**Project Report**

**Student Grades Management System**

**Project Title:** Student Grades Management System

**Course:** Introduction to Problem Solving and Python Programming

**Submission Date**: 22/11/2025

**Submitted By:**

**Name:** Swara Narendra Kale

**Registration Number:** 25BAI11072

**Department:** School of Computer Science

and Engineering [AI ML]

**University:** VIT Bhopal University

### 1. Introduction

The **Student Grades Management System** is a Python-based console application designed to assist teachers and administrators in managing student academic records efficiently. In an educational setting, tracking student performance is crucial. This project provides a lightweight, user-friendly interface to perform fundamental operations such as adding new student records, updating existing marks, deleting records, and viewing the entire class list. The system utilizes Python's core data structures to ensure fast data retrieval and manipulation.

**2. Problem Statement**

Traditional methods of recording grades (such as paper-based ledgers or unorganized spreadsheets) are prone to errors, difficult to update, and lack immediate accessibility. There is a need for a digital tool that allows for the immediate recording and modification of student marks in a structured format. This project aims to solve this by automating the Grade Management process through a Command Line Interface (CLI).

**3. Functional Requirements**

The system fulfills the following functional requirements:

1. **Add Student:** The system must allow the user to input a student's name and their corresponding grade.
2. **Update Student:** The system must allow the user to modify the grade of an existing student.
3. **Delete Student:** The system must allow the user to remove a student's record permanently.
4. **View All Students:** The system must display a list of all students and their grades.
5. **Search/Validation:** The system must check if a student exists before attempting to update or delete.
6. **Exit:** The system must provide a way to terminate the program.

### 4. Non-functional Requirements

1. **Usability:** The application provides a clear, text-based menu for navigation.
2. **Performance:** Operations (Add, Update, Delete) are performed in O(1) time complexity on average due to the use of Hash Maps (Dictionaries).
3. **Reliability:** The system handles basic logic errors, such as attempting to update a non-existent user, by providing informative feedback.
4. **Portability:** The script can run on any machine with a Python interpreter installed.

### 5. System Architecture

The system follows a **Modular Architecture** design pattern, ensuring a clear "Separation of Concerns." Instead of a single monolithic script, the application is divided into three distinct modules, each responsible for a specific aspect of the program.

* **5.1 Presentation Layer (Interface)**
  + **Module:** main.py
  + **Responsibility:** This module serves as the entry point of the application. It manages the user interface (CLI), displays the menu options, and handles the primary control flow (the while loop). It does not process data directly; instead, it delegates tasks to the Logic and Utility modules.
* **5.2 Business Logic & Data Layer**
  + **Module:** operations.py
  + **Responsibility:** This module contains the core functionality of the system. It manages the global state (the student\_grades dictionary) and executes CRUD operations (Create, Read, Update, Delete). By isolating this logic, the data remains protected from direct manipulation by the user interface.
* **5.3 Utility Layer (Helper Functions)**
  + **Module:** utils.py
  + **Responsibility:** This module handles input validation and error prevention. It includes functions like get\_valid\_integer() and get\_valid\_name() which utilize try-except blocks to ensure the system does not crash due to invalid user input (e.g., entering text when a number is required).

**Bonus: Updated Section 8 (Implementation Details)**

**Core Code Snippet:**

Python

# main.py - The Controller

import operations

import utils

def main():

while True:

# Using utils module to safely get the menu choice

choice = utils.get\_valid\_integer("Enter your choice (1-5): ")

if choice == 1:

# Using utils for validation and operations for logic

name = utils.get\_valid\_name("Enter student name: ")

grade = utils.get\_valid\_integer("Enter student grade: ")

operations.add\_student(name, grade)

elif choice == 4:

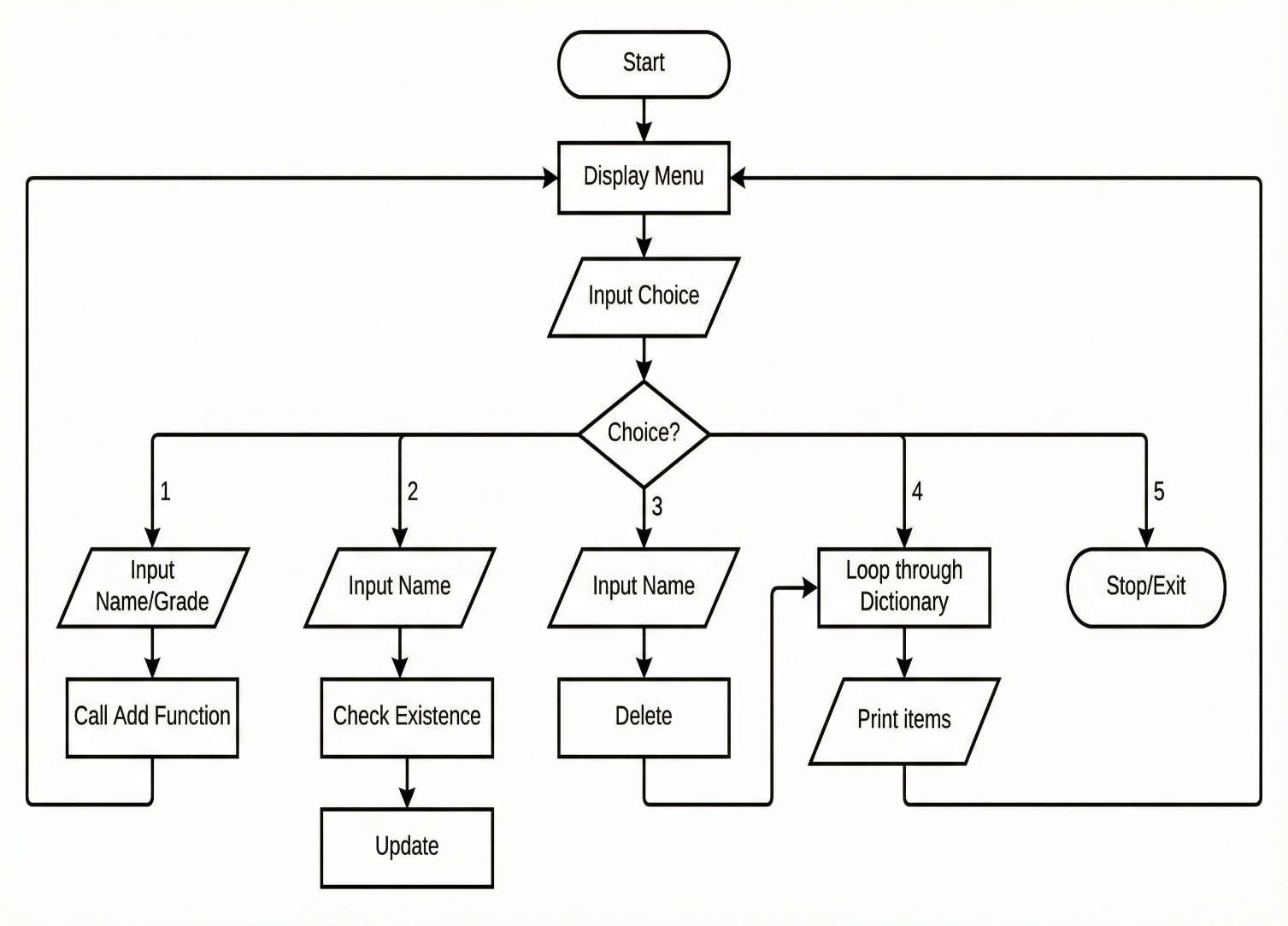
operations.display\_all\_students()

### 6. Design Diagrams

1. **Case Diagram**

****

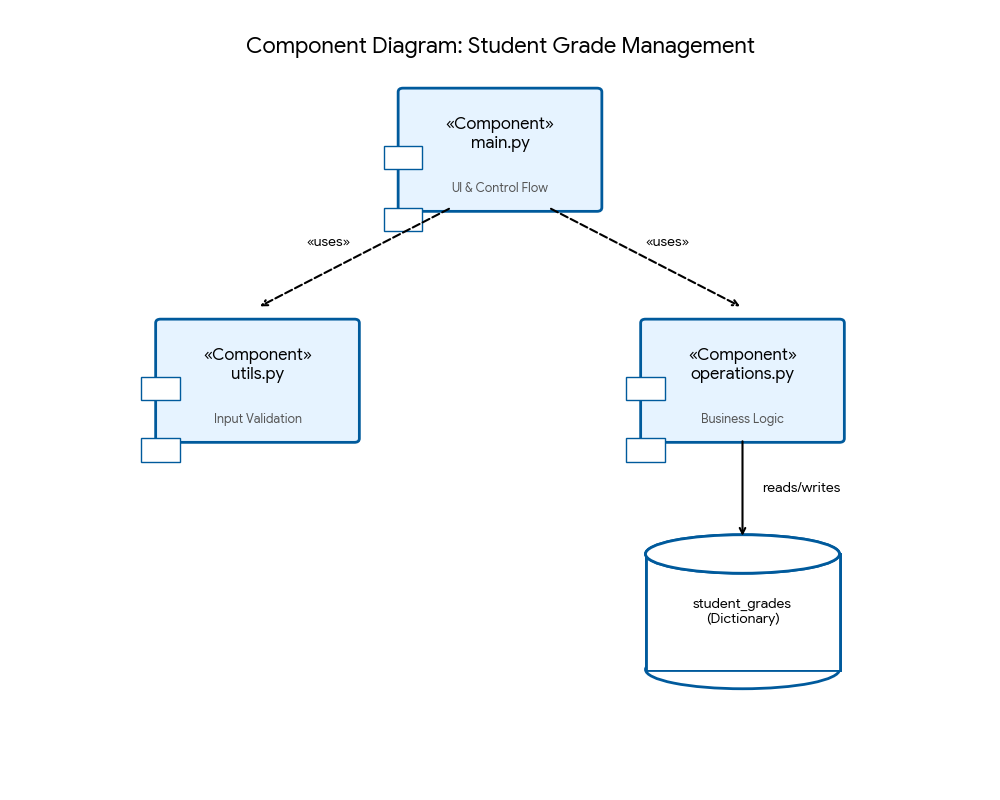
1. **Workflow Diagram (Flowchart)**



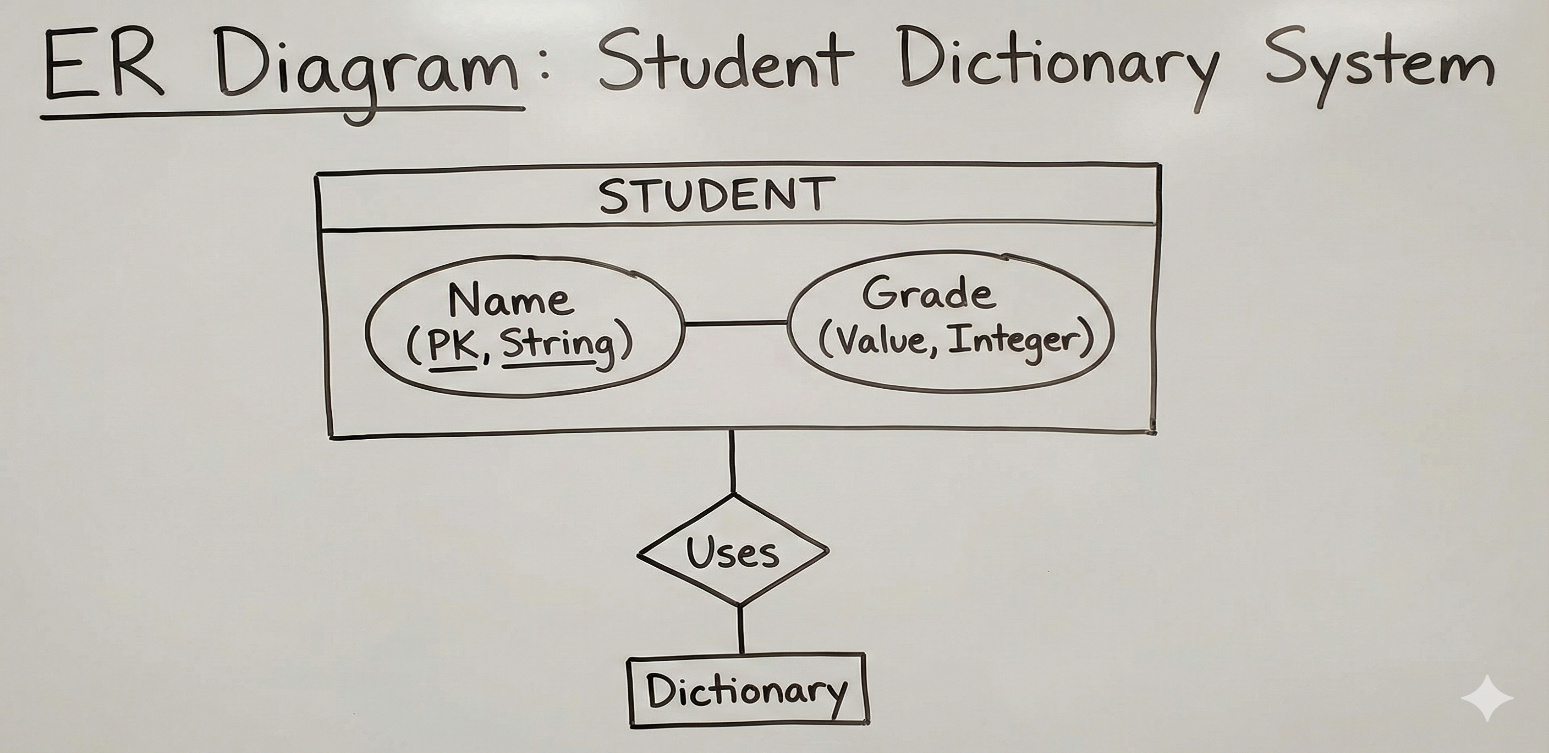
1. **Sequence Diagram (For "Update Student")**



1. **Class/Component Diagram**

****

**E. ER Diagram**



### 7. Design Decisions & Rationale

1. **Language Selection:** **Python** was chosen for its readability and ease of handling data structures.
2. **Data Structure Selection:** A **Dictionary** ({}) was chosen over a List.
   * *Rationale:* Dictionaries allow for Key-Value pairing. Searching for a student by name in a List would require iterating through the whole list , whereas a Dictionary allows looking up a name in constant time .
3. **User Interface:** A **CLI (Command Line Interface)** was chosen.
   * *Rationale:* It is lightweight, easy to implement for a fundamental project, and requires no external UI libraries.

### 8. Implementation Details

**Core Code Snippet:**

Python

# main.py - The Controller

import operations

import utils

def main():

while True:

# Using utils module to safely get the menu choice

choice = utils.get\_valid\_integer("Enter your choice (1-5): ")

if choice == 1:

# Using utils for validation and operations for logic

name = utils.get\_valid\_name("Enter student name: ")

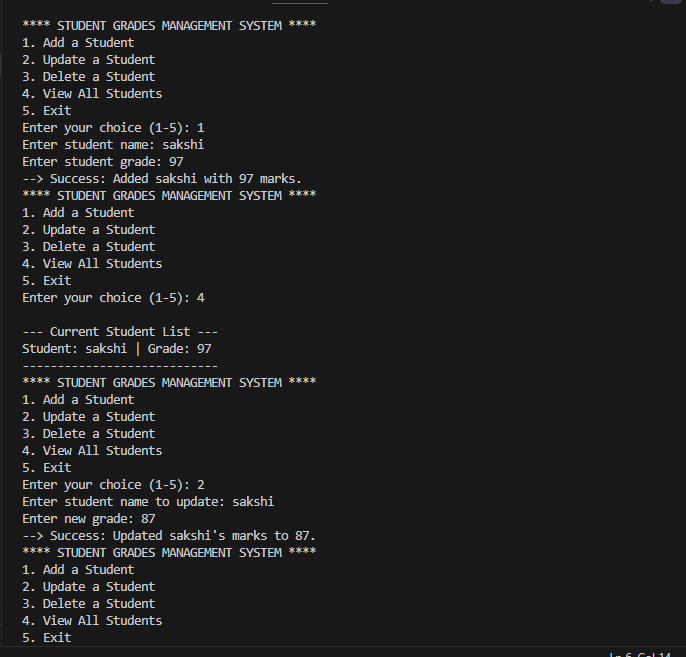
grade = utils.get\_valid\_integer("Enter student grade: ")

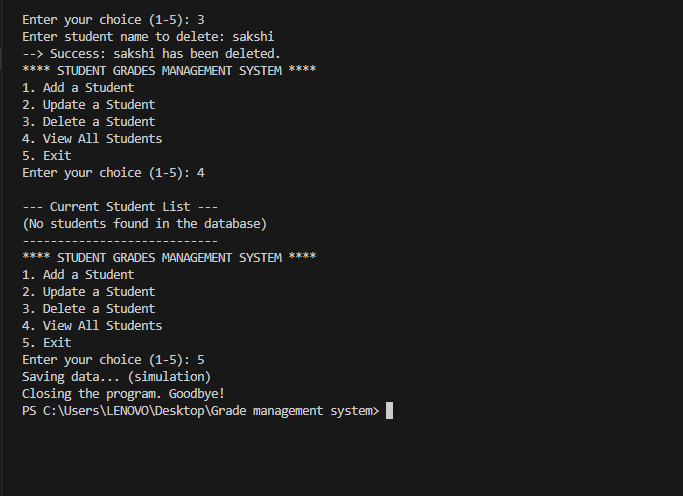
operations.add\_student(name, grade)

elif choice == 4:

operations.display\_all\_students()

**9. Screenshots / Results**

****



### 10. Testing Approach

* **Unit Testing:** Each function (add, update, delete) was tested individually to ensure it modifies the dictionary correctly.
* **Boundary Testing:** Tested what happens if the dictionary is empty (Handled by the if student\_grades: check in the display function).
* **Negative Testing:** Tested inputting a name that does not exist during Update/Delete operations. The system correctly outputs "Name is not found!" rather than crashing.

### 11. Challenges Faced

1. **Data Persistence:** Currently, the data is stored in RAM (variables). Closing the program erases the data. Learning how to keep data saved was a conceptual challenge.
2. **Input Handling:** Ensuring the user inputs an integer for the "Choice" and "Grade". If a user enters text, the program raises a ValueError.
3. **Scope Management:** Understanding how to access the global student\_grades dictionary inside local functions.

### 12. Learnings & Key Takeaways

1. **Python Syntax:** Mastered the use of functions, loops, and conditional statements.
2. **Data Structures:** Gained a deep understanding of how **Dictionaries** work and why they are efficient for lookups.
3. **Modular Programming:** Learned how to break a large problem into smaller, manageable functions.
4. **f-strings:** Learned how to format strings dynamically for better user output.

### 13. Future Enhancements

1. **File Handling:** Integrate .txt or .csv file handling to save student data permanently so it isn't lost hen the program closes.
2. **GUI:** Build a graphical interface using **Tkinter** or **Streamlit** for a more modern look.
3. **Advanced Analytics:** Add features to calculate the class average, highest grade, and lowest grade.

### 14. References

Python 3 Documentation: <https://docs.python.org/3/>