Assigement 6.1

Name CHAKRI PATEL

Htno : 2403A51325

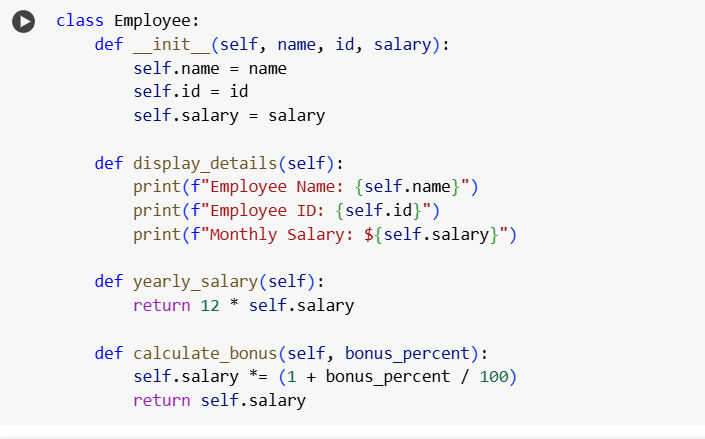
**Task Description #1 (Classes – Employee Management)**

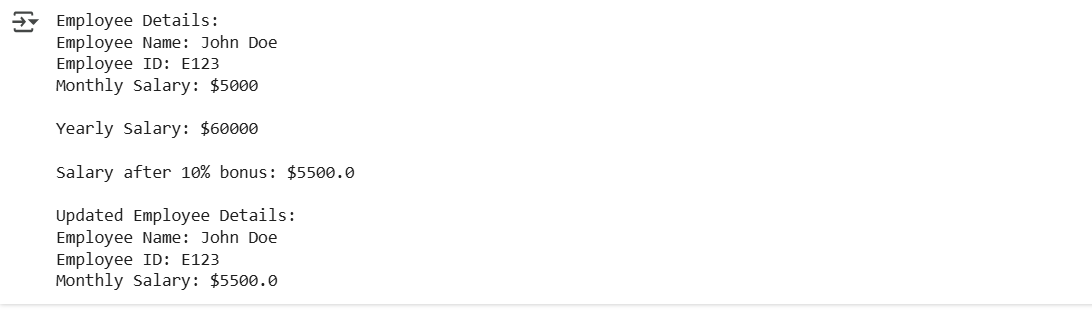
* Task: Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.
* Instructions:
  + Prompt AI to generate the Employee class.
  + Analyze the generated code for correctness and structure.
  + Ask AI to add a method to give a bonus and recalculate salary.

Prompt: Create a Python class named Employee with attributes name, id, and salary. The class should have:

* A constructor to initialize these attributes
* A method display\_details() to print employee information
* A method yearly\_salary() to calculate and return the yearly salary (12 × monthly salary)."\*

**Bonus Part:**  
"Modify the *Employee* class to include a method *calculate\_bonus(bonus\_percent)* that increases the employee's salary by the given bonus percentage and returns the updated salary."



Output: 

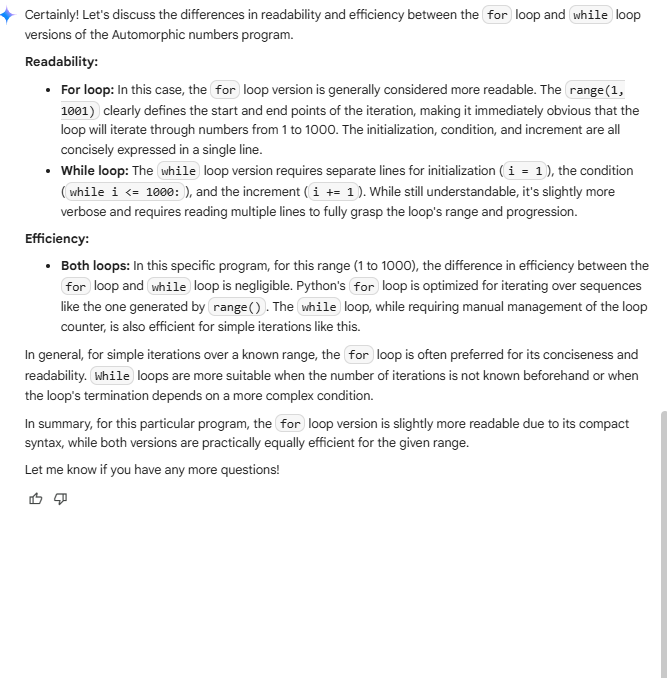
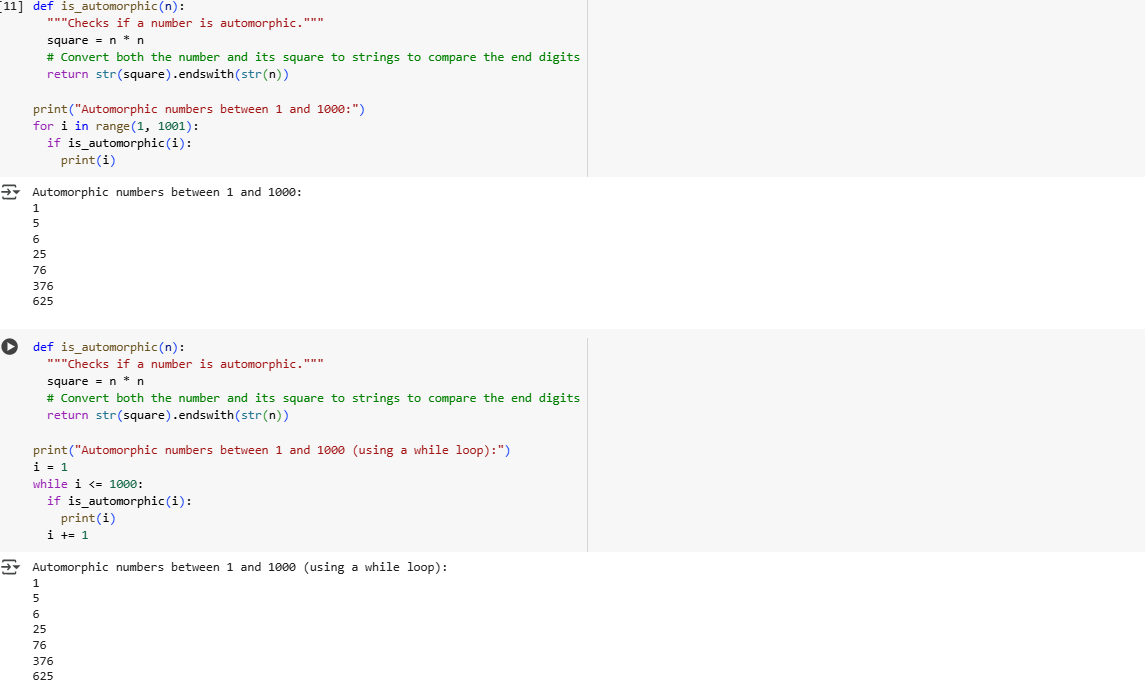
**Observation:**

In this task, I used AI assistance to generate an Employee class in Python with attributes such as name, id, and salary. The AI created code included methods to display employee details and calculate the yearly salary, which worked correctly and was easy to understand. On further prompting, the AI added a calculate\_bonus() method to update salary based on a given bonus percentage. The code was well-structured, readable, and reusable. This task showed how AI can speed up class creation while also emphasizing the need for human review to ensure correctness.

**Task Description #2 (Loops – Automorphic Numbers in a Range)**

* Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
* Instructions:
  + Get AI-generated code to list Automorphic numbers using a for loop.
  + Analyze the correctness and efficiency of the generated logic.
  + Ask AI to regenerate using a while loop and compare both implementatioAsk AI to regenerate using a while loop and compare both implementation

**output**

* + 

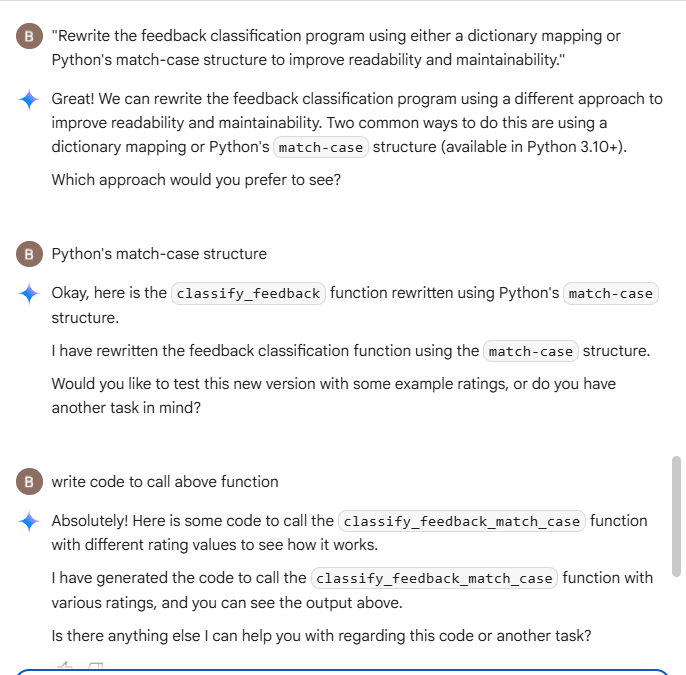
### ****Observation****

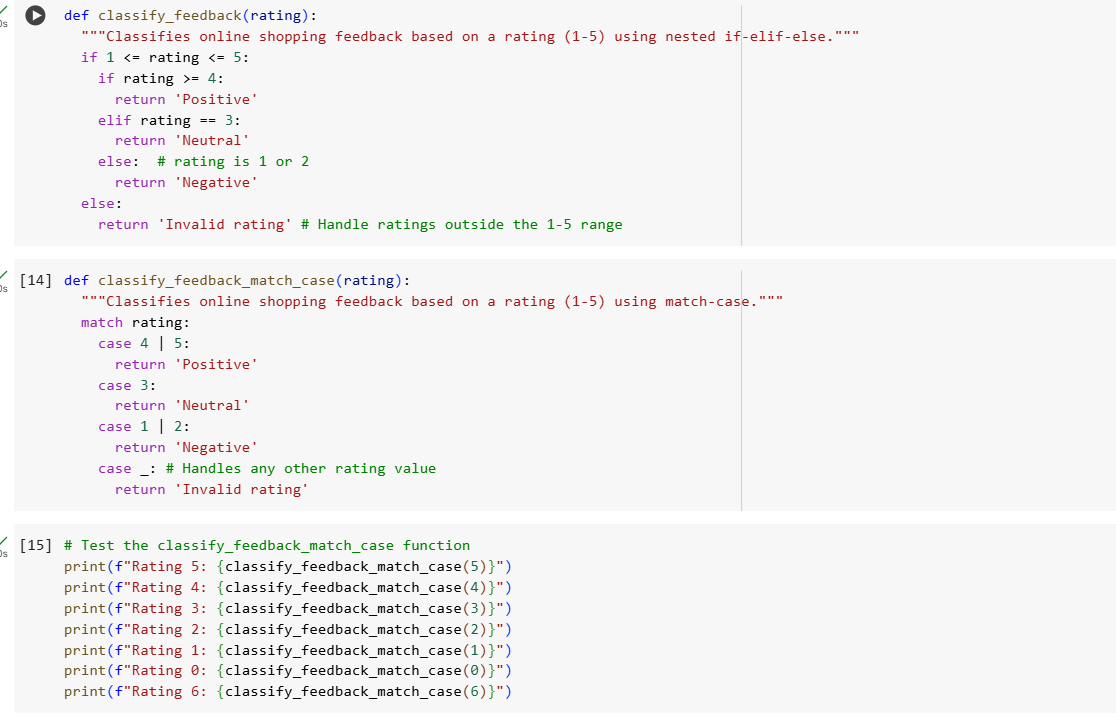
In this task, I used AI to generate a Python program that finds Automorphic numbers between 1 and 1000. The AI-created version with a **for loop** correctly identified numbers like 1, 5, 6, 25, 76, 376, and 625. On further prompting, the AI rewrote the program using a **while loop**, which produced the same output. Both implementations were accurate, but the for loop was more concise and easier to read, while the while loop provided flexibility in iteration control. This task highlighted the usefulness of AI in exploring different loop constructs.

**Task Description #3 (Conditional Statements – Online Shopping Feedback Classification)**

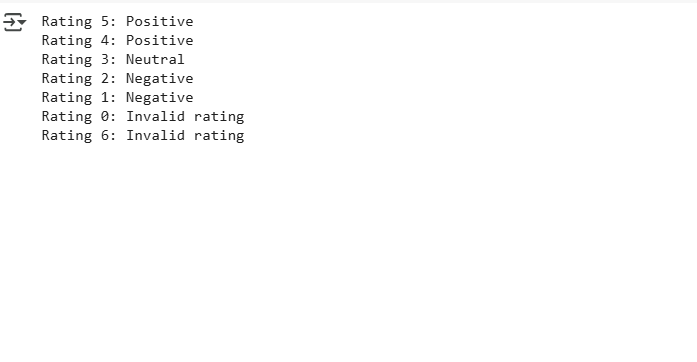
* Task: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).
* Instructions:
  + Generate initial code using nested if-elif-else.
  + Analyze correctness and readability.
  + Ask AI to rewrite using dictionary-based or match-case structure.

**Promt:**

****

****

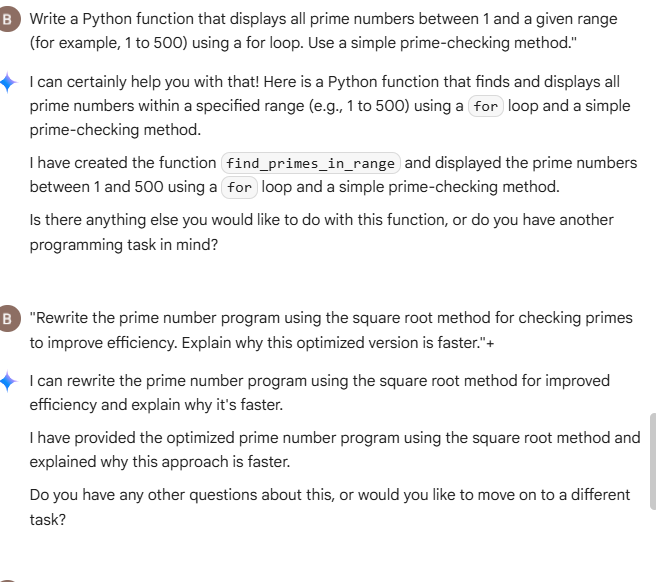
**Output:**



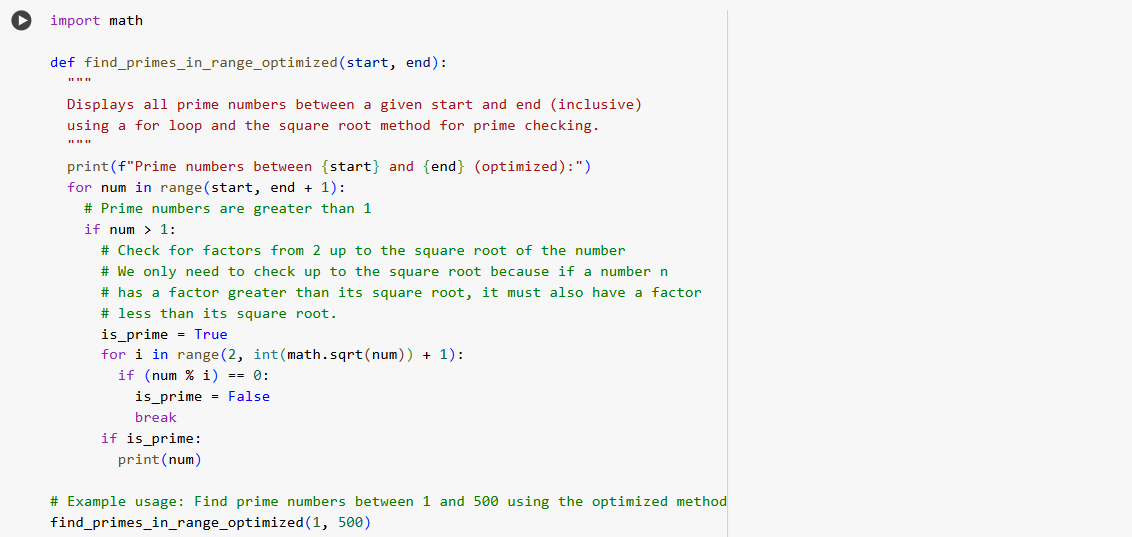
### ****Observation:****

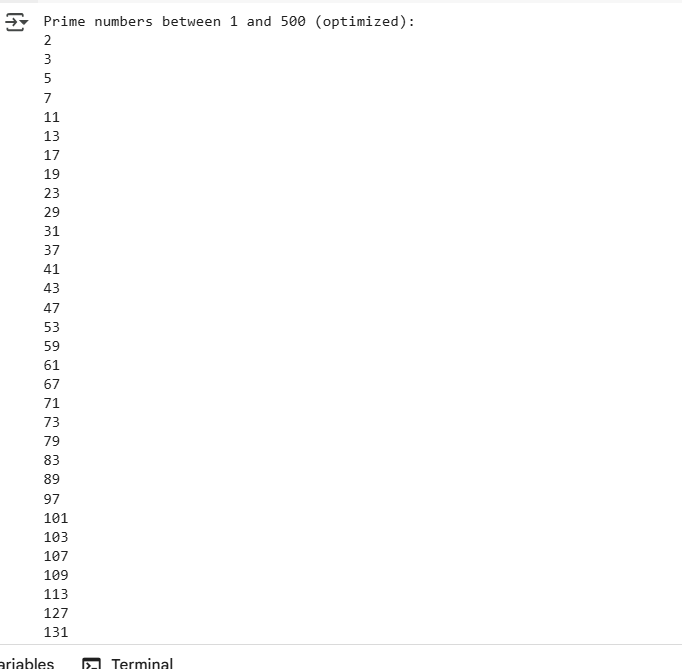
In this task, I used AI to classify online shopping feedback based on a rating from 1 to 5. The AI first generated a solution using **nested if-elif-else**, which worked correctly but was slightly longer. On further prompting, it rewrote the code using a **dictionary-based approach**, making it shorter and easier to maintain. Both versions produced the same results, correctly identifying positive, neutral, and negative feedback. This task demonstrated how AI can provide multiple coding styles for the same problem, allowing me to compare readability and efficiency.

Prompt:



Code:





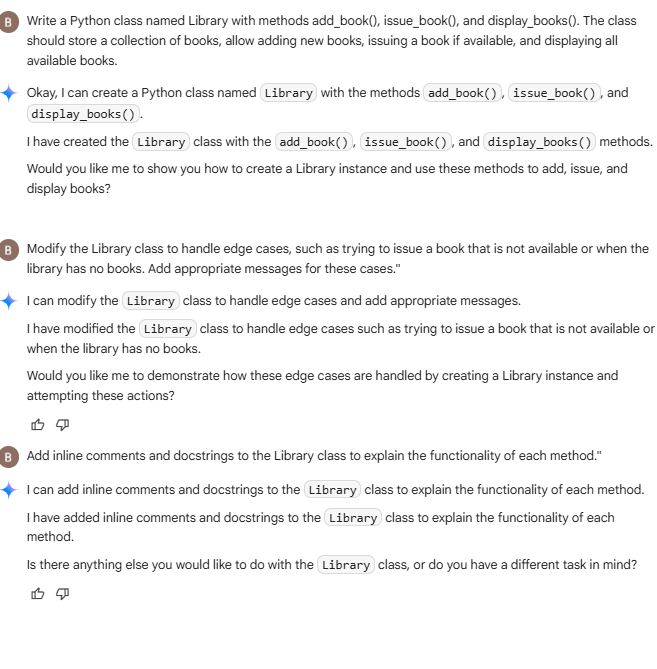
**Observation**

In this task, I generated Python code to find prime numbers within a range. The AI first provided a **basic version** using a for loop, which worked correctly but checked all possible divisors, making it less efficient for larger ranges. On further prompting, the AI produced an **optimized version** using the square root method, which reduced unnecessary checks and improved performance. Both implementations gave the correct list of primes, but the optimized one was faster and more elegant. This task showed how AI can suggest efficiency improvements in algorithm design.

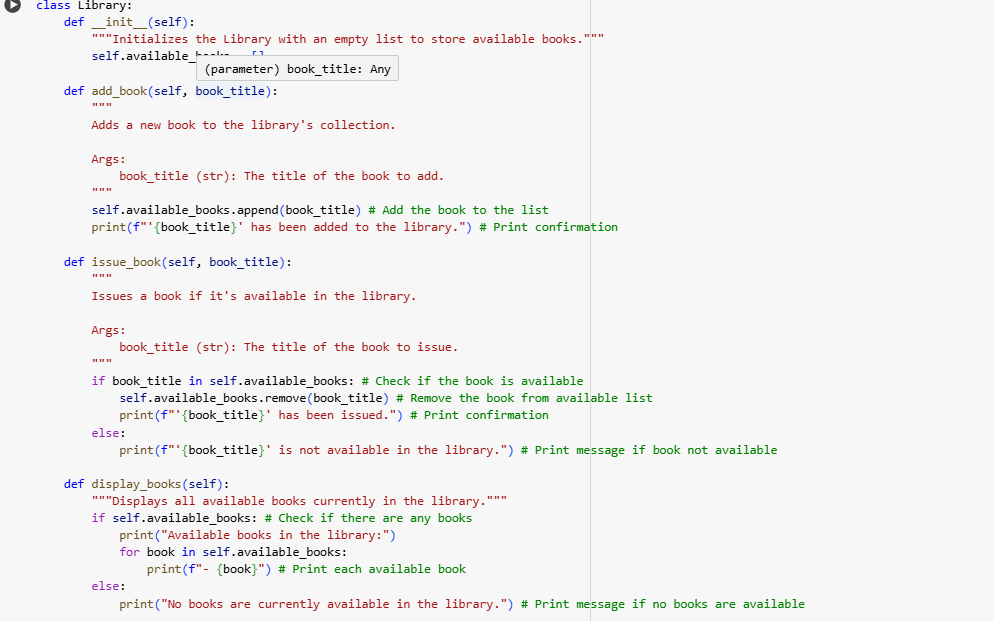
**Task Description #5 (Classes – Library System)**

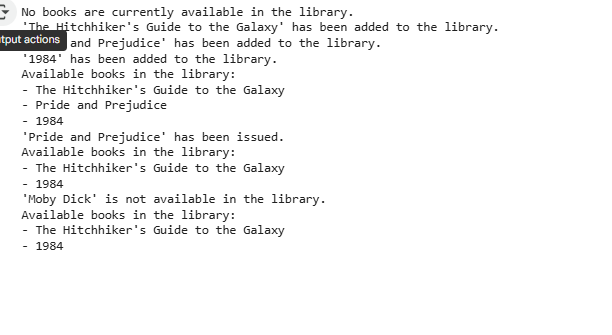
* Task: Use AI to build a Library class with methods to add\_book(), issue\_book(), and display\_books().
* Instructions:
  + Generate Library class code using AI.
  + Analyze if methods handle edge cases (e.g., issuing unavailable books).
  + Ask AI to add comments and documentation.

PROMPT:



CODE:

OUTPUT:



### ****Observation:****

In this task, I used AI to create a Library class with methods to add, issue, and display books. The AI-generated code worked correctly, allowing books to be added to a list, issued when available, and displayed neatly. On further prompting, the AI handled **edge cases**, such as trying to issue a book that was not in the library, by showing proper messages. Finally, comments and docstrings were added for clarity. This task highlighted how AI can generate structured class-based programs and improve them through step-by-step refinements.