**Project Presentation:**

The goal of this project is to code a quiz that generates exercises directly from a code base. The principle involves creating multiple lists in JSON format and implementing them under different names, either based on difficulty or chapter. With a selection menu for difficulty levels, users can progressively enhance their learning.

This project will use JSON and Python to create a quiz based on keys defined in JSON files, allowing the Python code to retrieve information about the questions, possible answers, and correct answers. To the basic quiz, additional features will be added, such as the option to choose the difficulty level and a replay button, to explore the customizable options that can be implemented in the quiz.Code

**Description:**

The code uses several commands to manage the quiz. First, the json module is imported to enable the loading and manipulation of JSON files containing the quiz questions, using the open and json.load commands.

* The load\_quiz function loads data from a JSON file and handles errors when the file is not found.
* The quiz is executed using the run\_quiz function. while and for loops are used to prompt the user to select the number of questions and display them one by one.
* try-except blocks handle input errors.

With input, the player can choose the difficulty, provide answers, and replay the quiz if desired.  
if-elif-else conditions are used to verify input validity and make decisions, such as loading the appropriate JSON file based on the selected difficulty.  
Finally, the code uses basic functions like print to display text on the screen. The main program structure is defined by the if \_\_name\_\_ == "\_\_main\_\_": block, which executes the main function when the script is run.

**Main Code descriptions :**

**Implementation of JSON Files:**

First, we need to understand what a JSON file is: it stands for JavaScript Object Notation and is useful for storing information and data in a file. With Python, we can implement the code and use it (in this case, to create a quiz).

We need to use JSON syntax to code a list with three keys: "answer", "option", and "question". Using curly braces, we define an object, while square brackets define an array. The JSON list is then created and must be named according to the difficulty level it represents. (Acsany, 2024)

**Code Execution:**

First, the quiz must be loaded from the JSON file using the load\_quiz function:

This function links the JSON file to the main.py script, allowing access to the three keys defined in the JSON file. (Acsany, 2024)

**Lines 4 to 6**: These lines read and load the content of a specified JSON file using the file\_path. The open(file\_path, 'r') command opens the file in read mode ('r'), and the with keyword ensures that the file is automatically closed after being read, even if an error occurs. Once the file is opened, the json.load(file) function is used to read and convert its JSON content into a Python structure. This structure is then returned using the return statement, allowing the rest of the program to access the file's data for use, such as posing questions in the quiz. These lines form the core of JSON data loading within the program. (Acsany, 2024)

**Player Inputs:**

This block of code starts by prompting the user to input their answer using the command:  
user\_choice = int(input("Enter your answer (1, 2, etc.): "))  
The input is converted to an integer to serve as an index for the options list. The chosen option is retrieved using:  
user\_answer = question["options"][user\_choice - 1]  
The user's response is then stored in the question dictionary under the key 'user\_answer':  
question['user\_answer'] = user\_answer

Next, the code compares the user's response to the correct answer stored under the 'answer' key. If they match, the program displays the message "Correct!" and increments the score:  
score += 1  
If the response is incorrect, the program displays "Wrong!" along with the correct answer. If an error occurs (e.g., invalid input or an out-of-range number), the except (ValueError, IndexError) block catches the error, displays the message "Invalid input. Skipping this question.", and logs "No Answer" as the user's response for that question. This mechanism ensures the program's robustness against input errors while allowing the quiz to continue. (Tiwari, 2024)

**Adding Replay Functionality:**

The replay functionality is implemented in lines 87 to 90. This block manages the option for the user to replay the quiz after completing a session.  
replay\_choice = input("\nDo you want to play again? (yes/no): ").strip().lower()  
The strip() method removes any leading or trailing spaces, while lower() converts the input to lowercase to make the comparison case-insensitive (e.g., "Yes" or "YES" will be interpreted as "yes").

Then, the condition:  
if replay\_choice not in ['yes', 'y']:  
checks if the input is not "yes" or its abbreviated form "y". If true, the program displays the message "Thank you for playing! Goodbye!" and breaks out of the main loop using break, ending the program. This block ensures smooth interaction with the user, allowing them to either replay or exit the quiz. (Tiwari, 2024)

**Difficulty Selection:**

This block manages the user's choice of difficulty level. Upon starting the quiz, the program prompts the user to select a difficulty level from the available options. The input is processed using:  
difficulty\_choice = input("Select a difficulty (easy, medium, hard): ").strip().lower()  
The strip() and lower() methods ensure robust handling of user input by ignoring extra spaces and making the comparison case-insensitive. Based on the input, the program loads the corresponding JSON file with:  
if difficulty\_choice == "easy":  
file\_path = "easy.json"  
elif difficulty\_choice == "medium":  
file\_path = "medium.json"  
elif difficulty\_choice == "hard":  
file\_path = "hard.json"  
else:  
print("Invalid difficulty selected. Defaulting to 'easy'.")  
file\_path = "easy.json"

This mechanism ensures that the appropriate quiz questions are loaded based on the selected difficulty, improving the user experience by tailoring the quiz to their preferred level.

**Adding Parameters to the Code:**

The replay functionality is implemented in lines 87 to 90. This block manages the option for the user to replay the quiz after completing a session.  
The line:  
replay\_choice = input("\nDo you want to play again? (yes/no): ").strip().lower()  
prompts the user to indicate whether they want to replay by entering “yes” or “no.” The .strip() method removes leading and trailing spaces, while .lower() converts the input to lowercase, ensuring the comparison is case-insensitive (e.g., “Yes” or “YES” will be interpreted as “yes”).

The condition:  
if replay\_choice not in ['yes', 'y']  
checks whether the response is not “yes” or its abbreviated form “y.” If this is the case, the program displays the message:  
"Thank you for playing! Goodbye!"  
and breaks the main loop using break, effectively ending the program. This block ensures smooth interaction with the user, allowing them to replay or exit the quiz according to their choice.

**Difficulty Selection:**

This block manages the user’s choice of difficulty level. At the start of the quiz, the program prompts the user to select a difficulty level. The line:  
difficulty\_choice = input("Select a difficulty (easy, medium, hard): ").strip().lower()  
ensures that the input is processed robustly by ignoring extra spaces and converting it to lowercase.

Depending on the input, the program loads the appropriate JSON file:

* if difficulty\_choice == "easy":  
  file\_path = "easy.json"
* elif difficulty\_choice == "medium":  
  file\_path = "medium.json"
* elif difficulty\_choice == "hard":  
  file\_path = "hard.json"
* else:  
  print("Invalid difficulty selected. Defaulting to 'easy'.")  
  file\_path = "easy.json"

This ensures the correct questions are loaded based on the user’s selected difficulty, improving the overall experience.

**Conclusion:**

This code allows the creation of a quiz based on a JSON list. While simple, the quiz provides multiple functionalities, with the most notable ones detailed in this code. These features include:

Quiz execution ;Error handling ; Difficulty selection ; JSON file implementation ;Summary of results.

This ensures the program is both versatile and user-friendly. Moreover b adding more json quiz for different courses the code is easily adapted and ready to work for the new lists

Table of aids :

|  |  |  |
| --- | --- | --- |
| ChaatGPT | * Aid to debug and adapt code * Paraphrase and translate to French to English * Formulate clear explanation | * In the code section * In all parts of this documents * Over the code |
| Replit.com | * For coding | * In the code section |

Bibliography :

Acsany, P. (2024, July 3). *Working with JSON data in Python*. Real Python. <https://realpython.com/python-json/>

Tiwari, Y. (2024). *Create a simple quiz game program in Python with score*. AllinPython. <https://allinpython.com/create-a-simple-python-quiz-game-with-score/>