**QUIZ GAME**

NAME: ANJALI ARUN CHAVAN

ORGANIZATION: MICRO IT

TOPIC: GUIZ GAME

**INTRODUCTION**

A quiz game in Python is an interactive program that tests a user's knowledge on various subjects by asking multiple-choice questions. The code you provided is a great example of such a quiz game, offering a user-friendly experience with several advanced features.

**OBJECTIVES OF THE PROJECT**

This Python quiz application aims to create an interactive learning experience with three core goals:

🎯 Core Objectives

Knowledge Assessment

Test users' understanding in Python programming, Data Structures, and General Knowledge

Evaluate proficiency across easy, moderate, and hard difficulty levels

Adaptive Learning

Automatically adjust question distribution based on user-specified total marks

Provide weighted scoring (1/2/3 marks per difficulty level)

Enable self-paced learning through answer reviews

User Engagement

Offer randomized questions for varied attempts

Support early exits with progress preservation

Deliver instant feedback and performance metrics

💡 Key Value Propositions

Scalable Design: Easily expandable question banks

Educational Tool: Helps identify knowledge gaps through post-quiz review

In this project, the quiz game is an interactive, command-line Python application that allows users to test their knowledge in various subjects by answering multiple-choice questions. The game is designed to be both educational and engaging, providing immediate feedback and a summary of results.

Key Elements of the Quiz Game

1. Subject Choice

The user can select from three subjects:

Python

General Knowledge

Data Structure

2. Difficulty Levels

Each subject contains questions categorized as:

Easy

Moderate

Hard

The distribution of questions is based on the total marks specified by the user.

3. Randomized Questions

Questions are randomly selected from the question bank for each difficulty level, making each quiz attempt unique.

4. Scoring System

Points are awarded based on the difficulty of the question:

Easy: 1 mark

Moderate: 2 marks

Hard: 3 marks

The user's total score and percentage are calculated at the end.

5. User Interaction

The user answers each question by typing the option (a/b/c/d).

The user can exit the quiz at any time by typing 'exit'.

After the quiz, the user can choose to see the correct answers for all questions.

6. Learning and Feedback

The quiz provides instant feedback on the user's performance.

The option to review correct answers helps users learn and improve.

**TYPES OF INVESTMENT ASSETS**

The Python code you provided is an educational quiz application, not a finance or investment project. It focuses on testing knowledge in three subjects:

Python Programming

General Knowledge

Data Structures

The "assets" in this project are:

Questions and Answers (categorized by subject and difficulty

User input and scoring logic

There are no financial or investment assets such as:

Stocks

Bonds

Real estate

Mutual funds

Commodities

Cryptocurrencies

**RISK AND RETURN ANALYSIS**

1. What is Risk and Return Analysis?

In finance, risk and return analysis refers to evaluating the potential risks (chances of loss) and expected returns (gains) of an investment asset, such as stocks, bonds, or real estate.

It involves statistical tools and financial metrics like standard deviation, Sharpe ratio, beta, expected return, etc.

2. What Does This Project Do?

This Python project is a quiz game that:

Lets users choose a subject (Python, General Knowledge, Data Structures)

Asks questions of varying difficulty

Scores the user's answers

Displays the total score and percentage

Optionally shows correct answer

3. Is There Any Risk or Return?

No financial risk or return is involved.

The only "risk" is the challenge of answering questions incorrectly, which might affect the user's score.

The only "return" is the knowledge gained and the quiz score.

4. If You Want to Draw an Analogy:

Risk: The chance of answering a question incorrectly (lower score).

Return: The marks/score achieved by answering correctly.

But: This is just a metaphor; there is no real-world financial risk or reward

**QUIZ GAME CONSTRUCTION PROCESS**

1. Subject and Question Bank Setup

Subjects: The game offers three subjects: Python, General Knowledge, and Data Structures.

Question Bank: Each subject contains a dictionary of questions, further categorized into three difficulty levels: easy, moderate, and hard.

Question Format:

Each question is a key, and its value is a list containing four options (a–d) and the correct answer.

2. User Interaction and Input

The user is prompted to select a subject by entering a number (1/2/3).

The program checks if the input is valid; if not, it exits.

The user is then asked to enter the total marks for the quiz, which determines the number of questions from each difficulty.

3. Question Distribution Logic

The total marks are split into:

Easy: 50% of total marks

Moderate: 20% of total marks

Difficult: 30% of total marks

The number of questions for each level is calculated based on the marks and the scoring system (easy=1, moderate=2, hard=3 marks per question).

4. Random Question Selection

The program randomly selects questions from each difficulty pool using random.sample(), ensuring no repeats and a unique quiz each time.

5. Quiz Execution Loop

For each selected question:

The question and its options are displayed.

The user inputs their answer (a/b/c/d) or can type 'exit' to quit early.

The answer is checked against the correct one:

If correct: marks are added according to difficulty.

If incorrect: only the total possible marks are incremented.

User responses are stored for later review.

6. Result Calculation and Display

After the quiz (or early exit), the program calculates:

Total marks obtained

Percentage score

These are displayed to the user.

7. Review of Correct Answers

The user is asked if they want to see all correct answers.

If yes, the program displays each question, the user's answer, and the correct answer.

**TOOLS USED IN QUIZ GAME MANAGEMENT**

1. C Programming Language

The entire calculator is written in C, a powerful and efficient programming language suitable for console-based applications.

2. Standard Input/Output Library (stdio.h)

#include <stdio.h>: This header file is essential for using input/output functions like printf() and scanf().

3. Arrays

double numbers;: An array is used to store up to 100 numbers entered by the user for calculations.

4. Looping Constructs

While Loop (while (1)): Used to keep the calculator running until the user chooses to exit.

For Loop (for (i = 0; i < n; i++)): Used to input numbers and perform operations on them.

5. Conditional Statements

If-Else Statements: Used for menu selection, input validation (number of inputs), and division by zero checks.

Switch Statement: Used to select the operation (addition, subtraction, multiplication, division) based on user input.

6. User Input Handling

scanf(): For reading user choices and numbers.

Input validation ensures the user enters a valid number of operands and avoids division by zero.

7. Arithmetic Operations

Addition, Subtraction, Multiplication, Division: Performed using respective operators (+, -, \*, /) in loops.

8. Error Handling

Checks for invalid menu choices.

Checks for division by zero and notifies the user if such an error occurs.

9. Result Display

printf(): Used to display the operation and result in a user-friendly format.

**KEY LEARNING FROM INTERNSHIP**

1. Python Programming Skills

Data Structures:

Learned to use dictionaries and lists to organize complex data (subjects, questions, options, answers).

Randomization:

Used the random module to implement random sampling, ensuring each quiz session is unique.

Input/Output Handling:

Gained experience with user input validation and formatted output for a smooth user experience.

2. Logic and Control Flow

Conditional Statements:

Applied if-else and switch-like logic to control quiz flow and validate user choices.

Loops:

Used loops to iterate over questions and manage repeated user interactions.

3. User Experience Design

Interactive CLI Design:

Built a command-line interface that guides users step-by-step, making the quiz engaging and easy to use.

Error Handling:

Implemented checks for invalid inputs and provided user-friendly error messages.

4. Modular and Scalable Code

Separation of Concerns:

Structured the code so that adding more subjects or questions is straightforward.

Reusability:

Designed the question bank and quiz logic to be easily reusable for other topics or quiz formats.

5. Assessment and Feedback Mechanisms

Scoring System:

Developed a weighted scoring system based on question difficulty, providing a fair assessment.

Review Feature:

Added a post-quiz review option to help users learn from their mistakes.

6. Project Management and Documentation

Requirements Analysis:

Understood how to translate requirements (e.g., subject selection, random questions, scoring) into working code.

Documentation:

Practiced commenting and explaining code for clarity and future maintenance.

7. Professional and Soft Skills

Problem-Solving:

Tackled challenges like dynamic question allocation, randomization, and user input errors.

Attention to Detail:

Ensured accurate scoring, proper question selection, and robust error handling.

**CONCLUSION**

The quiz game project developed in Python successfully demonstrates the design and implementation of an interactive, subject-based assessment tool. Through this project, a robust command-line quiz system was created that allows users to select from multiple subjects, answer questions of varying difficulty, and receive immediate feedback on their performance.