**Module 2: Python Programming**

**Project Report**

**Project Title - Quiz Application**

**Name - FLORENCE C TUNGOE**

A report submitted in part fulfilment of the certificate of

**Artificial Intelligence Programming Assistance**

**(2024-2025)**

**Guidance:** Ms. Arpita Roy

****

**NSTI (W) Kolkata**

**Date**-

# **Abstract**

This project, titled **Quiz Application Using Python**, presents an interactive command-line learning tool aimed at testing foundational Python programming knowledge. Unlike many GUI or web-based platforms, this tool emphasizes simplicity and offline functionality using core Python and the Colorama library.

Designed primarily for beginners and students, the application delivers a structured quiz with 10 multiple-choice questions focused on Python concepts such as data types, syntax, functions, loops, and error handling. Each question includes four options, and users receive instant color-coded feedback. green for correct, red for incorrect, based on their selection.

What distinguishes this project is its lightweight nature and practical design. Requiring only a Python interpreter and the Colorama package, it is accessible on virtually any system, including low-resource environments. The quiz system stores questions, options, and answers in tuples and processes them using loops and conditional logic.

This application serves as both a revision tool and a foundational programming exercise, with potential for expansion into GUI or web-based formats. It encourages active learning through interactive engagement and real-time feedback.

# **Acknowledgement**

With deep respect and genuine appreciation, I extend my heartfelt thanks to **Ms. Arpita Roy**, whose unwavering support and insightful mentorship guided me throughout the development of this project. Her ability to balance technical precision with compassionate teaching inspired me to approach programming not just as a skill, but as a craft that combines logic, clarity, and creativity.

This project, though seemingly simple, became a gateway for me to understand how foundational coding practices translate into real-world applications. It was under her guidance that I learned to think beyond syntax—understanding user experience, code structure, and the importance of writing purposeful programs.

I would also like to thank **NSTIW Kolkata**, not merely as an institution, but as a space where curiosity is nurtured and learners are empowered. The access to learning resources, the collaborative environment, and the encouragement to build independent projects made this journey all the more meaningful.

Lastly, I acknowledge the silent contributions of every bug, every unexpected output, and every line of code that refused to run because it was through solving these problems that I grew not just as a programmer, but as a patient and persistent learner.

# **Table of content**

Table of Contents

[**Abstract** 2](#_Toc202431018)

[**Acknowledgement** 3](#_Toc202431019)

[**Table of content** 5](#_Toc202431020)

[**Table of figures** 6](#_Toc202431021)

[**Problem Statement** 6](#_Toc202431022)

[**Literature Review** 7](#_Toc202431023)

[**Proposed Solution** 8](#_Toc202431024)

[**Requirements** 8](#_Toc202431025)

[**Test Scenarios** 15](#_Toc202431026)

[**Observations** 16](#_Toc202431027)

[**Steps to Deploy:** 16](#_Toc202431028)

[**Summarize:** 18](#_Toc202431029)

[**Future ideas:** 18](#_Toc202431030)

[**References** 19](#_Toc202431031)

[**Appendix** 19](#_Toc202431032)

# **Table of figures**

| **Figure No.** | **Title** | **Page No.** |
| --- | --- | --- |
| 1. | Program Flow of Quiz Application | 11-12 |
| 2. | Terminal View: Sample Quiz Question | 18 |
| 3. | Terminal Output: Final Results Display | 19 |

# **Problem Statement**

* **What is the use case?**

The primary use case of the **Quiz Application** is to provide a lightweight, offline-friendly, and interactive tool for learners to test their knowledge of Python programming fundamentals. It is particularly useful for:

* **Students** preparing for coding exams, interviews, or class assessments.
* **Beginners** learning Python who want to validate their understanding through quick quizzes.
* **Instructors** who want to demonstrate basic Python input/output, control flow, and data handling through a hands-on example.
* **Self-learners** seeking a terminal-based tool that runs without needing complex installations or internet access.

The application functions as both a revision tool and a real-time feedback system, reinforcing correct knowledge and clarifying misconceptions instantly.

* **Who benefits?**

This Quiz Application is designed to benefit a broad range of users, especially those engaged in learning or teaching Python:

* **Students** in computer science or vocational training programs who need a practical, interactive way to assess their understanding of Python fundamentals.
* **Educators and Trainers** who want a simple, no-setup-required tool to demonstrate multiple-choice quiz logic or basic Python constructs like loops, conditionals, and user input/output.
* **Self-learners and Beginners** who are exploring Python and need a quick, offline tool to test their knowledge without relying on online platforms.
* **Institutes** looking to incorporate terminal-based mini projects into coursework to strengthen coding logic and foundational programming concepts.

The tool encourages independent learning and can also be used in classroom environments as a live demonstration or hands-on assignment.

# **Literature Review**

In recent years, interactive learning tools such as quizzes have become a common method to reinforce knowledge, especially in computer science education. Online platforms like **HackerRank**, **Codecademy**, and **W3Schools** have popularized quiz-based learning by providing rich interfaces that test a learner’s understanding of programming concepts. These tools, while effective, often rely on internet connectivity and browser-based environments, which can limit accessibility for learners with limited resources or bandwidth.

From an academic standpoint, multiple-choice questions (MCQs) have been proven effective in evaluating recall, comprehension, and application-level knowledge. MCQ-based assessments are widely used in e-learning systems because they provide quick feedback, scalability, and measurable learning outcomes. Most of the available platforms, however, are web-based and do not focus on minimal computing environments.

In contrast, this project explores the design of a **terminal-based Python quiz application** that delivers a similar learning experience without a graphical user interface or network dependency. The implementation draws inspiration from basic quiz logic but adapts it for offline use in a local command-line environment. Additionally, the use of Python’s Colorama library enhances user experience with color-coded feedback, despite the absence of a GUI.

This project stands out by focusing on the **educational utility of MCQs delivered through simple Python constructs**, making it especially accessible for beginners and environments with limited computing capabilities.

# **Proposed Solution**

To address the need for a simple, interactive, and offline tool for practicing Python programming concepts, this project proposes a **command-line based Quiz Application** developed in Python. The core idea is to deliver multiple-choice questions in an engaging format that runs entirely in the terminal, without requiring any external interface, internet access, or advanced software setup.

The application is designed with the following key features:

* 10 MCQs covering essential Python concepts
* Answer choices displayed per question
* Immediate correctness feedback
* Score and time reported at end
* Randomized order of questions using random.shuffle()
* Simple, readable flow using for loops and dictionaries

This solution serves as a practical tool for **self-assessment**, **revision**, and **demonstrating programming logic** using Python’s core constructs. It is especially well-suited for educational institutions, beginners, and offline learning environments.

# **Requirements**

**Technology Stack:**

 **Programming Language**: Python 3

*  **Library Used**:

random – to shuffle question order

time – to calculate total time taken

**Hardware:**

 A basic computer or laptop with any operating system (Windows, Linux, or macOS)

 Minimum 512MB RAM and a standard keyboard for input

**Software:**

 Python 3.x installed on the system

**Deployment Environment:**

Runs in any terminal or command-line interface (Command Prompt, PowerShell, Bash, etc.)

 No web server, database, or GUI environment is required

 Offline execution supported—ideal for classrooms, training labs, or self-learning setups

**Design Documentation**

The design of the Quiz Application follows a simple, linear structure that guides the user through a series of predefined multiple-choice questions using Python’s core programming constructs. The focus is on readability, maintainability, and clarity of user interaction in a terminal environment.

**Key Components of the Design:**

* **Questions** are stored as a list of dictionaries with keys: question, choices, and answer
* Questions are shuffled at runtime
* A loop prints the question and options, and receives user input
* The answer is compared against the correct choice and feedback is printed
* Score is incremented per correct answer
* Timer records start and end time using time.time()
* Final result includes score and total time taken

**Overall Structure:**  
The program is sequential and event-driven, with clearly defined input-output phases and feedback mechanisms. It is designed to be intuitive for users and easy to extend or adapt for future improvements.

**Implementation Details**

The implementation of the Quiz Application is straightforward, using procedural programming principles and Python’s built-in data types. Below are the main components and how they are implemented:

import random

import time

questions = [

{"question": "What is the output of the following code?\nprint(2 \*\* 3)",

"choices": ["A. 6", "B. 8", "C. 9", "D. 12"], "answer": "B"},

{"question": "Which of the following is a mutable data type in Python?",

"choices": ["A. Tuple", "B. String", "C. List", "D. Integer"], "answer": "C"},

{"question": "What is the correct syntax to create a function in Python?",

"choices": ["A. def functionName():", "B. function functionName()",

"C. create function functionName()", "D. functionName()"], "answer": "A"},

{"question": "How do you insert an element at the end of a list in Python?",

"choices": ["A. list.add(element)", "B. list.append(element)",

"C. list.insert(element)", "D. list.push(element)"], "answer": "B"},

{"question": "What is the output of the following code?\nx = [1, 2, 3]\nprint(x[1])",

"choices": ["A. 1", "B. 2", "C. 3", "D. Error"], "answer": "B"},

{"question": "Which of the following is used to handle exceptions in Python?",

"choices": ["A. try and except", "B. catch and throw",

"C. handle and except", "D. try and catch"], "answer": "A"},

{"question": "What is the output of the following code?\nfor i in range(3):\n print(i)",

"choices": ["A. 0 1 2", "B. 1 2 3", "C. 0 1 2 3", "D. 1 2 3 4"], "answer": "A"},

{"question": "Which method removes whitespace from beginning and end of a string?",

"choices": ["A. strip()", "B. trim()", "C. remove()", "D. delete()"], "answer": "A"},

{"question": "What is the output of the following code?\nx = {'a': 1, 'b': 2}\nprint(x['a'])",

"choices": ["A. 1", "B. 2", "C. 'a'", "D. Error"], "answer": "A"},

{"question": "Which is the correct way to import a module in Python?",

"choices": ["A. import module", "B. include module",

"C. require module", "D. using module"], "answer": "A"}

]

random.shuffle(questions)

score = 0

start\_time = time.time()

for q in questions:

print(q["question"])

for choice in q["choices"]:

print(choice)

answer = input("Your answer: ")

if answer.upper() == q["answer"]:

print("Correct!")

score += 1

else:

print(f"Wrong! The correct answer is {q['answer']}.")

print()

end\_time = time.time()

total\_time = end\_time - start\_time

minutes, seconds = divmod(total\_time, 60)

print(f"Your final score is: {score}/{len(questions)}")

print(f"Time taken: {int(minutes)} minutes and {int(seconds)} seconds")

**Testing**

The Quiz Application was tested through manual execution in various terminal environments to ensure its functionality, correctness, and user interaction flow. The testing process focused on validating question display, input handling, answer checking, score computation, and terminal output formatting.

### **Test Scenarios**

* User answers are validated correctly (case insensitive)
* Time taken and score displayed at end
* Input is raw and accepts unexpected entries (e.g. "E", numbers)
* Questions appear in different order each time due to random.shuffle()

### **Observations**

* All major functionalities worked as expected.
* The application handles continuous input and feedback smoothly.
* Lack of input validation (e.g., preventing inputs outside A–D) is noted but does not break the quiz.

**Deployment**

The deployment of the Quiz Application is straightforward, as it is a terminal-based Python script with no external dependencies beyond a single third-party library. It can be executed on any system that supports Python 3.

### **Steps to Deploy:**

1. **Prerequisites:**

Ensure Python 3 is installed on the system.

No third-party libraries are needed.

1. **Copy the Script**

Place quiz\_application.py in any folder.

1. **Run the Application**

Open a terminal or command prompt.

Navigate to the script directory.

Execute the command:

* + python quiz\_application.py

1. **Execution Environment**

Verified to work on:

Windows Command Prompt and PowerShell

Linux and macOS terminals

IDE-integrated terminals like VS Code and PyCharm

1. **No Internet or Server Required**

No GUI, database, or internet required.

**Challenges Faced**

During the development of the Quiz Application, several challenges were encountered, each contributing to a deeper understanding of programming logic, user interaction, and terminal behavior:

1. **Input Handling**
   * Lowercase inputs like 'a' were converted to uppercase using .upper() to match answer keys.
2. **Input Validation**
   * No strict validation for A–D; invalid entries were treated as wrong.
3. **Code Reusability**
   * The current script is not modular. Functions could be used to improve readability and reuse.
4. **Static Content**
   * Questions are hardcoded. A scalable version should load questions from external files.
5. **UI Limitations**
   * Being CLI-based, there is no graphical interface. However, it ensures compatibility with low-resource systems.

Despite these challenges, all functionalities worked well, and the application provides a solid base for future feature expansion.

**Conclusions and Future Work**

### **Summarize:**

**What worked well?**

* The core logic — displaying questions, capturing user input, validating answers, and calculating scores — worked smoothly.
* The use of Python dictionaries provided a clean and structured approach to question and answers management.
* The inclusion of random.shuffle() enabled variability in quiz order for every attempt.
* Minimal setup and dependency-free architecture ensured broad compatibility.

**What needs improvement?**

* The application currently accepts invalid inputs like "E" or empty responses; stricter input validation is needed.
* The questions are hardcoded within the script, making it less scalable for large sets or frequent updates.
* Lack of modular structure makes future feature additions or testing less maintainable.
* No support for saving results or exporting performance summaries

### **Future ideas:**

* Add input validation for only A–D
* Allow saving results to a file
* Add GUI support using Tkinter
* Support timed modes or difficulty levels
* Option to load questions from external JSON or CSV files

# **References**

**Tutorials followed:**

* Python Docs (https://docs.python.org)
* W3Schools Python MCQs
* RealPython tutorials

# **Appendix**

**Include:**

* **Code snippets:**

1. **Question and Option Definitions**  
   Questions are stored as a list of dictionaries. Each dictionary contains:

* question: The quiz question
* choices: A list of answer options (strings)
* answer: The correct answer letter (e.g., "A")

questions = [

{

"question": "What is the output of the following code?\nprint(2 \*\* 3)",

"choices": ["A. 6", "B. 8", "C. 9", "D. 12"],

"answer": "B"

},

...

]

1. **Question Loop and Answer Validation**  
   Each question is displayed in a shuffled order. User input is collected and compared case-insensitively to the correct answer:

python

Copy code

import random

import time

random.shuffle(questions)

score = 0

start\_time = time.time()

for q in questions:

print(q["question"])

for choice in q["choices"]:

print(choice)

answer = input("Your answer: ")

if answer.upper() == q["answer"]:

print("Correct!")

score += 1

else:

print(f"Wrong! The correct answer is {q['answer']}.")

print()

1. **Final Score and Timing Display**

end\_time = time.time()

total\_time = end\_time - start\_time

minutes, seconds = divmod(total\_time, 60)

print(f"Your final score is: {score}/{len(questions)}")

print(f"Time taken: {int(minutes)} minutes and {int(seconds)} seconds")

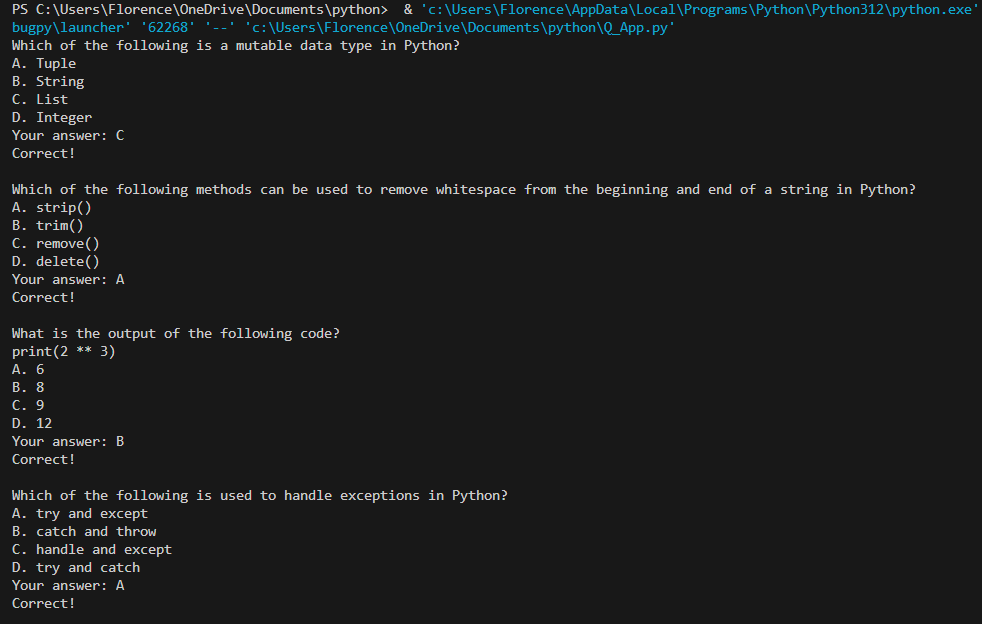
1. **Libraries Used**

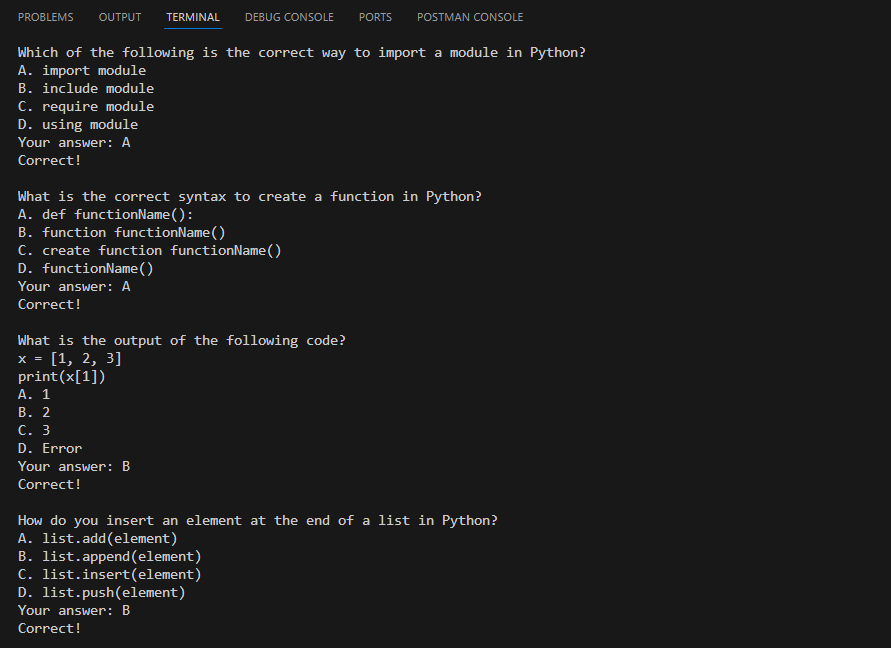
import random

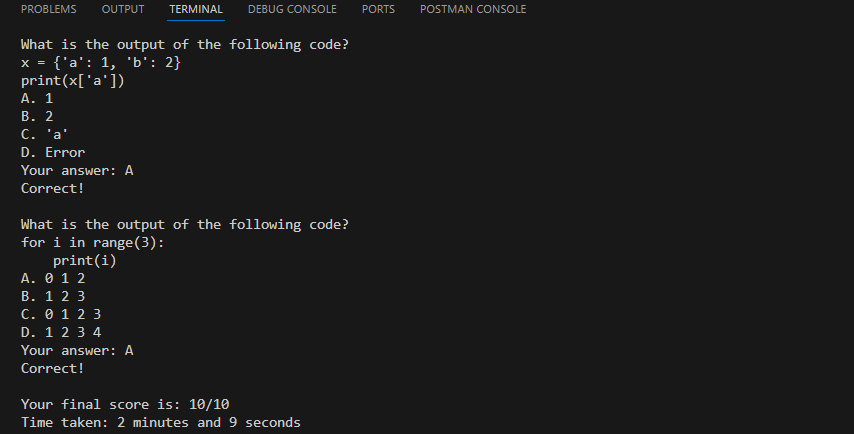
import time

**Output**

* Displays each question with multiple choices
* Accepts user input (case-insensitive)
* Gives instant feedback on correctness
* Shows final score and time taken

****

****

****

* **Additional graphs:**

This is a CLI-based project; therefore, no graphs were generated. Graphs could be considered for future versions with GUI enhancements or score analytics.

* **GitHub link:** [Kiki27tungs/Quiz\_App](https://github.com/Kiki27tungs/Quiz_App)