# PROJECT REPORT 2024
# _QUIZ GAME_

.

Pooja Anbalagan
22CDS0412

# Contents

# _Introduction_

The provided Python script (console based application) is an interactive quiz game developed for students aged 13-19. Designed as a fun educational tool, the game tests players' general knowledge through a series of five random questions. Players earn points for correct answers, which are stored in a cumulative score table for future reference.

## Features

❶ Player Eligibility: Only players between the ages of 13 and 19 are allowed to participate.

❷ Randomized Questions: A pool of diverse questions ensures variability in gameplay.

❸ Scoring System: Each correct answer earns 20 points, with a maximum score of 100 points.

❹ Data Persistence: Player details and scores are stored in a GAME_SUMMARY.csv file for record-keeping.

❺ Leaderboard: Displays sorted results by points and announces the top scorer(s).

❻ Friendly Interface: Clear prompts guide the player through the game process.

## Game flow

❶ Welcome Screen: Displays rules and eligibility criteria.

❷ Player Registration: Collects player information, including Index No, Name, and Age.

❸ Eligibility Check: Verifies if the player is a teenager. Non-teenagers cannot proceed.

❹ Question Rounds:

Five questions are chosen randomly from a predefined list.

The player must answer each question correctly to proceed to the next.

❺ Score Calculation: Updates points for each correct answer.

❻ Data Storage: Saves player data into a CSV file, appending new records.

❼ Summary Display: Optionally shows all players' scores, sorted leaderboard, and top scorer(s).

## Requirements

❶ Running the Script

Install Python and ensure pandas is installed (pip install pandas).

Import os and random modules.

Run the script using a terminal or IDE.

- python QUIZ_GAME.py.

❷ Interpreting Results

Players are guided through registration and gameplay via prompts.

After gameplay, a CSV file (GAME_SUMMARY.csv) logs all players' information.

If requested, the leaderboard and top scorer(s) are displayed

# _Design choices_

## ❶ User Interface

This application is designed as a console-based program. Despite being text-based, various decorative elements such as bullet points, dashed underlines (----), and symbols are incorporated to enhance the clarity and readability of the output. These features provide a structured and organized presentation, improving the overall user experience.

## ❷ Repetition and Replay-ability

The quiz game supports multiple play through using the same index number and name. However, a conditional mechanism ensures data integrity by deleting duplicate entries. A *while* loop is employed to manage the repetition process and to prompt the user for input on whether they wish to replay the game. Additionally, users can reset and restart the quiz game seamlessly when desired.

## ❸ Data Handling

The application utilizes the Pandas library for managing and presenting user inputs and player details in a structured format. This approach ensures clean and well-organized data output. Functionalities such as displaying a sorted table upon request, identifying the highest scores, generating a new sorted dataset, identifying top scorers, removing duplicate entries, and deleting records from the table demonstrate the extensive use of Pandas for efficient data handling and analysis.

# _Technical Requirements_

## ❶ Variables and Data Types

In developing the quiz game application, I utilized a variety of variables and data types, each named uniquely to clearly represent its purpose. For instance***, indexno, name***, and ***age*** are some examples. The variable ***indexno*** represents the players (or student's) index number, ***name*** signifies the player's name, and ***age*** indicates their age. The corresponding data types for these variables are as follows:

***indexno* = integer , *name* = String and *age* = integer**

## ❷ Control Flow Statements

Control flow statements were indispensable in the logic of this game. Here are some examples:

- ***while (q_count <= 5)***
- ***if (user_input.lower() == Answer.lower())***
- ***if (opinion.lower() == "no")***
- ***while (True)***

In this application, the game iterates through five questions, using the variable *q_count* to track the number of questions asked. Execution continues until this count reaches five. In the second example, the *if* statement is employed to verify whether the player's answer (*user_input*) matches the correct one (*Answer)*. If the response is accurate, the player earns points; otherwise, they lose the game.

## ❸ Functions

I incorporated functions to modularize the application. For instance:

***def game(question, Answer)***

The ***game*** function encapsulates the logic for determining if the answer is correct. It is invoked iteratively for each round until all five questions are completed or an incorrect answer terminates the game.

## ❹ Exception handling

To ensure valid user inputs and enforce the correct data types, I implemented exception handling.
For example:

```
try:
   indexno = int(input("Enter Your Index No*: ").strip())
except ValueError:
   print("")
   indexno = int(input("--Index number should be a number--\n\nEnter Your Index No*: ").strip())
```

In this instance, if the input for ***indexno*** is not an integer, the program displays an error message and prompts the user for a valid input. This mechanism guarantees data integrity and enhances user experience.

## ❺ <u>Data Structures</u>

I employed two primary data structures: lists and dictionaries.

### **Dictionaries**

A data dictionary was utilized to store player details, such as index number, name, age, and points. For instance:

```
data = pd.concat([pd.DataFrame({"Index No": [indexno], "Name": [name], "Age": [age], "Points": '-'})])
```

This dictionary was then appended to a CSV file to maintain structured records.

### **Lists**

Lists played a pivotal role in storing questions and answers, streamlining the code significantly. For example:

```
question_list = ["Which scientist is known for the Theory of Relativity?", “Who is the author of the 'Harry Potter' series?", "Who painted the 'Mona Lisa'?” …]

Answer_list = ["Albert Einstein", "J.K. Rowling", "Leonardo Da Vinci", "Argentina” …]
```

Their simplicity and versatility made them ideal for this purpose.

## ❻ <u>File I/O</u>

To store and retrieve player details, I utilized file input/output operations with a CSV file named ***GAME_SUMMARY.csv.***

For example:

```
file_name = "GAME_SUMMARY.csv"
df = pd.read_csv("GAME_SUMMARY.csv")
data.to_csv(file_name, index=False, mode="a", header=False)
```

Pandas proved invaluable for structuring and presenting the data, ensuring a clean and organized output. The final line demonstrates how player details were appended to the CSV file efficiently.

## _Challenges and Solutions_

❶ **Challenge** - Ensure the user inputs and validate.

**Solution** - Exception handling and using while loop until gets the valid input.

❷ **Challenge** - Store questions and answers efficiently and well organized.

**Solution** - Using data structures such as lists and data dictionaries.

❸ **Challenge** - Selecting random questions.

**Solution** - Using random module.

❹ **Challenge** - Store the players' data.

**Solution** - Using CSV files.

❺ **Challenge** - File operations such as display the game summary in a structured way, sort the values, delete the duplicates and find highest score and players.

**Solution** - Using pandas as data frame.

THANK YOU!!!