

QUIZE COMPITITION USING PYTHON

¹Aryan Sandalwar, ²Anurag Khobragade, ³Mohit Gothe, ⁴Kartik Chandankhede

¹²³⁴Department of Electronics and Telecommunication Engineering, Jagadambha College of Engineering and Technology, Yavatmal, India

Abstract - This competition serves as a platform for students to showcase their academic prowess and quick-thinking abilities across a diverse range of subjects, including science, history, literature, and current affairs. Designed to foster a spirit of healthy competition and intellectual curiosity, the event utilizes a multi-round format transitioning from written preliminaries to a high-stakes buzzer round. By challenging participants to retrieve information under pressure, the [Name of Competition] aims to promote critical thinking, teamwork, and the pursuit of knowledge beyond the standard curriculum. A quiz competition is an effective method for testing knowledge, improving memory, and developing quick decision-making skills. This project presents a Quiz Competition system using Python that converts a traditional quiz into a structured digital application. The system can present questions to participants, provide multiple answer options, record responses, calculate scores automatically, and display results at the end of the competition. The proposed system is designed to support educational and competitive environments such as schools, colleges, classrooms, training programs, and online learning activities. A Python-based implementation provides a simple and flexible foundation for managing questions and answers. Features such as timers, random question selection, instant feedback, score tracking, and graphical user interaction can be added according to the requirements of the competition. The project also emphasizes proper preparation and organization of quiz content before implementation. Questions, options, correct answers, categories, marks, and time limits should be stored in a structured form so that the application remains easy to maintain and expand. Python data structures such as lists and dictionaries can be used for small quizzes, while JSON or text files can be used for larger question banks. The system encourages practical programming skills because the developer must combine data handling, conditional logic, functions, user input, interface design, scoring logic, and result generation. Therefore, the project demonstrates how Python can be applied to create a useful educational application while improving both programming knowledge and interactive learning.

Keywords - Quiz competition, Python, HTML, JavaScript

I. INTRODUCTION

A quiz competition is an interactive activity in which participants answer questions within a defined set of rules and time limits. Quizzes are commonly used for academic assessment, entertainment, awareness programs, technical competitions, and skill development. Traditional quiz competitions often require printed question papers, manual checking, physical scoreboards, and significant effort from organizers.

With the advancement of computer technology, these activities can be converted into digital systems. A computer-based quiz can display questions on a screen, accept answers from participants, check responses automatically, and calculate scores without requiring manual evaluation. This reduces repetitive work and improves consistency.

Python is suitable for this type of project because it has a simple syntax, extensive libraries, and strong support for data structures. A basic console version can be created using standard Python features, while a graphical version can be developed using Tkinter or other GUI frameworks. Web-based extensions can also use HTML and JavaScript for the interface while Python handles application logic or server-side processing.

The proposed Quiz Competition using Python can be organized into several stages: participant entry, quiz instructions, question presentation, answer selection, time management, answer evaluation, score calculation, and final result display. The modular structure makes it possible to add new features without rewriting the entire application.

II. OBJECTIVES OF THE PROJECT

- To develop a digital quiz competition system using Python.
- To reduce manual effort involved in checking answers and calculating scores.
- To provide a simple and user-friendly method for conducting quizzes.
- To organize questions, options, answers, categories, and scores systematically.
- To provide immediate feedback and final results to participants.

- To demonstrate practical use of Python programming concepts.
- To support different subjects and question categories.
- To make the quiz system flexible enough for future improvements.
- To improve participant engagement through timers, random questions, and interactive interfaces.
- To provide a foundation that can later be extended to web or database-based quiz applications.

III. SCOPE OF THE PROJECT

The scope of the project covers the design and implementation of a Python-based quiz competition system. The application can be used for academic quizzes, technical events, general knowledge competitions, classroom activities, and practice tests.

- Question bank management.
- Multiple-choice question support.
- Automatic answer verification.
- Automatic score calculation.
- Timer-based questions or complete quiz timing.
- Random selection of questions.
- Category-wise quizzes.
- Participant information collection.
- Result and score display.
- Graphical interface using Tkinter where required.
- Storage of questions using lists, dictionaries, JSON files, or text files.
- Future integration with databases, authentication, leaderboards, and online competitions.

IV. EASE OF USE

Ease of use is an important requirement of a quiz application because participants should be able to concentrate on answering questions rather than understanding complicated controls. The system should provide clear instructions, readable questions, visible answer options, and simple navigation.

4.1 Easy to Start and Play

Users should be able to start the quiz with a simple button or command. The application can display a welcome screen followed by participant instructions and the first question. Unnecessary configuration should be avoided.

4.2 Clear Instructions and Questions

Questions and answer options should be displayed in a readable format. Instructions should explain the number of

questions, marks for each answer, time limits, and rules before the competition begins.

4.3 Instant Feedback

The application can provide immediate feedback after an answer is submitted. It can indicate whether the response was correct and update the score. In a formal competition, feedback may instead be hidden until the final result to avoid revealing answers.

4.4 Automatic Scoring System

Automatic scoring removes the need for manual calculation. When the participant submits an answer, the program compares it with the stored correct answer and updates the score according to the defined marking scheme.

4.5 Simple Navigation

Navigation buttons such as Start, Next, Previous, Submit, and Finish can be used in a graphical interface. The participant should always know which question is currently active and how many questions remain.

V. SYSTEM REQUIREMENTS

5.1 Hardware Requirements

- Computer or laptop with a basic modern processor.
- At least 4 GB RAM for a simple Python application.
- Keyboard and mouse or touch-enabled input where supported.
- Display monitor for graphical quiz interfaces.
- Optional network connection for online features.

5.2 Software Requirements

- Python 3.x.
- Python IDE such as IDLE, VS Code, PyCharm, or another suitable editor.
- Tkinter for a basic Python graphical interface, if required.
- HTML and JavaScript for a browser-based interface, if required.
- JSON or text files for question storage.
- Operating system such as Windows, Linux, or macOS.

VI. PREPARATION OF QUIZ CONTENT

Before writing the program, the quiz content should be prepared carefully. The quality of the application depends heavily on the quality and accuracy of its questions.

Preparing the content first also prevents unnecessary changes during coding.

6.1 Define the Objective

The developer should decide whether the quiz is intended for education, entertainment, competitive assessment, practice, or general awareness. The objective determines the difficulty level, question type, scoring rules, and time limits.

6.2 Prepare Questions and Answers

Questions should be clear, grammatically understandable, and relevant to the selected subject. Every question must have a verified correct answer. For multiple-choice questions, incorrect options should be plausible but clearly distinguishable from the correct answer.

6.3 Choose Question Format

- Multiple-choice questions with four or more options.
- True or false questions.
- Short-answer questions.
- Numerical questions.
- Category-based questions.

6.4 Set Rules and Scoring

The organizer should define marks for correct answers, treatment of incorrect answers, whether negative marking is used, time limits, tie-breaking rules, and the conditions for completing the quiz.

6.5 Test Content Before Implementation

All questions, spellings, options, answers, marks, and categories should be checked before they are inserted into the program. A single incorrect answer in the question bank can produce an incorrect result even when the program itself is working correctly.

VII. ORGANIZATION OF QUIZ DATA

Proper data organization makes the quiz system easier to maintain and extend. Python provides lists, dictionaries, tuples, and other data structures that can represent questions and answers.

7.1 Using Lists

Lists are suitable for small quizzes. Questions, options, and answers can be stored as separate lists or as nested lists. This approach is simple and easy to understand for beginners.

7.2 Using Dictionaries

A dictionary can store a question together with its options and correct answer. For example:

```
quiz = {
    "What is the capital of India?": {
        "options": ["Mumbai", "Delhi", "Kolkata", "Chennai"],
        "answer": "Delhi"
    },
    "2 + 2 = ?": {
        "options": ["3", "4", "5", "6"],
        "answer": "4"
    }
}
```

7.3 Using JSON Files

JSON files are useful when the question bank becomes larger. Questions can be edited without modifying the main Python program. The program reads the JSON file at runtime and converts the stored information into Python data structures.

7.4 Using Text Files

Text files provide a simple storage method for basic applications. Questions and answers can be stored line by line and read by Python. This approach is easy to implement but provides less structure than JSON for complex question banks.

VIII. SYSTEM WORKFLOW

The complete quiz process can be divided into a sequence of stages. A clear workflow reduces logical errors and makes the program easier to understand.

- Launch the quiz application.
- Display the welcome screen and competition instructions.
- Collect participant information if required.
- Select the quiz category or subject.
- Load the question bank.
- Randomize questions when random selection is enabled.
- Display the current question and available options.
- Start or update the timer.
- Accept the participant's answer.
- Compare the selected answer with the stored correct answer.
- Update the score according to the marking scheme.
- Move to the next question.

- Repeat the process until all questions are completed or the time expires.
- Calculate the final score and percentage.
- Display the result and relevant feedback.
- Store or export the result if required.

IX. MAIN MODULES OF THE SYSTEM

9.1 User/Participant Module

This module collects basic participant information such as name, roll number, class, or team name. The information can be displayed with the final score.

9.2 Question Bank Module

The question bank stores questions, answer options, correct answers, categories, difficulty levels, and marks. Separating question data from program logic makes future updates easier.

9.3 Quiz Engine

The quiz engine controls the main sequence of the competition. It loads questions, displays them, accepts responses, checks answers, updates scores, and moves between questions.

9.4 Timer Module

A timer can be implemented to create time pressure. It can operate per question or for the complete quiz. When the time expires, the application can automatically submit the quiz or move to the result screen.

9.5 Scoring Module

The scoring module calculates marks based on the competition rules. It can support positive marks, negative marks, unanswered questions, and percentage calculation.

9.6 Result Module

At the end of the quiz, the result module displays the participant's score, number of correct answers, incorrect answers, unanswered questions, and percentage. It can also provide a performance summary.

9.7 User Interface Module

A graphical interface can be developed using Tkinter. The interface can include labels, buttons, radio buttons, progress indicators, timers, and result panels. A web version can use HTML and JavaScript for a browser-based interface.

X. USING A PROGRAMMING TEMPLATE

A programming template provides a predefined structure for organizing the quiz application. It separates the major functions of the program and makes the code easier to read, test, and modify.

- Provides a predefined structure for the quiz.
- Separates question data, quiz logic, and interface code.
- Reduces development time.
- Makes customization easier.
- Improves consistency and code readability.
- Reduces repeated coding and common implementation errors.
- Makes it easier to add new features later.

10.1 Example Program Structure

```
def start_quiz():
    print("Welcome to Quiz Competition")

def ask_questions():
    # display questions and process answers
    pass

def calculate_score():
    # calculate final score
    pass

def show_result(score):
    print("Your Score:", score)

# Main Program
start_quiz()
ask_questions()
score = calculate_score()
show_result(score)
```

XI. PYTHON IMPLEMENTATION CONCEPTS

11.1 Variables

Variables store information such as the participant name, current question number, selected answer, score, and remaining time.

11.2 Conditional Statements

If-else conditions are used to determine whether the selected answer matches the correct answer and to apply the appropriate score.

11.3 Loops

Loops can be used to process multiple questions without writing the same code repeatedly. A for loop is useful when the number of questions is known, while a while loop can be useful for condition-based execution.

11.4 Functions

Functions divide the program into manageable components. Functions such as `start_quiz()`, `ask_questions()`, `calculate_score()`, and `show_result()` make the code easier to test and maintain.

11.5 Randomization

Python's random module can be used to shuffle questions or answer options. Randomization reduces the possibility that every participant receives questions in exactly the same sequence.

11.6 File Handling

Python file handling allows the application to read questions from external files and store results. JSON is particularly useful for structured question data.

XII. SAMPLE SCORING LOGIC

A simple scoring system may award one mark for every correct answer and zero marks for an incorrect or unanswered question. The following example illustrates the logic conceptually:

- Correct answer: +1 mark.
- Incorrect answer: 0 mark.
- Unanswered question: 0 mark.
- Final score: total marks obtained.
- Percentage: $(\text{marks obtained} / \text{maximum marks}) \times 100$.

The actual marking scheme can be changed according to the competition requirements. Negative marking can also be added if the organizer defines such a rule.

XIII. GRAPHICAL USER INTERFACE

A graphical user interface makes the quiz easier to operate than a command-line interface. Tkinter is included with standard Python installations and can be used to create windows, labels, buttons, radio buttons, frames, and other controls.

- Welcome screen with quiz title.
- Participant information form.
- Category selection.
- Question and answer display area.

- Next and submit controls.
- Timer or progress indicator.
- Current score display where appropriate.
- Final result screen.
- Restart or exit option.

The existing document also contains a screenshot on page 3 showing a graphical quiz-style dashboard with menu options and categories. This demonstrates how a GUI can present the system more clearly than a plain text interface.

XIV. TESTING AND ERROR HANDLING

Testing is necessary to ensure that the quiz produces correct results in different situations. The program should be tested with normal inputs as well as unexpected inputs. Check that every question has the correct answer stored.

- Check that only valid options can be selected.
- Test the first and last question.
- Test submission without selecting an answer.
- Test timer expiration.
- Test random question selection.
- Verify score calculation manually with known test cases.
- Check that the application does not crash when the question bank is empty or incorrectly formatted.
- Test result generation after completing all questions.
- Test restarting the quiz and resetting the score.

XV. ADVANTAGES

- Automatic answer checking reduces manual work.
- Automatic scoring reduces calculation errors.
- Questions can be stored and updated easily.
- The application can support different subjects and categories.
- Randomized questions can improve competition fairness.
- Timers can increase engagement and simulate competitive conditions.
- Python provides a simple development environment.
- The system can be extended with GUI, database, web, and leaderboard features.
- The same basic architecture can be reused for different quizzes.

XVI. LIMITATIONS

- A basic local Python application may not support multiple participants simultaneously.
- Question quality depends on the accuracy of the prepared question bank.

- A simple file-based system provides limited security.
- Online competitions require additional networking and server-side components.
- Advanced authentication and database functionality require additional development.
- A GUI made only for desktop Python may not automatically work as a mobile application.

XVII. FUTURE SCOPE

The project can be expanded considerably beyond a basic Python quiz. A database can be introduced to store users, questions, categories, results, and competition history. Authentication can allow participants to log in securely. A leaderboard can rank participants according to competition rules.

- Online multiplayer quiz competitions.
- Real-time leaderboard.
- Admin panel for question management.
- Database integration using SQLite, MySQL, or another database.
- User authentication and role-based access.
- Difficulty-based question selection.
- Adaptive quizzes based on participant performance.
- Result history and performance analytics.
- Export results to PDF or Excel.
- Web and mobile versions.
- Audio and image-based questions.
- AI-assisted question generation with human verification.
- Automatic category and difficulty tagging.
- Cloud-based question bank and synchronization.

XVIII. IMPORTANCE OF PREPARING THE CONTENT

Preparing the quiz content before development is one of the most important steps in the project. It ensures that programming effort is spent on implementation rather than repeatedly changing questions and rules.

- Ensures accuracy of questions and answers.
- Saves development time.
- Improves the overall quality of the quiz.
- Makes question data easier to organize.
- Reduces logical and content-related errors.
- Creates a smooth sequence from start to result.
- Makes future updates easier.
- Improves the participant experience.

XIX. CONCLUSION

The Quiz Competition using Python project demonstrates how a traditional academic or competitive activity can be converted into a structured digital application. Python provides the programming foundation required for question management, answer validation, scoring, timing, randomization, and result generation.

The project also demonstrates practical programming concepts such as variables, lists, dictionaries, functions, loops, conditional statements, file handling, and graphical user interfaces. With proper preparation of quiz content and a modular program structure, the application can be adapted for different subjects and competition formats.

The basic system can serve as the foundation for a larger online quiz platform with databases, authentication, leaderboards, analytics, and multi-user support. Thus, the project provides both an educational programming exercise and a practical application for conducting knowledge-based competitions.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to everyone who supported me in completing this project on "Quiz Competition using Python." I would like to thank my teacher/guide for valuable guidance, encouragement, and continuous support throughout the development of the project. Their suggestions helped improve the technical understanding and presentation of the work.

I am also thankful to my friends and classmates for their help, ideas, testing, and feedback during the project work. Their participation helped identify areas where the quiz system could be improved.

I would also like to acknowledge the online documentation, tutorials, programming resources, and study materials that helped in understanding Python programming and application development. Finally, I am grateful to my family for their support and encouragement.

REFERENCES

- 1) Mehta A, Tripathi K, Bansal H. Quiz as an adjunctive modality in teaching clinical microbiology for undergraduate medical students. *Adv Biosci Clin Med* 9: 31, 2021.
- 2) Priego-Quesada JI, Jimenez-Perez I, Cibrián Ortiz de Anda RM, González-Peña R, Salvador Palmer R. Effect of in-class group clicker-quiz competition on student final exam performance. *Adv Physiol Educ* 43: 430–434, 2019.
- 3) Anonymous. Best Evidence Medical Education (BEME): report of meeting—3–5 December 1999, London, UK. *Med Teach* 22: 242–245, 2000.

- 4) Taware G, Tapadia J. Importance of quiz competitions in teaching and learning. *J Dental Med Sci* 14: 19–23, 2015.
- 5) Shanmugavelu G, Ariffin K, Vadivelu M, Mahayudin Z, Sundaram MA. Questioning techniques and teachers' role in the classroom. *Education* 8: 45–49, 2020.
- 6) Barthakur R, Srivastava R, Thourani S, Thourani MK, Dawka S. Quiz gaming competitions for undergraduate medical students: questioning the MediQuiz. *Arch Med Biomed Res* 1: 103–112, 2014.
- 7) Cheng HM. Stimulating student interest in physiology: the intermedical school physiology quiz. *Adv Physiol Educ* 34: 20–21, 2010.
- 8) Bell K, Gerrits R, Hood S. The effect of cooperative quizzes on student performance and anxiety in community college human physiology courses. *HAPS Educ* 25: 24–35, 2021.
- 9) Jensen M, Moore R, Hatch J. Cooperative learning: part I: cooperative quizzes. *Am Biol Teach* 64: 29–34, 2002.
- 10) Johnson DW, Johnson RT, Smith KA. *Active Learning: Cooperation in the College Classroom*. Edina, MN; Interaction Book Company, 2006.
- 11) Aslan KT, Ay P, Kaş D, Tosun F, Yürükcü R, Kekeç E, Şahin MF, Apaydın Kaya I. Adaptation and validation of the Turkish version of the vaccine hesitancy 5 point Likert scale. *Hum Vaccin Immunother* 17: 5176–5182, 2021.
- 12) Talsania N, Barot D, Chaudhari A. Quiz versus didactic lecture on undergraduate students of BJ Medical College, Ahmedabad: a cross-sectional, comparative, and interventional study. *Int J Sci Study* 3: 86–91, 2015.
- 13) Devi K. Quiz as an innovative approach in teaching community medicine to medical students. *Natl J Comm Med* 5: 182–185, 2014. Google Scholar
- 14) Dengri C, Gill A, Chopra J, Dengri C, Koritala T, Khedr A, Korsapati AR, Adhikari R, Jain S, Zec S, Chand M, Kashyap R, Pattan V, Khan SA, Jain N. A review of the quiz, as a new dimension in medical education. *Cureus* 13: e18854, 2021.
- 15) Pallant J. *SPSS Survival Manual*. Maidenhead, UK: Open University Press, 2020.