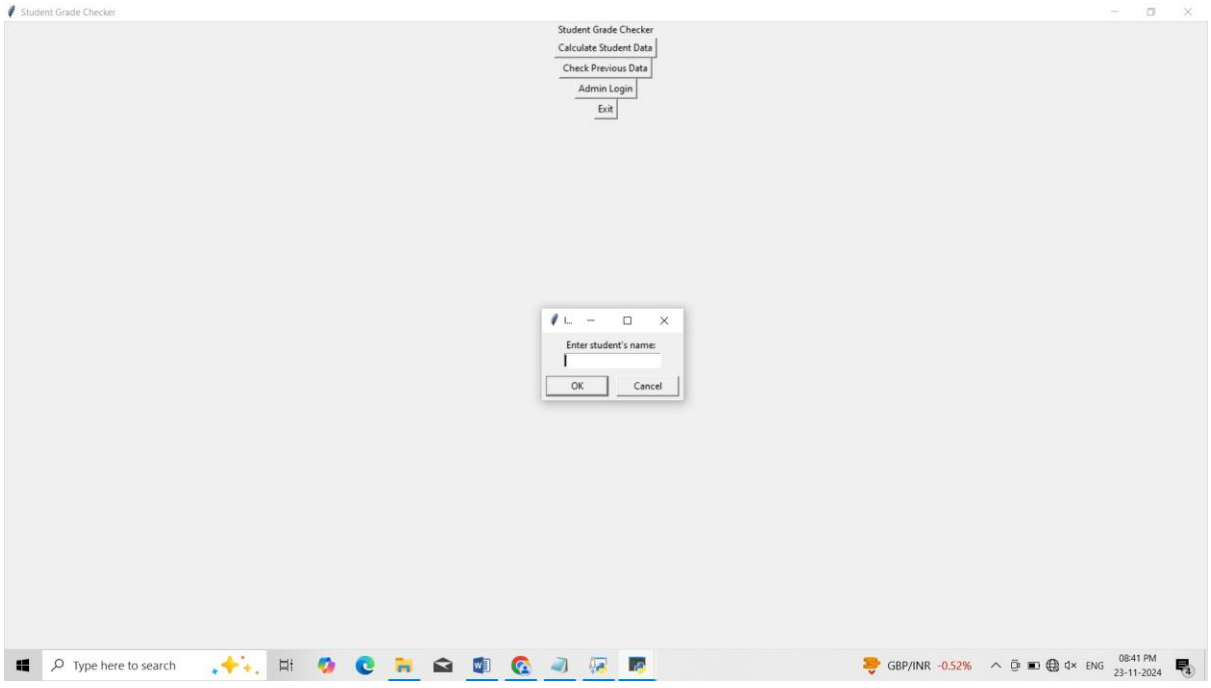

Fig 5.2.: Student name taken after clicking on ‘Calculate Student Data’

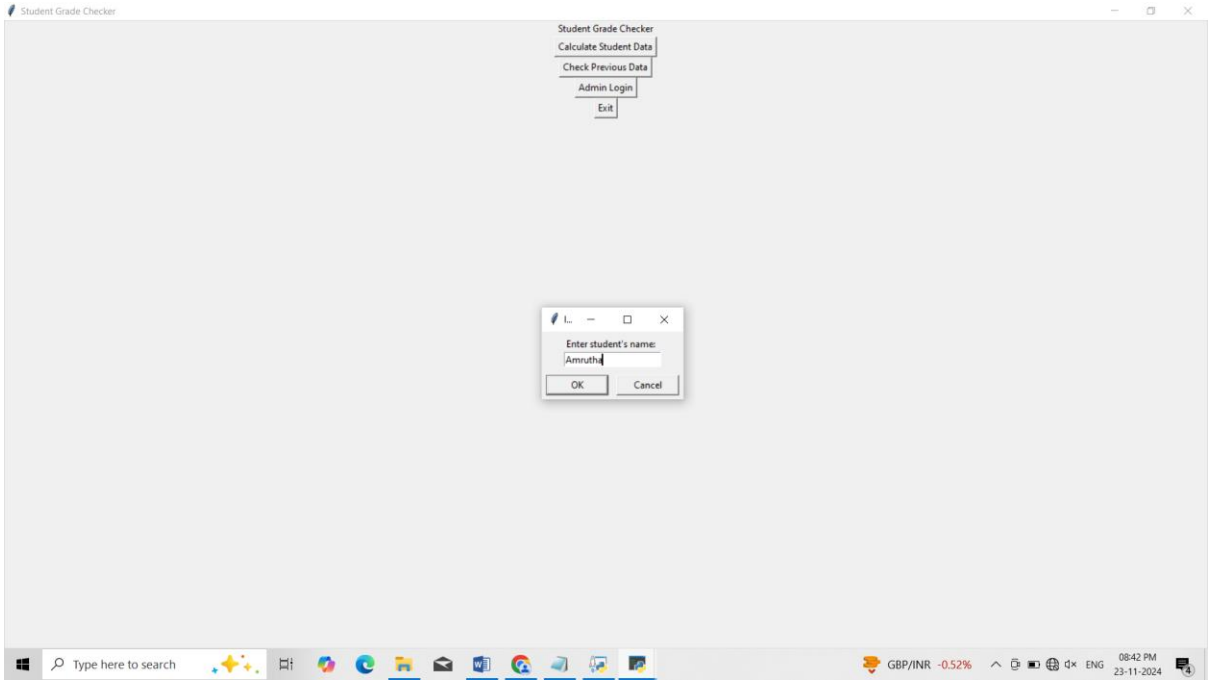

Fig 5.3.: Student name taken after clicking on ‘Calculate Student Data’

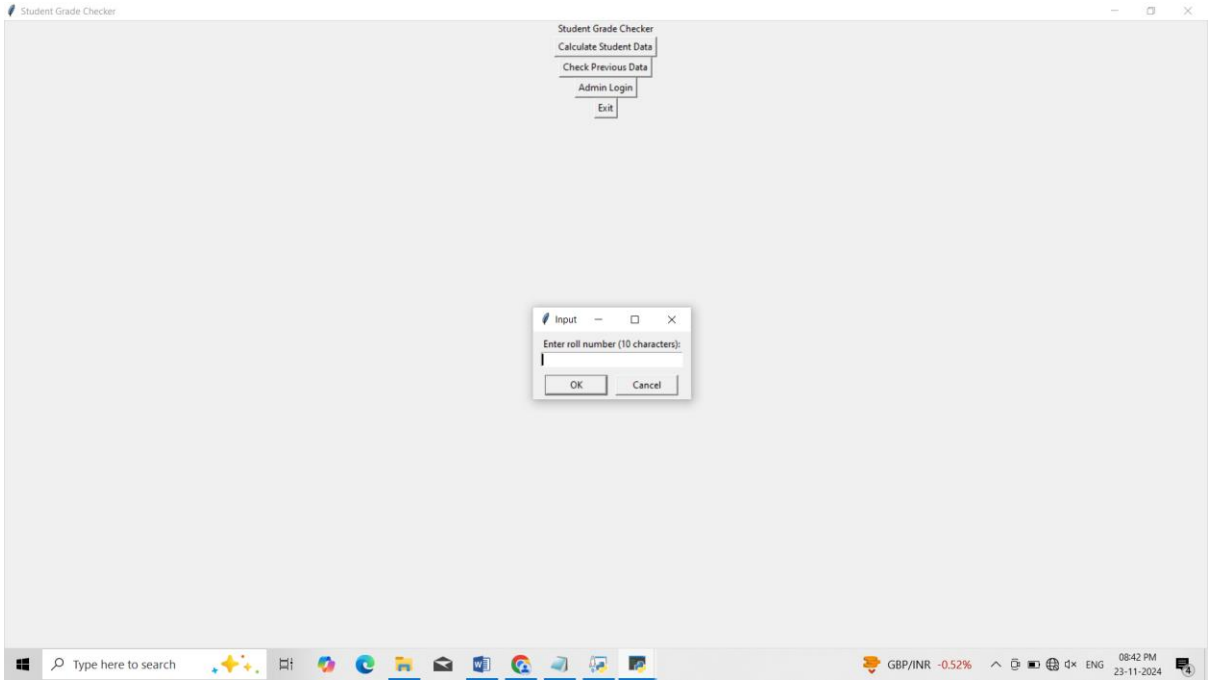

Fig 5.4.: Student roll number taken after clicking on ‘Calculate Student Data’

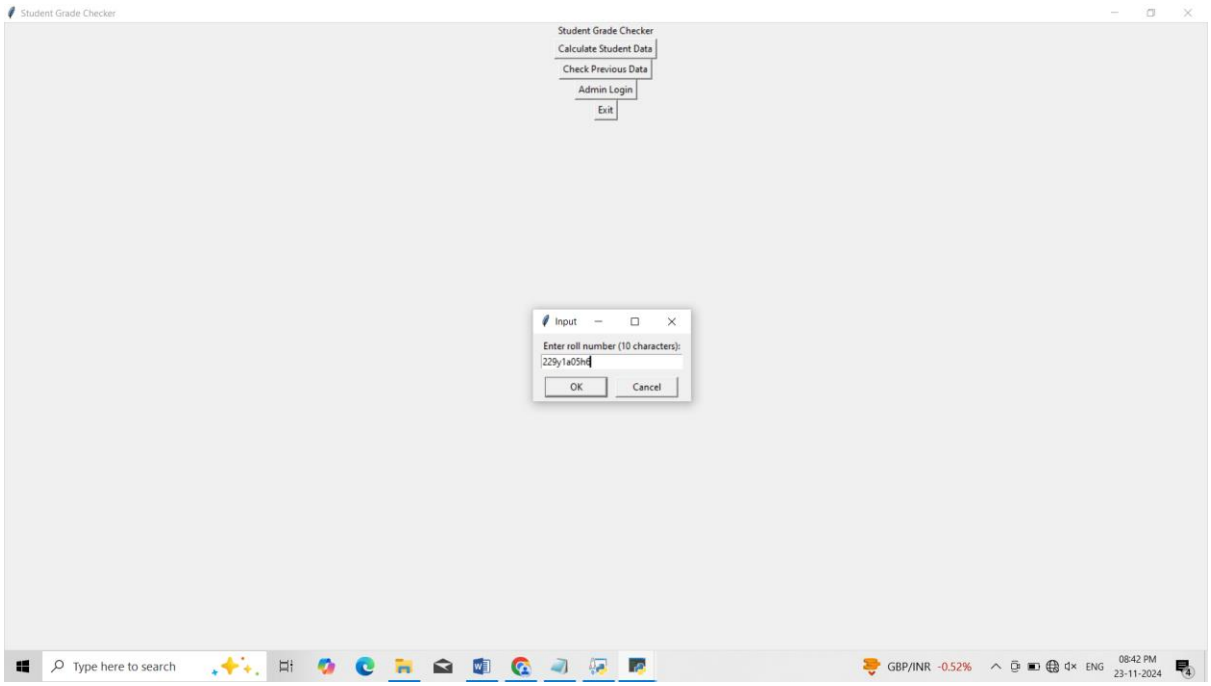

Fig 5.5.: Student roll number taken after clicking on ‘Calculate Student Data’

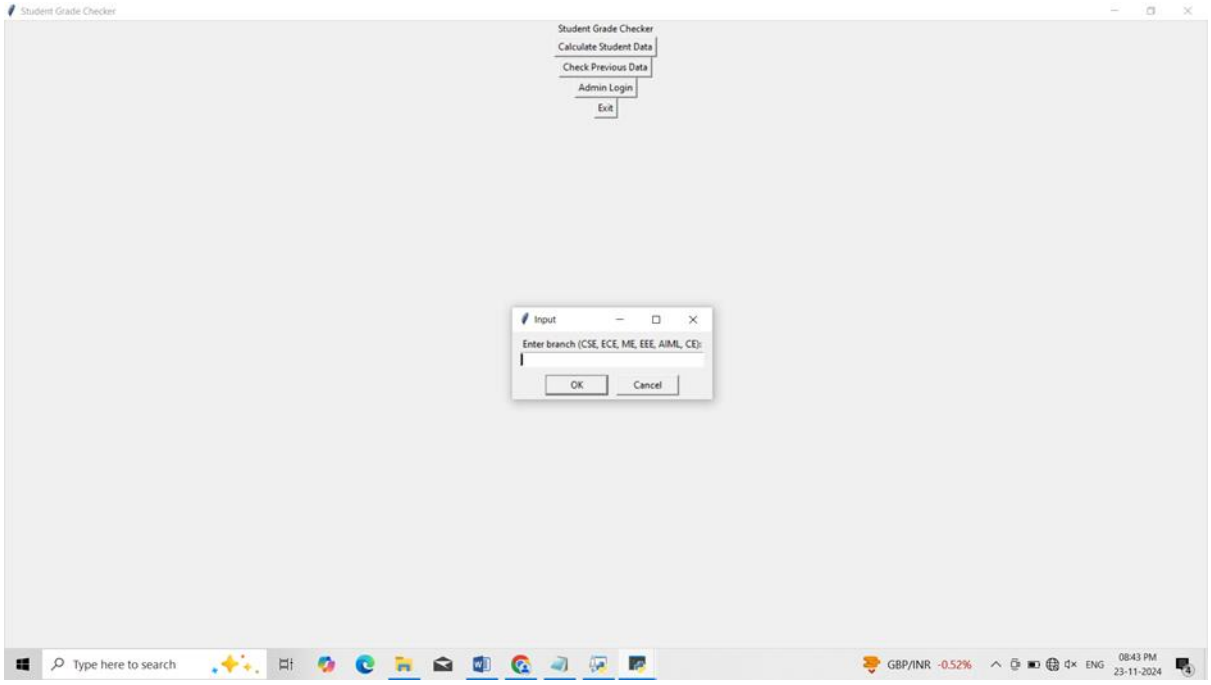

Fig 5.6.: Student branch taken after clicking on ‘Calculate Student Data’

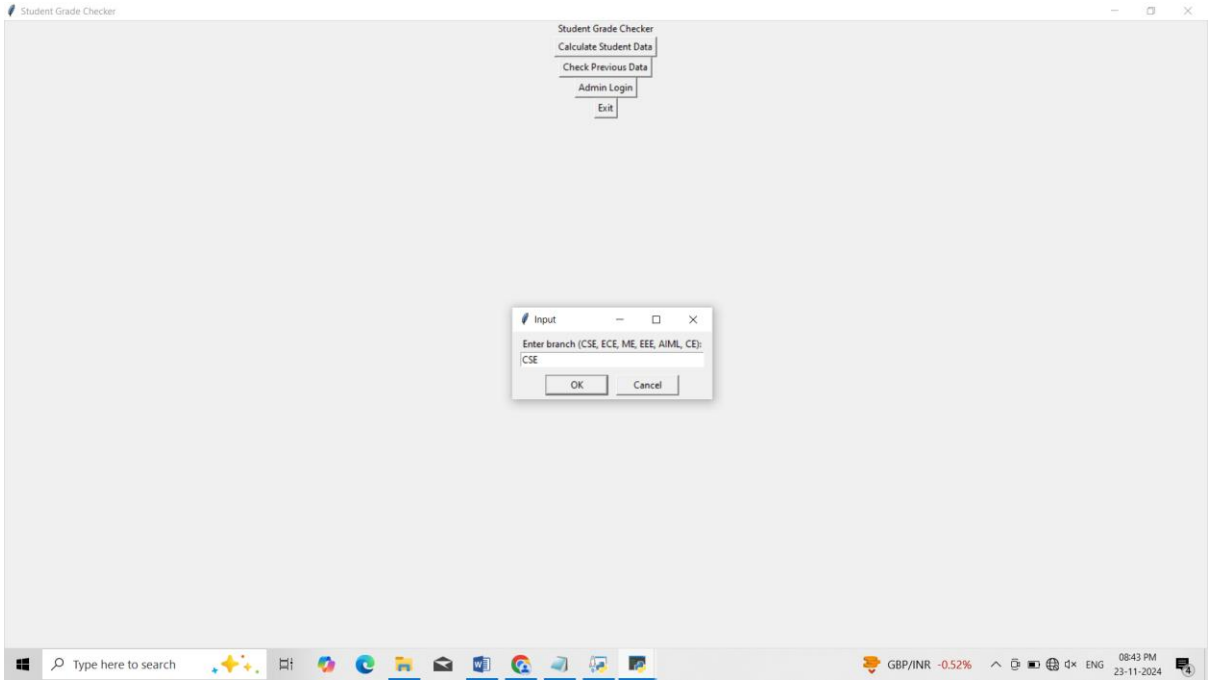

Fig 5.7.: Student branch taken after clicking on ‘Calculate Student Data’

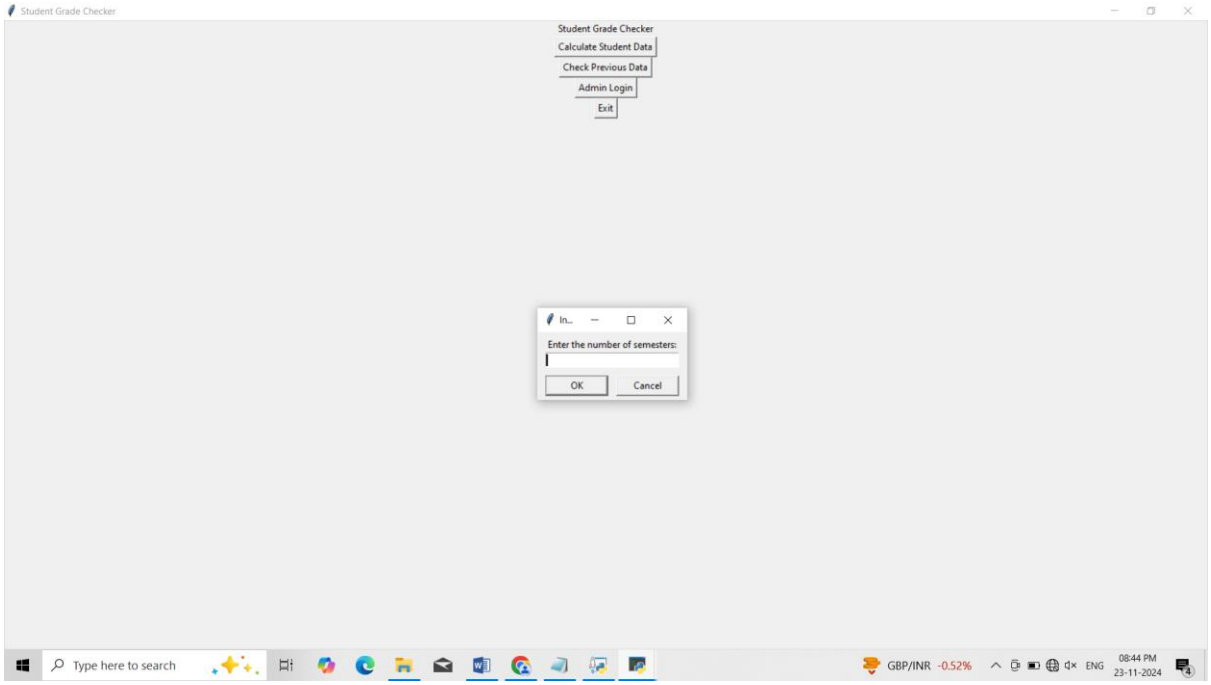

Fig 5.8.: No . of semesters taken after clicking on ‘Calculate Student Data’

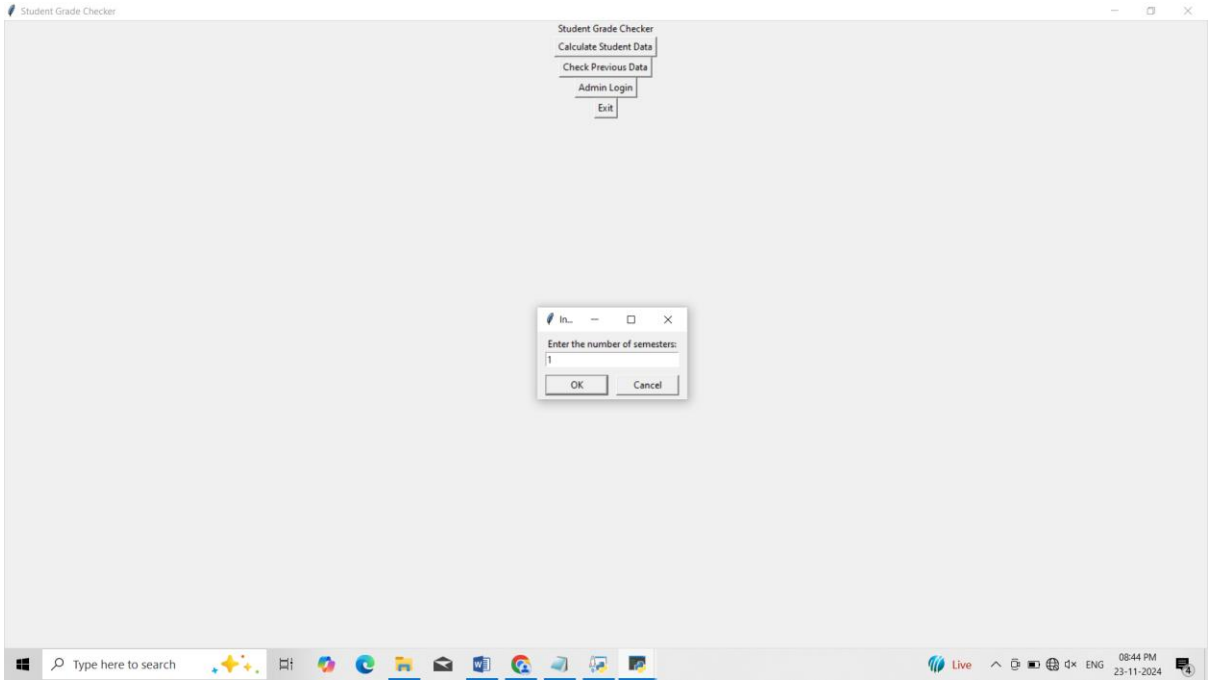

Fig 5.9.: No . of semesters taken after clicking on ‘Calculate Student Data’

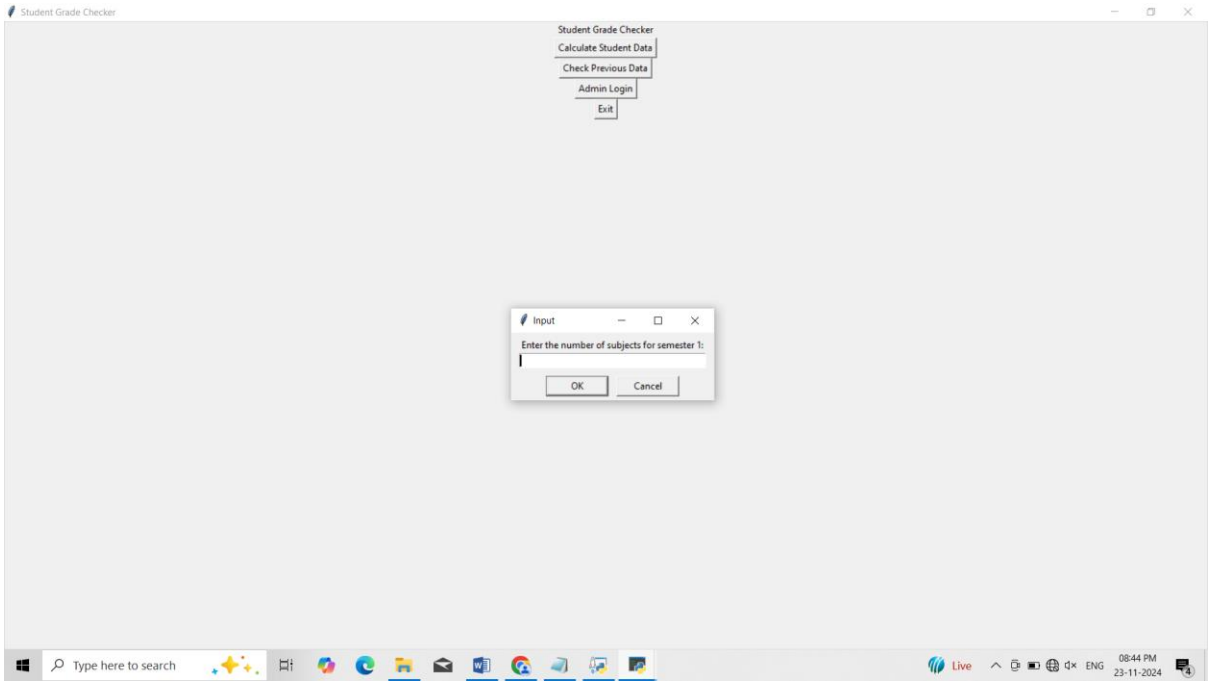

Fig 5.10.: No . of subjects taken after clicking on ‘Calculate Student Data’

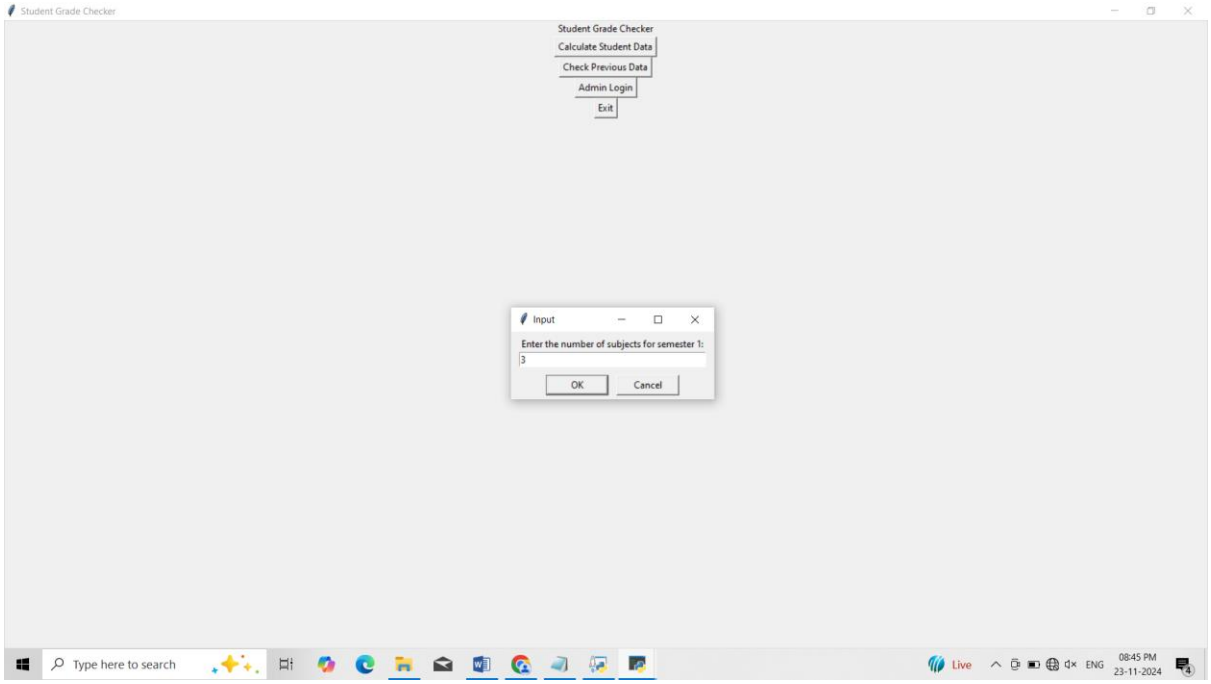

Fig5.11.: No . of subjects taken after clicking on ‘Calculate Student Data’

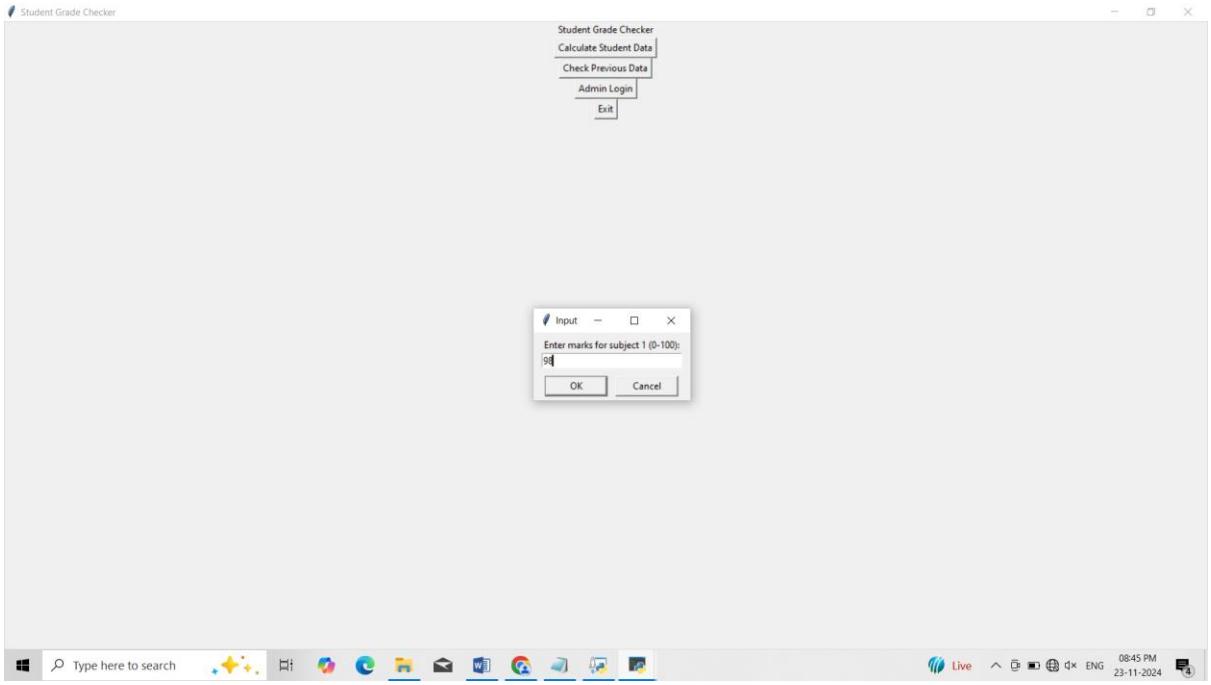

Fig 5.12.: Marks of subject taken after clicking on ‘Calculate Student Data’

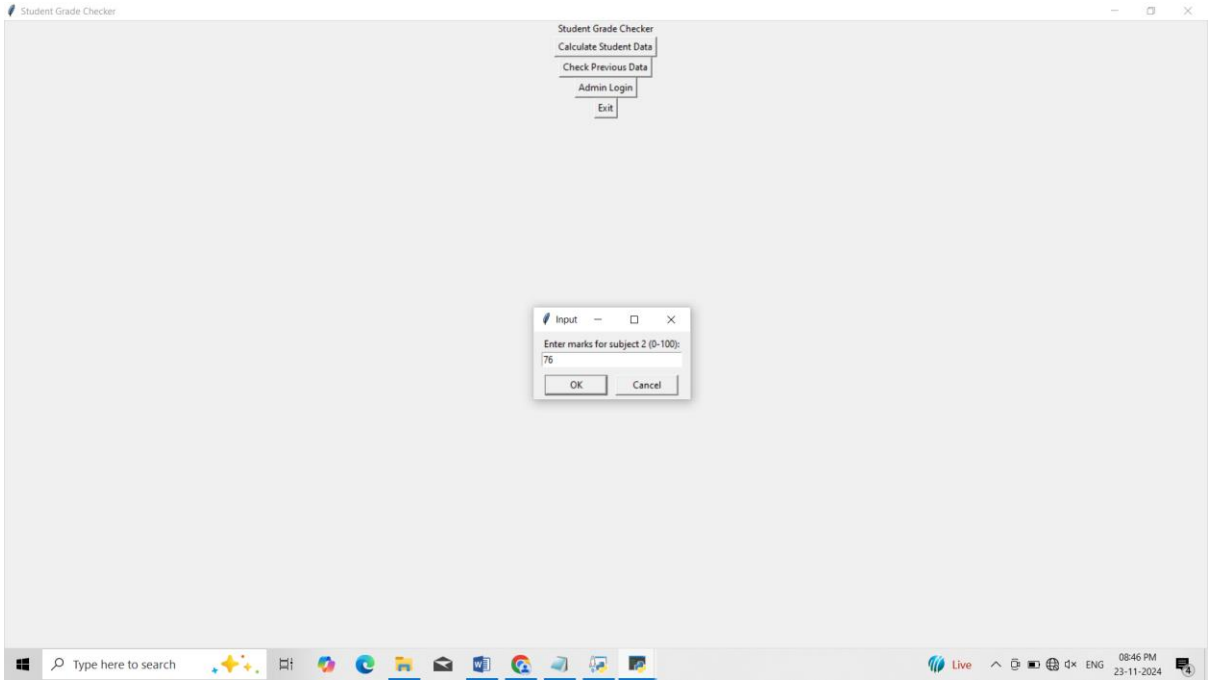

Fig 5.13.: Marks of subject taken after clicking on ‘Calculate Student Data’

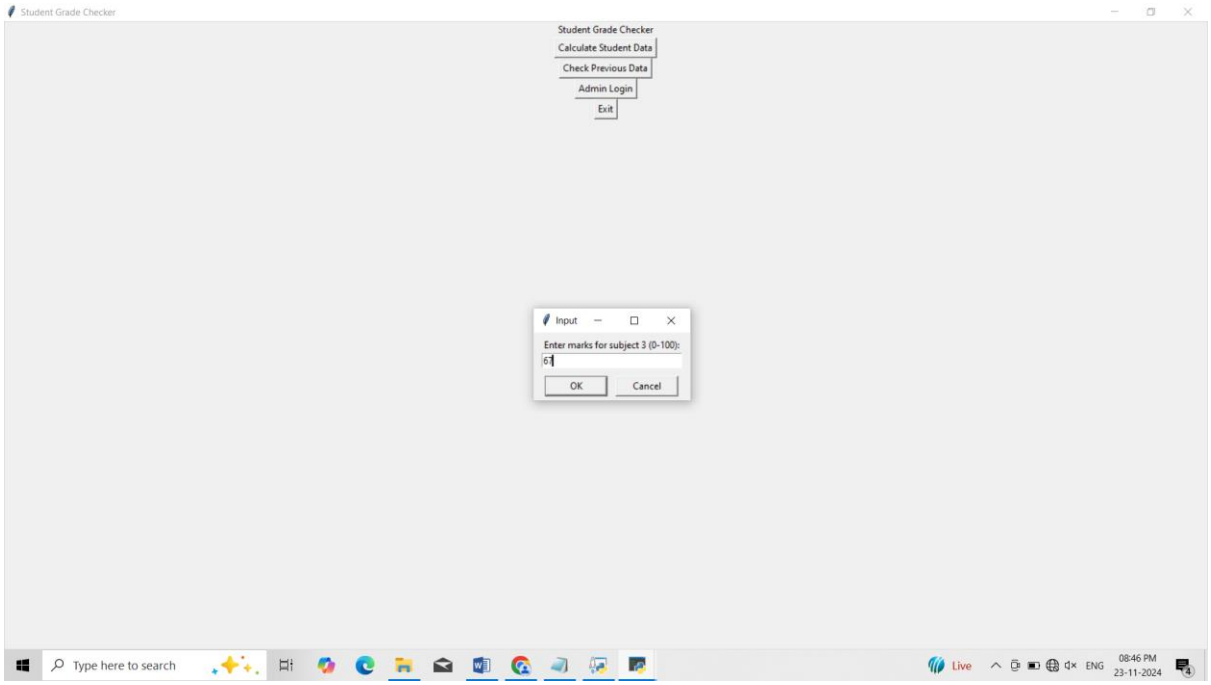

Fig 5.14.: Marks of subject taken after clicking on ‘Calculate Student Data’

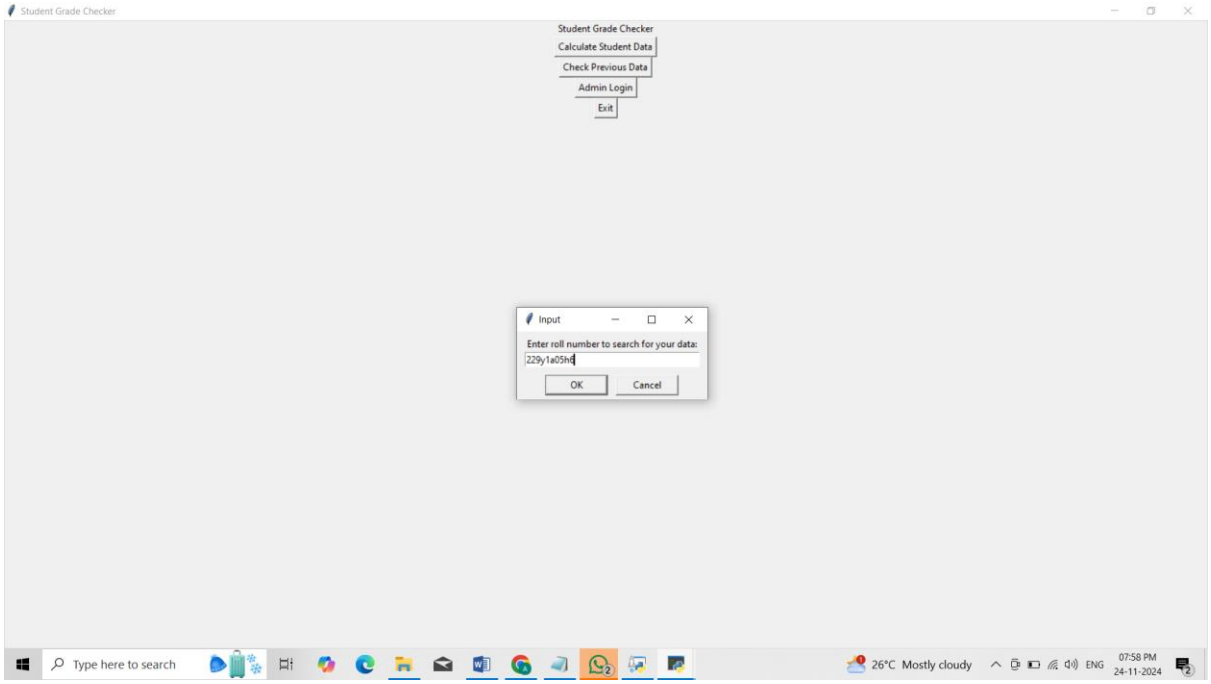

Fig 5.15.: Prompt page for ‘Checking Student Previous Data’

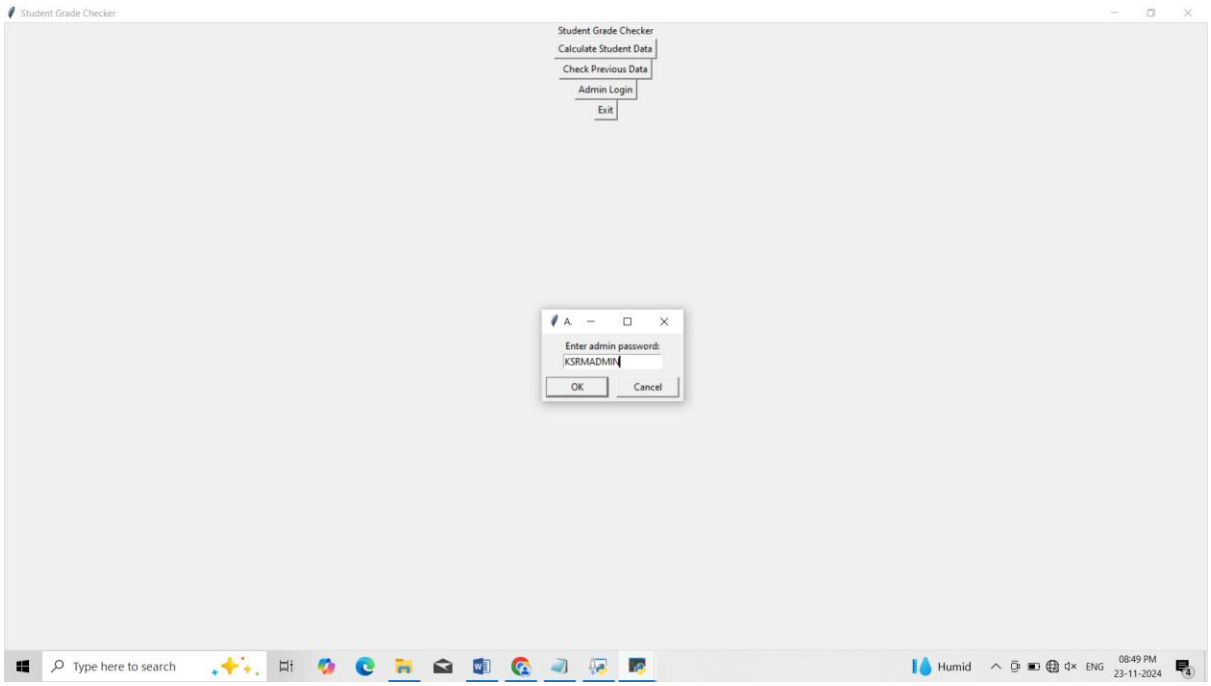

Fig 5.16.: Prompt page for viewing ‘Admin login’

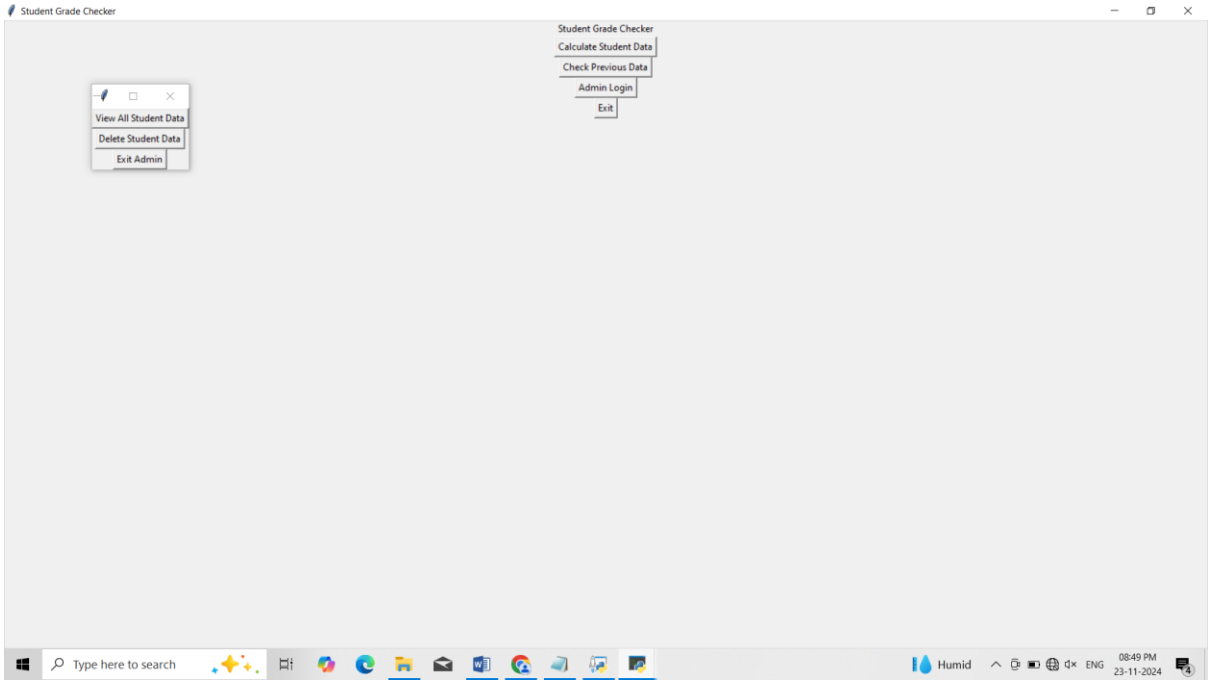

Fig 5.17.: Admin login Page

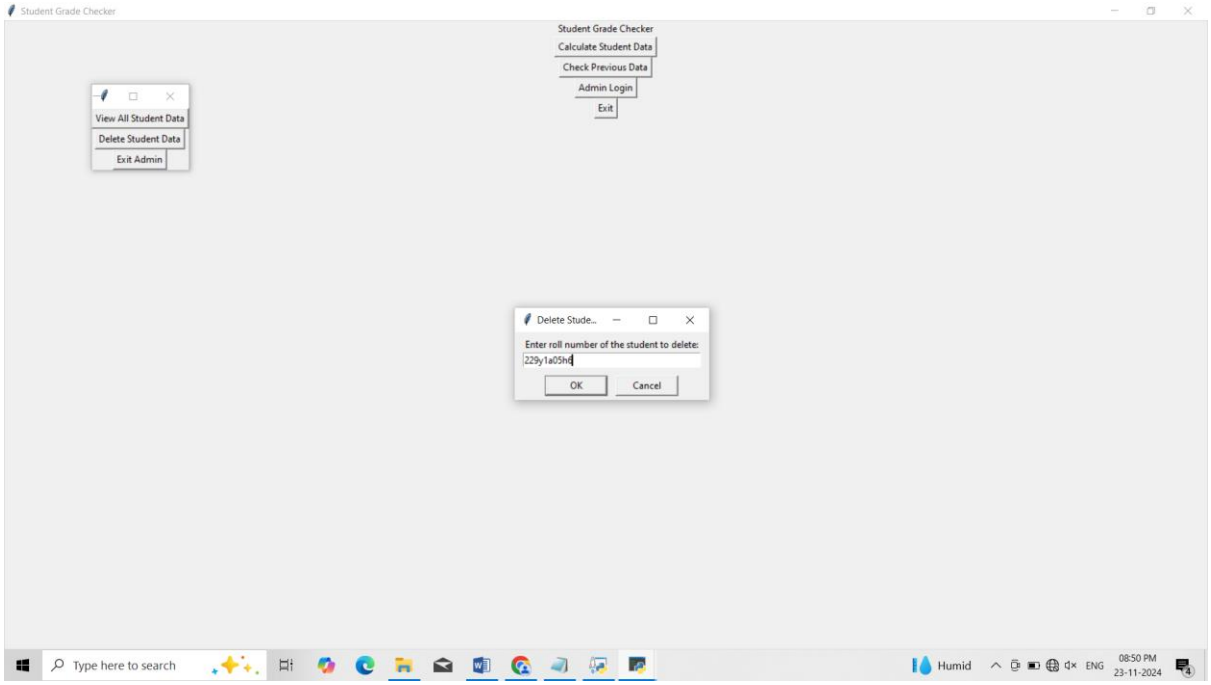

Fig 5.18.: Deleting Student Data in Admin login Page

6. CONCLUSION

The Student Grade Checker project serves as a practical tool for educational institutions, including schools and universities, to manage and evaluate student performance efficiently. By automating the process of calculating SGPA and CGPA, the application reduces the administrative burden on educators and allows them to focus on teaching and mentoring students. Additionally, the ability to store and retrieve student records provides a reliable way to maintain academic history, which is essential for academic counseling and planning.

1. Sorting Algorithm: Quick Sort

Concept: Quick Sort is a divide-and-conquer algorithm that sorts an array or list by selecting a 'pivot' element and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.

Use in the Program: The quick_sort function is implemented to sort student records based on total marks in descending order. This is useful for displaying student data in a way that highlights the highest achievers. The function takes a list of student records and a key (in this case, 'total_marks') and sorts the records accordingly. This allows the admin to view all student data sorted by performance.

2. Search Algorithm: Linear Search

Concept: Linear Search is a straightforward search algorithm that checks each element in a list sequentially until the desired element is found or the list ends.

Use in the Program: The linear_search function is used to find a student's record by their roll number. When a user wants to check their previous data, they input their roll number, and the program uses linear search to iterate through the list of student records to find a match. This is a simple and effective method for searching in small datasets, as is the case here, where student records are stored in a text file.

7. FUTURE ENHANCEMENTS

User Authentication and Role Management:

Enhancement: Implement a user authentication system that allows students to create accounts and log in securely. Different roles (students, admins) can have different access levels.
Benefit: This will improve security and allow for personalized user experiences.

Data Visualization:

Enhancement: Integrate data visualization tools (e.g., graphs and charts) to display students' performance over semesters.
Benefit: Visual representations of data can help students and administrators easily identify trends and areas for improvement.

Mobile Application Development:

Enhancement: Develop a mobile version of the application for iOS and Android platforms.
Benefit: This will make the application more accessible, allowing students to check their grades and data on the go.

Enhanced Reporting Features:

Enhancement: Allow users to generate detailed reports in PDF format that can be printed or shared.
Benefit: This will provide students with a formal document of their academic performance, which can be useful for job applications or further studies.

Integration with Learning Management Systems (LMS):

Enhancement: Integrate the application with popular LMS platforms (e.g., Moodle, Blackboard) to automatically fetch grades and attendance.

Benefit: This will streamline the data entry process and ensure that the application always has the latest information.

Notifications and Reminders:

Enhancement: Implement a notification system that reminds students of important dates, such as exam schedules or deadlines for grade submissions.
Benefit: This feature can help students stay organized and manage their time effectively.

Multi-Language Support:

Enhancement: Add support for multiple languages to cater to a diverse user base.
Benefit: This will make the application more inclusive and accessible to non-English speaking users.

Performance Analytics:

Enhancement: Introduce features that allow students to analyze their performance based on various criteria (e.g., subject-wise performance, comparison with peers).
Benefit: This can help students identify their strengths and weaknesses, guiding their study efforts.

Backup and Restore Functionality:

Enhancement: Implement a backup and restore feature that allows users to save their data securely and recover it in case of data loss.

Benefit: This will enhance data security and user confidence in using the application.

Customizable Grading System:

Enhancement: Allow institutions to customize the grading system according to their specific requirements (e.g., different grade point scales).

Benefit: This will make the application adaptable to various educational institutions and grading policies.

8. REFERENCES

1. **Tkinter Documentation**:
    - https://docs.python.org/3/library/tkinter.html
2. **Python Official Documentation**:
    - https://docs.python.org/3/
3. **Python File Handling:**
    - https://realpython.com/read-write-files-python/
4. **Grade Calculation and GPA/CGPA Concept:**
    - https://en.wikipedia.org/wiki/Grade_point_average
5. **Quick Sort Algorithm:**
    - https://www.geeksforgeeks.org/quick-sort/
6. **Python Simple dialog and Message box**:
    - https://python-course.eu/tkinter/dialogs-and-message-boxes.php
7. **Admin Authentication and Security in Python**:
    - https://realpython.com/python-password-validation/
8. **Software Engineering Best Practices**:
    - https://sourcemaking.com/architecture-patterns
9. **Tkinter Tutorial for Beginners**:
    - https://realpython.com/python-gui-tkinter/
10. TextBook: Ellis Horowitz, Sartaj Sahni and Sanguthevar Rajasekaran,"**Fundamentals of Computer Algorithms"**, Galgotia Publications.

These references provide both the technical foundation and theoretical background to support the development and functionality of the **Student Grade Checker**. They also help in understanding the tools, libraries, and algorithms used to implement the system efficiently.