

Grading App

DOCUMENTATION

NERURKAR, TANMAY ASHAY

Overview

The Grade Management App is designed to help you manage student information and assignments within different courses. You can add students, create assignments, and track grades easily using this application.

Getting Started

1. **Installation:**

- Ensure you have Python installed on your system.
- Download the source code for the Grade Management App.
- Install any required dependencies (e.g., SQLite library).

2. **Database Setup:**

- The app uses an SQLite database to store data.
- Run the app to create the necessary tables (users, students, assignments, scores).

1. Course Selection

Every course has a password. You can register for a new course, and assign it a password.

2. Viewing Students and Assignments

Once inside a course, you'll see a list of enrolled students and their assigned assignments.

Click on a student's name to view their assigned scores.

3. Assigning Grades

Select Student and Assignment: Choose a student and the corresponding assignment.

Enter the score.

Click on confirm to save the score.

If a score is already assigned, you will be asked if you wish to modify it.

4. Adding Students and Assignments

To add new students or assignments:

Click the "Add Student" button to enroll a new student.

Click the "Add Assignment" button to create a new assignment.

Click the "Delete Assignment" button to delete the selected assignment.

5. Calculating Overall Percentage Grade

The app automatically calculates the overall percentage grade for each student based on the assigned grades.

Grades are displayed in the format:

1. Student name
2. Homework total percentage
3. Test Total percentage
4. Overall percentage
5. Overall grade

NOTE:

- Homework are weighted at 30%
- Tests are weighted at 70%

Structure of Program:

```
class Student:
```

```
    def __init__(self, name, rocketid):
        self.id = rocketid
        self.name = name
```

```
class Assignment:
```

```
    def __init__(self, assignmentId, name, type):
        self.name = name
        self.type = type
```

```
class Course:
```

```
    def __init__(self, name):
        self.name = name
        self.students = []
        self.assignments = []
```

```
    def add_student(self, student):
```

```
    def remove_student(self, student):
```

```
    def add_assignment(self, assignment):
```

```
    def remove_assignment(self, assignment):
```

```
class GradeManager:
```

```
    def __init__(self):
        self.courses = []
        self.selectedCourse = Course
```

```
    # Create the Users table if it does not exist
```

```
    def createUserTable(self):
```

```
    # Create the Students table if it does not exist
```

```
    def createStudentsTable(self):
```

```
    # Create the Assignments table if it does not exist
```

```
    def createAssignmentsTable(self):
```

```
    # Create the Scores table if it does not exist
```

```
    def createScoresTable(self):
```

```
    # Initialization
```

```
    def initialize(self):
```

```
        # Scans grades table, adds courses, students, and assignments found.
```

```
# Database Functions
```

```
def modifyRecord(self, inpStr):
```

```
def getRecords(self, inpStr):
```

```
def getFirstRecord(self, inpStr):
```

```
# Returns objects using name parameter
```

```
def get_objects(self, course_name, student_name, assignment_name):
```

```
# Enters grade into Database
```

```
def grade_assignment(self, course_name, student_name, assignment_name, score):
```

```
# Calculate grade using course and student
```

```
def calculate_grade(self, course_name, student_name):
```

```
##### GUI #####
```

```
class MyGrades(tk.Tk):
```

```
class LoginPage(tk.Frame):
```

```
class Registration(tk.Frame):
```

```
class MainPage(tk.Frame):
```

```
class GradesPage(tk.Frame):
```

```
##### Main #####
```

```
# Connect to or create Database
```

```
conn = sqlite3.connect('gradesdatabase.db')
```

```
if conn:
```

```
    print("Opened database successfully")
```

```
else:
```

```
    print("Error creating database")
```

```
# Create grademanager
```

```
g1 = GradeManager()
```

```
g1.createUserTable()
```

```
g1.createStudentsTable()
```

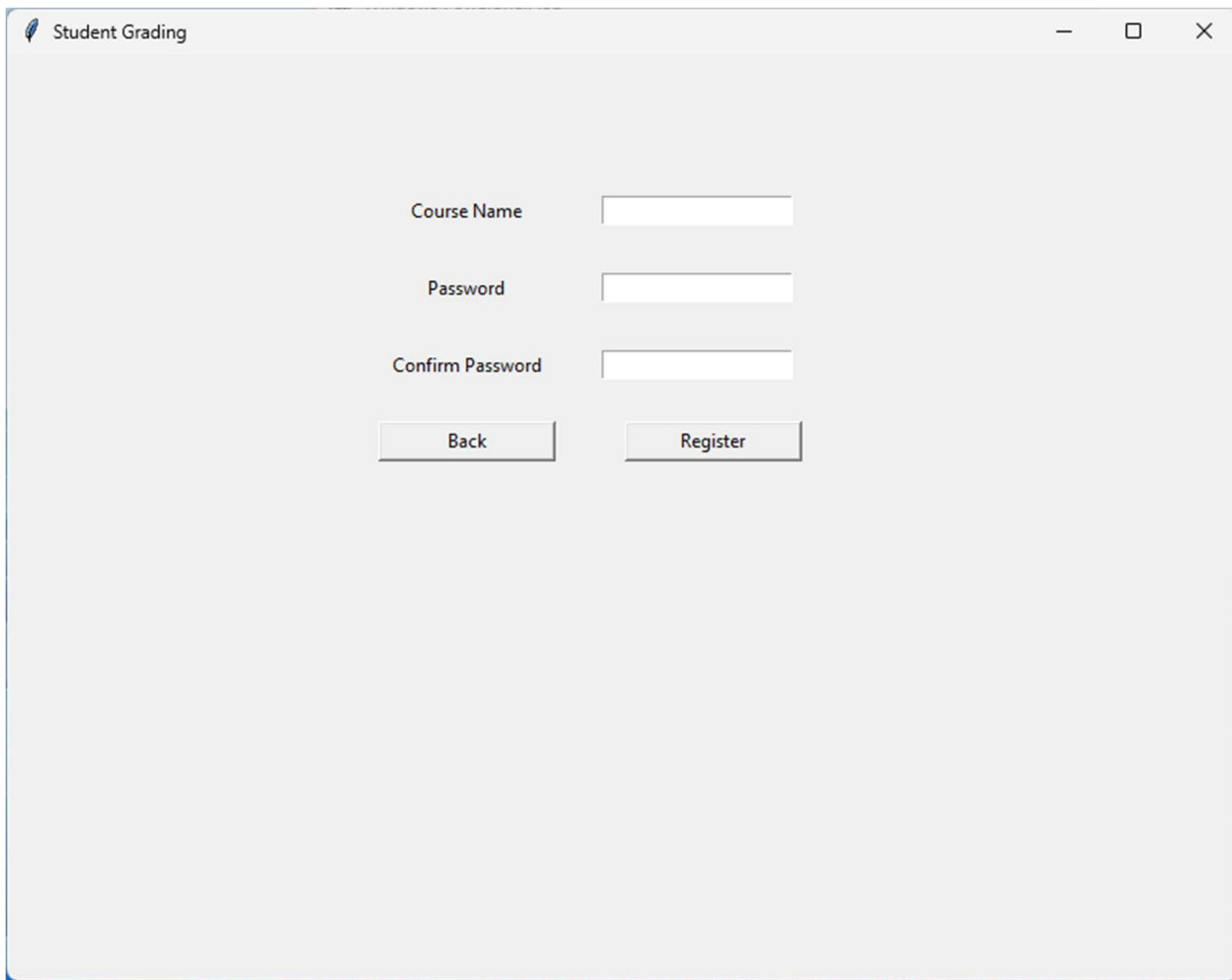
```
g1.createAssignmentsTable()
```

```
g1.createScoresTable()
```

```
g1.initialize()
```

```
# Create Window and display  
app = MyGrades()  
app.title("Student Grading")  
app.geometry("800x600")  
app.mainloop()
```

Screenshots:



A screenshot of a web application window titled "Student Grading". The window contains a registration form with three input fields: "Course Name", "Password", and "Confirm Password". Below the fields are two buttons: "Back" and "Register".

Student Grading

Course Name

Password

Confirm Password

Registration Page:

1. Checks all fields.
2. Checks Password.
3. Checks whether course already exists in DB.
4. If yes, displays error
5. If not, user is registered

Course Name

Password

Confirm Password

Back

Register

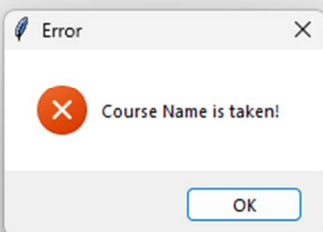
Course Name

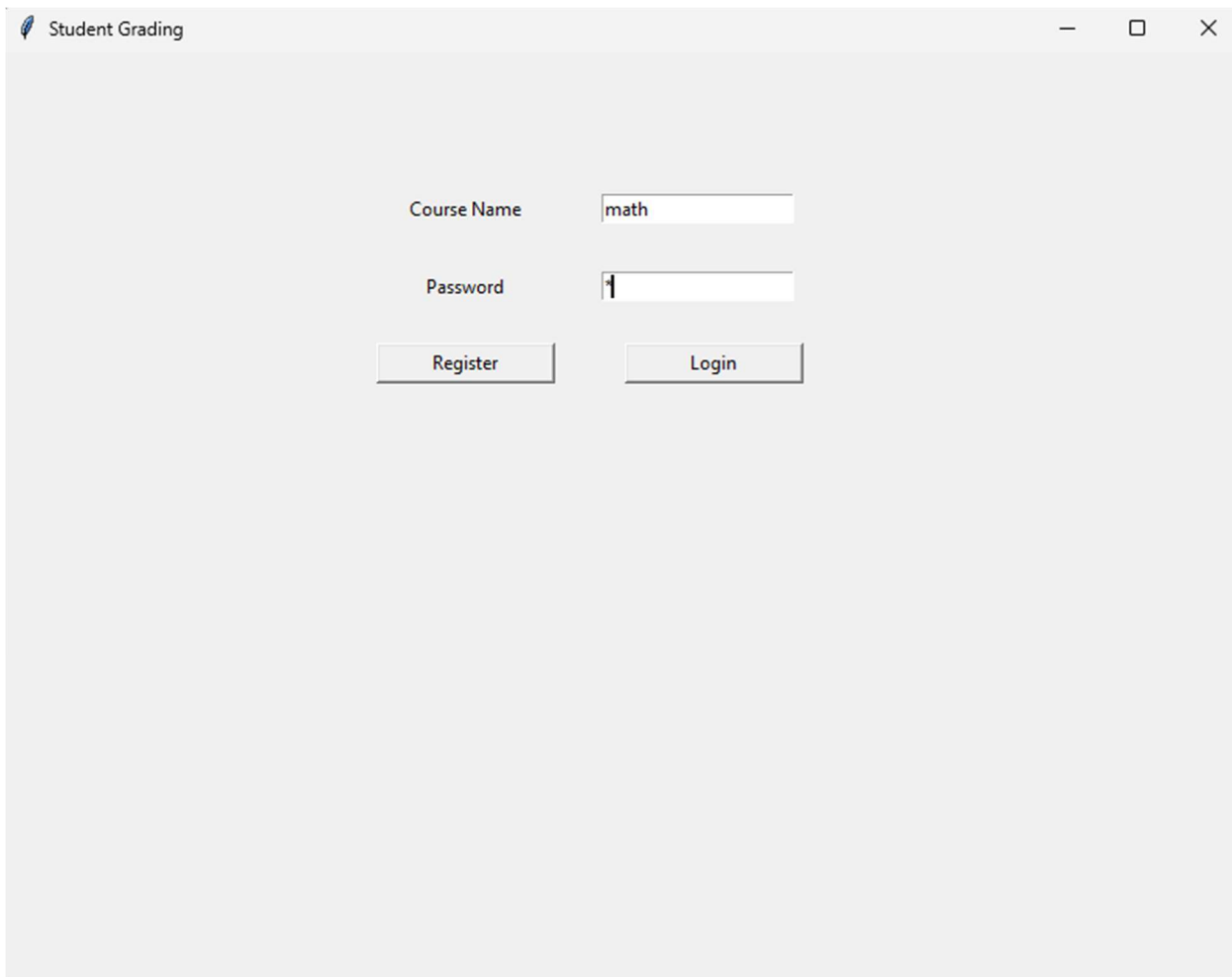
Password

Confirm Password

Back

Register





A screenshot of a software application window titled "Student Grading". The window has a light gray background and standard window controls (minimize, maximize, close) in the top right corner. In the center, there is a registration/login form. The form consists of two text input fields: "Course Name" with the value "math" and "Password" with a single character "1". Below these fields are two buttons: "Register" and "Login".

Student Grading

Course Name

Password

Login Page:

1. Checks fields
2. Checks credentials.

Student Grading

Student

OM
TANMAY

Add student

Name	Type	Student Score
HW1	HW	
HW2	HW	
T1	TEST	

Add Assignment

Delete Assignment

Score (100)

Confirm Score

Delete Score

Back

Calculate Class Grades

Main Page:

1. Course is now selected.
2. You can add students to the course.
3. Scores of any student can be displayed by selecting the student
4. Assignments can be added and deleted by selecting any assignment.
5. Scores can be added and modified by selecting any student assignment combination.
6. Class grade can be calculated by clicking on the button

Student
OM
TANMAY

Name	Type	Student Score
HW1	HW	98
HW2	HW	88
T1	TEST	92

Score (100)

Student Grading

Student

OM

TANMAY

Name	Type	Student Score
HW1	HW	98
HW2	HW	88
T1	TEST	92

Student Information

Student ID:

R12

Student Name:

BOB

Submit

Score (100)

Confirm Score

Delete Score

Delete Assignment

Back

Calculate Class Grades

Student Grading

Student

OM

TANMAY

BOB

Name

HW1

HW2

T1

Type

HW

HW

TEST

Student Score

None

None

None

Assignment Information

Assignment Name:

☐ HW

☒ TEST

Score

Delete Assignment

Delete Score

Back

Calculate Class Grades

Student Grading

Student

OM

TANMAY

BOB

Add student

Name	Type	Student Score
HW1	HW	68
HW2	HW	99
T1	TEST	100
TEST2	TEST	None

Add Assignment

Delete Assignment

Score (100) 99

Delete Score

Back

Calculate Class Grades

Alert

Select an assignment

OK

Student
OM
TANMAY
BOB

Name	Type	Student Score
HW1	HW	None
HW2	HW	None
T1	TEST	None
TEST2	TEST	None

Score (100)

Student Grading

Student

OM

TANMAY

BOB

Add student

Score (100)

99

Back

Name	Type	Student Score
HW1	HW	68
HW2	HW	5
T1	TEST	100
TEST2	TEST	None

Modify?

?

Record Exists. Do you wish to modify it?

Yes

No

Confirm Score

Delete Score

Calculate Class Grades

Student
OM
TANMAY
BOB

Name	Type	Student Score
HW1	HW	68
HW2	HW	99
T1	TEST	100
TEST2	TEST	None

Score (100)

Student Name	HW %	TEST %	Overall %	Overall Grade
OM	93.0	92.0	92.30000000000001	A
TANMAY	36.5	100.0	80.95	B
BOB	43.0	63.5	57.34999999999999	F

[Back](#)

Student Grading

Student

OM

TANMAY

BOB

Add student

Name	Type	Student Score
HW1	HW	98
HW2	HW	88
T1	TEST	92
TEST2	TEST	None

Add AssignmentDelete Assignment

Score (100)

1001

ore

Delete Score

Back

Calculate Class Grades

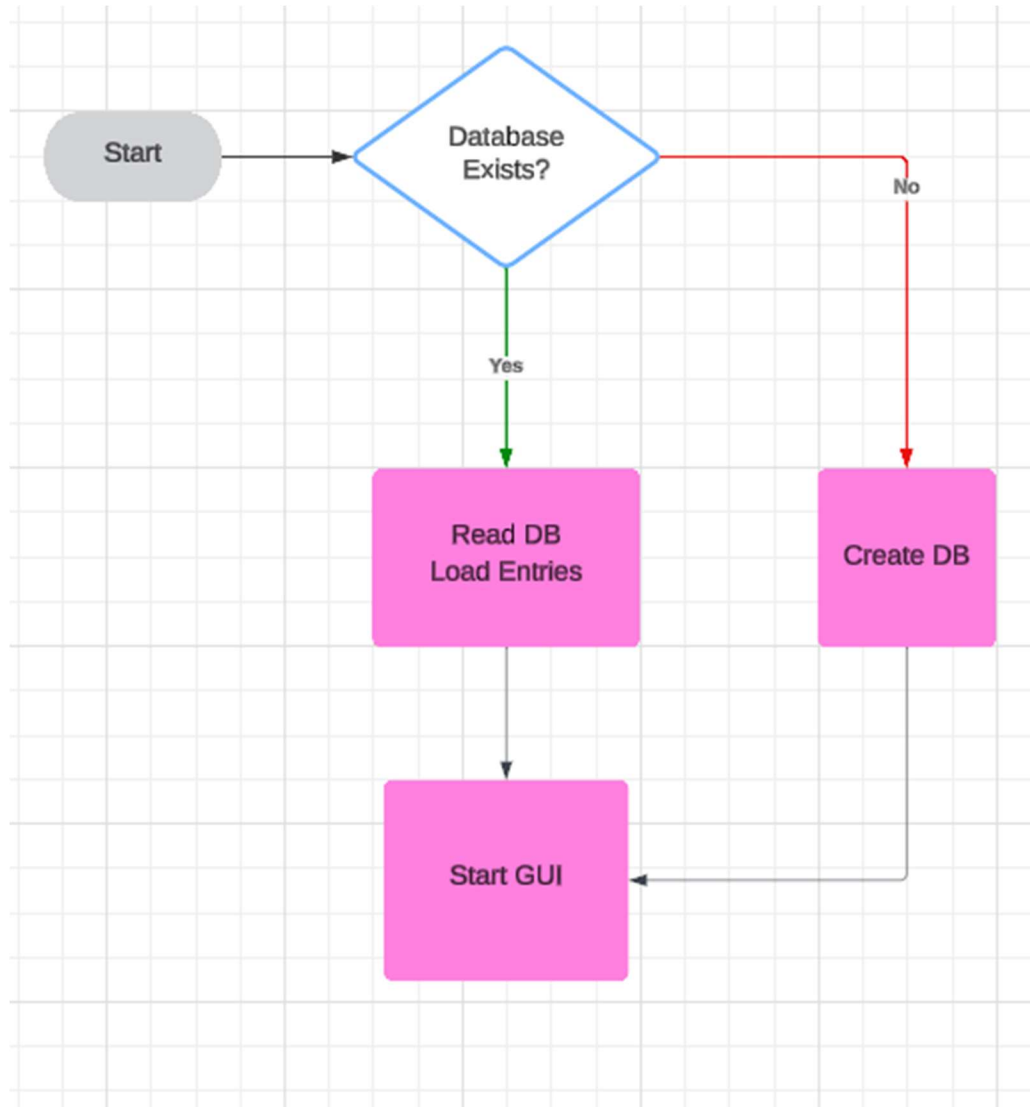
Alert

Invalid Score!

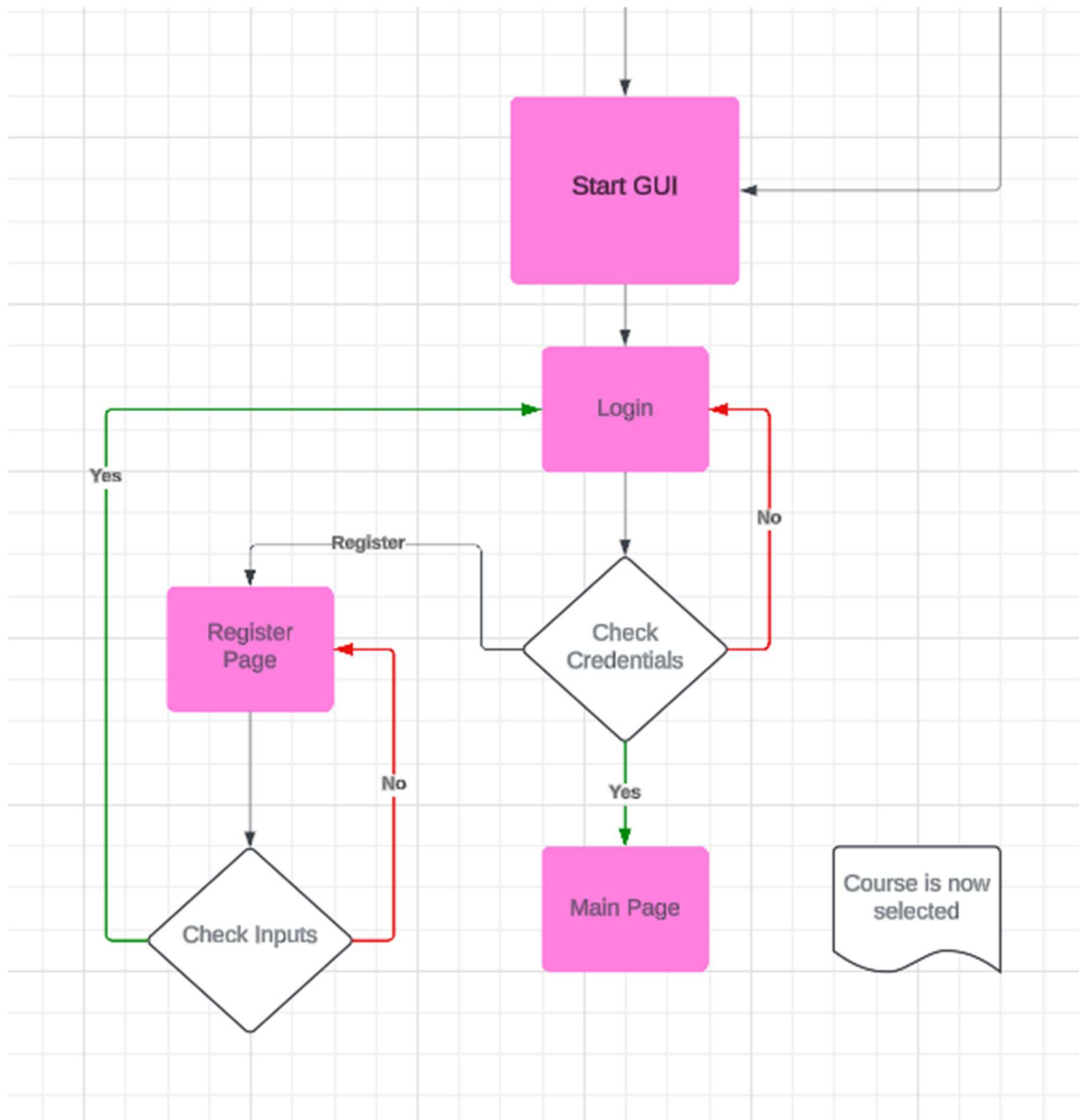
OK

Flowcharts

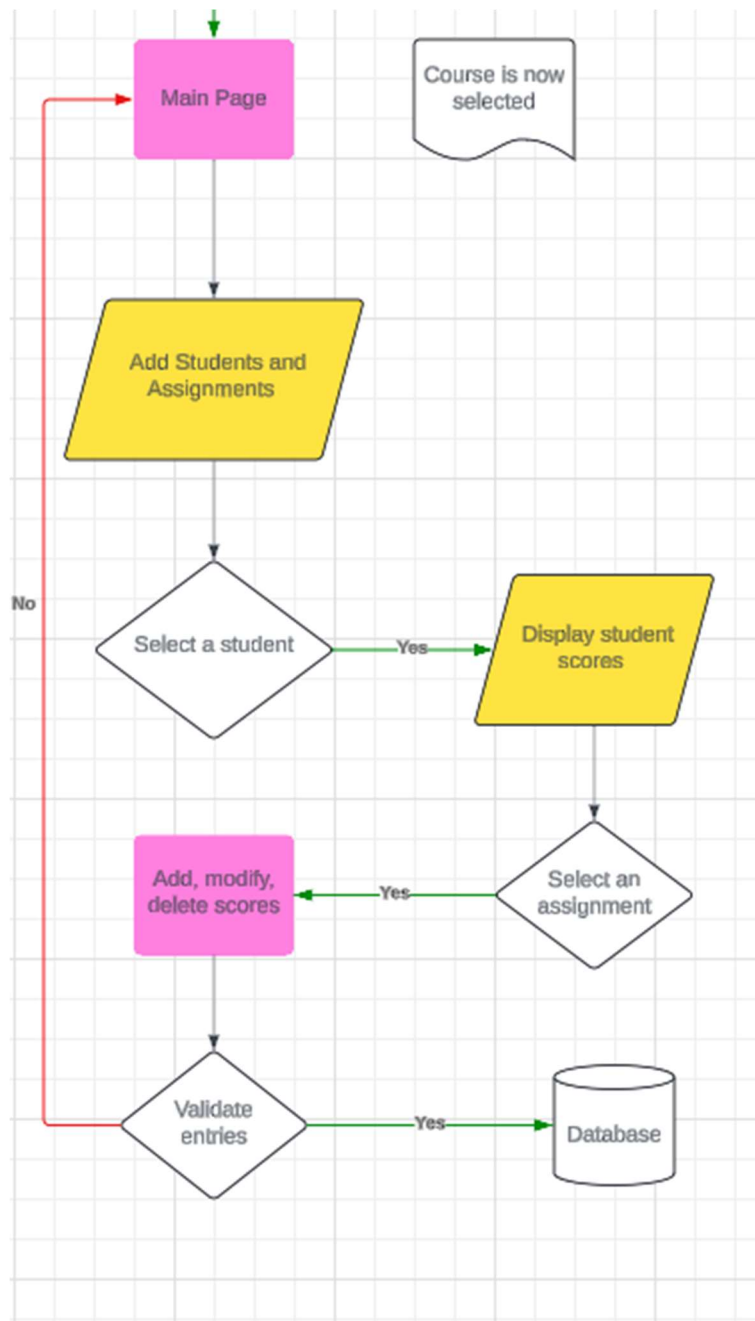
Startup



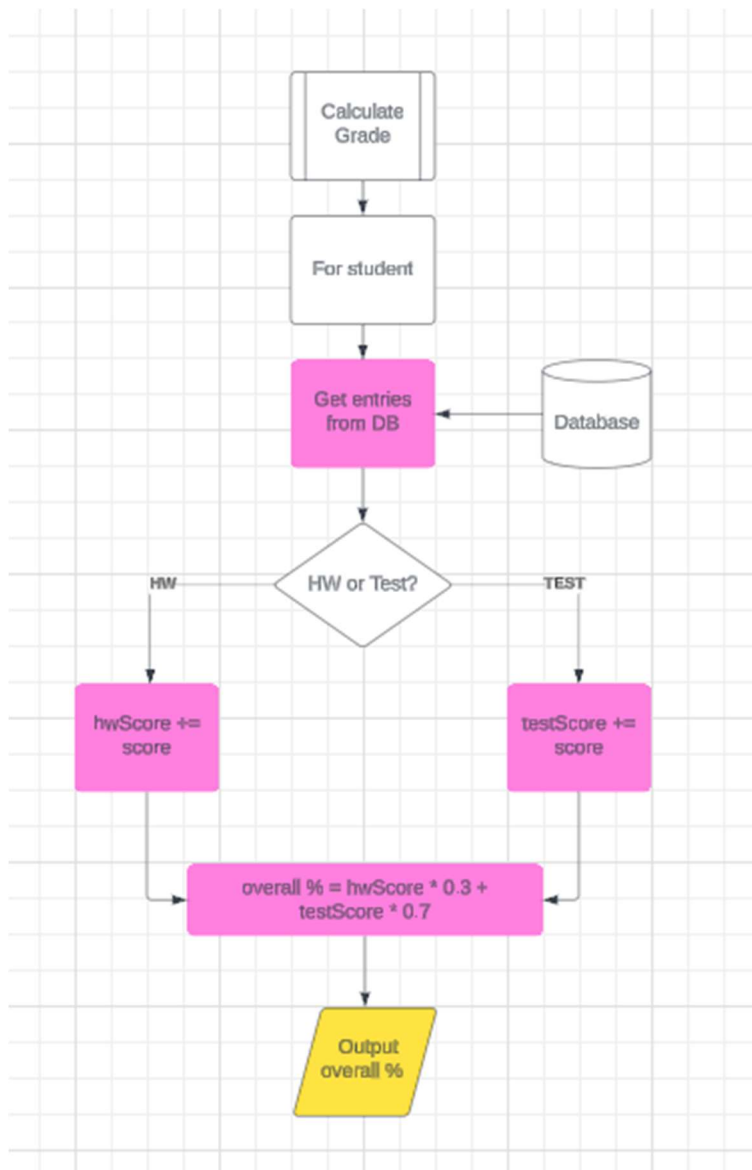
GUI Flow

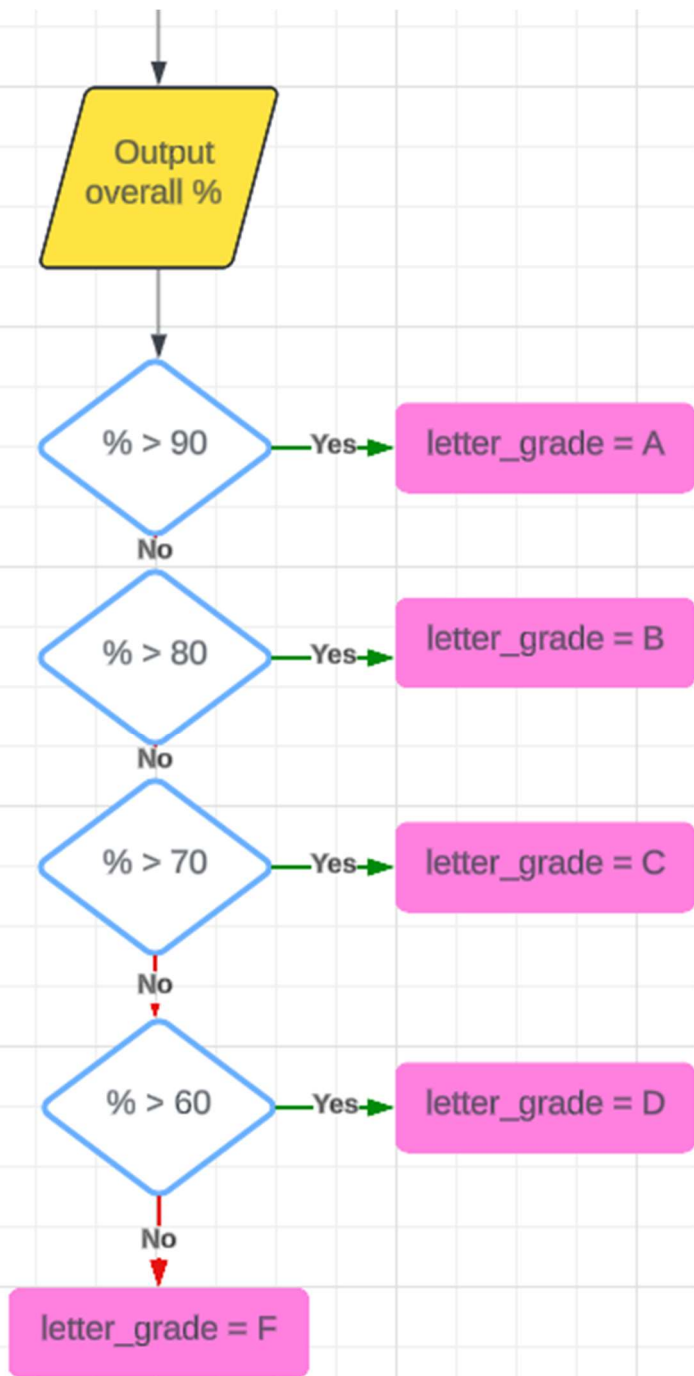


Insert Data



Calculations





Installer Script

```
cd\  
$Dirname="c:\GradingApp"  
if (-NOT (Test-Path $Dirname))  
{md GradingApp}
```

```
Copy-Item -Path "$home\Downloads\4250" -Destination "c:\GradingApp" -Recurse
```

```
cd GradingApp
```

```
$SourceFilePath = "c:\GradingApp\4250\dist\main.exe"  
$ShortcutPath = "$home\Desktop\Grades.lnk"  
$WScriptObj = New-Object -ComObject ("WScript.Shell")  
$shortcut = $WscriptObj.CreateShortcut($ShortcutPath)  
$shortcut.TargetPath = $SourceFilePath  
$shortcut.Save()
```

History of Scripting Languages

Origins:

- Early Scripts: Scripting languages have roots in early computers where batch processing scripts were used to automate tasks. Examples include shell scripts and job control languages in mainframe systems.
- Shell Scripts: In the 1960s, UNIX introduced shell scripts, which allowed users to automate the command line interface. The Bourne Shell, created in the late 1970s, is a notable example.

1980s-1990s:

- Perl (1987): Created by Larry Wall, Perl emerged as a versatile scripting language with powerful text processing capabilities.
- Tcl (1988): John Ousterhout developed Tcl for embedding within C programs and for rapid prototyping.
- JavaScript (1995): Brendan Eich created JavaScript for web browsers, which became the standard scripting language for web pages.
- PHP (1995): Rasmus Lerdorf initially designed PHP as a simple set of Common Gateway Interface (CGI) binaries for web development.

History of Python

Early Development:

- 1980s: Python's development began in the late 1980s. Guido van Rossum, working at the Centrum Wiskunde & Informatica (CWI) in the Netherlands, started working on Python as a successor to the ABC language.
- 1991: Python 0.9.0 was released to the public, including features like exception handling and functions.

Python 1.x Era:

- 1994: Python 1.0 released with functional programming tools like lambda, map, filter, and reduce.
- 1995-2000: Python grew in popularity and was adopted by various open-source communities. Python 1.6, the last 1.x version, was released in 2000.

Python 2.x Era:

- 2000: Python 2.0 introduced features like list comprehensions and garbage collection.
- 2008: Python 2.6 was released with features for forward compatibility with Python 3.

Python 3.x Era:

- 2008: Python 3.0 (Python 3000) was released, aiming to rectify design flaws and remove redundancies. However, it introduced backward incompatibilities that made it challenging for many projects to transition from Python 2.x.
- 2019-2020: Python 2 reached its end of life, officially making Python 3.x the only supported version.

Comparison of Languages

Readability:

1. Python:

- Python emphasizes readability with its clean and concise syntax. Its code is often more readable due to its use of indentation to denote blocks of code.

2. Java:

- Java also prioritizes readability, though its syntax can be more verbose compared to Python. However, its strong type system and explicit declarations can enhance readability in larger projects.

3. JavaScript:

- JavaScript's readability can vary based on coding style and project structure. Its flexible nature allows for concise code, but this can sometimes lead to less readable code if not properly structured.

4. C++:

- C++ syntax tends to be more complex compared to the other languages listed. Its use of pointers, memory management, and various language features can make code less readable, especially for beginners.

Writeability:

1. Python:

- Python excels in writeability due to its simplicity and ease of use. Its high-level abstractions and dynamic typing allow developers to write code quickly and with fewer lines compared to other languages.

2. Java:

- Java offers good writeability, though it can be more verbose compared to Python. However, its strong type system and comprehensive standard library provide developers with tools for efficient development.

3. JavaScript:

- JavaScript is highly writeable, especially for web development. Its lightweight syntax and dynamic typing enable rapid prototyping and iteration.

4. C++:

- C++ offers writeability, but it can be more challenging due to its lower-level constructs and stricter syntax. Writing efficient and correct C++ code often requires more effort compared to higher-level languages like Python or JavaScript.

Reliability:

1. Python:

- Python emphasizes simplicity and readability, which can contribute to reliability. Its extensive standard library and automatic memory management reduce the likelihood of common programming errors.

2. Java:

- Java's strong type system and compile-time checks enhance reliability. Its robust exception handling and garbage collection contribute to stable and predictable performance.

3. JavaScript:

- JavaScript's reliability can vary depending on the environment and runtime. While modern JavaScript engines provide optimizations and error handling, the dynamic nature of the language can lead to runtime errors if not properly managed.

4. C++:

- C++ offers high performance and control but requires more careful management of resources, which can increase the risk of memory leaks, segmentation faults, and other runtime errors compared to higher-level languages.

Cost:

1. Python:

- Python is free and open-source, making it cost-effective for development. There are no licensing fees, and a wide range of free libraries and frameworks are available for various tasks.

2. Java:

- Java is free to use, but certain enterprise features and tools may require licensing fees. However, the availability of open-source alternatives and the extensive Java ecosystem mitigate costs for many projects.

3. JavaScript:

- JavaScript is free and widely supported across web browsers and platforms. Development tools and frameworks are often open-source, reducing costs associated with building web applications.

4. C++:

- C++ compilers and tools are typically free to use, but developing and maintaining C++ projects can be costlier due to the need for highly skilled developers and the potential for complex debugging and optimization processes. Additionally, C++ may require more hardware resources for efficient execution compared to higher-level languages.

Space Cost:

1. Python:

- Python is an interpreted language, which means it typically requires more memory than compiled languages like C++ or Java. Additionally, Python's dynamic typing and runtime overhead can increase memory usage compared to statically typed languages.

2. Java:

- Java programs run on the Java Virtual Machine (JVM), which can consume significant memory, especially for larger applications. However, JVM optimizations and garbage collection mechanisms help manage memory efficiently.

3. JavaScript:

- JavaScript code executed in web browsers generally has lower memory overhead compared to desktop or server applications. However, memory management in JavaScript can be less predictable due to automatic garbage collection and the potential for memory leaks in complex web applications.

4. C++:

- C++ offers more control over memory management compared to higher-level languages, which can lead to more efficient memory usage. However, manual memory management in C++ can also introduce the risk of memory leaks and other memory-related issues if not handled carefully.

Time Cost:

1. Python:

- Python is often slower than compiled languages like C++ or Java due to its interpreted nature and dynamic typing. While Python's high-level abstractions and built-in data structures contribute to developer productivity, they can also result in slower execution speed for performance-critical applications.

2. Java:

- Java programs typically have faster execution speeds compared to Python due to the Just-In-Time (JIT) compilation performed by the JVM. Additionally, Java's static typing and optimizations at runtime contribute to improved performance for certain types of applications.

3. JavaScript:

- JavaScript execution speed can vary depending on the runtime environment and the efficiency of the JavaScript engine. Modern JavaScript engines like V8 (used in Chrome) and SpiderMonkey (used in Firefox) employ Just-In-Time compilation and other optimizations to improve performance, but JavaScript may still be slower than compiled languages for CPU-intensive tasks.

4. C++:

- C++ is known for its high performance and low-level control, making it well-suited for applications where speed is critical. C++ code is typically compiled directly to machine code, resulting in efficient execution and minimal runtime overhead. However, the trade-off for this performance is often increased development time and complexity compared to higher-level languages.

Illustrations and explanations of the language:

i. Syntax

Python's syntax is known for its simplicity and readability. It uses indentation to define blocks of code rather than curly braces like many other programming languages.

ii. Control structures

Python supports common control structures like if-else statements, loops (for and while), and also has additional constructs like list comprehensions.

```
if overall_grade_percentage >= 90:
    letter_grade = "A"
elif overall_grade_percentage >= 80:
    letter_grade = "B"
elif overall_grade_percentage >= 70:
    letter_grade = "C"
elif overall_grade_percentage >= 60:
    letter_grade = "D"
else:
    letter_grade = "F"
```

iii. Data structures

Python provides built-in data structures like lists, tuples, dictionaries, and sets. These are fundamental for organizing and manipulating data efficiently.

```
class Assignment:
    def __init__(self, assignmentId, name, type):
        self.assignmentId = assignmentId
        self.name = name
        self.type = type
```

iv. I/O

Python supports various methods for input and output operations, including reading from and writing to files, console input/output, and more advanced operations like serialization. Input is taken using GUI for this code

v. Parameter passing methods

In Python, parameters can be passed to functions either by value (immutable objects like numbers, strings, tuples) or by reference (mutable objects like lists, dictionaries). However, Python doesn't have explicit mechanisms like pass-by-value or pass-by-reference.

```
def alertMsg(inpStr):
    messagebox.showerror("Error", inpStr)
```


vi. Scope rules

Python follows the LEGB (Local, Enclosing, Global, Built-in) rule for variable scope resolution.

vii. Stack/Heap – Constructors/Destructors

Python handles memory management internally through its memory manager. Objects are created on the heap and Python automatically deallocates memory when an object is no longer referenced (garbage collection).

```
st = Student (student_name, student_id)
```

viii. Error handling procedures

Python provides try-except blocks for handling exceptions. This allows you to gracefully handle errors that might occur during program execution.

```
if not username or not password or not conPassword:
    alertMsg("All fields are mandatory!")
    return 1
elif not username.isalnum():
    alertMsg("Course Name must be Alphanumeric")
    return 2
elif g1.getFirstRecord(f"SELECT ID FROM USERS WHERE USERNAME = '{username.upper()}"):
    alertMsg("Course Name is taken!")
    return 2
elif password != conPassword:
    alertMsg("Password and Confirm Password do not match!")
    return 3
else:
    return 0
```

Appendix

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter import simpledialog
import sqlite3

# Global Variables
userId = 0
LARGE_FONT = ("Verdana", 20)

class Student:
    def __init__(self, name, rocketid):
        self.id = rocketid
        self.name = name

    def __repr__(self):
        return (f"Student name: {self.name}")

    def __str__(self):
        return (f"Student name: {self.name}")

class Assignment:
    def __init__(self, assignmentId, name, type):
        self.assignmentId = assignmentId
        self.name = name
        self.type = type

    def __repr__(self):
        return (f"Assignment name: {self.name}")

    def __str__(self):
        return (f"Assignment name: {self.name}")

class Course:
    def __init__(self, name):
        self.name = name
        self.students = []
        self.assignments = []
        self.homeworks = []
        self.tests = []

    def add_student(self, student):
        self.students.append(student)
```

```

    g1.modifyRecord(f"INSERT INTO STUDENTS (COURSE, STUDENT_ID, STUDENT_NAME) VALUES
('{self.name}', '{student.id}', '{student.name}')"

def remove_student(self, student):
    self.students.remove(student)
    g1.modifyRecord(f"DELETE FROM STUDENTS WHERE COURSE = '{self.name}' AND
STUDENT_NAME = '{student.name}'")

def add_assignment(self, assignment):
    self.assignments.append(assignment)
    g1.modifyRecord(f"INSERT INTO ASSIGNMENTS (COURSE, ASSIGNMENT_NAME,
ASSIGNMENT_TYPE) VALUES ('{self.name}', '{assignment.name}', '{assignment.type}')"
    if assignment.type == "HW":
        self.homeworks.append(assignment)
    if assignment.type == "TEST":
        self.tests.append(assignment)

def remove_assignment(self, assignment):
    self.assignments.remove(assignment)
    g1.modifyRecord(f"DELETE FROM ASSIGNMENTS WHERE COURSE = '{self.name}' AND
ASSIGNMENT_NAME = '{assignment.name}'")
    g1.modifyRecord(f"DELETE FROM SCORES WHERE COURSE = '{self.name}' AND
ASSIGNMENT_NAME = '{assignment.name}'")

def __repr__(self):
    return (f"Course name: {self.name}")

def __str__(self):
    return (f"Course name: {self.name}")

class GradeManager:
    def __init__(self):
        self.courses = []
        self.selectedCourse = Course

    # Create the Users table if it does not exist
    def createUserTable(self):
        try:
            conn.execute("CREATE TABLE USERS \
                (ID      INTEGER  PRIMARY KEY AUTOINCREMENT, \
                USERNAME  TEXT   NOT NULL UNIQUE, \
                PASSWORD  TEXT   NOT NULL); ")
            print("Table created successfully")

```

```
except:
    print("User table exists")
```

Create the Students table if it does not exist

```
def createStudentsTable(self):
```

```
    try:
        conn.execute("CREATE TABLE STUDENTS \
            (ID      INTEGER  PRIMARY KEY AUTOINCREMENT, \
            COURSE   TEXT    NOT NULL, \
            STUDENT_ID TEXT    NOT NULL, \
            STUDENT_NAME TEXT  NOT NULL); ")
        print("Table created successfully")
```

```
    except:
        print("Students table exists")
```

Create the Assignments table if it does not exist

```
def createAssignmentsTable(self):
```

```
    try:
        conn.execute("CREATE TABLE ASSIGNMENTS \
            (ID      INTEGER  PRIMARY KEY AUTOINCREMENT, \
            COURSE   TEXT    NOT NULL, \
            ASSIGNMENT_NAME TEXT  NOT NULL, \
            ASSIGNMENT_TYPE TEXT  NOT NULL); ")
        print("Table created successfully")
```

```
    except:
        print("Assignments table exists")
```

Create the Grades table if it does not exist

```
def createScoresTable(self):
```

```
    try:
        conn.execute("CREATE TABLE SCORES \
            (USER_ID INTEGER, \
            RECORD_ID INTEGER PRIMARY KEY AUTOINCREMENT, \
            COURSE TEXT, \
            STUDENT_ID TEXT, \
            STUDENT_NAME TEXT, \
            ASSIGNMENT_NAME TEXT, \
            ASSIGNMENT_TYPE TEXT, \
            SCORE INTEGER, \
            TIMESTAMP DATETIME DEFAULT CURRENT_TIMESTAMP, \
            FOREIGN KEY (USER_ID) REFERENCES USERS(ID)); ")
        print("Table created successfully")
```

```
    except:
        print("Grades table exists")
```

Initialization

```
def initialize(self):
    # Scans grades table, adds courses, students, and assignments found.
    courses = g1.getRecords(
        f"Select username from users")
    print(courses)
    for crs in courses:
        g1.courses.append(Course(crs[0].upper()))

    students = g1.getRecords(
        f"Select * from students")
    print(students)
    for st in students:
        cr = g1.get_objects(st[1], "", "")
        sto = Student(st[3], st[2])
        cr[0].students.append(sto)

    assignments = g1.getRecords(
        f"Select * from assignments")
    print(assignments)
    for assi in assignments:
        cr = g1.get_objects(assi[1], "", "")
        assign = Assignment("", assi[2], assi[3])
        cr[0].assignments.append(assign)
```

Database Functions

```
def modifyRecord(self, inpStr):
    conn.execute(f"{inpStr}")
    conn.commit()
```

```
def getRecords(self, inpStr):
    cur = conn.cursor()
    cur.execute(f"{inpStr}")

    rows = cur.fetchall()
    print(rows)
    return rows
```

```
def getFirstRecord(self, inpStr):
    cur = conn.cursor()
    cur.execute(f"{inpStr}")

    rows = cur.fetchall()
    print(rows)
```

```

for row in rows:
    return row[0]

def get_objects(self, course_name, student_name, assignment_name):
    course_name, student_name, assignment_name = course_name.upper(), student_name.upper(),
assignment_name.upper()
    g1.selectedCourse = next((course for course in g1.courses if course.name == course_name), None)

    student = next((student for student in g1.selectedCourse.students if student.name ==
student_name), None)
    assignment = next((assignment for assignment in g1.selectedCourse.assignments if
assignment.name == assignment_name), None)
    print(g1.selectedCourse, student, assignment)
    return [g1.selectedCourse, student, assignment]

def grade_assignment(self, course_name, student_name, assignment_name, score):
    course, student, assignment = g1.get_objects(course_name, student_name, assignment_name)
    print(course.name, student.id, student.name, assignment.name, assignment.type, score)

    g1.modifyRecord(f"INSERT INTO SCORES (COURSE, STUDENT_ID, STUDENT_NAME,
ASSIGNMENT_NAME, ASSIGNMENT_TYPE, SCORE ) VALUES ({course.name}, {student.id},
'{student.name}', '{assignment.name}', '{assignment.type}', {score})")

def calculate_grade(self, course_name, student_name):
    objs = g1.get_objects(course_name, student_name, "")
    course = objs[0]
    print(course.homeworks, course.tests)
    hw_total = 0

    test_total = 0

    overall_grade_percentage = 0

    hw_scores = g1.getRecords(f"SELECT SCORE FROM SCORES WHERE COURSE = '{course_name}'
AND STUDENT_NAME = '{student_name}' AND ASSIGNMENT_TYPE = 'HW'")
    total_hws = len(hw_scores) * 100
    test_scores = g1.getRecords(f"SELECT SCORE FROM SCORES WHERE COURSE =
'{course_name}' AND STUDENT_NAME = '{student_name}' AND ASSIGNMENT_TYPE = 'TEST'")
    total_tests = len(test_scores) * 100
    hw_per = 0
    test_per = 0

    for hwscore in hw_scores:
        hw_total += hwscore[0]

```

```

for tscore in test_scores:
    test_total += tscore[0]
print(hw_total, test_total)

if len(hw_scores) == 0 and len(test_scores) == 0:
    hw_per = None
    test_per = None
    overall_grade_percentage = 0
elif len(hw_scores) == 0:
    hw_per = None
    test_per = (test_total / total_tests) * 100
    overall_grade_percentage = (test_total / total_tests)
elif len(test_scores) == 0:
    hw_per = hw_total / total_hws * 100
    test_per = None
    overall_grade_percentage = (hw_total / total_hws) * 100
else:
    overall_grade_percentage = ((hw_total / total_hws) * 0.3 + (test_total / total_tests) * 0.7) * 100
    hw_per = hw_total / total_hws * 100
    test_per = test_total / total_tests * 100
    print((hw_total / total_hws), (test_total / total_tests))

```

```

if overall_grade_percentage >= 90:
    letter_grade = "A"
elif overall_grade_percentage >= 80:
    letter_grade = "B"
elif overall_grade_percentage >= 70:
    letter_grade = "C"
elif overall_grade_percentage >= 60:
    letter_grade = "D"
else:
    letter_grade = "F"

print("Grades", student_name, hw_per, test_per, overall_grade_percentage, letter_grade)
return student_name, hw_per, test_per, overall_grade_percentage, letter_grade

```

```

def calculate_overall_grade(self, course):
    courseGrades = []
    for student in course.students:
        courseGrades.append(g1.calculate_grade(course.name, student.name))

```

```
return courseGrades
```

```
# Create the main class and define show frame functions
```

```
class MyGrades(tk.Tk):
```

```
    def __init__(self, *args, kwargs):
```

```
        tk.Tk.__init__(self, *args, kwargs)
```

```
        container = tk.Frame(self)
```

```
        container.pack(side="top", fill="both", expand=True)
```

```
        container.grid_rowconfigure(0, weight=1)
```

```
        container.grid_columnconfigure(0, weight=1)
```

```
        self.frames = {}
```

```
        for F in (LoginPage, MainPage, RegistrationPage, GradesPage):
```

```
            frame = F(container, self)
```

```
            self.frames[F] = frame
```

```
            frame.grid(row=0, column=0, sticky="nsew")
```

```
        self.show_frame(LoginPage)
```

```
    def show_frame(self, cont):
```

```
        frame = self.frames[cont]
```

```
        frame.tkraise()
```

```
    def get_page(self, page_class):
```

```
        return self.frames[page_class]
```

```
# Login Page
```

```
class LoginPage(tk.Frame):
```

```
    def __init__(self, parent, controller):
```

```
        tk.Frame.__init__(self, parent)
```

```
    # Checks whether username and password are correct
```

```
    def checkPassword():
```

```
        # Check if user exists in database
```

```
        username = usernameEntry.get().upper()
```

```
        password = passwordEntry.get()
```

```
        print(username, password)
```



```

        if g1.getFirstRecord(f"SELECT ID FROM USERS WHERE USERNAME = '{username}' AND
PASSWORD = '{password}'"):
            return True
        else:
            return False

# Prints whether login is successful or not to the app
def sucUnsuc():
    if checkPassword():
        # If authenticated
        username = usernameEntry.get()
        global userId
        userId = g1.getFirstRecord(f"Select id from users where username = '{username}'")
        g1.selectedCourse = g1.get_objects(username, "", "")[0]
        print(g1.selectedCourse)
        usernameEntry.delete(0, 'end')
        passwordEntry.delete(0, 'end')
        label1 = tk.Label(self, text=" ", width=30)
        canvas1.create_window(200, 250, window=label1)
        View()
        controller.show_frame(MainPage)
    else:
        # If authentication fails
        label1 = tk.Label(self, text="Invalid credentials!")
        canvas1.create_window(200, 250, window=label1)

# Login Page Layout:
canvas1 = tk.Canvas(self, width=400, height=400)
canvas1.pack()

# Username
usernameLabel = tk.Label(self, text="Course Name")
canvas1.create_window(100, 100, window=usernameLabel)
usernameEntry = tk.Entry(self)
canvas1.create_window(250, 100, window=usernameEntry)

# Password
passwordLabel = tk.Label(self, text="Password")
canvas1.create_window(100, 150, window=passwordLabel)
passwordEntry = tk.Entry(self, show="*")
canvas1.create_window(250, 150, window=passwordEntry)

# Login button
loginButton = tk.Button(self, width=15, text="Login", command=sucUnsuc)

```

```

canvas1.create_window(260, 200, window=loginButton)

# Register button
registerButton = tk.Button(self, width=15, text="Register", command=lambda:
controller.show_frame(RegistrationPage))
canvas1.create_window(100, 200, window=registerButton)

# Registration Page
class RegistrationPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

    def alertMsg(inpStr):
        messagebox.showerror("Error", inpStr)
        # label2 = tk.Label(self, text=inpStr)
        # canvas1.create_window(200, 300, window=label2)

    # Checks whether username exists and passwords match
    def regCheckPassword():
        username = usernameEntry.get()
        password = passwordEntry.get()
        conPassword = passwordConEntry.get()
        print(username, password, conPassword)

        if not username or not password or not conPassword:
            alertMsg("All fields are mandatory!")
            return 1
        elif not username.isalnum():
            alertMsg("Course Name must be Alphanumeric")
            return 2
        elif g1.getFirstRecord(f"SELECT ID FROM USERS WHERE USERNAME = '{username.upper()}"):
            alertMsg("Course Name is taken!")
            return 2
        elif password != conPassword:
            alertMsg("Password and Confirm Password do not match!")
            return 3
        else:
            return 0

    # Prints whether registration is successful or not to the app
    def regSucUnsuc():
        username = usernameEntry.get().upper()
        password = passwordEntry.get()
        if regCheckPassword() == 0:

```

```

# Initialize records in both tables
g1.modifyRecord(f"INSERT INTO USERS (USERNAME, PASSWORD) VALUES ('{username}',
'{password}')"')
newId = g1.getFirstRecord(f"Select id from users where username = '{username}'")
# g1.modifyRecord(
#   f"INSERT INTO SCORES (USER_ID, COURSE) VALUES ({newId}, '{username}')"')

# Add course to courses
g1.courses.append(Course(username))
print(g1.courses)

# Clear the form
usernameEntry.delete(0, 'end')
passwordEntry.delete(0, 'end')
passwordConEntry.delete(0, 'end')

# Go back to login Page
controller.show_frame(LoginPage)

# Registration Page Layout:
canvas1 = tk.Canvas(self, width=400, height=300)
canvas1.pack()

# Username
usernameLabel = tk.Label(self, text="Course Name")
canvas1.create_window(100, 100, window=usernameLabel)
usernameEntry = tk.Entry(self)
canvas1.create_window(250, 100, window=usernameEntry)

# Password
passwordLabel = tk.Label(self, text="Password")
canvas1.create_window(100, 150, window=passwordLabel)
passwordEntry = tk.Entry(self, show="*")
canvas1.create_window(250, 150, window=passwordEntry)

# Confirm Password
passwordConLabel = tk.Label(self, text="Confirm Password")
canvas1.create_window(100, 200, window=passwordConLabel)
passwordConEntry = tk.Entry(self, show="*")
canvas1.create_window(250, 200, window=passwordConEntry)

# Register button
registerButton = tk.Button(self, width=15, text="Register", command=regSucUnsuc)
canvas1.create_window(260, 250, window=registerButton)

```

```

# Back button
backButton = tk.Button(self, width=15, text="Back", command=lambda:
controller.show_frame(LoginPage))
canvas1.create_window(100, 250, window=backButton)

# Main Page
class MainPage(tk.Frame):
    def __init__(self, parent, controller):

        # Table
        global View
        def View():
            print("Building Tree...")
            print(g1.selectedCourse)

            for row in student_tree.get_children():
                student_tree.delete(row)

            for student in g1.selectedCourse.students:
                student_tree.insert(parent = '', index='end', values= student.name)

            for row in assignment_tree.get_children():
                assignment_tree.delete(row)

            for assignment in g1.selectedCourse.assignments:
                assignment_tree.insert(parent = '', index='end', values= (assignment.name, assignment.type))

        def clearView():
            for row in student_tree.get_children():
                student_tree.delete(row)
            for row in assignment_tree.get_children():
                assignment_tree.delete(row)
            scoreEntry.delete(0, 'end')

        def alertMsg(myString):
            messagebox.showerror("Alert", myString)
            print(myString)

        # Check whether all entries in the form are valid
        def checkForm():
            selected_student = student_tree.item(student_tree.focus(), 'values')
            selected_assignment = assignment_tree.item(assignment_tree.focus(), 'values')
            score = scoreEntry.get()
            if not selected_student:
                alertMsg("Please select a student")

```

```

    return 1
elif not selected_assignment:
    alertMsg("Please select an Assignment")
    return 1

try:
    score = int(score)
    if score < 0 or score > 100:
        alertMsg("Invalid Score!")
        return 1
    else:
        print(selected_student, selected_assignment, score)
        return 0
except:
    alertMsg("Invalid Score!")
    return 1

def submitForm():
    if checkForm() == 0:
        student_name = student_tree.item(student_tree.focus(), 'values')[0]
        assignment_name = assignment_tree.item(assignment_tree.focus(), 'values')[0]
        score = scoreEntry.get()
        print(student_name, assignment_name, score)
        print("Submitted successfully")

        existing = g1.getFirstRecord(
            f"Select record_id from scores where course = '{g1.selectedCourse.name}' AND
STUDENT_NAME = '{student_name}' AND ASSIGNMENT_NAME = '{assignment_name}'")
        print(existing)

        if not existing:
            g1.grade_assignment(g1.selectedCourse.name, student_name, assignment_name, score)
            select_item(0)
        else:
            print("Record exists")
            res = messagebox.askyesno("Modify?", "Record Exists. Do you wish to modify it?")
            print(res)
            if res:
                g1.modifyRecord(f"UPDATE SCORES SET SCORE = {score} WHERE COURSE =
'{g1.selectedCourse.name}' AND STUDENT_NAME = '{student_name}' AND ASSIGNMENT_NAME =
'{assignment_name}'")
                View()

```

```

def logMeOut():
    # Clear all displayed data and go to Login Page
    clearView()
    controller.show_frame(LoginPage)

def delScore():
    selected_student = student_tree.item(student_tree.focus(), 'values')
    selected_assignment = assignment_tree.item(assignment_tree.focus(), 'values')

    if not selected_student:
        alertMsg("Please select a student")
        return 1
    elif not selected_assignment:
        alertMsg("Please select an Assignment")
        return 1
    else:
        print(selected_student, selected_assignment)
        if selected_assignment[2] == 'None':
            print("Record does not exist")
            return 1
        else:
            g1.modifyRecord(f"DELETE FROM SCORES WHERE COURSE = '{g1.selectedCourse.name}'
AND STUDENT_NAME = '{selected_student[0]}' AND ASSIGNMENT_NAME =
'{selected_assignment[0]}'")
            select_item(0)

def calcGrade():
    # Clear all displayed data and go to Login Page
    dispGrades()
    controller.show_frame(GradesPage)

def get_student_info():
    student_window = tk.Toplevel(self)
    student_window.title("Student Information")
    student_window.geometry("400x200")

    canvas = tk.Canvas(student_window, width=200, height=150)
    canvas.pack()

    label_id = tk.Label(student_window, text="Student ID:")
    canvas.create_window(25, 30, window=label_id)
    entry_id = tk.Entry(student_window)

```

```
canvas.create_window(150, 30, window=entry_id)
```

```
label_name = tk.Label(student_window, text="Student Name:")
```

```
canvas.create_window(25, 60, window=label_name)
```

```
entry_name = tk.Entry(student_window)
```

```
canvas.create_window(150, 60, window=entry_name)
```

```
def submitstu():
```

```
    try:
```

```
        student_id = entry_id.get()
```

```
        student_name = entry_name.get().upper()
```

```
        if student_id.isalnum() and student_name.isalpha():
```

```
            print("Student ID:", student_id)
```

```
            print("Student Name:", student_name)
```

```
            existing_students = []
```

```
            for st_name in g1.selectedCourse.students:
```

```
                existing_students.append(st_name.name)
```

```
            print(existing_students)
```

```
        if student_name not in existing_students:
```

```
            st = Student(student_name, student_id)
```

```
            course = g1.get_objects(g1.selectedCourse.name, "", "")
```

```
            course[0].add_student(st)
```

```
            student_window.destroy()
```

```
            View()
```

```
        else:
```

```
            error_label = tk.Label(student_window, text="Student Already Exists", fg="red")
```

```
            canvas.create_window(100, 150, window=error_label)
```

```
    else:
```

```
        error_label = tk.Label(student_window, text="Invalid Entries", fg="red")
```

```
        canvas.create_window(100, 150, window=error_label)
```

```
except:
```

```
    print("Invalid entries exception")
```

```
    error_label = tk.Label(student_window, text="Invalid Entries", fg="red")
```

```
    canvas.create_window(100, 150, window=error_label)
```

```
submit_button = tk.Button(student_window, text="Submit", command=submitstu)
```

```
canvas.create_window(100, 100, window=submit_button)
```

```
def get_assignment_info():
```

```
    student_window = tk.Toplevel(self)
```

```
    student_window.title("Assignment Information")
```

```
    student_window.geometry("400x200")
```

```

canvas = tk.Canvas(student_window, width=200, height=300)
canvas.pack()

label_name = tk.Label(student_window, text="Assignment Name:")
canvas.create_window(25, 30, window=label_name)
entry_name = tk.Entry(student_window)
canvas.create_window(150, 30, window=entry_name)

# Radio buttons
var = tk.StringVar()

radio_hw = tk.Radiobutton(student_window, text="HW", variable=var, value="HW")
canvas.create_window(50, 90, window=radio_hw)
radio_test = tk.Radiobutton(student_window, text="TEST", variable=var, value="TEST")
canvas.create_window(200, 90, window=radio_test)

def submitassi():
    try:
        assignment_name = entry_name.get().upper()
        assign_type = "HW"
        if assignment_name.isalnum() and var.get():
            print("Assign Name:", assignment_name)
            assign_type = var.get()
            existing_assignments = []
            for as_name in g1.selectedCourse.assignments:
                existing_assignments.append(as_name.name)
            print(existing_assignments)

            if assignment_name not in existing_assignments:
                assi = Assignment("", assignment_name, assign_type)
                course = g1.get_objects(g1.selectedCourse.name, "", "")
                course[0].add_assignment(assi)
                student_window.destroy()
                View()
            else:
                error_label = tk.Label(student_window, text="Assignment Already Exists", fg="red")
                canvas.create_window(100, 170, window=error_label)
        else:
            error_label = tk.Label(student_window, text="Invalid Entries", fg="red")
            canvas.create_window(100, 170, window=error_label)
    except:
        print("Invalid entries exception")
        error_label = tk.Label(student_window, text="Invalid Entries ex", fg="red")

```



```
canvas.create_window(100, 170, window=error_label)
```

```
submit_button = tk.Button(student_window, text="Submit", command=submitassi)  
canvas.create_window(100, 140, window=submit_button)
```

```
def delAssignment():  
    selected_assignment = assignment_tree.item(assignment_tree.focus(), 'values')  
    if not selected_assignment:  
        alertMsg("Select an assignment")  
        return 1  
    print(selected_assignment)  
    objs = g1.get_objects(g1.selectedCourse.name, "", selected_assignment[0])  
    objs[0].remove_assignment(objs[2])  
    View()
```

```
def select_item(a):  
    course_name = g1.selectedCourse.name  
    student_name = student_tree.item(student_tree.focus(), 'values')[0]  
    for row in assignment_tree.get_children():  
        assignment_tree.delete(row)  
  
    for assignment in g1.selectedCourse.assignments:  
        score = g1.getFirstRecord(f"SELECT SCORE FROM SCORES WHERE COURSE =  
'{course_name}' AND STUDENT_NAME = '{student_name}' AND ASSIGNMENT_NAME =  
'{assignment.name}'")  
        assignment_tree.insert(parent = '', index='end', values= (assignment.name, assignment.type,  
score))
```

```
tk.Frame.__init__(self, parent)
```

```
# Main Page Layout:
```

```
canvas1 = tk.Canvas(self, width=400, height=700)  
canvas1.pack()
```

```
# Student tree
```

```
student_tree = ttk.Treeview(self, column=("c1"), show='headings')  
student_tree.column("#1", anchor=tk.W, width=200)  
student_tree.heading("#1", text="Student")  
student_tree.bind('<ButtonRelease-1>', select_item)  
student_tree.pack()  
canvas1.create_window(0, 150, window=student_tree)
```

```
# Assignment tree
```

```

assignment_tree = ttk.Treeview(self, column=("c1", "c2", "c3"), show='headings')
assignment_tree.column("#1", anchor=tk.W, width=100)
assignment_tree.heading("#1", text="Name")
assignment_tree.column("#2", anchor=tk.W, width=100)
assignment_tree.heading("#2", text="Type")
assignment_tree.column("#3", anchor=tk.W, width=100)
assignment_tree.heading("#3", text="Student Score")
assignment_tree.pack()
canvas1.create_window(350, 150, window=assignment_tree)

# addStudent button
addStudent = tk.Button(self, width=20, text="Add student", command=get_student_info)
canvas1.create_window(0, 300, window=addStudent)

# # editAssignments button
# editAssignments = tk.Button(self, width=20, text="Edit Assignments",
command=editAssignments)
# canvas1.create_window(350, 300, window=editAssignments)

# addAssignment button
addAssignment = tk.Button(self, width=20, text="Add Assignment",
command=get_assignment_info)
canvas1.create_window(270, 300, window=addAssignment)

# delAssignment button
delAssignment = tk.Button(self, width=20, text="Delete Assignment", command=delAssignment)
canvas1.create_window(430, 300, window=delAssignment)

# Score
scoreLabel = tk.Label(self, text="Score (100)", width=20)
canvas1.create_window(-60, 400, window=scoreLabel)
scoreEntry = tk.Entry(self, width=24)
canvas1.create_window(50, 400, window=scoreEntry)

# Confirm button
confirmButton = tk.Button(self, width=20, text="Confirm Score", command=submitForm)
canvas1.create_window(270, 400, window=confirmButton)

# Delete Score button
delScoreButton = tk.Button(self, width=20, text="Delete Score", command=delScore)
canvas1.create_window(430, 400, window=delScoreButton)

# Logout button
logoutButton = tk.Button(self, width=20, text="Back", command=logMeOut)
canvas1.create_window(0, 550, window=logoutButton)

```

```

# Calculate Class Grade button
calcButton = tk.Button(self, width=30, text="Calculate Class Grades", command=calcGrade)
canvas1.create_window(350, 550, window=calcButton)

# Login Page
class GradesPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)

        global dispGrades
        def dispGrades():
            for row in grades_tree.get_children():
                grades_tree.delete(row)

            course_grades = g1.calculate_overall_grade(g1.selectedCourse)

            for grade in course_grades:
                print(grade)
                grades_tree.insert(parent = "", index='end', values= (grade[0],
grade[1],grade[2],grade[3],grade[4]))

# Main Page Layout:
canvas1 = tk.Canvas(self, width=400, height=700)
canvas1.pack()

# Assignment tree
grades_tree = ttk.Treeview(self, column=("c1", "c2", "c3", "c4", "c5"), show='headings')
grades_tree.column("#1", anchor=tk.W, width=150)
grades_tree.heading("#1", text="Student Name")
grades_tree.column("#2", anchor=tk.W, width=150)
grades_tree.heading("#2", text="HW %")
grades_tree.column("#3", anchor=tk.W, width=150)
grades_tree.heading("#3", text="TEST %")
grades_tree.column("#4", anchor=tk.W, width=150)
grades_tree.heading("#4", text="Overall %")
grades_tree.column("#5", anchor=tk.W, width=150)
grades_tree.heading("#5", text="Overall Grade")
grades_tree.pack()
canvas1.create_window(200, 200, window=grades_tree)

# Back button

```

```
        backButton = tk.Button(self, width=15, text="Back", command=lambda:
controller.show_frame(MainPage))
        canvas1.create_window(200, 400, window=backButton)

# Connect to or create Database
conn = sqlite3.connect('gradesdatabase.db')
if conn:
    print("Opened database successfully")
else:
    print("Error creating database")

# Create grademanager
g1 = GradeManager()
g1.createUserTable()
g1.createStudentsTable()
g1.createAssignmentsTable()
g1.createScoresTable()
g1.initialize()

print(g1.courses)

# Create Window and display
app = MyGrades()
app.title("Student Grading")
app.geometry("800x600")
app.mainloop()
```