****

**Project Title:**

**Window Examination (Summary)**

**Project Requirement(Technologies used):**

**Frontened:** Python Tkinter for GUI Development.

**DBMS Software:**Mysql(My Structured query language)

**Project Done By:**

Gorrela.Manikanta Sai with the guidance of p.srinuvasurao sir

**ABSTRACT**

**Summary of the Window Examination System with Tkinter** **and** **MySQL**

**Abstract:**

**Introduction:**

This project presents an **Window Examination System** designed to automate and streamline the assessment process for educational institutions. Developed using **Python** with the **Tkinter** library for the graphical user interface (GUI) and **MySQL** for database management, the system provides a secure and efficient platform for conducting computer-based tests.

The application features a **multi-section examination** consisting of **Arithmetic, Reasoning, and English** (10 questions each, totaling 30 marks). Key functionalities include:

* **Secure login authentication** using hall ticket numbers and candidate names.
* **Interactive exam interface** with real-time question navigation and automated scoring.
* **Malpractice prevention** through mandatory agreement checkboxes before exam commencement.
* **Instant result calculation**, with a passing threshold set at >15/30.
* **Database integration** to store candidate scores for future reference.

**Purpose of the Window Examination System Code**

This Python program creates a **desktop-based Window Examination System** using Tkinter for the GUI and MySQL for database management. Its primary purposes are:

1. **Automated Digital Testing**
   * Provides a computerized platform for conducting exams with three sections:
     + **Arithmetic** (10 questions)
     + **Reasoning** (10 questions)
     + **English** (10 questions)
   * Eliminates manual paper-based evaluations.
2. **Secure Candidate Authentication**
   * Validates users via **hall ticket numbers** and **names** (predefined credentials).
   * Prevents unauthorized access with an eligibility check.
3. **Anti-Malpractice Features**
   * Requires candidates to agree to exam rules via checkboxes before starting.
   * Ensures fair testing conditions.
4. **Real-Time Scoring & Instant Results**
   * Automatically calculates scores as candidates answer questions.
   * Declares pass/fail results immediately (passing threshold: **>15/30**).
   * Displays detailed score summaries.
5. **Database Integration**
   * Stores candidate details (hall ticket number, name, score) in a **MySQL database** for record-keeping and analysis.
   * Uses table student\_scores to maintain exam records.
6. **User-Friendly Interface**
   * Navigable sections with "Next" buttons.
   * Clear visual feedback for selected answers.
   * Light green-themed GUI for reduced eye strain.
7. **Educational Use Case**
   * Demonstrates practical applications of:
     + **Tkinter GUI development**
     + **MySQL database connectivity**
     + **Event-driven programming**
   * Scalable for institutional exam management.

**Technical Stack**

* **Frontend**: Tkinter (Python)
* **Backend**: MySQL
* **Logic**: Score tracking via global variables, radio button-based Q&A.

This system streamlines exam administration while ensuring security, efficiency, and transparency in evaluations.

**Methodology for the Window Examination System Project**

This project follows a **structured development approach** combining **GUI design, database integration, and event-driven programming**. Below is the step-by-step methodology:

**1. Requirement Analysis**

* **Objective**: Develop a secure, automated exam platform with three sections (Arithmetic, Reasoning, English).
* **Key Requirements**:
  + User authentication (hall ticket + name).
  + Multi-section exam (30 questions total).
  + Real-time scoring and results.
  + Anti-malpractice measures.
  + Database storage for results.

**2. Technology Stack Selection**

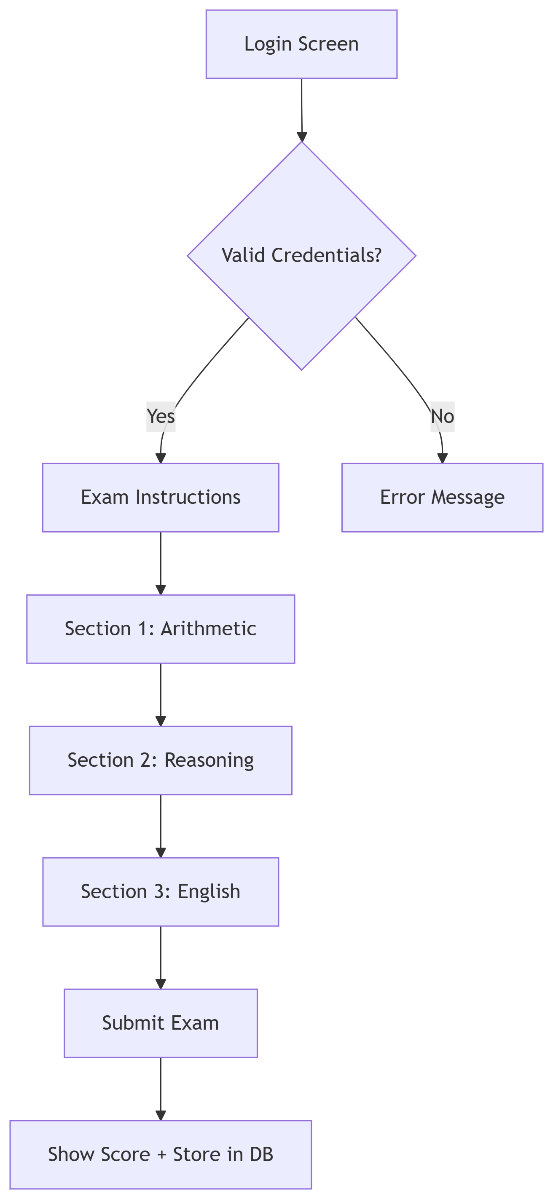
* **Frontend**: Tkinter (Python) for GUI.
* **Backend**: MySQL for data persistence.
* **Logic**: Python functions with global variables for score tracking.

**3. System Design**

**A. Module Breakdown**

1. **Login Module**
   * Validates credentials against hardcoded values (for demo purposes).
   * Shows error for invalid/missing inputs.
2. **Exam Instructions Module**
   * Displays exam rules via Toplevel() window.
   * Implements mandatory checkboxes for agreement.
3. **Question Modules**
   * **Section 1 (Arithmetic)**: 10 questions with radio buttons.
   * **Section 2 (Reasoning)**: 10 questions with radio buttons.
   * **Section 3 (English)**: 10 questions with radio buttons.
   * Each question updates a global score variable.
4. **Scoring Module**
   * Tracks correct/incorrect answers in real-time.
   * Pass/fail threshold: **>15/30**.
5. **Database Module**
   * Stores hall\_ticket\_number, name, and score in MySQL.

**B. Workflow**



**Exam Portal Workflow (Simplified)**

1. **Login Screen**
   * Enter credentials (username, password).
   * If valid → Proceed to exam.
   * If invalid → Show error.
2. **Exam Instructions**
   * Read rules & click "Start Exam."
3. **Exam Sections**
   * **Section 1:** Arithmetic (Math questions)
   * **Section 2:** Reasoning (Logic questions)
   * **Section 3:** English (Grammar/Comprehension)
   * Must complete in order.
4. **Submit Exam**
   * Confirm submission → Calculates score.
5. **Result & Storage**
   * Displays score (total + section-wise).
   * Saves to database

**4. Implementation Steps**

**A. GUI Development**

1. **Main Window** (Tk()):
   * Login form with Entry widgets for hall ticket/name.
   * "Next" button triggers validation (call\_result()).
2. **Exam Screens** (Toplevel()):
   * Each section opens in a new window.
   * Questions use Radiobutton widgets with IntVar() for selections.
   * Navigation via "Next" buttons.
3. **Dynamic Feedback**:
   * Labels update with selected options (label.config(text=selection)).

**B. Scoring Logic**

* Global score variable increments for correct answers.
* Example:

def question1():

global score

if radio1.get() == 3: *# Correct option*

score += 1

**C. Database Integration**

1. **Setup**:

mydb = mysql.connector.connect(

host="localhost", user="root",

password="root123", database="crttext"

)

1. **Table Creation**:

CREATE TABLE IF NOT EXISTS student\_scores (

hall\_ticket\_number VARCHAR(20) PRIMARY KEY,

name VARCHAR(100),

score INT

)

1. **Insertion**:

python

sql = "INSERT INTO student\_scores VALUES (%s, %s, %s)"

val = (current\_rollno, current\_name, score)

mycursor.execute(sql, val)

**5. Testing Strategy**

1. **Unit Testing**:
   * Verify login with valid/invalid credentials.
   * Test scoring logic for each question.
2. **Integration Testing**:
   * Ensure seamless transition between sections.
   * Validate database entries post-submission.
3. **User Acceptance Testing (UAT)**:
   * Demo with 5 predefined users (e.g., "111:mani").

**6. Error Handling**

* **Input Validation**:
  + Empty fields show messagebox.showerror().
  + Checkbox agreement enforced before exam start.

**7. Deployment**

1. **Dependencies**:
   * Install mysql-connector-python.

pip install mysql-connector-python

1. **Execution**:
   * Run the Python script (main.py).

**8. Maintenance Plan**

* **Scalability**:
  + Replace hardcoded credentials with a full user database.
  + Add admin panel for question management.
* **Enhancements**:
  + Timer for each section.
  + Randomized question order.

**Key features of your window examination system project:**

1. **User Authentication**

* Validates student credentials (Hall Ticket Number and Name) against a predefined list.
* Shows appropriate error messages for invalid or missing inputs.

2. **Multi-Section Exam Structure**

* Divided into 3 sections (Arithmetic, Reasoning, English).
* Each section contains 10 questions (30 questions total).

3. **Interactive GUI**

* Built with Tkinter for a user-friendly interface.
* Clean layout with consistent styling (light green background, blue text).
* Responsive design with proper widget alignment.

4. **Question Presentation**

* Multiple-choice questions with radio buttons for selections.
* Immediate feedback showing the selected option.
* Questions are properly labeled and organized by section.

5. **Scoring System**

* Tracks correct answers in real-time (1 point per correct answer).
* Global score variable maintains the total throughout the exam.
* Final score calculation out of 30.

6. **Navigation Flow**

* Sequential progression through sections with "NEXT" buttons.
* Final submission button to view results.

7. **Result Evaluation**

* Pass/Fail determination (threshold set at score>=15).
* Displays final score with a congratulatory or improvement message.

8. **Data Persistence**

* Stores results in MySQL database with:
  + Hall Ticket Number (Primary Key)
  + Student Name
  + Exam Score
* Uses error handling for database operations.

9. **Exam Instructions**

* Clear guidelines displayed before starting:
  + Section-wise marks distribution
  + Exam rules and expectations
  + Checkbox agreement for terms

10. **Error Handling**

* Validates checkbox selections before exam start.
* Shows warning messages for incomplete inputs/agreements.

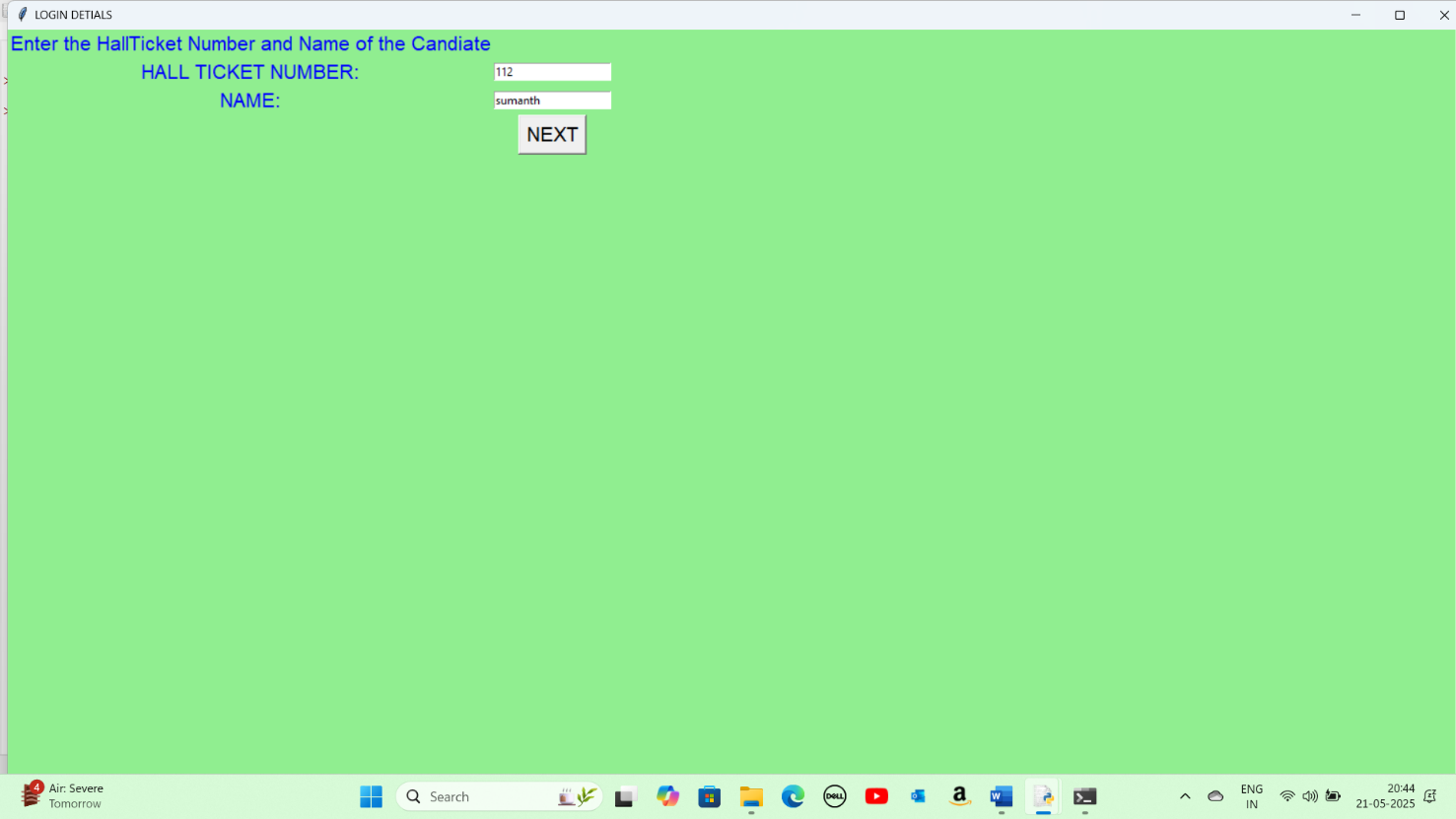
11. **Window Management**

* Uses Toplevel() for secondary windows.
* Proper window closing functionality.

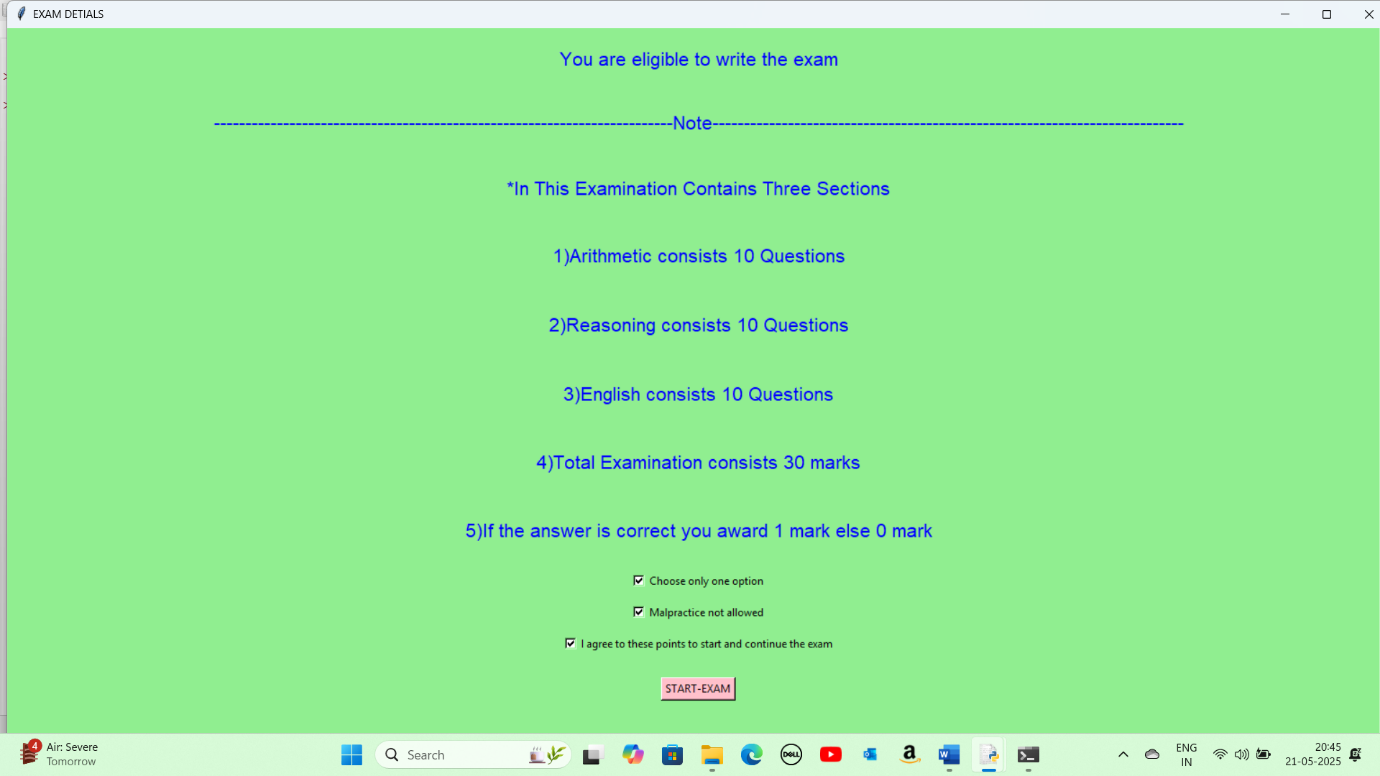
**Technical Components Used:**

* **Tkinter**: For GUI development
* **MySQL Connector**: For database integration
* **functools.partial**: For parameter binding in button commands
* **Global variables**: For maintaining state (score, current user)

**Results/Outcome of the Window Examination System Project**

****

1. **Header**: "LOGIN DETAILS" appears at the top, likely representing an examination or testing system.
2. **Instruction**: "Enter the HallTicket Number and Name of the Candidate" appears as a subheading.
3. **Input Fields**:
   * **HALL TICKET NUMBER**: A blank field for entering the candidate's examination hall ticket number (with "[112]" possibly being a placeholder or example)
   * **NAME**: A blank field for entering the candidate's name (with "[sumanth]" possibly being a placeholder or example)
4. **Navigation**:
   * A horizontal line separates the input section from a "NEXT" button, which would presumably proceed to the next step after entering credentials.



**You are eligible to write the exam**

**Note**

This examination consists of **three sections**, with a total of **30 marks**:

1. **Arithmetic** – 10 Questions
2. **Reasoning** – 10 Questions
3. **English** – 10 Questions

**Marking Scheme**

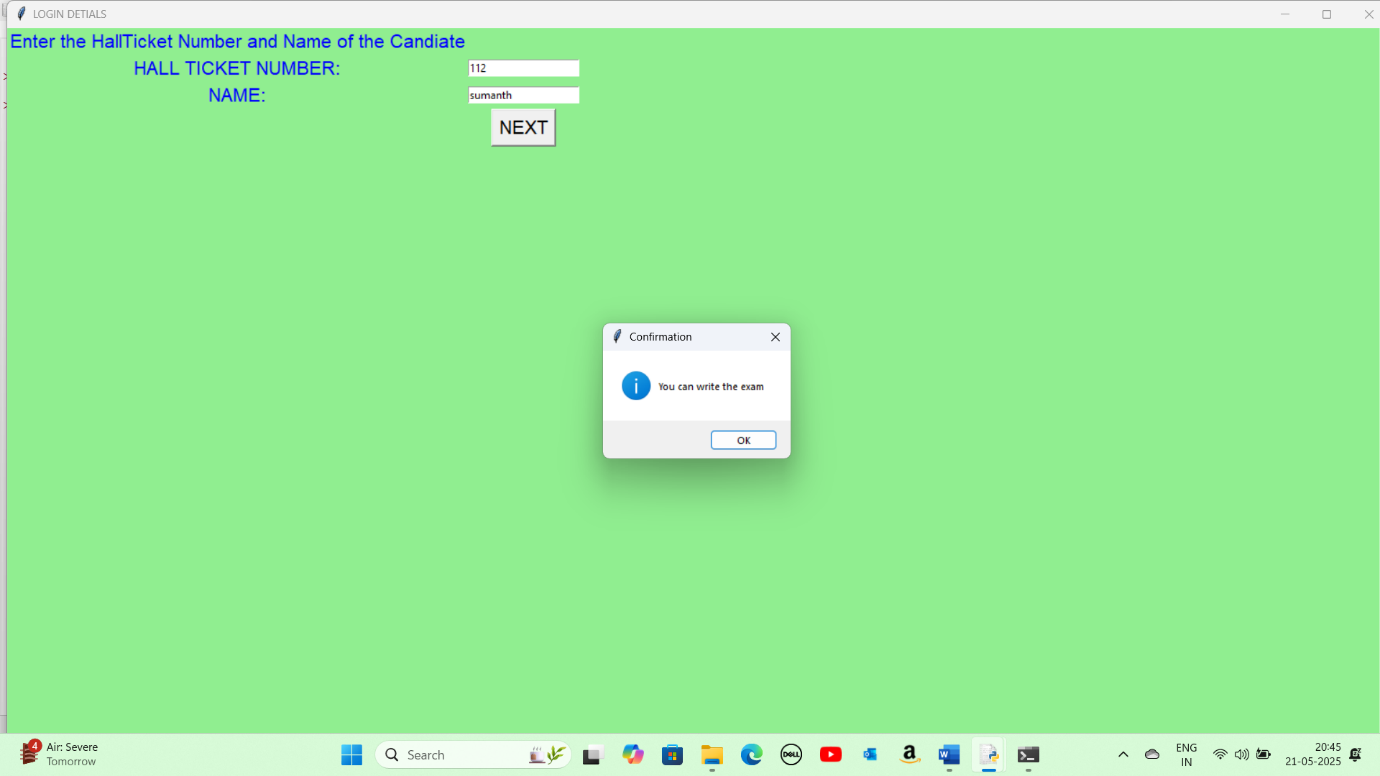
* **Correct Answer:** +1 mark
* **Incorrect Answer:** 0 marks

**Exam Rules & Guidelines**

* Choose only **one option** per question (likely referring to multiple-choice questions).
* **Malpractice is not allowed** (Note: The screenshot says "Malpractice and allowed," which may be a typo; it likely means malpractice is **prohibited**).
* **I agree to these points** to start and continue the exam.

**Proceed to Exam**

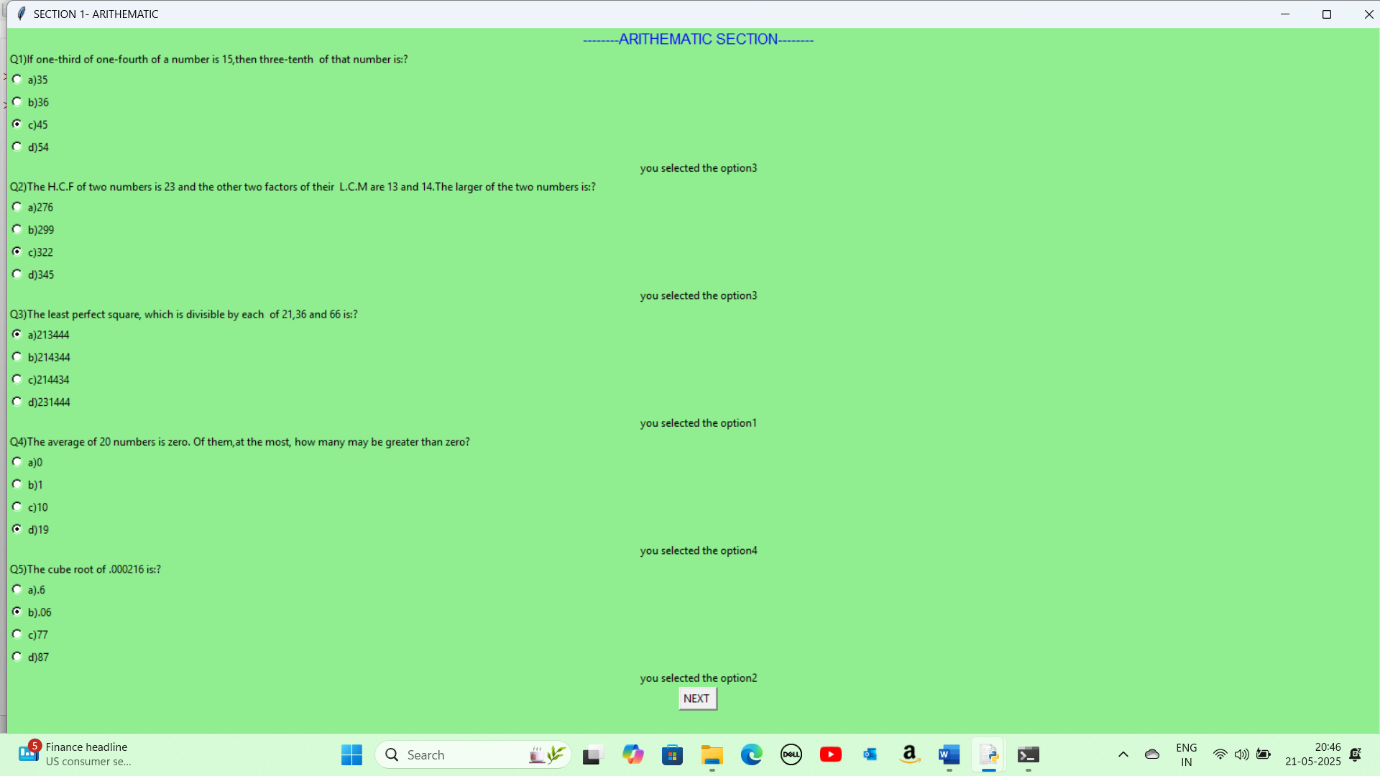
* **Button:** **"START EXAM"** – Clicking this will begin the test.



* Checks if either the **hall ticket number (**r**)** or **name (**n**)** is empty.
* Displays an error message ("Please enter all details") if any field is blank.
* Stops further execution (return) if validation fails.
* Compares the entered (r, n) against a **hardcoded list of valid candidates**.
* **If matched**, calls show\_exam\_instructions() (likely a function to display exam rules).
* **If no match**, shows a warning: "You are not eligible".

**Key Observations**

* **Predefined Valid Candidates**:
  + Only 5 candidates are allowed (e.g., "111", "mani").
  + This is a **hardcoded** list, meaning new candidates cannot be added dynamically.
* **Case-Sensitive Matching**:
  + The name must match exactly (e.g., "sumanth" ≠ "Sumanth").
* **Security Note**:
  + Hardcoding credentials is **not secure** for production systems. A database lookup would be better.
* **User Flow**:
  + Blank inputs → Error: "Please enter all details."
  + Invalid credentials → Error: "You are not eligible."
  + Valid credentials → Proceeds to exam instructions.



This Python/Tkinter code creates an interactive arithmetic test section with 5 multiple-choice questions. It tracks user selections and calculates a score based on correct answers.

**Global Score Tracking**

* Initializes a global variable to track the test-taker's score.

**Question Structure (5 Questions)**

Each question follows this pattern:

**Question Label** - Displays the problem

**Radio Buttons** - 4 options (a-d)

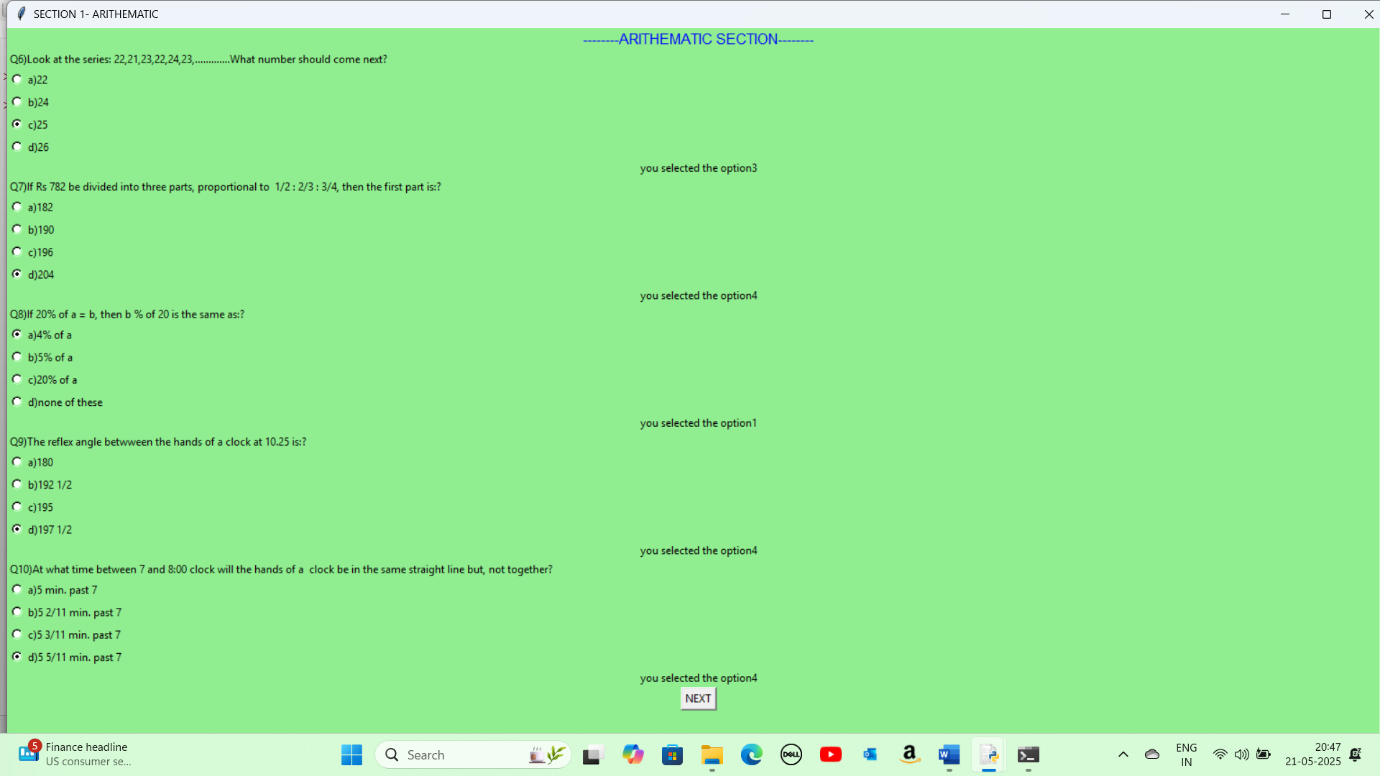
**Selection Handler** - Updates score and shows selection

**Navigation**

* Proceeds to next section (presumably "section2")

**Scoring Behaviour**

+1 if correct, else reset to 0



This Python/Tkinter code creates an interactive arithmetic test section with 5 multiple-choice questions. It tracks user selections and calculates a score based on correct answers.

**Global Score Tracking**

* Initializes a global variable to track the test-taker's score.

**Question Structure (5 Questions)**

Each question follows this pattern:

**Question Label** - Displays the problem

**Radio Buttons** - 4 options (a-d)

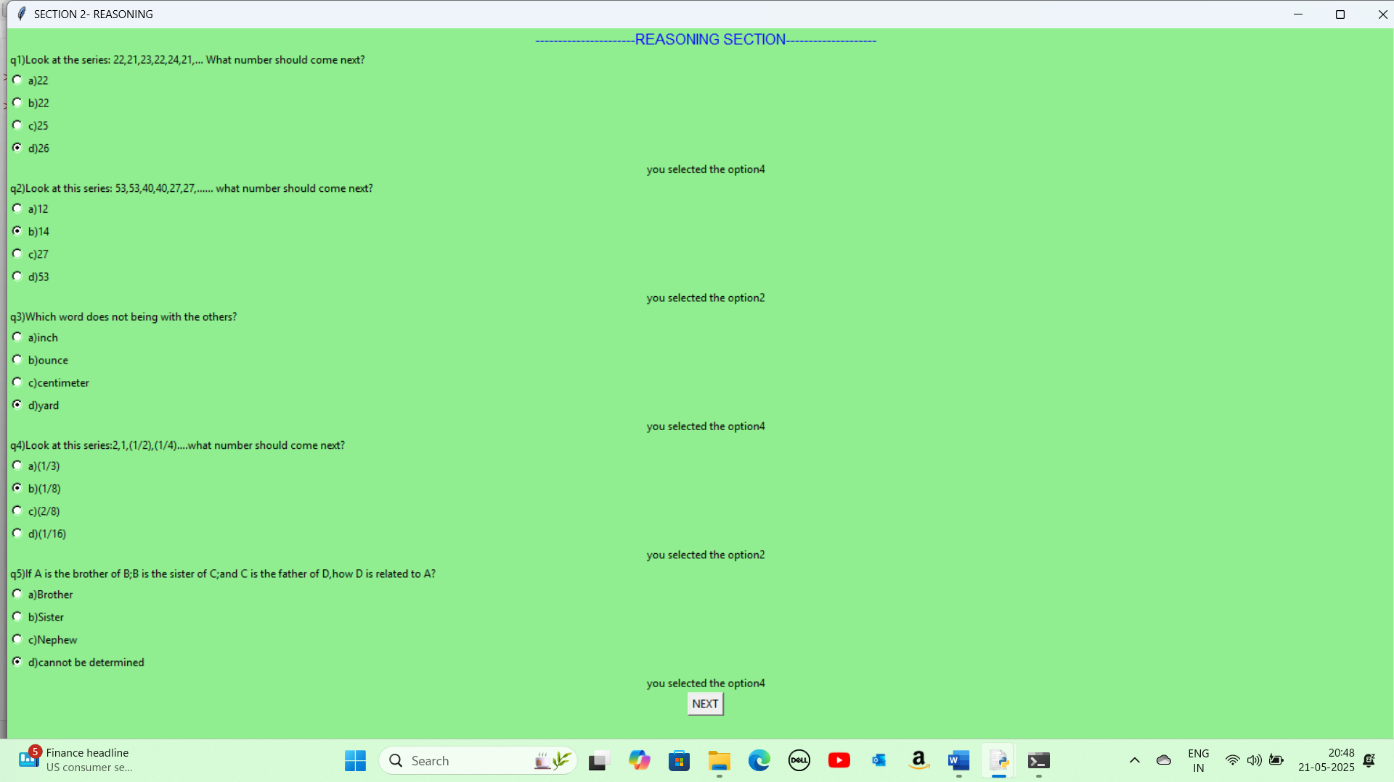
**Selection Handler** - Updates score and shows selection

**Navigation**

* Proceeds to next section (presumably "section3”)

**Scoring Behaviour**

+1 if correct,else reset to 0



This Python/Tkinter code creates the reasoning section of an exam with 5 multiple-choice questions, following a similar structure to the arithmetic section but with different content.

**Global Score Tracking**

Continues tracking the cumulative score from previous sections.

Each question has:

**Question Label** - Presents the reasoning problem

**Radio Buttons** - 4 options (a-d)

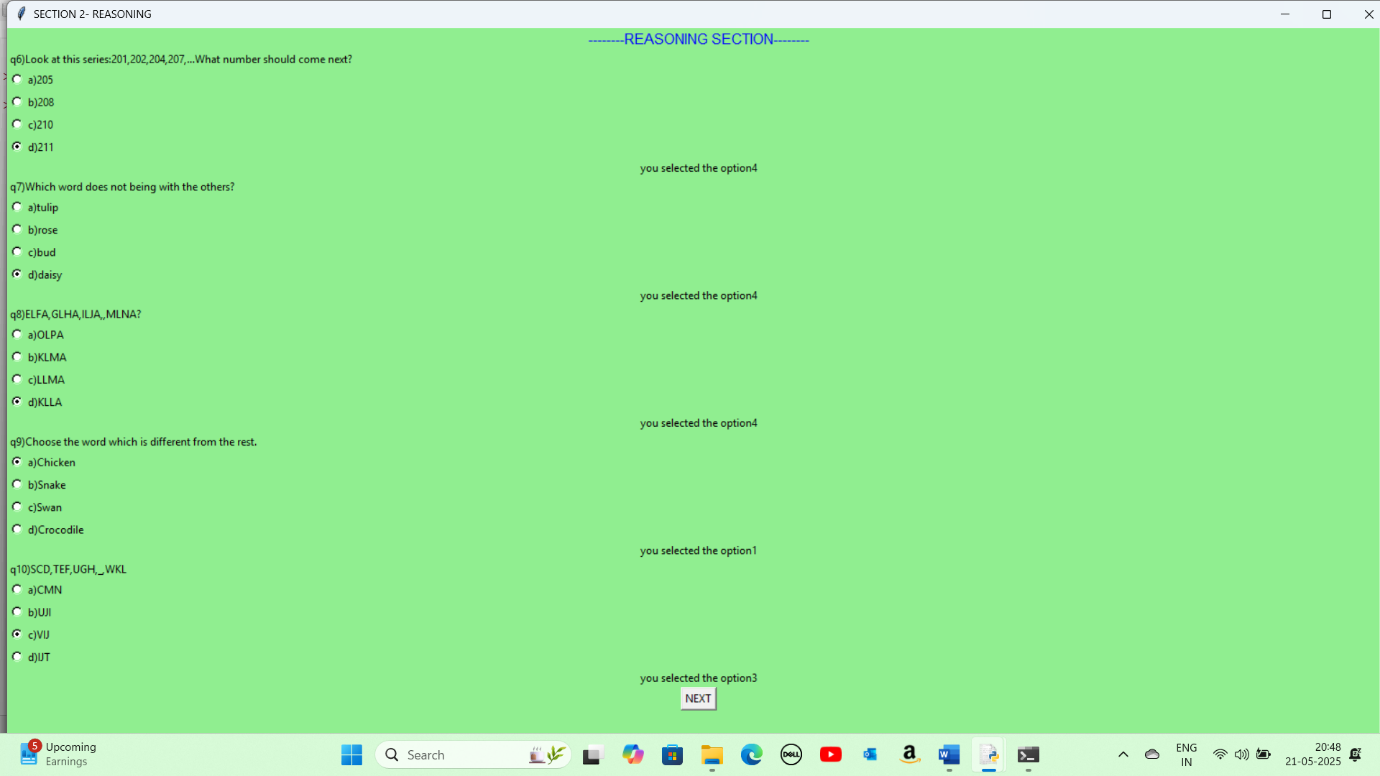
**Selection Handler** - Updates score and shows selection

**Navigation**

Proceeds to next section (presumably "section4")

**Scoring Behaviour**

+1 if correct, else reset to 0



This Python/Tkinter code creates the reasoning section of an exam with 5 multiple-choice questions, following a similar structure to the arithmetic section but with different content.

Global score tracking

Continues tracking the cumulative score from previous sections.

Each question has:

**Question Label** - Presents the reasoning problem

**Radio Buttons** - 4 options (a-d)

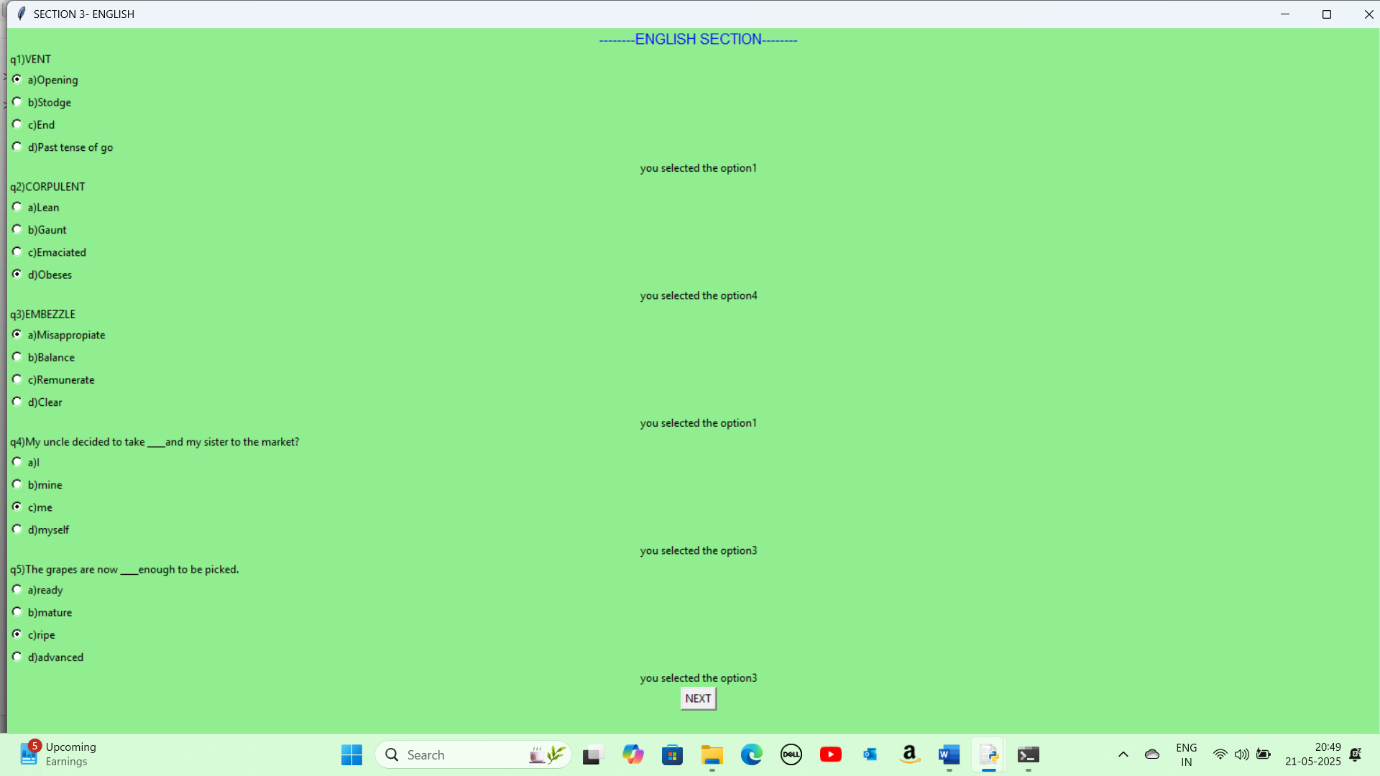
**Selection Handler** - Updates score and shows selection

**Navigation**

Proceeds to next section (presumably "section5")

**Scoring Behaviour**

+1 if correct, else reset to 0



This Python/Tkinter code creates the English section of an exam with 5 multiple-choice questions, following a similar structure to the arithmetic section and reasoning section but with different content.

Continues tracking the cumulative score from previous sections.

Each question has:

**Question Label** - Presents the reasoning problem

**Radio Buttons** - 4 options (a-d)

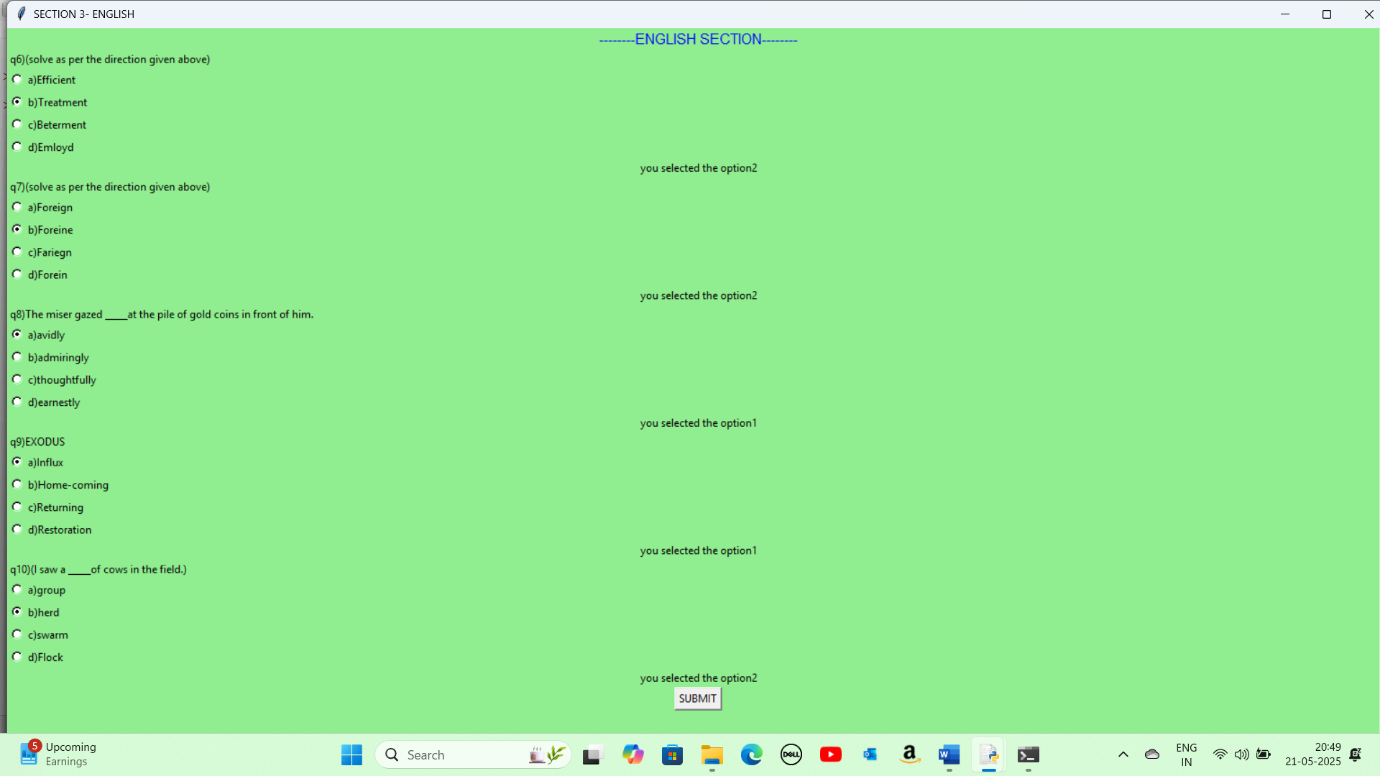
**Selection Handler** - Updates score and shows selection

**Navigation**

Proceeds to next section (presumably "section6")

**Scoring Behaviour**

+1 if correct, else reset to 0



This Python/Tkinter code creates the English section of an exam with 5 multiple-choice questions, following a similar structure to the arithmetic section and reasoning section but with different content.

Continues tracking the cumulative score from previous sections.

Each question has:

**Question Label** - Presents the reasoning problem

**Radio Buttons** - 4 options (a-d)

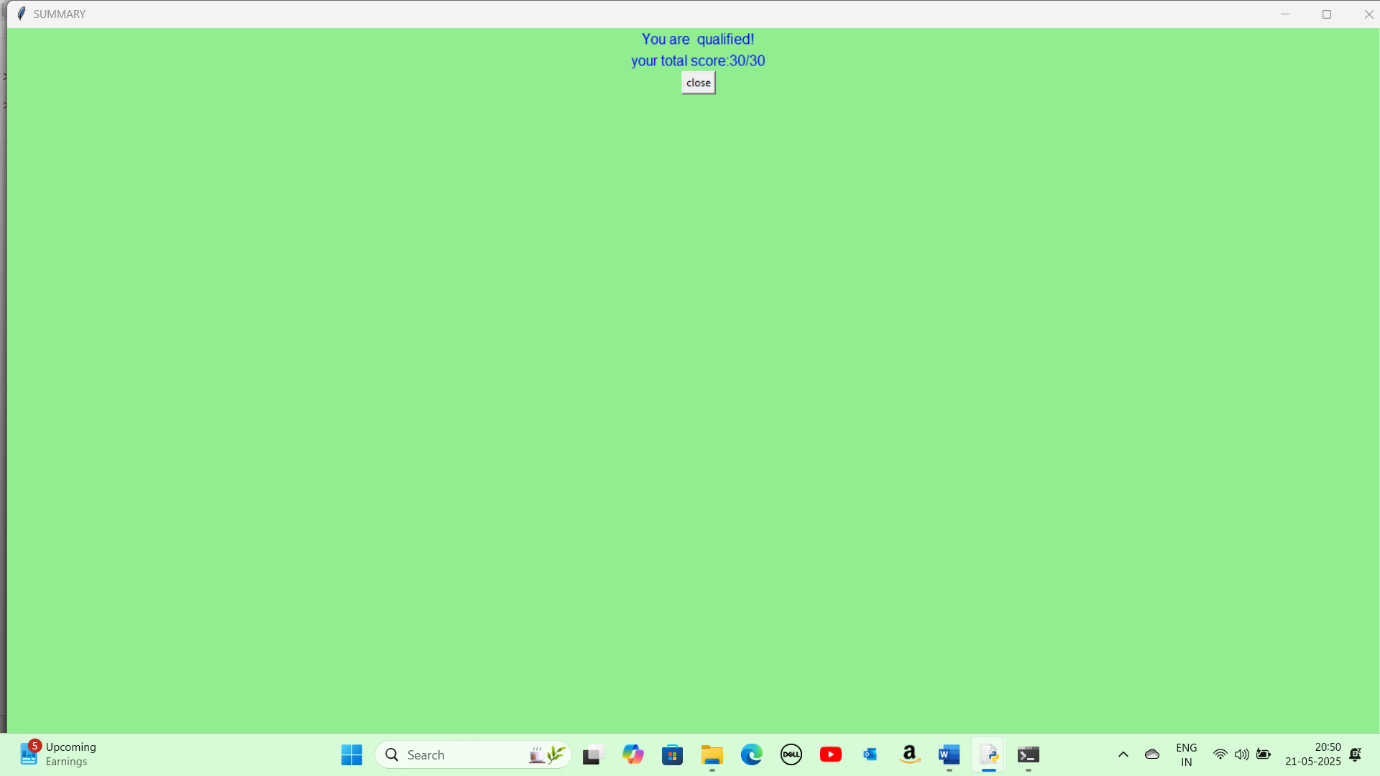
**Selection Handler** - Updates score and shows selection

**Navigation**

Proceeds to submit exam (presumably "submit")

**Scoring Behaviour**

+1 if correct, else reset to 0



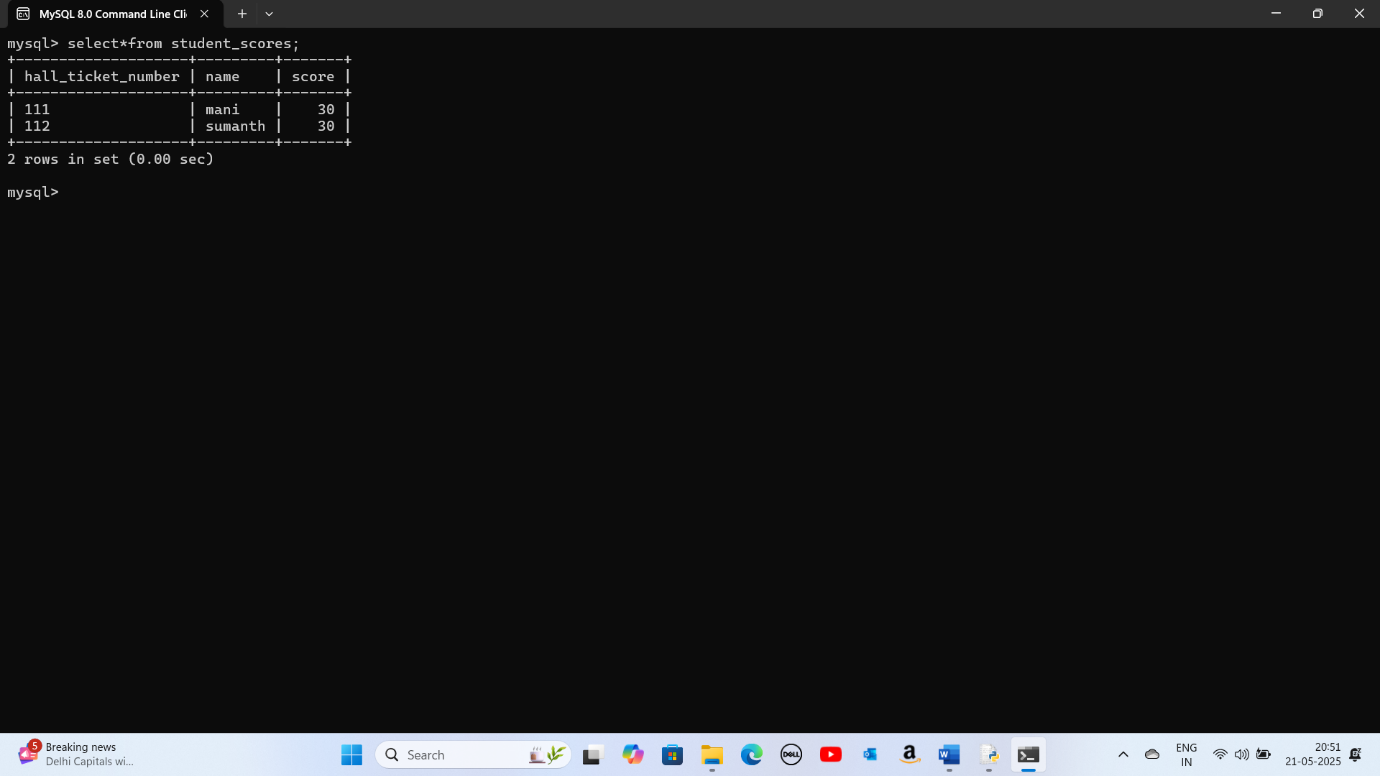
**Exam Summary Report**

**Candidate Qualification Status**

**"You are qualified"** – when the score is greater than or equal to 15.

And displayed your total score

**“You are disqualified”-**When the score is less than than 15.



**Database Integration for Student Scores Management**

This Python code snippet demonstrates how to connect to a MySQL database and manage student score records. Here's an explanation of what this code does:

**Database Connection Setup**

The code establishes a connection to a MySQL database named "crttext" on the localhost using the root credentials. This connection will be used to execute SQL queries for storing student score data.

Table Creation

The code first creates a table called student\_scores if it doesn't already exist, with three columns using **CREATE TABLE** command:

* hall\_ticket\_number (VARCHAR, primary key)
* name (VARCHAR)
* score (INT)

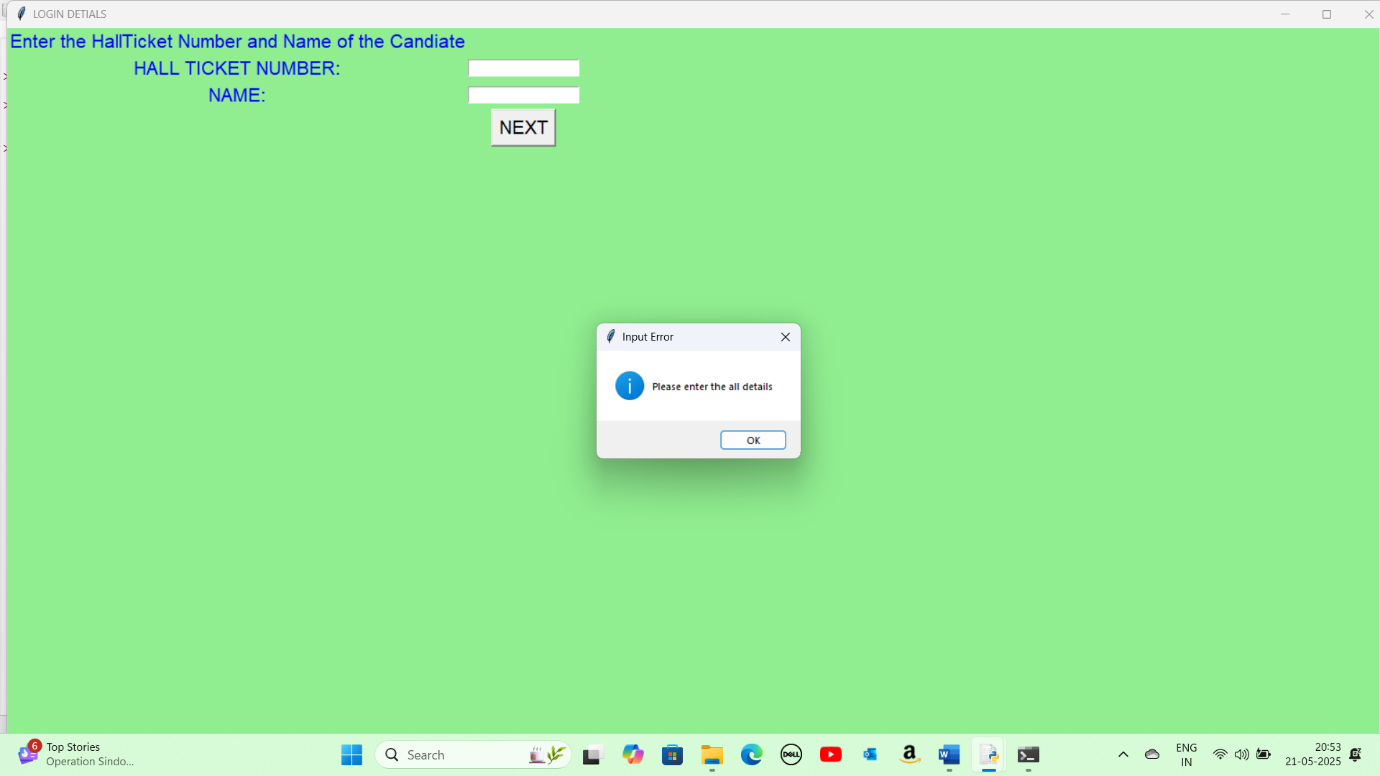
**Data Insertion**

The code then inserts a new record into the student\_scores table with values for table using **INSERT INTO** command :

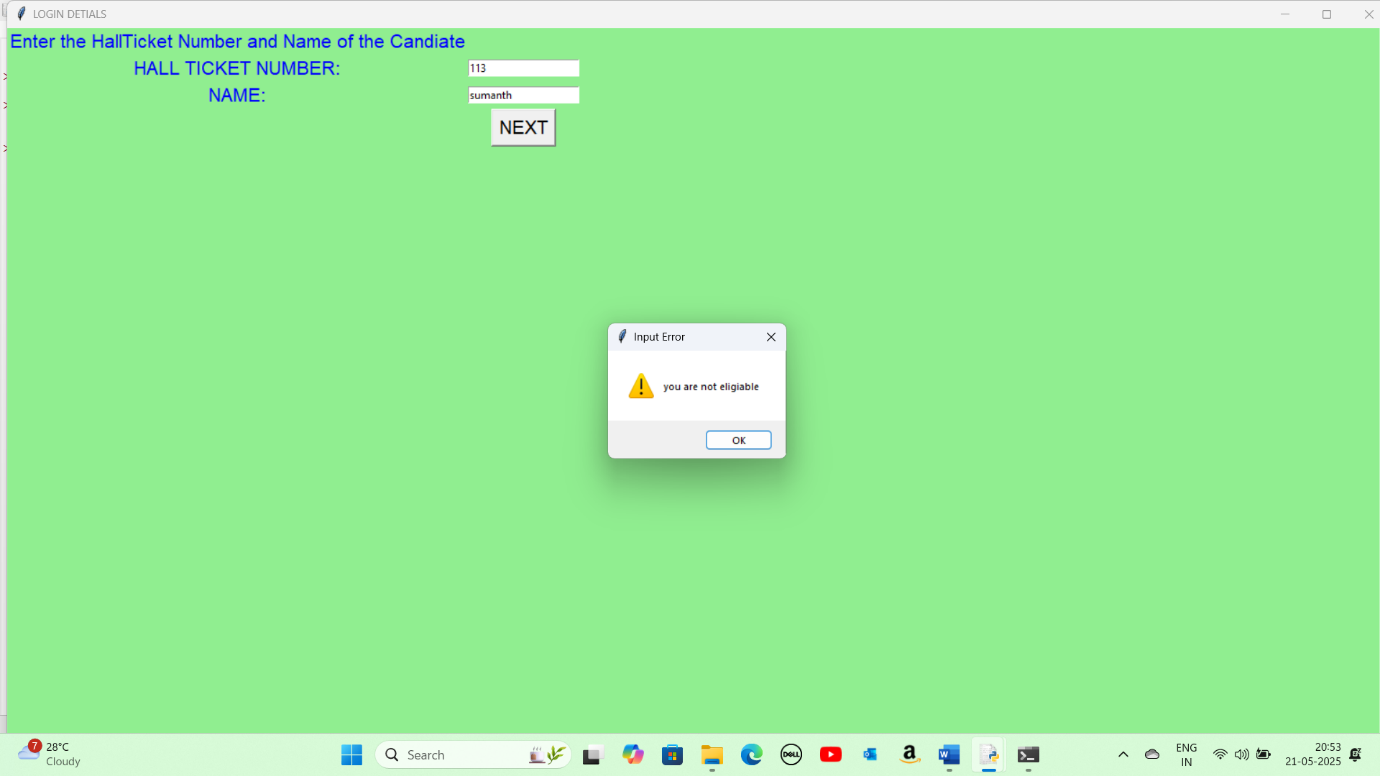
* Hall ticket number (from variable current\_rollno)
* Student name (from variable current\_name)
* Score (from variable score)

**Transaction Handling**

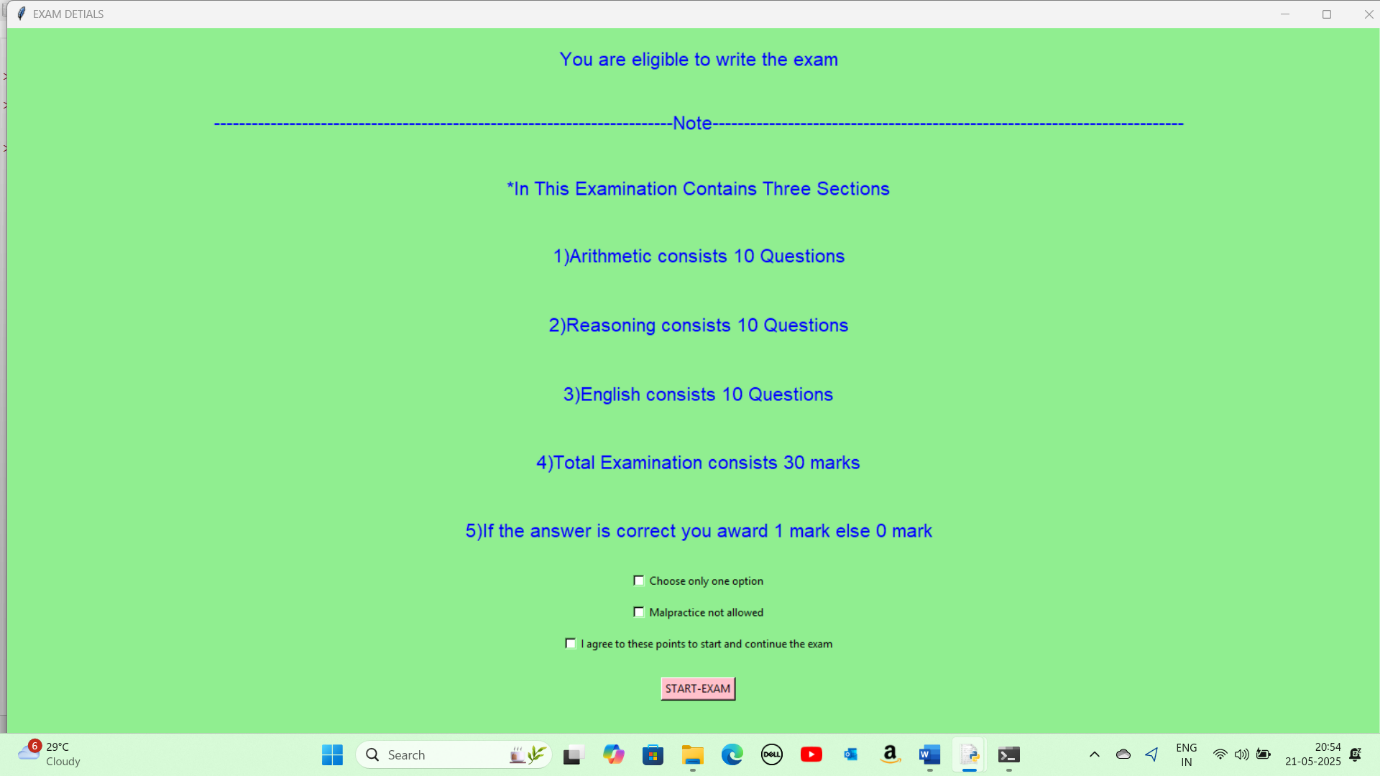
After executing the insert statement, the code commits the transaction to ensure the data is permanently saved to the database.

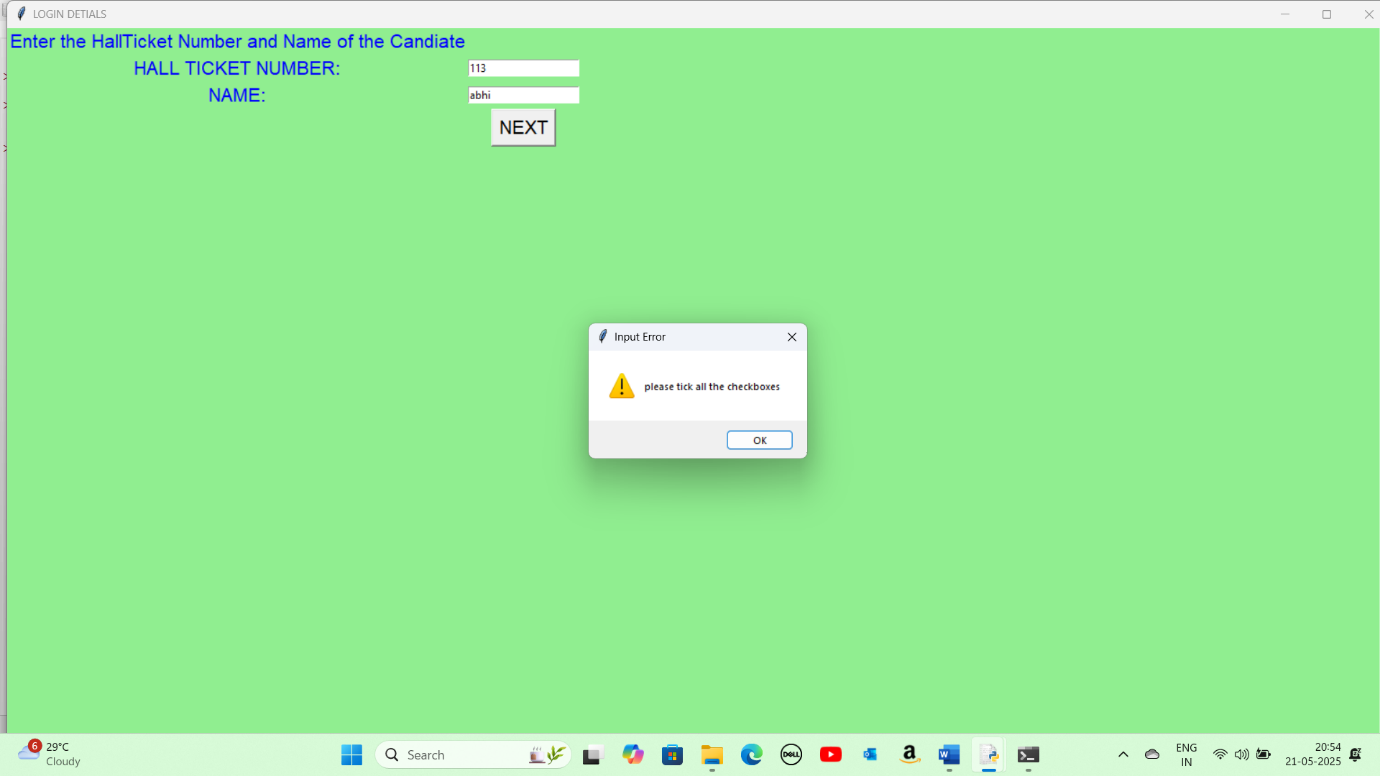
The primary key constraint on hall\_ticket\_number ensures each student has only one score record in the table.

* If the user doesn’t enter the his “**HALL TICKET NUMBER “** and “**Name**” throws an message box “**Please enter the details**” after clicking the next button.

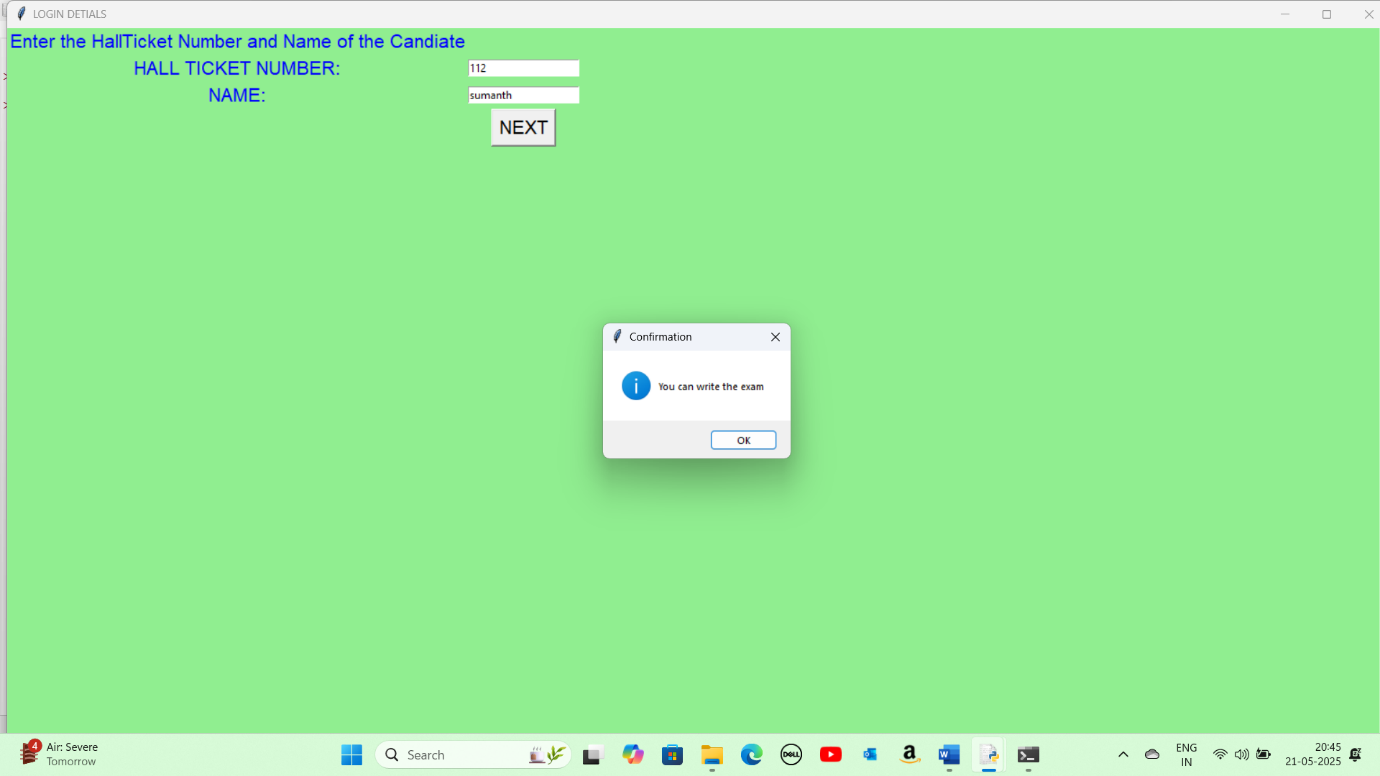


If the student user is entered the details wrongly then it throws an message box “**Input Error** ” is the message box title and warning message box “**you are not eligible**”.





If the user is does not ticked all the checkboxes it throws an message box **“please tick all the checkboxes”**.Other wise it throws the message box **“You can write the exam”.**



**THE END**