INSTITUTE OF COMPUTER SCIENCE AND DIGITAL INNOVATION

ACADEMIC SESSION: MAY - SEPTEMBER 2025

BIC1114/BBA1134: INTRODUCTION TO PROGRAMMING/PROGRAMMING FOR FINTECH

ASSIGNMENT **DEADLINE: 5TH AUGUST 2025 5.00 PM**

INSTRUCTIONS TO CANDIDATES

1. This assignment will contribute **30%** to your final grade.
2. This coursework is a group assignment. You may have 5 members in one group.
3. You are required to submit the codes and screenshot of all the possible the results.

IMPORTANT

Plagiarism is not allowed, and any student found to commit such an act, 0 mark will be awarded.

Coursework must be submitted on their due dates. If a coursework is submitted after its due date, the following penalty will be imposed:

- **ONE day late : 3 marks deducted from the total marks awarded.**
- **TWO days late : 5 marks deducted from the total marks awarded.**
- **THREE or more days late: Assignment will not be marked and 0% will be awarded.**

***For example**: A student scores 10 marks for an assignment that has a total mark of 10 If the assignment is submitted one day late, the marks awarded will be 7 marks.*

You are also required to attach the screenshot of the results from the command prompt console.

Student Name:	Ronald Lee Kai Ren	**Student ID:**	1002162634
Student Name:	Chee Kin Hoe	**Student ID:**	1002163440
Student Name:	Alvin Tan Jun Hong	**Student ID:**	1002266261
Student Name:	Eric Ding Sheng Kiet	**Student ID:**	1002266424
Student Name:	ChzenZiXi	**Student ID:**	1002577887

Subject Name: Introduction To Programming

Subject Code: BIC1114

Lecturer/Tutor: **ASST. PROF. DR. SAMAR GHAZAL**

Assignment Number/Title: **Assignment**

Due Date: **5TH AUGUT 2025**

Assignment Weightage: **30%**

All work must be submitted by the due date.

Student's Statement:

I hereby declare that the work submitted is my own. I confirm that I have read and understood the University regulations with regard to plagiarism, and that Plagiarism, Collusion and Cheating in this work will be penalized.

Note:

1) You are expected to retain copies of your assignment report.
2) If you require an acknowledgement receipt of this assignment, please prepare a duplicate copy of this form.

Student Signature [illegible]

Date 5 August 2025

Table of Contents

1. Introduction

1.1 System Overview

The Smart Online Quiz System (SOQS) is a command-line-based quiz application developed in Python, aimed at simulating a real-world multiple-choice quiz platform. It enables users to take quizzes, receive immediate scores and feedback, and view their rankings on a dynamic leaderboard. The system is built using a modular architecture, separating functionalities into distinct components for user interaction, quiz management, administrative control, and utility support. One of the key features is the password-protected Admin Mode, which allows authorized personnel to securely manage the quiz database by adding, editing, searching, or deleting questions from a persistent JSON file. SOQS is designed to serve a range of users, from students preparing for assessments to educators managing question banks, and can be adapted for training, self-assessment, or classroom use.

1.2 Objective and Motivation

The main objective of this project is to design and implement a modular Python application that demonstrates a comprehensive understanding of core programming concepts. These include:

- Data structures (e.g., lists, dictionaries)
- File handling with JSON
- Modular programming and function design
- Control flow and logical decision-making
- Robust error handling and input validation

The motivation behind developing SOQS stems from the desire to bridge theoretical learning with practical application. By building a working system from scratch, the team aimed to gain hands-on experience in solving real-world problems, applying object-oriented and procedural programming principles, and collaborating effectively in a development environment.

2. System Design

This section outlines the architectural design of the Smart Online Quiz System (SOQS), detailing its modular components, file structure, and logical flow.

2.1 Description of Modules and Their Purposes

The SOQS is developed with a modular approach to separate concerns, improving code organization, maintainability, and collaboration. The system is broken down into the following core Python modules:

- `main.py` (Main Controller)
  - Purpose: This script serves as the primary entry point and central controller of the application. It is responsible for displaying the main menu to the user, capturing their initial choice, and directing the program flow to the appropriate module based on user input (e.g., starting a quiz, viewing the leaderboard, or entering admin mode).
- `quiz.py` (User-Facing Quiz Logic)
  - Purpose: This module encapsulates all functionalities related to the user's quiz-taking experience. Its responsibilities include:
    - `load_questions()`: Loading the pool of questions from the `questions.json` file.
    - `take_quiz()`: Presenting a random selection of 10 questions to the user, capturing their answers, and managing the quiz session.
    - `calculate_score()`: Evaluating the user's answers, calculating the final score, and generating performance feedback.
    - `save_result()` & `view_leaderboard()`: Saving the user's name, score, and timestamp to the `leaderboard.json` file and displaying the formatted, sorted leaderboard.
- `admin.py` (Administrative Functions)

- Purpose: This module contains all the backend functionalities reserved for administrators. Access to this module is password-protected to prevent unauthorized modifications to the quiz database. Its functions include:
    - `add_question()`: Allows an admin to add new questions with validated options and a correct answer.
    - `search_question()`: Provides a keyword-based search to find specific questions.
    - `edit_question()`: Allows an admin to modify the text or answers of an existing question.
    - `delete_question()`: Allows an admin to remove a question from the database after a confirmation prompt.

- `utils.py` (Utility and Helper Functions)
  - Purpose: This is a supporting module that contains common, reusable functions utilized across the application to avoid code duplication. This can include functions for input validation, screen clearing, implementing countdown timers, or formatting text for display.

2.2 File Structure

The project adheres to a structured and organized file system, which is crucial for managing the source code and its associated data files. The layout is as follows:

```
smart-online-quiz-system/
│
├── main.py            # Main application entry point
├── quiz.py            # Module for user quiz functions
├── admin.py           # Module for admin functions
├── utils.py           # Module for shared helper functions
│
├── data/
│   ├── questions.json    # Database of quiz questions and answers
│   └── leaderboard.json   # Stores user scores and timestamps
│
└── README.md            # Project overview and instructions
```

- Root Directory: Contains the main Python script files (.py).
- data/ Directory: This sub-directory is dedicated to storing the application's data, separating it from the source code. questions.json is used for its structured format, which is ideal for storing complex objects like questions with multiple options. leaderboard.json is used for its simplicity and ease of appending new records.

2.3 Flow Diagrams

The following diagrams illustrate the logical flow of the application from the user's perspective.

2.3.1 Main Application Flow

This diagram shows the primary navigation path from the main menu.

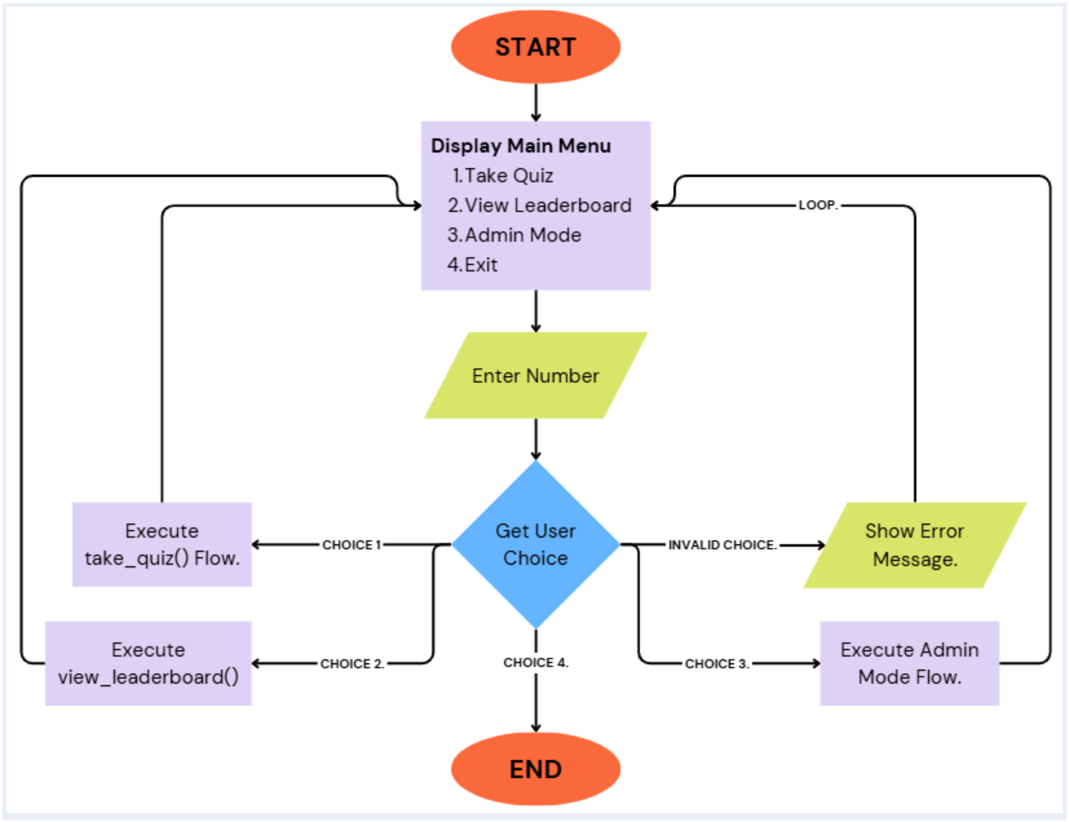


Diagram 1: Main Application Flow Chart

2.3.2 Take Quiz Flow

This diagram details the process a user goes through when taking a quiz.

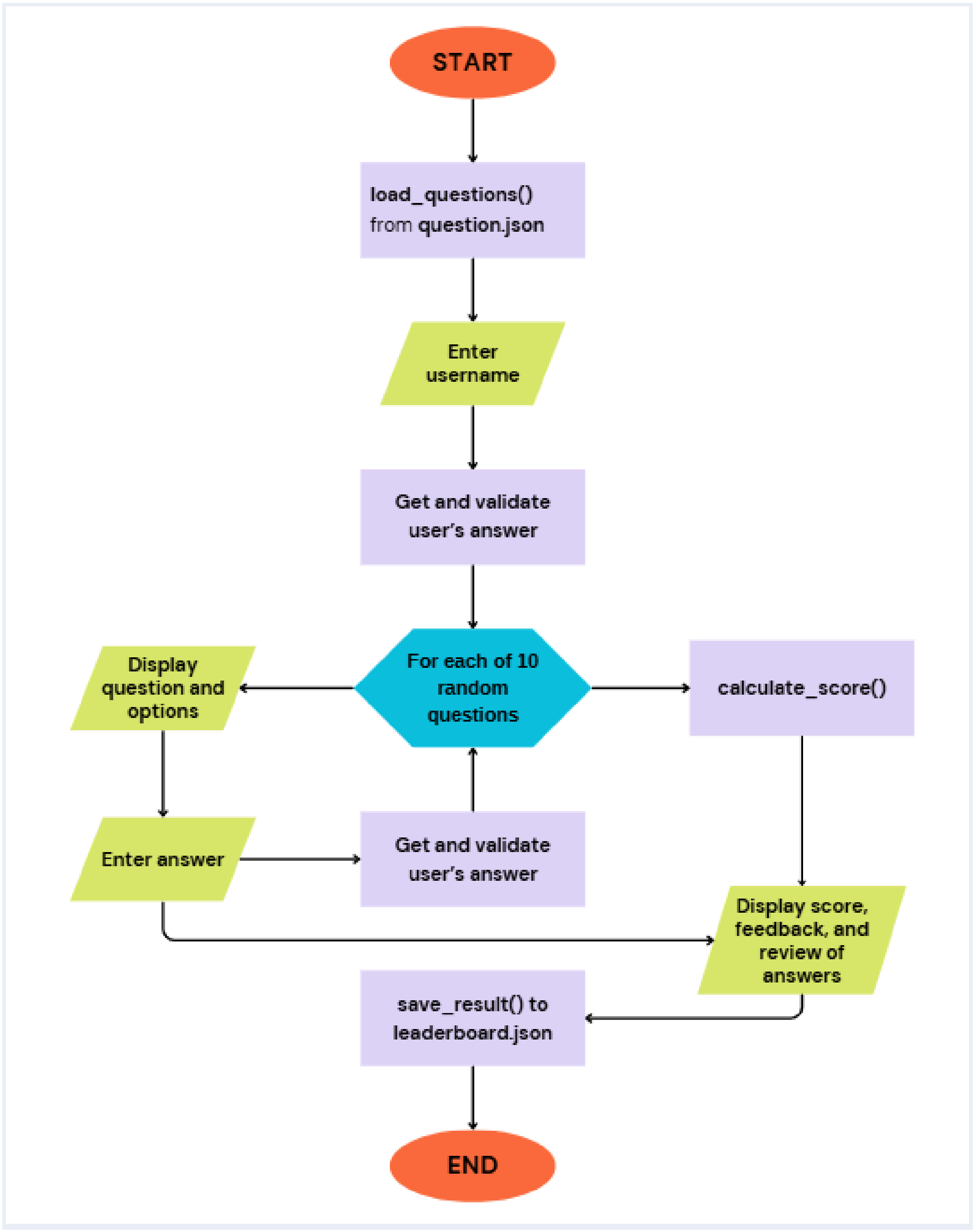

START
load_questions() from question.json
Enter username
Get and validate user's answer
For each of 10 random questions
Display question and options
calculate_score()
Enter answer
Get and validate user's answer
Display score, feedback, and review of answers
save_result() to leaderboard.json
END

Diagram 2: Take Quiz Flow Chart

2.3.3 Admin Mode Flow

This diagram shows the workflow for the password-protected administrative section.

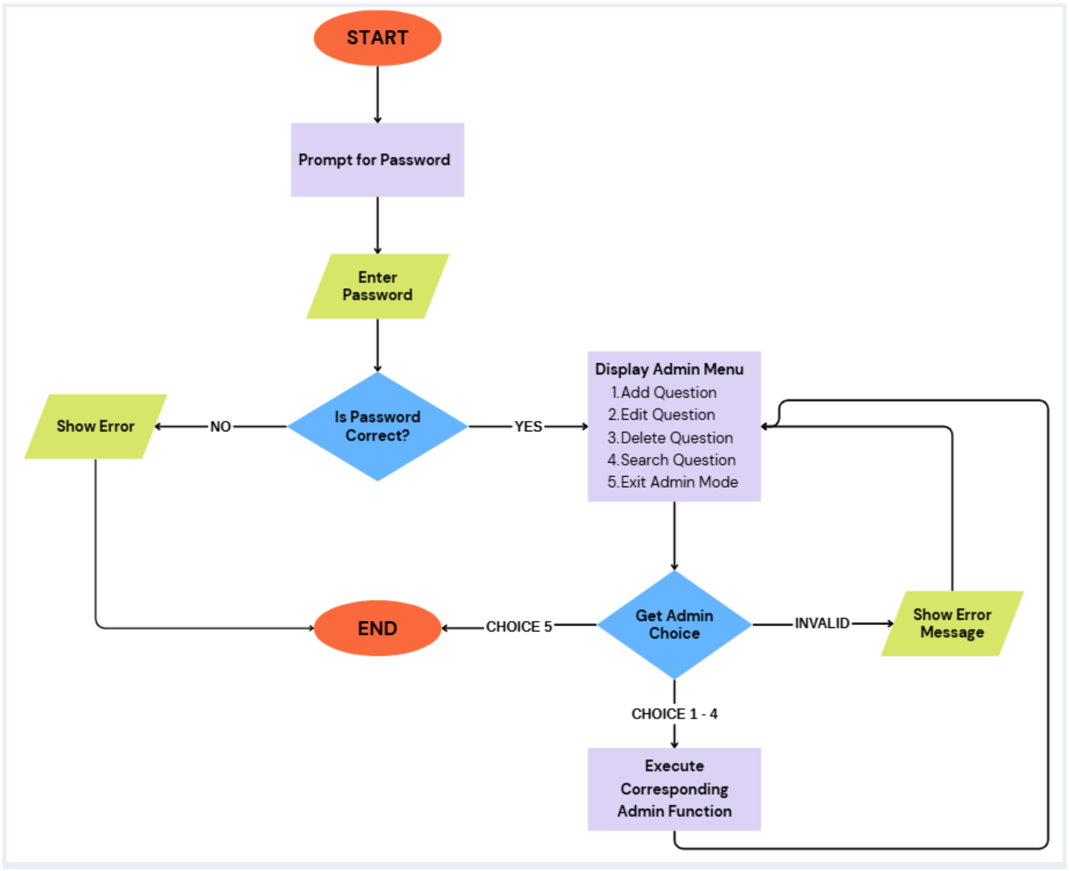
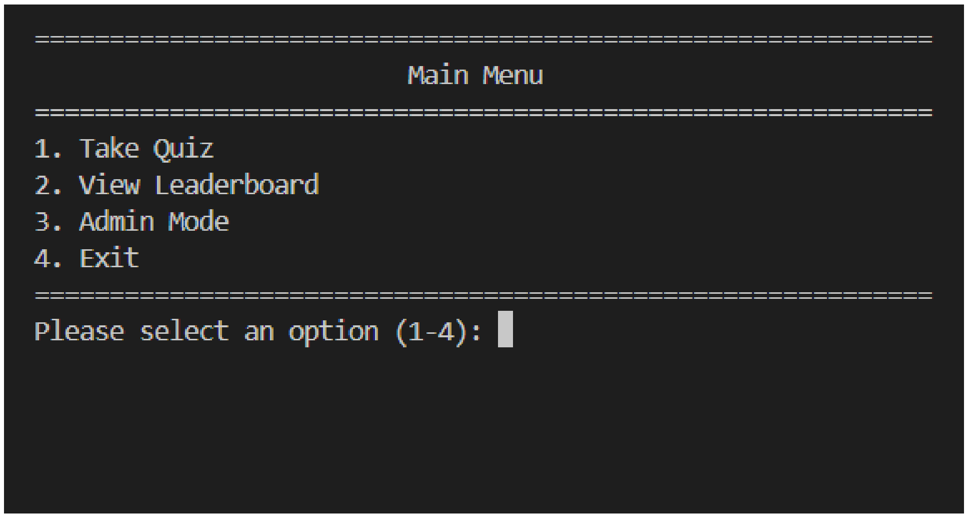


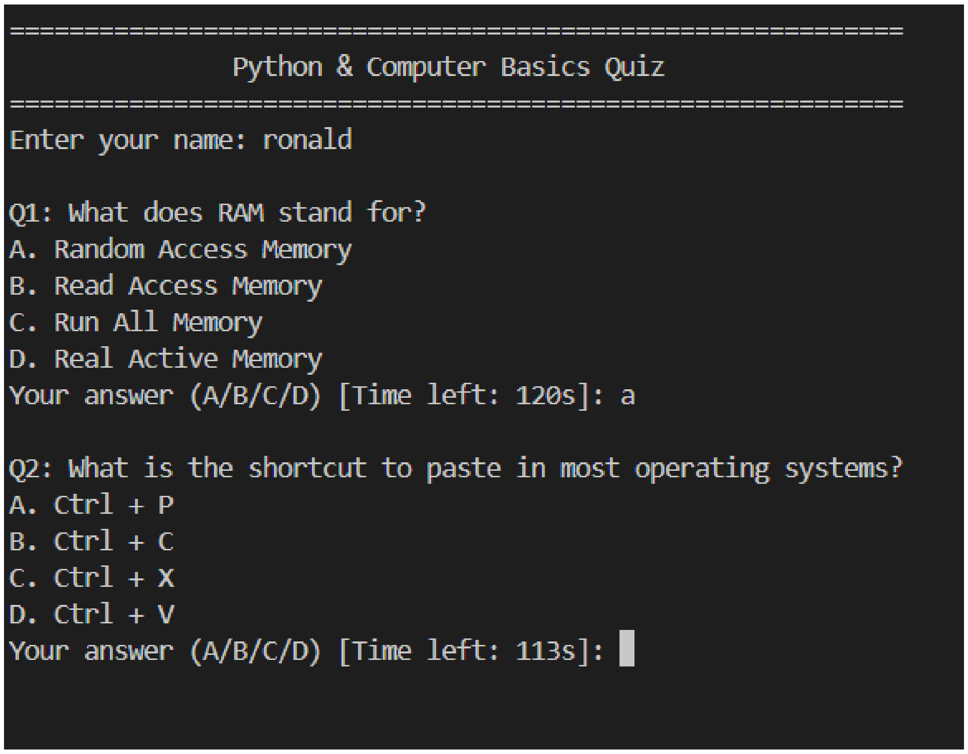
Diagram 3: Admin Mode Flow Chart

2.4 Screenshots of Sample Runs



```
============================================================
                        Main Menu
============================================================
1. Take Quiz
2. View Leaderboard
3. Admin Mode
4. Exit
============================================================
Please select an option (1-4):
```

Diagram 4: Main Menu

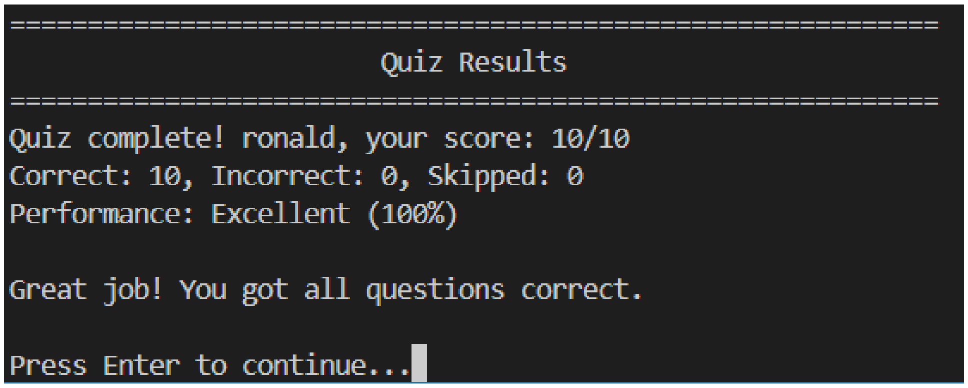


```
============================================================
              Python & Computer Basics Quiz
============================================================
Enter your name: ronald

Q1: What does RAM stand for?
A. Random Access Memory
B. Read Access Memory
C. Run All Memory
D. Real Active Memory
Your answer (A/B/C/D) [Time left: 120s]: a

Q2: What is the shortcut to paste in most operating systems?
A. Ctrl + P
B. Ctrl + C
C. Ctrl + X
D. Ctrl + V
Your answer (A/B/C/D) [Time left: 113s]:
```

Diagram 5: Quiz Page

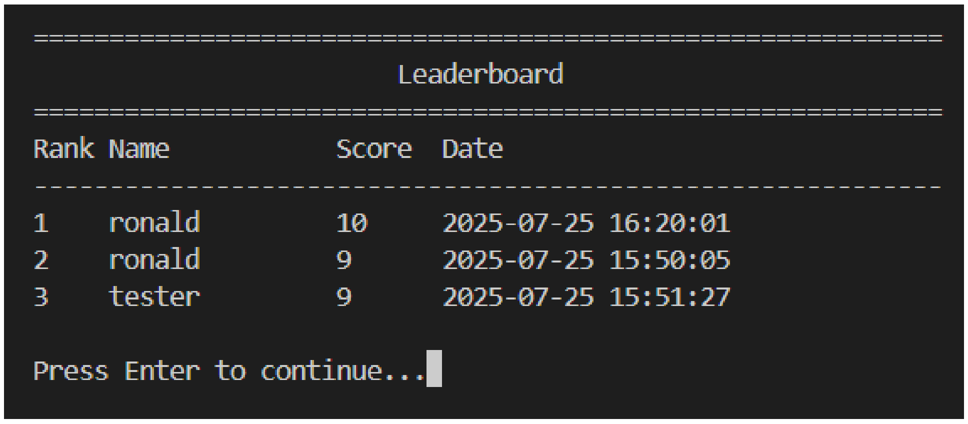


```
============================================================
                        Quiz Results
============================================================
Quiz complete! ronald, your score: 10/10
Correct: 10, Incorrect: 0, Skipped: 0
Performance: Excellent (100%)

Great job! You got all questions correct.

Press Enter to continue...
```

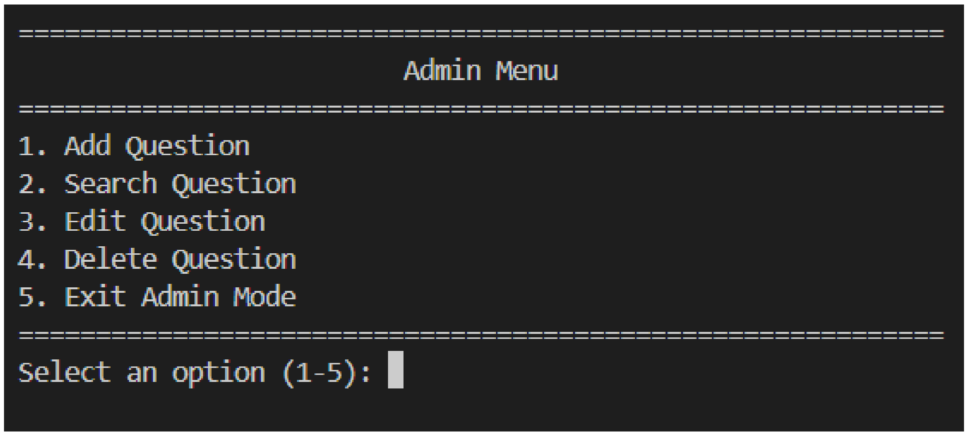
Diagram 6: Quiz Results



```
============================================================
                        Leaderboard
============================================================
Rank Name            Score  Date
------------------------------------------------------------
1    ronald          10     2025-07-25 16:20:01
2    ronald          9      2025-07-25 15:50:05
3    tester          9      2025-07-25 15:51:27

Press Enter to continue...
```

Diagram 7: Leaderboard



```
============================================================
                        Admin Menu
============================================================
1. Add Question
2. Search Question
3. Edit Question
4. Delete Question
5. Exit Admin Mode
============================================================
Select an option (1-5):
```

Diagram 8: Admin Menu

3. Individual Contributions

Effective collaboration and a clear division of labor were critical to the successful completion of the Smart Online Quiz System. The project was divided among five team members, with each member taking ownership of specific modules and responsibilities, from initial design and coding to testing and final documentation.

3.1 Task Division Among Team Members

To ensure all aspects of the project were covered systematically, the team assigned specific roles to each member. This approach allowed for focused development and accountability. The table below outlines the primary role and key responsibilities for each of the five team members.

Team Member Name	**Role**	**Key Responsibilities**
Ronald Lee Kai Ren	Project Lead / Architect	● Oversaw the overall project direction and architecture. ● Designed the initial file structure and data flow. ● Developed the main controller (`main.py`) and integrated all modules. ● Managed team coordination and resolved integration issues.
Alvin Tan Jun Hong	Quiz Module Lead	● Developed all core functionalities within `quiz.py`. ● Implemented question loading, quiz-taking logic, scoring, and feedback mechanisms. ● Handled the logic for randomizing questions.

Team Member Name	**Role**	**Key Responsibilities**
Aldred Chee Kin Hoe	Admin Module Lead	● Developed the password-protected admin.py module. ● Implemented all administrative functions: adding, searching, editing, and deleting questions. ● Ensured secure and robust handling of the question database.
Eric Ding Sheng Kiet	Data & Utilities Lead	● Developed the helper functions in utils.py (e.g., input validation, screen clearing). ● Designed and managed the data storage files (questions.json, leaderboard.json). ● Implemented the leaderboard saving and viewing logic.
ChzenZiXi	QA & Documentation Lead	● Conducted thorough testing of all system modules. ● Created test cases and documented bugs and their resolutions. ● Wrote the final project report, including system design, flow diagrams, and conclusion. ● Prepared the README.md file.

3.2 Who Worked on Which Module

The following provides a more detailed breakdown of which team member was responsible for the primary development of each specific component of the system:

- `main.py`: The main application driver was primarily developed by *Ronald Lee Kai Ren*, who ensured that all other modules were correctly imported and called from the central menu.

- `quiz.py`: All user-facing quiz logic was developed by *Alvin Tan Jun Hong*. This included loading questions, managing the quiz session, calculating scores, and providing user feedback.

- `admin.py`: The administrative functions were the responsibility of *Aldred Chee Kin Hoe*, who handled the secure, password-protected logic for managing the quiz questions.

- `utils.py`: The shared utility functions were created by *Eric Ding Sheng Kiet* to support the other modules with common tasks like input validation and data formatting.

- Data Files (`questions.json`, `leaderboard.json`): The structure and management of the data files were handled by *Eric Ding Sheng Kiet*, while the content and integrity were tested and validated by *ChzenZiXi*.

- Project Report and `README.md`: The comprehensive final report and the user-facing `README.md` file were compiled and written by *ChzenZiXi*, with input and review from all team members.

This division of tasks allowed the team to work in parallel, maximizing efficiency while ensuring that each component was developed with a high level of detail and quality.

To streamline collaboration and avoid version conflicts, the team used GitHub as a version control platform. This enabled team members to push and pull code updates, resolve merge conflicts, and track changes across all development branches effectively. It also ensured safe backup and code synchronization, especially during integration and debugging stages.

4. Testing and Results

Thorough testing was conducted to ensure the Smart Online Quiz System (SOQS) functioned as intended, met all specified requirements, and provided a reliable user experience. This section outlines the testing methodology, provides example test cases, and refers to the visual evidence of test outcomes.

4.1 How the System Was Tested

The testing process involved a combination of manual testing and systematic verification of each module's functionality. The primary approach was black-box testing, where the system's external behavior was validated against the expected outcomes without knowledge of its internal code structure.

Key aspects of the testing methodology included:

- Unit Testing (Conceptual): Although formal unit testing frameworks were not used, each function and module (e.g., `load_questions()`, `take_quiz()`, `add_question()`) was individually tested to ensure it performed its specific task correctly before integration.
- Integration Testing: Once individual modules were deemed functional, they were integrated and tested together to verify seamless interaction and data flow between different parts of the system (e.g., loading questions, taking a quiz, and then saving the result to the leaderboard).
- Functional Testing: The core functionalities of the system, as outlined in the project requirements, were rigorously tested. This included:
    - Quiz Taking: Verifying that questions are displayed correctly, user input is accepted and validated, scores are calculated accurately, and feedback is provided.
    - Leaderboard Management: Ensuring results are saved correctly, the leaderboard displays the top scores in descending order, and timestamps are present.

  - Admin Mode: Testing password protection, adding new questions (with validation for duplicates, options, and correct answers), searching for questions, and editing/deleting questions (with confirmation).

- Error Handling Testing: Various invalid inputs and edge cases were deliberately introduced to test the system's error handling mechanisms (e.g., entering non-numeric input for choices, attempting to load a non-existent file, providing incorrect admin passwords).
- Usability Testing: The system's ease of use from a user perspective was informally assessed, focusing on clear prompts, understandable feedback, and intuitive navigation within the command-line interface.

4.2 Example Test Cases

Below are some example test cases used to validate the system's functionality:

Test Case ID	**Feature Tested**	**Test Description**	**Expected Outcome**
TC01	Quiz Taking	User starts a quiz and answers all 10 questions correctly	Final score is 100%, feedback = “Excellent”, results saved to leaderboard
TC02	Quiz Taking	User skips a question or enters an invalid answer (e.g., letter instead of number)	System prompts for valid input; does not crash
TC03	Quiz Taking	User completes quiz with mixed correct/incorrect answers	System calculates correct score, gives feedback, saves result
TC04	View Leaderboard	User selects “View Leaderboard” from main menu	Leaderboard displays sorted scores with usernames and timestamps

TC05	Admin Login	User enters correct admin password	Access to admin menu granted
TC06	Admin Login	User enters incorrect admin password	Access denied with error message
TC07	Add Question (Admin)	Admin adds a new question with a valid format	Question successfully saved to questions.json
TC08	Add Question (Admin)	Admin attempts to add a question with a missing option or answer	System shows validation error and does not save question
TC09	Edit Question (Admin)	Admin edits an existing question	Question content updated and saved correctly
TC10	Delete Question (Admin)	Admin deletes a question and confirms deletion	Question removed from database
TC11	Search Question (Admin)	Admin searches for a question using a keyword	System returns matching question(s) or notifies if none found
TC12	Error Handling	User enters invalid menu option (e.g., number out of range or letter)	Error message displayed, system prompts again without crashing
TC13	Data Persistence	User completes a quiz and exits the system	User result remains saved in leaderboard.json after restart
TC14	Exit System	User chooses "Exit" from main menu	Program terminates gracefully

5. Challenges Faced

Developing the Smart Online Quiz System (SOQS) presented several technical and team-related challenges that required innovative solutions and adaptive strategies. These challenges, and how the team collectively overcame them, are detailed below:

5.1 Technical Challenges: New to System Development and Python

Challenge:

A primary challenge for the team was the collective inexperience with developing a complete system from scratch, particularly using Python. Many team members were new to Python's intricacies, including its syntax, common libraries, and best practices for structuring larger applications. This led to initial difficulties in understanding how to integrate different modules, manage data flow, and implement robust error handling.

Solution:

To address this, the team adopted a proactive and collaborative learning approach:

- Leveraging Online Resources: Extensive use was made of online tutorials, documentation (e.g., Python's official docs, Stack Overflow), and educational platforms (e.g., YouTube, free online courses). Each team member was encouraged to research specific functionalities before attempting implementation.
- Pair Programming and Code Reviews: Regular pair programming sessions were conducted to tackle complex features, allowing knowledge transfer and immediate debugging. Code reviews were also implemented to catch errors early, ensure adherence to coding standards, and improve overall code quality.
- Incremental Development: The project was broken down into smaller, manageable tasks. This allowed the team to learn and apply new concepts incrementally, building confidence and momentum as each module was completed and integrated.

5.2 Team-Related Challenges: Task Allocation and Collaboration

Challenge:

As a group new to collaborative software development, the team faced initial uncertainty regarding effective task allocation and ensuring equitable distribution of workload. Without a clear framework, there was a risk of duplication of effort, missed functionalities, or an uneven burden on individual members. The project officially began on 9th July 2025, with a submission deadline of 5th August 2025. With a limited timeline of less than four weeks, it was important to plan and divide work efficiently.

Solution:

The team employed strategic tools and methods to streamline task allocation and foster effective collaboration:

- Utilizing AI and Online References: The team consulted AI tools like ChatGPT and various online project management guides to understand common roles and responsibilities in software development teams. This helped in identifying key modules and functionalities that could be assigned to specific individuals based on their strengths or areas they wished to develop.
- Defined Roles and Responsibilities: Based on the insights gained, clear roles (e.g., Project Lead, Quiz Module Lead, Admin Module Lead, Data & Utilities Lead, QA & Documentation Lead) were formally assigned to each member. This ensured that every component had a dedicated owner responsible for its development and integration.
- Regular Stand-up Meetings: Daily or bi-weekly stand-up meetings were held to discuss progress, identify roadblocks, and re-allocate tasks if necessary. This transparent communication ensured everyone was aware of the project status and could offer support where needed.
- GitHub was used as the version control platform to manage source code, track contributions, and support parallel development without code conflicts.

6. Conclusion and Reflection

The development of the Smart Online Quiz System (SOQS) was a significant learning experience for the team, providing practical exposure to software development principles and collaborative teamwork. Despite the initial challenges, the project successfully achieved its primary objective of creating a functional, modular, and interactive quiz application.

6.1 Lessons Learned

This project offered invaluable insights into various aspects of programming and project management:

- Importance of Modular Design: Breaking down the system into distinct modules (e.g., `quiz.py`, `admin.py`, `utils.py`) significantly improved code organization, readability, and maintainability. It also facilitated parallel development among team members.
- Practical Application of Python Concepts: The project provided a hands-on opportunity to apply core Python concepts such as data structures (lists, dictionaries for questions and leaderboard), file handling (JSON for persistence), control flow, functions, and error handling in a real-world context.
- Value of Collaborative Tools and Communication: Effective use of communication channels and, if applicable, version control systems (like Git/GitHub) was crucial for seamless integration of individual contributions and resolving conflicts. Regular meetings ensured everyone was aligned and aware of progress and roadblocks.
- Problem-Solving and Debugging Skills: Encountering and resolving technical issues, from file I/O errors to logical bugs in quiz scoring, honed the team's problem-solving and debugging capabilities. The iterative process of testing and refining was key to achieving a stable application.
- Adaptability and Resourcefulness: Being new to system development and Python, the team learned the importance of being resourceful, actively seeking out and utilizing online documentation, tutorials, and AI tools to bridge knowledge gaps.

6.2 Suggestions for Future Improvements

While the current version of the SOQS meets the assignment requirements, several areas have been identified for future enhancements to improve its functionality, user experience, and robustness:

- Graphical User Interface (GUI): Transitioning from a command-line interface to a GUI (e.g., using Tkinter, PyQt, or web-based frameworks like Flask/Django with HTML/CSS/JS) would significantly enhance user experience, making the system more intuitive and visually appealing.
- Enhanced Admin Features:
    - Question Editing Interface: Implement a more user-friendly interface for editing questions, allowing admins to easily select and modify specific parts of a question or its options.
    - Bulk Question Import: Add functionality to import questions from a CSV or Excel file, simplifying the process of populating the question database.
    - User Management: For a multi-user system, implement features for managing user accounts, including registration, login for regular users, and password reset options.
- Advanced Quiz Features:
    - Multiple Quiz Categories: Allow questions to be categorized, enabling users to select quizzes based on specific topics.
    - Question Difficulty Levels: Introduce difficulty levels for questions, allowing for adaptive quizzes or selection of quizzes based on desired challenge.
    - Detailed Performance Analytics: Provide more in-depth feedback to users, such as performance trends over time, areas of weakness, and personalized study recommendations.

- Improved Data Persistence and Security:
    - Database Integration: Migrate from JSON/CSV files to a more robust database system (e.g., SQLite, PostgreSQL) for better data management, scalability, and query performance, especially for larger datasets.
    - Enhanced Security: Strengthen the admin password protection with hashing and salting. For a web-based system, implement proper authentication and authorization mechanisms.
- Automated Testing: Implement unit tests and integration tests to automate the testing process, ensuring that new features or bug fixes do not introduce regressions.

These suggestions represent potential avenues for further development, transforming the SOQS into an even more comprehensive and user-friendly application. Overall, the SOQS project provided a valuable introduction to real-world software development, and we are proud of the results.

7. References

1.	Welcome to Python.org. (2025, July 22). Python.org. https://www.python.org
2.	W3Schools.com. (n.d.). https://www.w3schools.com/python
3.	freeCodeCamp.org. (2022, August 9). *Python for Beginners – Full course [Programming Tutorial]* [Video]. YouTube. https://www.youtube.com/watch?v=eWRfhZUzrAc
4.	JSON. (n.d.). https://www.json.org/json-en.html
5.	Make online quizzes in minutes - Quiz maker. (n.d.). https://www.quiz-maker.com
6.	Play & Create quiz - quiz.com. (n.d.). *Quiz.com*. https://quiz.com
7.	Home - Kahoot! (n.d.). *Create, Explore, and Host Kahoots | Kahoot!*. https://create.kahoot.it
8.	Your Sets | Quizlet. (n.d.). *Quizlet*. https://quizlet.com
9.	freeCodeCamp.org. (2020, May 28). *Git and GitHub for Beginners - Crash Course* [Video]. YouTube. https://www.youtube.com/watch?v=RGOj5yH7evk
10.	Team roles and contributions. (2025, July 24). *ChatGPT*. https://chatgpt.com/s/t_688183101e78819188518207abd609a8