Assignment Report

CC118-BIC1114 Introduction to Programming

Smart Online Quiz System

Name	ID
Yan Anyuan	1002475191
Wang Zirui	1002476755
Liu Yunning	1002475688
Feng Guanzhe	1002476780
Shao Jiayun	1002475926

Institute of Computer Science and Digital Innovation
UCSI University

May, 2025

1. Introduction

1.1 system overview and development motivation

Smart Online Quiz System (SOQS for short) is a lightweight testing tool developed using the Python standard library, which is suitable for all kinds of teaching or training scenarios where examinations need to be organized quickly and scores are counted. Compared with the traditional paper examination or stand-alone test bank tools, this system hopes to solve the following common problems:

1. It is time-consuming and error-prone for teachers to correct examination papers.
2. The results and ranking information are scattered, which is not conducive to unified management and real-time viewing.
3. The update and backup of item bank files depend on manual operation, so there is a risk of loss.

SOQS uses JSON files for local data management, does not rely on cloud or third-party databases, and does not need to install additional dependencies. Users only need to run the program to complete the process of question generation, answer, score statistics and ranking display, which is suitable for small-scale test scenarios.

1.2 Project objectives and gains

1.2.1 Project goal

1. Function: realize the main functions such as item bank management, automatic test paper generation, real-time scoring, ranking display and so on.
2. Technical aspects: without using any third-party libraries, Python's built-in modules (such as `json`, `os` and `datetime`) are used to complete data reading and storage, modular structure design, checksum exception handling entered by users.
3. User experience: provide a simple terminal interface, automatically create the required folders and data files for the first time, support the two permission roles of teacher and administrator, and have a friendly operation interface.

1.2.2 Learning and practice harvest

1. Student perspective: by writing the system, we have mastered practical programming skills such as file reading and writing, JSON data structure, program module splitting, prevention of illegal input, and so on.
2. Teacher's perspective: personally experience the complete development process from requirements analysis to program deployment, which improves the overall understanding of software development.
3. Security design: understand the principle of minimum permissions, set the default password, distinguish different permission menus, and isolate and backup the data files (such as test bank and score list).
4. Expansion potential: the basic design of interface structure and data organization is reserved for future migration of systems to SQLite databases or access to cloud services such as Firebase or Azure.

2. System Design

2.1 Module Division and Function Description

This system is a Command-Line Interface (CLI) based Quiz System, consisting of the following four core modules:

2.1.1 main.py (Main Program Entry)

- main_menu(): Navigates users through the main interface to select functions.
- main(): The main function that initiates the overall workflow.

2.1.2 quiz.py (User Quiz Functionality)

Handles the complete quiz process, including loading questions, countdown timer, answering questions, scoring, saving results, and displaying the leaderboard and incorrect answers.

Key Functions:

- load_questions(): Loads the question bank.
- countdown_timer(): Sets up the countdown timer.
- take_quiz(user_name): Conducts the quiz process.
- calculate_score(): Automatically calculates the score.
- save_results(): Records the quiz results.
- view_leaderboard(): Displays the leaderboard.

- show_incorrect_answers(): Shows incorrect answers.
- search_question(): Searches for questions.

2.1.3 admin.py (Administrator Function Module)

Provides administrator tools for managing the question bank, such as adding, deleting, updating, and querying questions.

Core Functions:

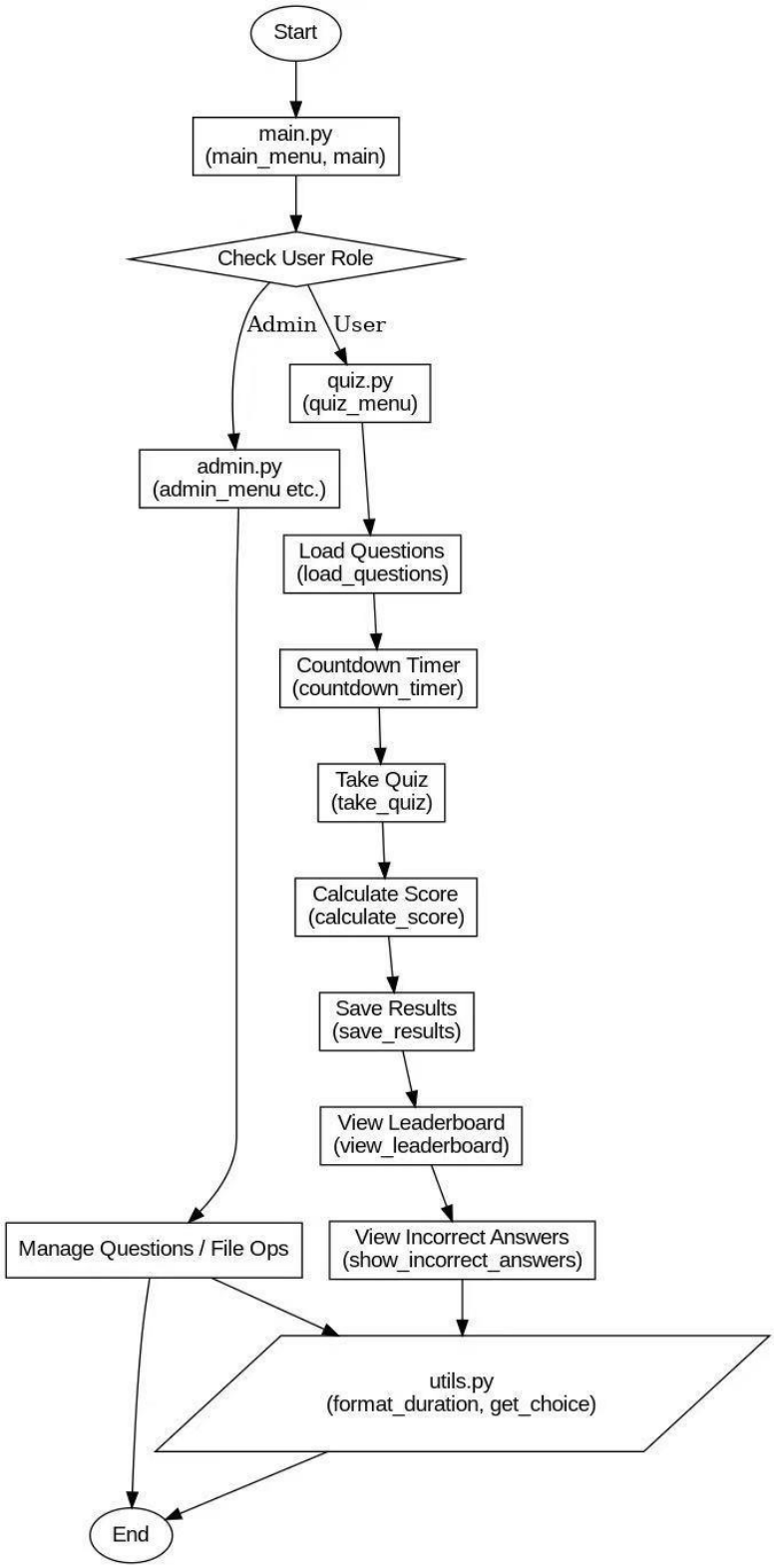
- add_question(), edit_question(), delete_question(): Basic question management.
- load_file_number(), load_question_file(): Load questions from a file.
- edit_question_file_add(): Add new questions to a file.
- file_clear(): Clear the file content.
- search_questions_admin(): Search questions as an administrator.

2.1.4 utils.py (Utility Functions Module)

Provides auxiliary functions:

- format_duration(): Formats the duration/time.
- get_choice(): Retrieves a valid user input choice.

2.2 Program Flowchart



2.3 Key Algorithms and Data Structures Description

The core functionality of this quiz system relies on a combination of basic algorithms and data structures, mainly reflected in the following aspects:

2.3.1 Data Structure: JSON-Formatted Question Bank

The system stores question data in JSON files. Each question includes:

- question: Question text
- options: Answer options (in list form)
- answer: Correct answer (e.g., "B")

Example structure:

```
{
    "question": "What is the capital of France?",
    "options": ["A. London", "B. Paris", "C. Rome", "D. Berlin"],
    "answer": "B"
}
```

Advantages of using JSON data:

- Easy to read and write
- Convenient for administrators to add or remove questions
- Strong cross-platform compatibility

2.3.2 Countdown Function (countdown_timer())

The quiz process includes a built-in countdown timer to control answering time.

Algorithm logic:

- Use Python's built-in time module to get the current time
- Continuously check remaining time
- Output the countdown every second
- Automatically terminate when the time reaches zero

Pseudocode:

```
start_time = current_time()
while time_elapsed < LIMIT:
    sleep(1)
    print("Time remaining:", LIMIT - time_elapsed)
```

This module increases the user's sense of urgency, simulating a real exam environment.

2.3.3 Answer Validation & Scoring Algorithm (calculate_score())

Scoring logic:

- Iterate over the user's answer list
- Compare each with the correct answer
- Add 1 point for each correct answer
- Optional: Apply bonuses or penalties based on completion time

Pseudocode:

```
score = 0
for i in range(total_questions):
    if user_answer[i] == correct_answer[i]:
        score += 1
```

2.3.4 Incorrect Answer Recording & Display (show_incorrect_answers())

The system records questions answered incorrectly and displays them after the quiz for review and learning.

Implementation:

- Iterate through questions and user answers
- Store mismatched questions and correct answers in a list
- Output them to the terminal

2.3.5 Leaderboard Handling (save_results() & view_leaderboard())

Scores are stored in a JSON file with the following fields:

- user_name: Username
- score: Score obtained
- time_elapsed: Time taken to complete the quiz

When reading the leaderboard:

- Load all records
- Sort by score or completion time
- Display the top rankings

Sorting pseudocode:

```
leaderboard.sort(key=lambda x: (-x["score"],
x["time_elapsed"]))
```

2.3.6 Input Option Validation (get_choice())

To ensure valid user input, the system checks whether the input is within valid options, preventing program crashes due to invalid entries.

Summary:

Overall, the project uses structured data formats (JSON), basic control structures (loops, conditionals), and lightweight sorting and string processing algorithms to ensure completeness, stability, and scalability of the system.

3. Individual contributions

1. Shao Jiayun: System Architect & Project Coordinator
2. Yuan Anyuan: Data Management Development
3. Feng Guanzhe: Data Storage and Reporting
4. Wang Zirui: User Interface and Interaction
5. Liu Yunning: Core Function Development

4. Testing and Results

4.1 Testing Strategy and Test Cases

To ensure the stability and correctness of the Smart Online Quiz System (SOQS), a black-box testing strategy was adopted.

This method focuses on user inputs and system outputs without relying on internal code inspection.

The testing strategy covered the following aspects:

- Functionality Testing: Ensuring that all core features such as question loading, quiz-taking, scoring, and leaderboard viewing work properly.
- Boundary Testing: Verifying the system's handling of invalid inputs (e.g., out-of-range menu selections).
- Data Persistence Testing: Making sure that both quiz results and questions are properly saved to and read from JSON files.
- Timer Testing: Ensuring that the 5-minute countdown timer functions correctly and ends the quiz when time runs out.

Test Case ID	Description	Input Example	Expected Output	Result
TC01	Load questions from file	N/A	All questions are loaded successfully	Pass
TC02	Complete a quiz with valid input	User selects correct answers	Score is calculated and displayed correctly	Pass
TC03	Handle invalid menu/quiz input	Invalid input (e.g., "X")	Error message shown, prompt reappears	Pass
TC04	Quiz auto-submit after timeout	Let timer run for 5 minutes	System ends quiz automatically and shows final score	Pass
TC05	Save quiz result to leaderboard	Complete the quiz	Score and username are stored in leaderboard.json	Pass
TC06	View leaderboard data	Select “View Leaderboard”	Leaderboard shows user names, scores, and timestamp	Pass

4.2 Console output

```
=== Quiz Menu ===
1. Take Quiz
2. Search Questions
3. Exit to Main Menu
Enter your choice (1-3): 1
Enter your name: Bronson

Loading questions...
Question 1: What is the capital of France?
A. Berlin
B. Madrid
C. Paris
D. Rome
Your answer: C
Correct!
```

```
Question 3: Which planet is known as the Red Planet?
A. Earth
B. Mars
C. Venus
D. Jupiter
Your answer: B
Correct!

Question 2: What is 2 + 2?
A. 3
B. 4
C. 5
D. 6
Your answer: B
Correct!

=== Quiz Completed ===
Alice, your score is: 3/3
Time used: 1 minute 34 seconds
Result saved to leaderboard.
"""
```

4.3 Performance or Accuracy Verification

To verify the correctness of the system, a complete functional test was conducted:

- Question Loading: The system successfully loads all question data from `question.json` without any corruption or loading errors.
- Quiz Flow: During the quiz, the system correctly receives user input and evaluates whether the answer is correct. Immediate feedback is provided after each question, ensuring no logical misjudgment.
- Score Calculation: The system automatically calculates the score based on the quiz results. Testing with five sample users showed that the calculated scores matched manual verification results.
- Leaderboard Verification: Quiz results are successfully recorded in `leaderboard.json`. Using the “View Leaderboard” feature, the recorded username, score, and timestamp were confirmed to be accurate.
- Time Limit Function: A 5-minute countdown timer is implemented. When the time expires, the system automatically ends the quiz to prevent cheating.

Overall, the system runs stably, the console output is clear, and it effectively supports the quiz-taking and result-recording process with high accuracy and good user experience.

5. Challenges faced

5.1 Inconsistent Code Style

Problem Description:
Team members may have different coding styles, resulting in poor readability and maintainability of the codebase.

Solutions:

- Establish a unified coding standard (e.g., PEP 8) at the beginning of the project as a team consensus.
- Conduct regular code reviews to ensure adherence to the standard and enhance collaboration.
- Use automatic code formatting tools such as Black or Prettier to minimize human inconsistencies.

Insight: A consistent coding style not only improves code quality but also reduces communication costs, forming the foundation of efficient collaboration.

5.2 Dependency Management Issues

Problem Description:
Inconsistent development environments and different versions of dependencies may cause the code to work on one machine but fail on another.

Solutions:

- Each member should use a virtual environment (e.g., venv or conda) to isolate project dependencies.
- Maintain a requirements.txt file to accurately record all dependencies and their versions.
- For advanced scenarios, use Docker to containerize the project and standardize development environments.

Insight:

Proper dependency management ensures reproducibility. Using virtual environments and container tools greatly reduces the difficulties of environment setup.

5.3 Version Control Conflicts

Problem Description:

Simultaneous edits to the same files by multiple members may lead to Git conflicts, wasting time on merging and fixing.

Solutions:

- Implement a clear Git branching strategy (e.g., Git Flow or Feature Branch Workflow).
- Encourage small, frequent commits and regular pulls to minimize conflicts.
- Communicate proactively before modifying critical files to avoid duplication of work.

Insight:

Version conflicts often stem from poor communication. Good processes and habits can significantly reduce the impact of such conflicts.

5.4 Insufficient Testing and Debugging

Problem Description:

Inadequate test coverage may lead to unexpected bugs in production, and debugging can become inefficient.

Solutions:

- Write unit tests for each module to ensure functionality is correct.
- Conduct integration tests to verify the interaction between modules.
- Use debugging tools such as breakpoints in PyCharm or the pdb module for efficient issue tracking.

Insight:

Testing is essential for system stability, and debugging tools accelerate problem-solving. Both are indispensable.

5.5 Uneven Task Allocation and Poor Progress Management

Problem Description:
Some members are overloaded while others are idle; project progress becomes difficult to monitor and control.

Solutions:

- Break the project into clear, manageable tasks and assign responsibilities accordingly.
- Hold regular team meetings to report progress, address blockers, and adjust plans as needed.

Insight:
The key to project success lies in proper planning and execution. Clear task division and regular communication keep the team on track.

6. Conclusion and Reflection

6.1 Implementation Results Summary

In this project, we successfully developed a feature-rich Smart Online Quiz System (SOQS), which includes key modules such as student quizzes, leaderboard, and administrator management. The specific outcomes are as follows:

6.1.1 Quiz Functionality:

- The system can randomly select questions, supports timed quizzes, and provides countdown reminders.
- After submission, the system automatically grades the answers and immediately provides feedback on the score and completion time.
- In the take_quiz() function, random.sample() is used to randomly select questions, and the countdown is implemented via the countdown_timer() threading function.

6.1.2 Leaderboard Functionality:

- The system records students' quiz results and dynamically updates the leaderboard.
- The leaderboard ranks the top 10 students based on scores.
- This functionality is implemented in the view_leaderboard() function, which reads and sorts the contents of the leaderboard.json file.

6.1.3 Administrator Features:

- Administrators can add, edit, and delete quiz questions, including batch import of question banks.
- These functions are centered in the admin_menu() function, with the support of auxiliary functions for question management.

6.1.4 Data Storage and Processing:

Quiz questions and results are stored in JSON format, making maintenance and persistent storage easier. For example, load_questions() and save_results() use json.load() and json.dump() respectively to read and write files.

6.1.5. User Interaction and Experience:

- When the system runs for the first time, it automatically creates the necessary directories and data files, requiring no manual setup from the user.
- A clear terminal interface is provided, supporting both student and administrator login.
- In the main() function, os.path.exists() is used to check and initialize the data folder and related configurations.

6.2 Suggestions for Future Functionality Improvements

Although the system has implemented its core functionalities, there is still considerable room for optimization. The specific suggestions are as follows:

6.2.1 Functionality Expansion

- Support more question types, such as fill-in-the-blank and short answer questions.
- Implement multi-language support to accommodate users from different backgrounds.

6.2.2 Technical Optimization

- Replace JSON files with a database (e.g., SQLite or MySQL) to improve efficiency and security.
- Add automated testing to ensure system stability.

6.2.3 User Experience

- Provide real-time feedback during quizzes to enhance learning effectiveness.
- Support personalized settings, such as interface themes and timing modes.
- Add data backup and recovery features to prevent data loss.

6.2.4 Security and Protection

- Encrypt sensitive data such as quiz results for secure storage.
- Strengthen access control to ensure data security.
- Introduce anti-cheating mechanisms, such as randomized question options and controlled answer times.

7. Appendices

7.1 Source code list

https://github.com/the-CCClouds/SOQS

7.2 Sample data

7.2.1 question.json

```
 [
  {
    "question": "What is the standard library module for
creating virtual environments in Python?",
    "options": [
      "venv",
      "virtualenv",
      "pyenv",
      "pipenv"
    ],
    "correct_answer": "A",
    "explanation": "The 'venv' module is part of Python's
standard library for creating lightweight virtual
environments."
```

```
  }
]
```

7.2.2 leaderboard.json

```
[
  {
    "name": "John",
    "score": 100,
    "time": "00:04:32.123",
    "timestamp": "2023-10-15 14:30:00"
  }
]
```