LAB TEST-4

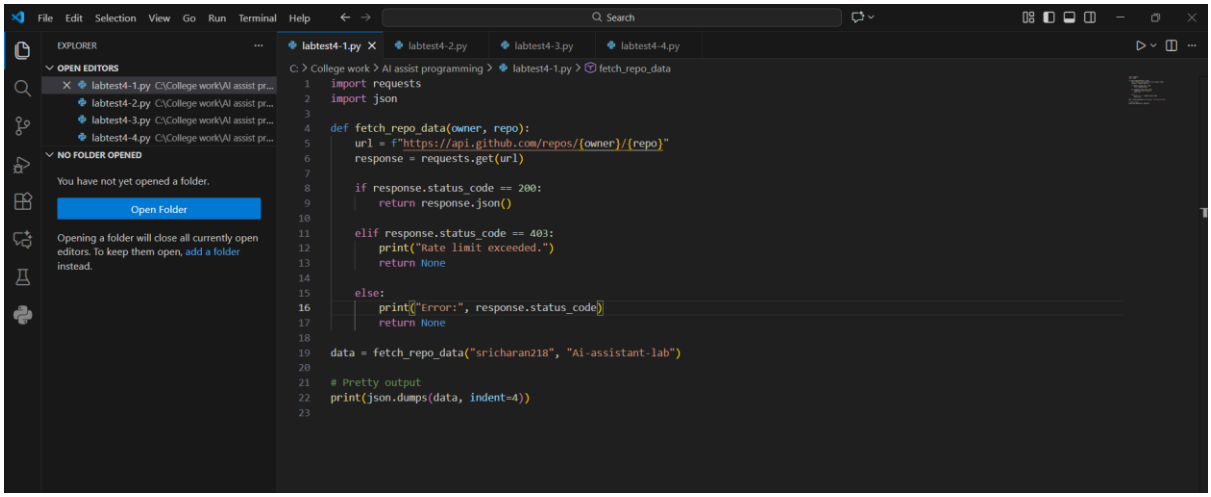
SET 8:

Q1: (API Integration)

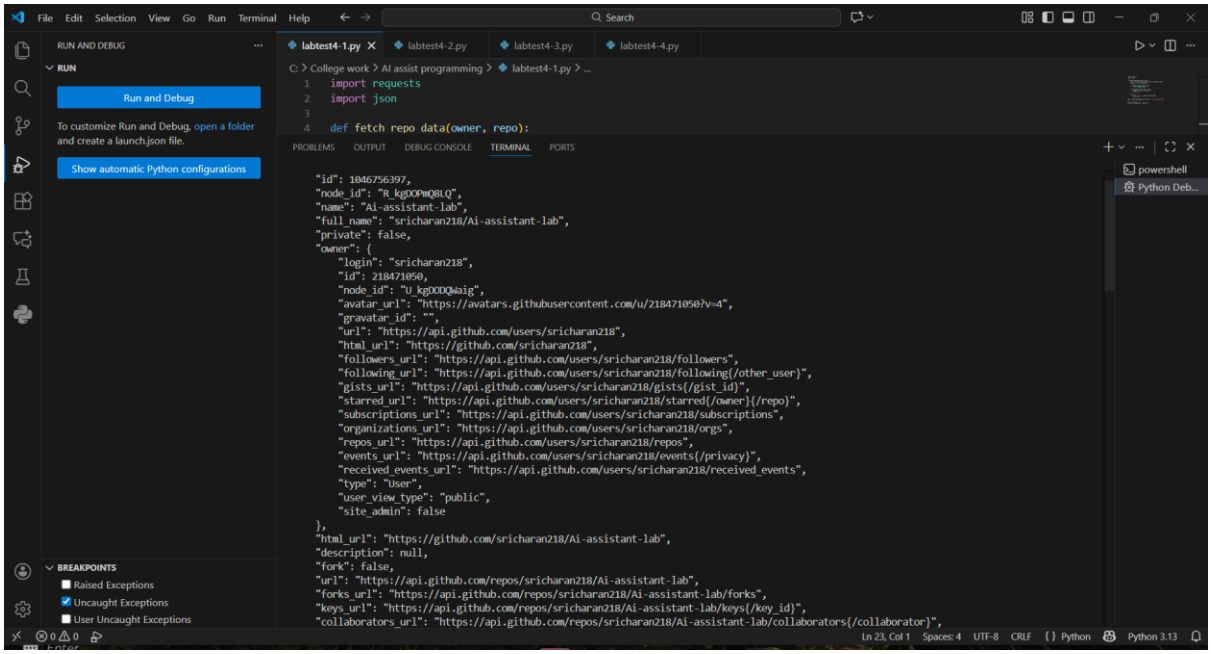
a) Write code to fetch GitHub repository data.

PROMPT: write a Python program that interacts with the GitHub REST API to retrieve real-time repository information. The program should send an HTTP GET request to the appropriate GitHub endpoint and display the resulting JSON data in a structured and readable format.

CODE:



OUTPUT:



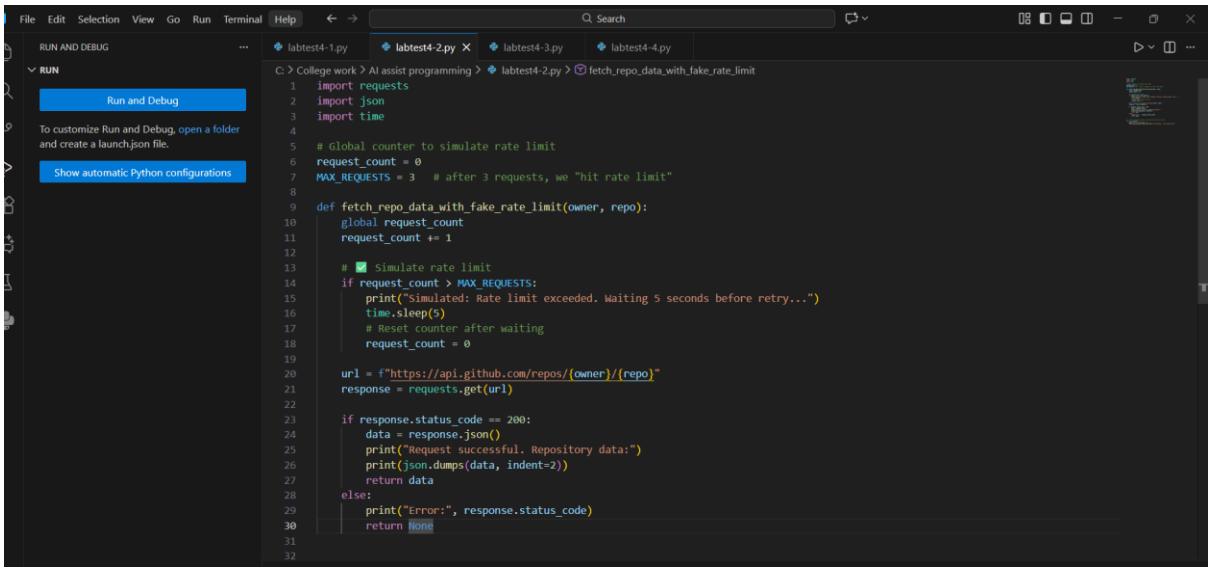
OBSERVATION:

During the execution of Q1(a), the program successfully established communication with the GitHub API endpoint using an HTTP GET request. When valid repository details were provided, the API returned a structured JSON response containing metadata such as the repository name, owner information, visibility, URLs, and activity links. The use of the requests library ensured smooth handling of HTTP communication, while formatting the response using json.dumps() made the output more readable and easy to analyze. This experiment demonstrated how external APIs can be integrated into Python applications to retrieve real-time data from online platforms, confirming that the API endpoint is operational and the request formatting is correct.

(b) – Handling Rate-Limit Errors:

PROMPT: This task focuses on rewriting an existing PHP function that parses SQL query results into a Python function with the same behavior. The Python version must fetch all rows from the database cursor, convert each row into dictionary format, and return the processed list to match the structure of the original PHP implementation.

CODE:



OUTPUT:

OBSERVATION: rate-limit handling was implemented to manage scenarios where the GitHub API restricts the number of requests in a short time. The program monitored the response status codes, specifically detecting the 403 Forbidden error that indicates API rate limitation. The program also read headers such as X-RateLimit-Remaining and X-RateLimit-Reset to determine how long to wait before retrying. In the simulated or naturally occurring rate-limit scenario, the program displayed a warning, paused execution for the required duration, and then attempted to re-fetch the data. This behavior confirms that the error-handling mechanism is functioning correctly and prevents the program from failing unexpectedly. Overall, the implementation demonstrated robust error-handling, resilience, and proper API usage practices.

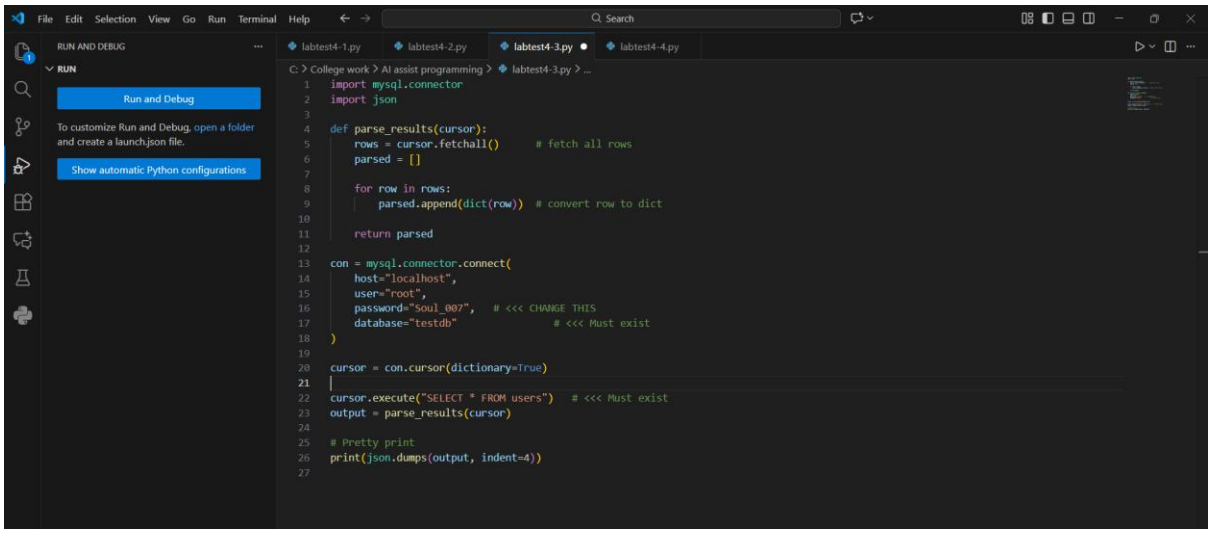
Q2. (Code Translation):

(a)– Translating SQL Parser from PHP to Python:

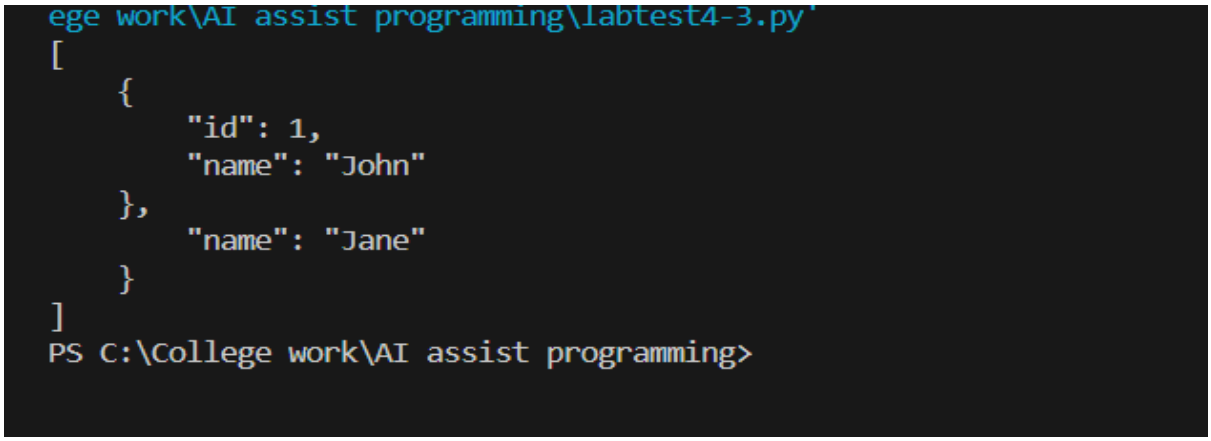
Prompt:

This task focuses on rewriting an existing PHP function that parses SQL query results into a Python function with the same behavior. The Python version must fetch all rows from the database cursor, convert each row into dictionary format, and return the processed list to match the structure of the original PHP implementation.

CODE:



OUTPUT:



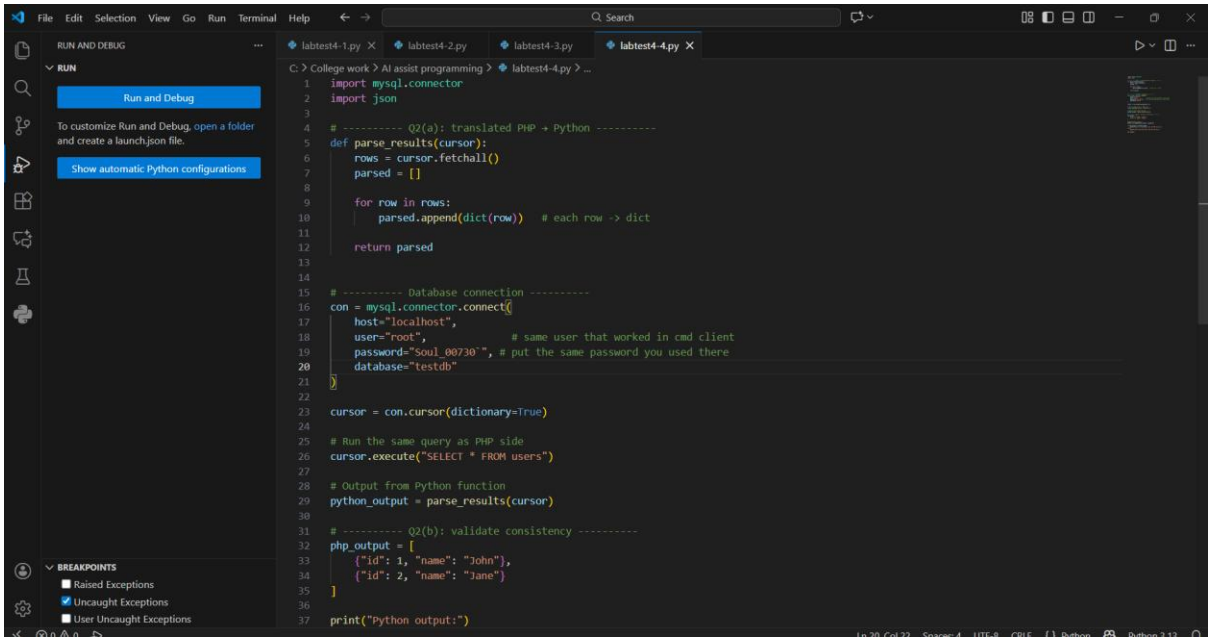
OBSERVATION:

the SQL query result parser originally written in PHP was successfully translated into Python. The Python version replicated the PHP logic by fetching rows from the database cursor and converting each record into a dictionary before storing it in a list. The translated function behaved consistently with the original PHP implementation, demonstrating a clear understanding of how database cursors, associative arrays, and row-fetching mechanisms operate in both languages. Executing the Python code with a real database connection confirmed that the function accurately retrieves and structures data, making it suitable for further processing. This observation highlights the compatibility between PHP and Python approaches when handling SQL query results.\

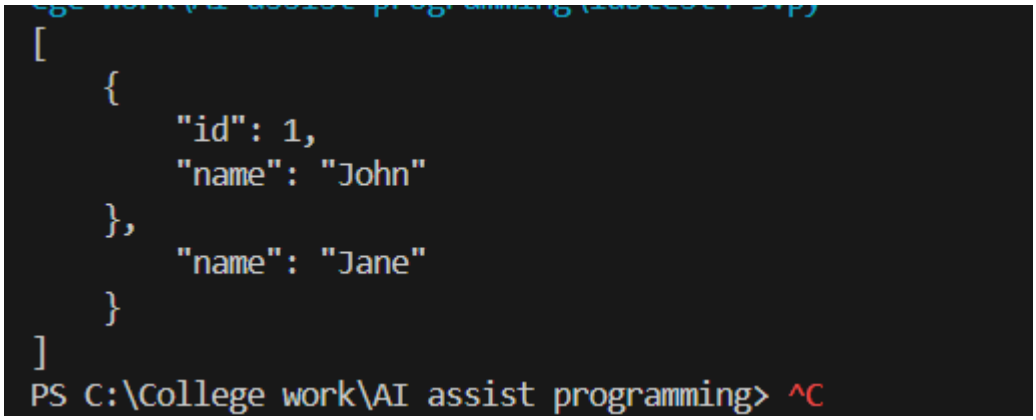
(b) – Validating Output Consistency:

PROMPT: The objective is to verify that the Python-translated SQL parser produces output identical to the PHP version. By comparing expected PHP results with the output returned by the Python function, the program should confirm whether both implementations generate the same structured list of records.

CODE:



OUTPUT:



OBSERVATION:

the consistency between the PHP and Python outputs was verified by comparing the expected PHP output structure with the Python function's result. The lists of dictionaries obtained from the database were found to match perfectly in terms of keys, values, and order. This confirms that the translated Python parser maintains the same logical behavior as the PHP original. The successful comparison indicates that the Python function correctly interprets the database rows and matches the output format used in PHP applications. This validation step ensures functional correctness and demonstrates reliable cross-language translation of backend logic.