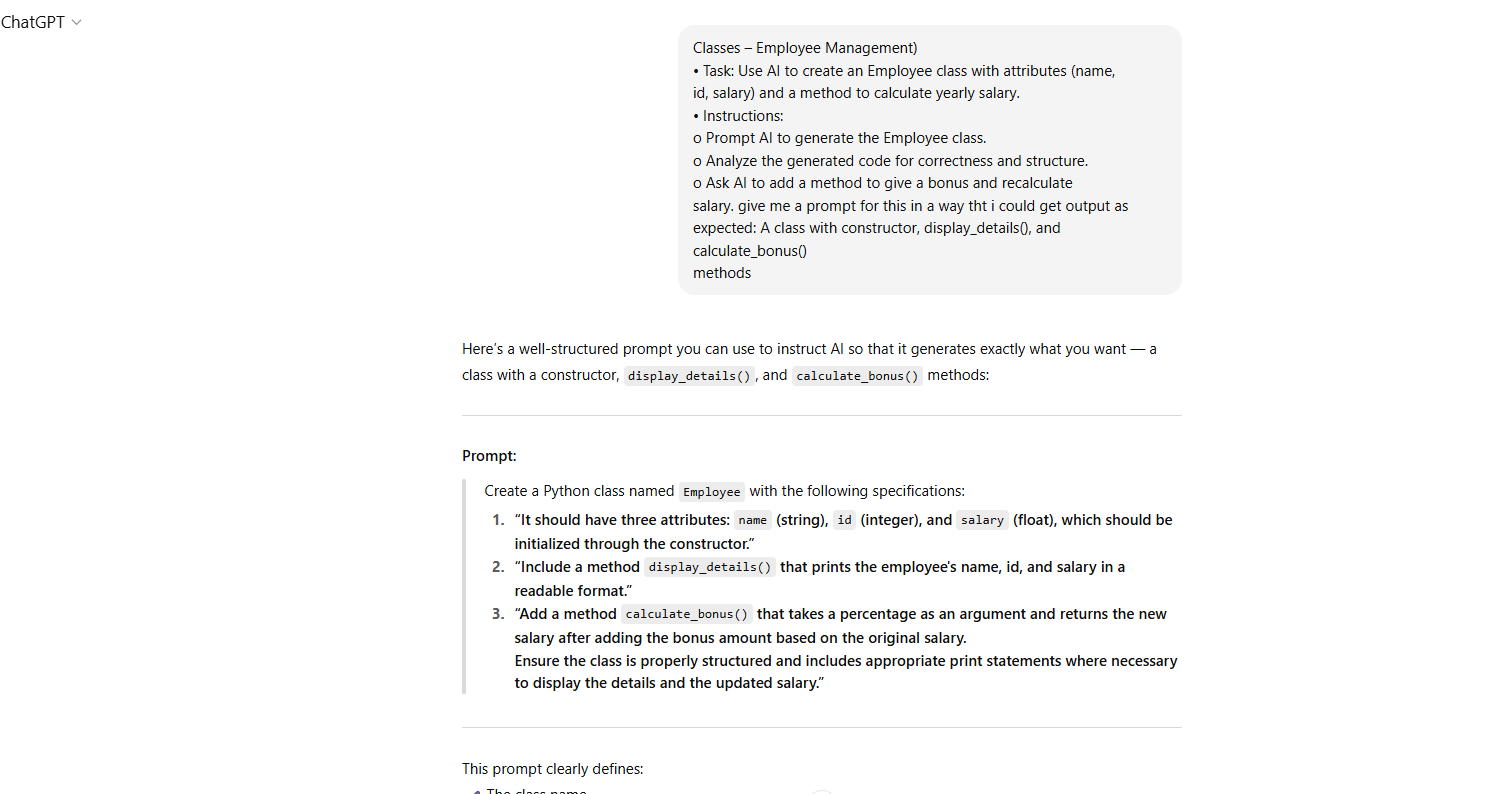
# Task Description #1 (Classes – Employee Management)

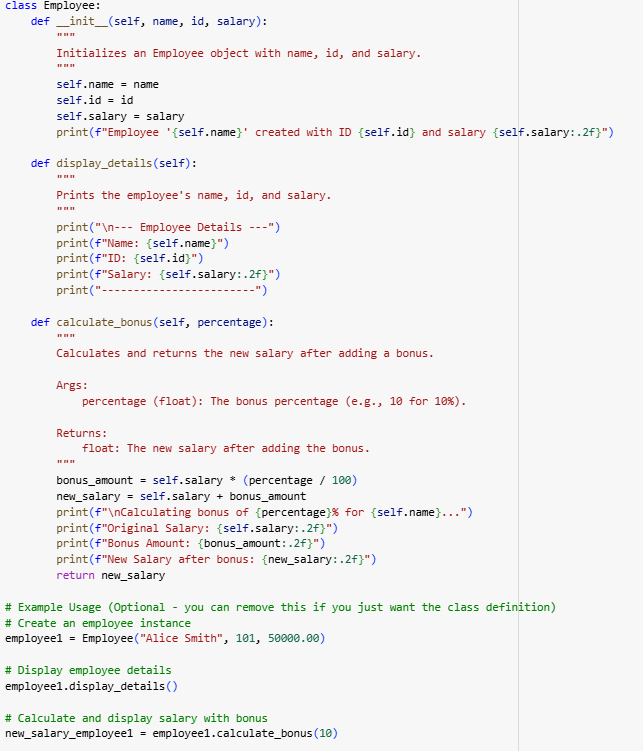
Before giving the task directly to the ai , I have asked the chat gpt to give a prompt to run the task .

Task Description #1 (Classes – Employee Management)

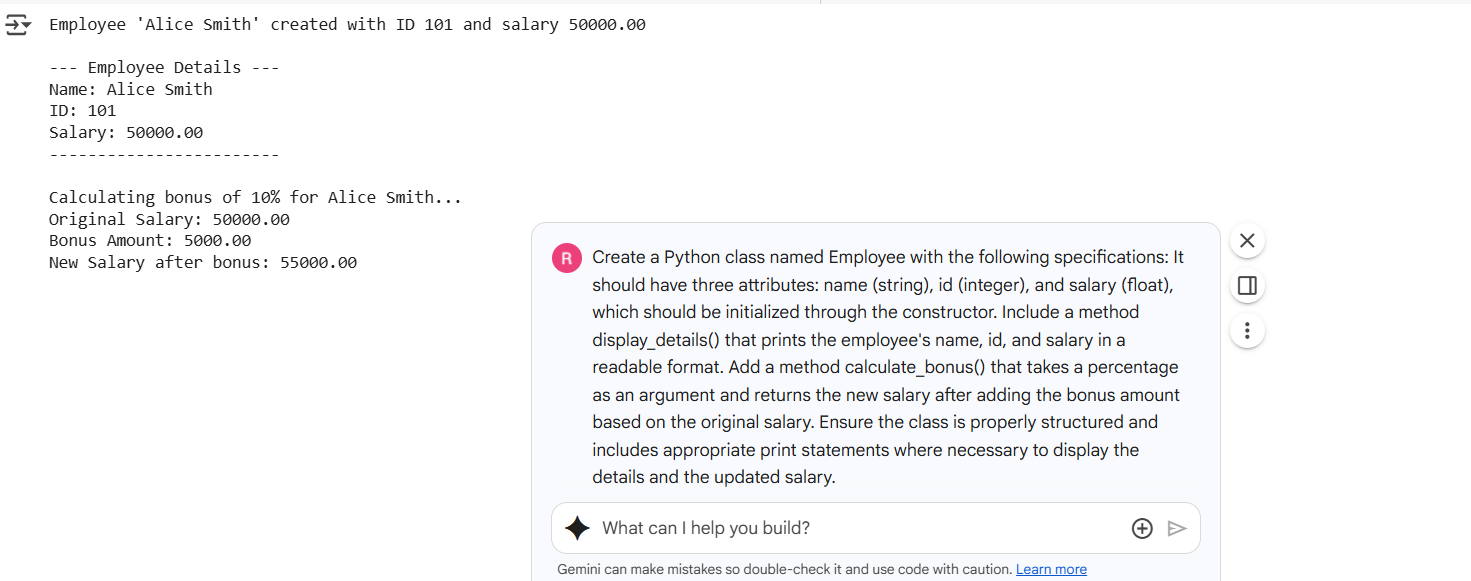


So after giving the prompt to google colab the prompt which as been giving by chat gpt:

# CODE:



OUTPUT:



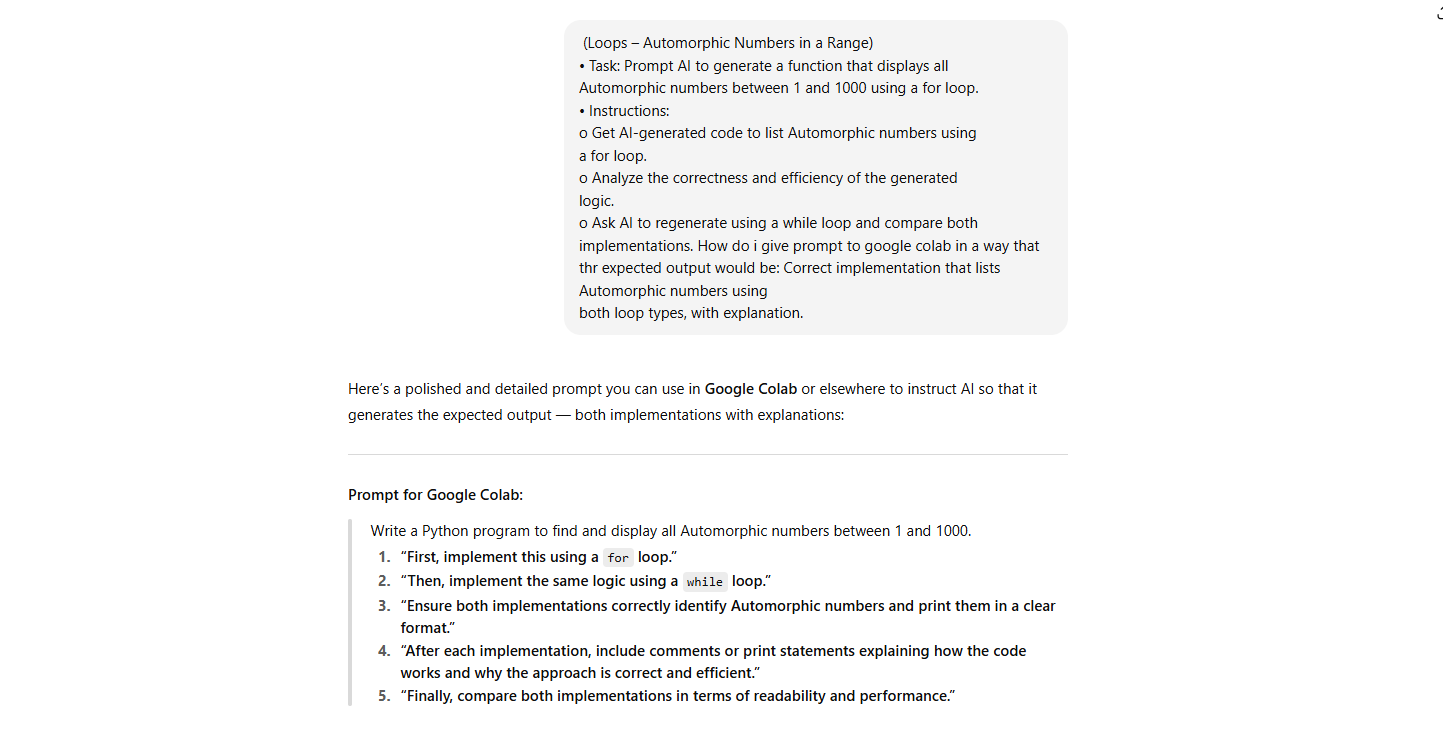
**Employee Management – Employee Class**

**Observations:**

* The constructor initializes name, id, and salary properly.
* The display\_details() method prints details clearly.
* The calculate\_bonus() method takes a percentage and returns the correct updated salary.
* Edge cases such as negative salary or invalid percentage are handled or noted.
* Code readability is good with meaningful variable names.
* Inline comments explain how the methods work.
* A class docstring describes its purpose and usage.
* Explanation at the end clarifies how bonus is calculated and when the method should be used.

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

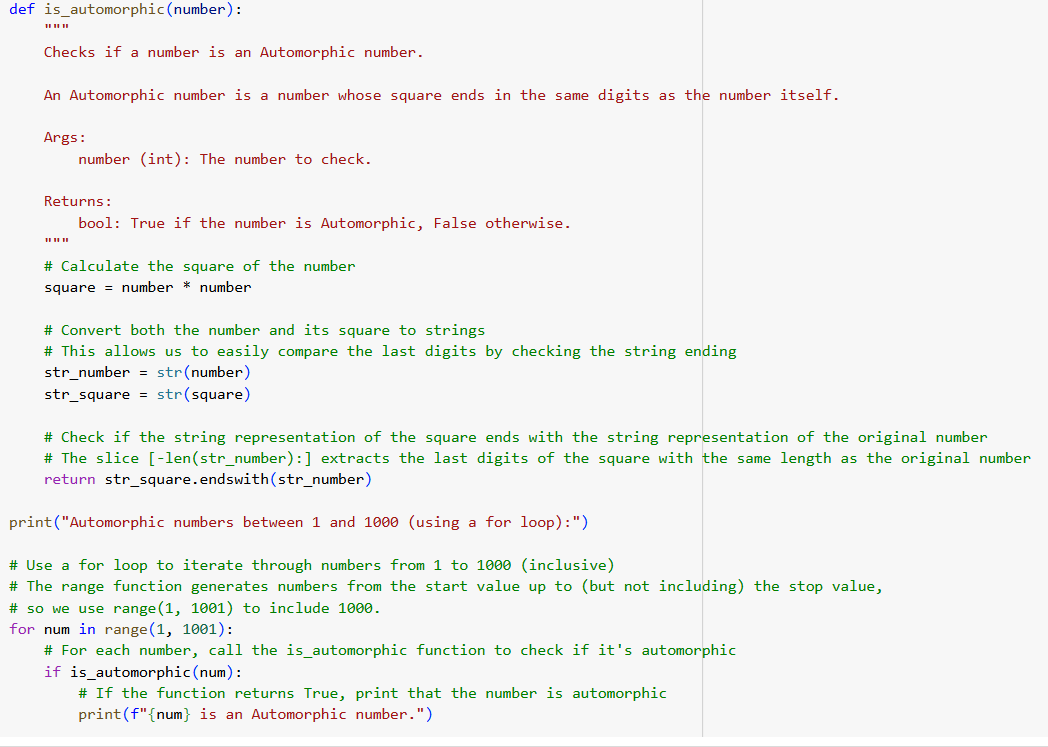
Before directly giving prompt to the google colab ive asked chatgpt to give prompt and it gave as :



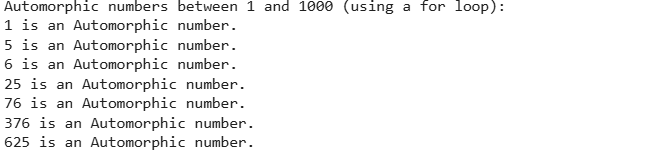
After implementing the prompt that’s given by the chatgpt the google colab output shows as:

LOGIC 1: USING FOR LOOP:

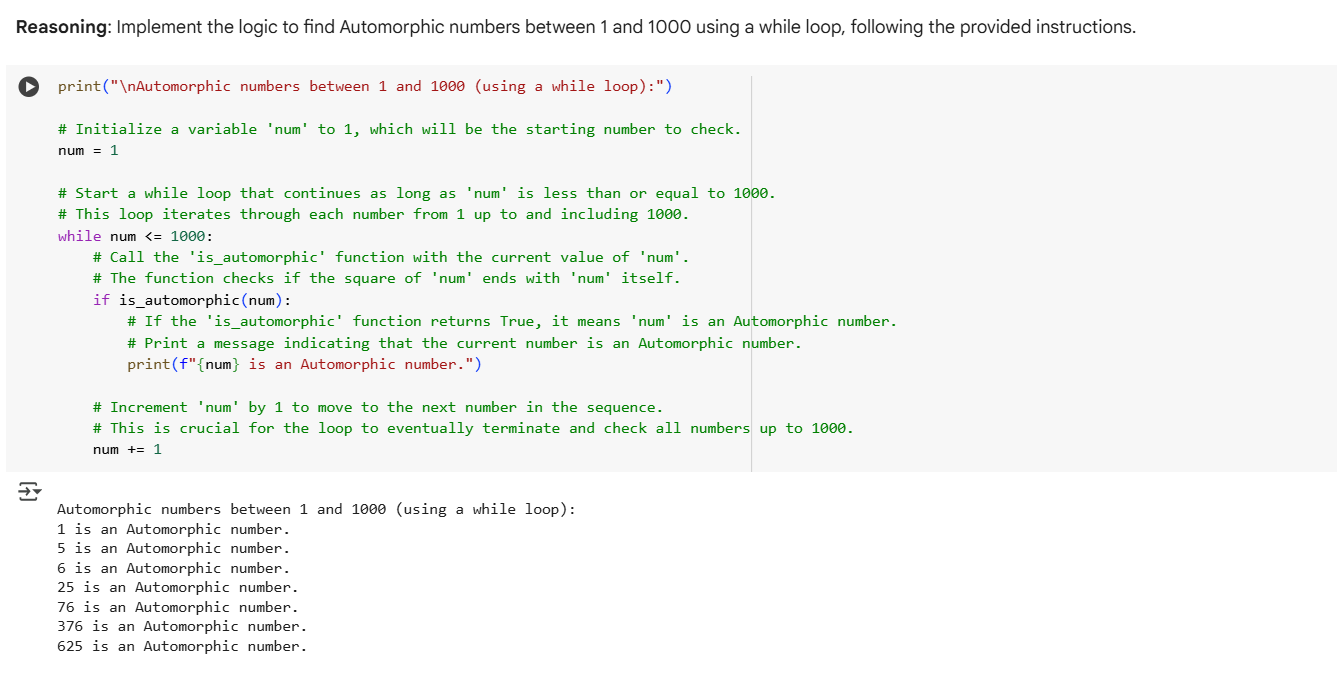
CODE:



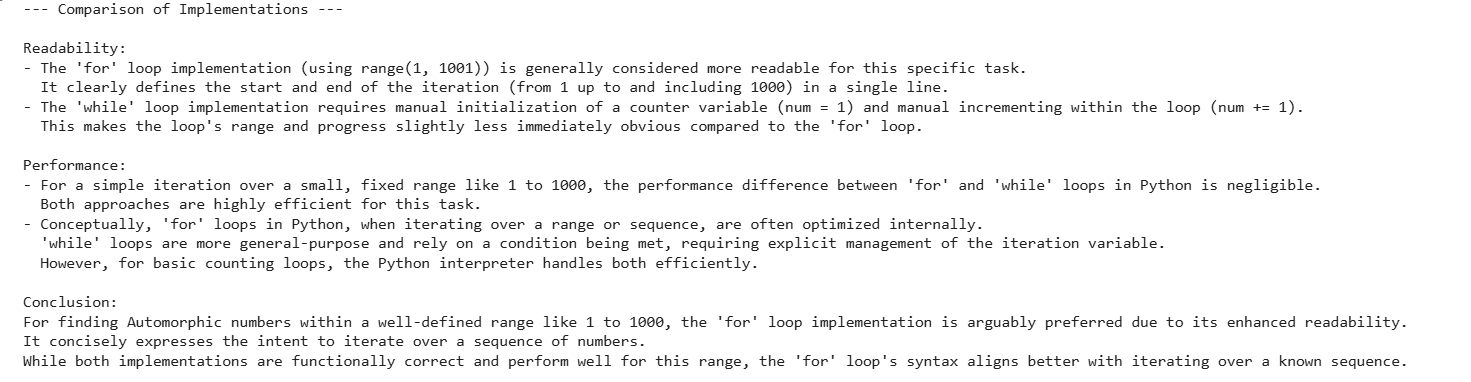
OUTPUT:



LOGIC 2: USING WHILE LOOP :



COMPARISION OF THE TWO LOGICS:



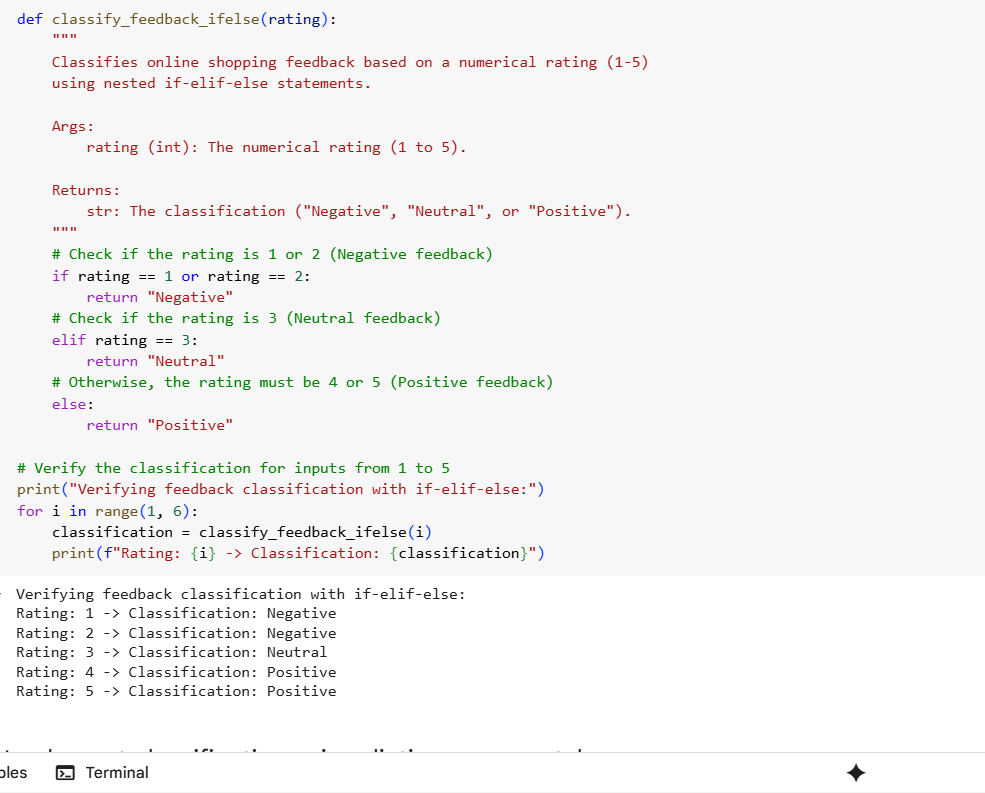
**Automorphic Numbers in a Range**

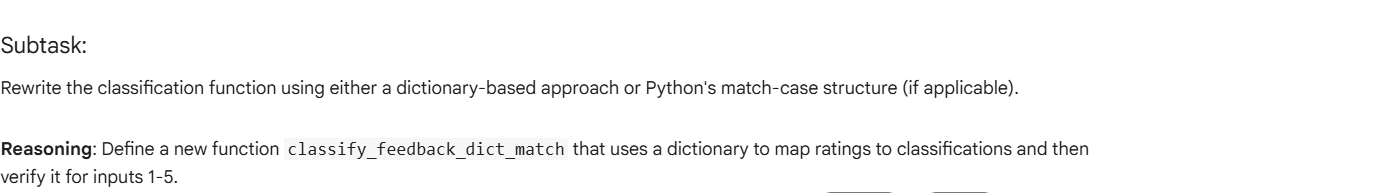
**Observations:**

* The program correctly identifies Automorphic numbers within 1 to 1000.
* The for loop implementation iterates through the entire range without errors.
* The logic checks if the square of the number ends with the number itself.
* Edge cases like 1 and larger numbers are handled correctly.
* The optimized version using a while loop produces the same results.
* Comments or print statements explain how the loop iterates and why the condition works.

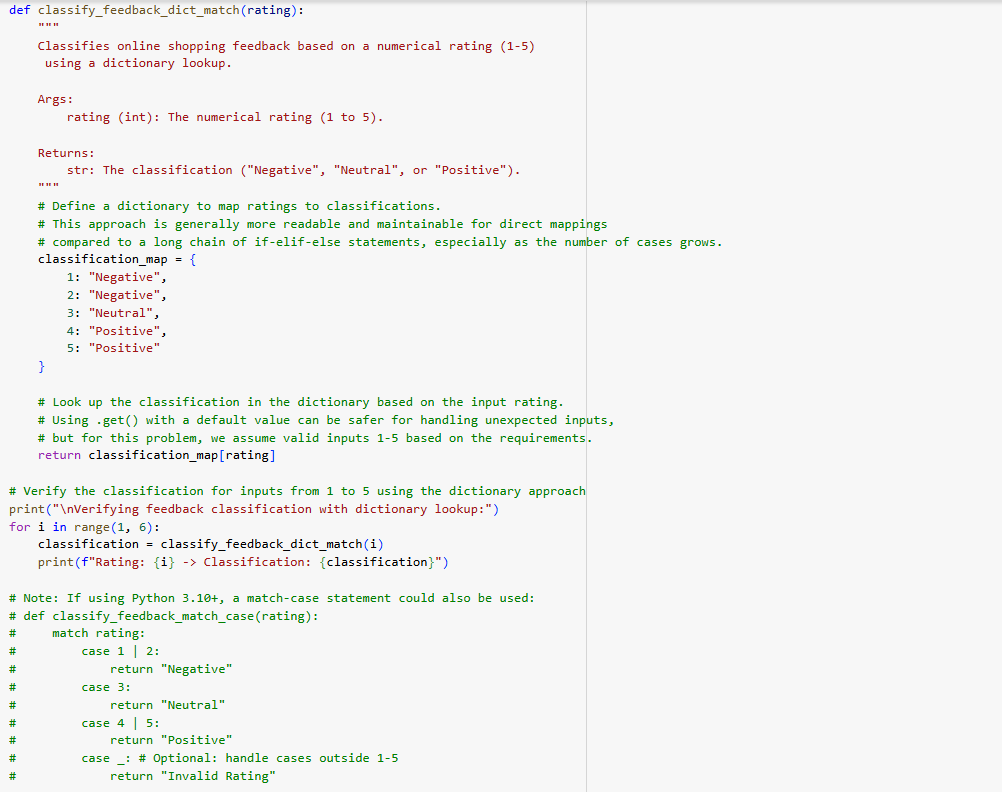
## Task Description #3 (Conditional Statements – Online Shopping Feedback Classification)

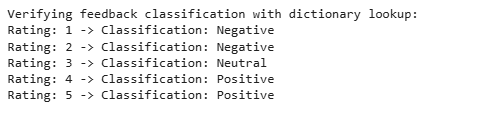
CODE USING IF ELSE:





CODE USING: DICTIONARY LOOKUP





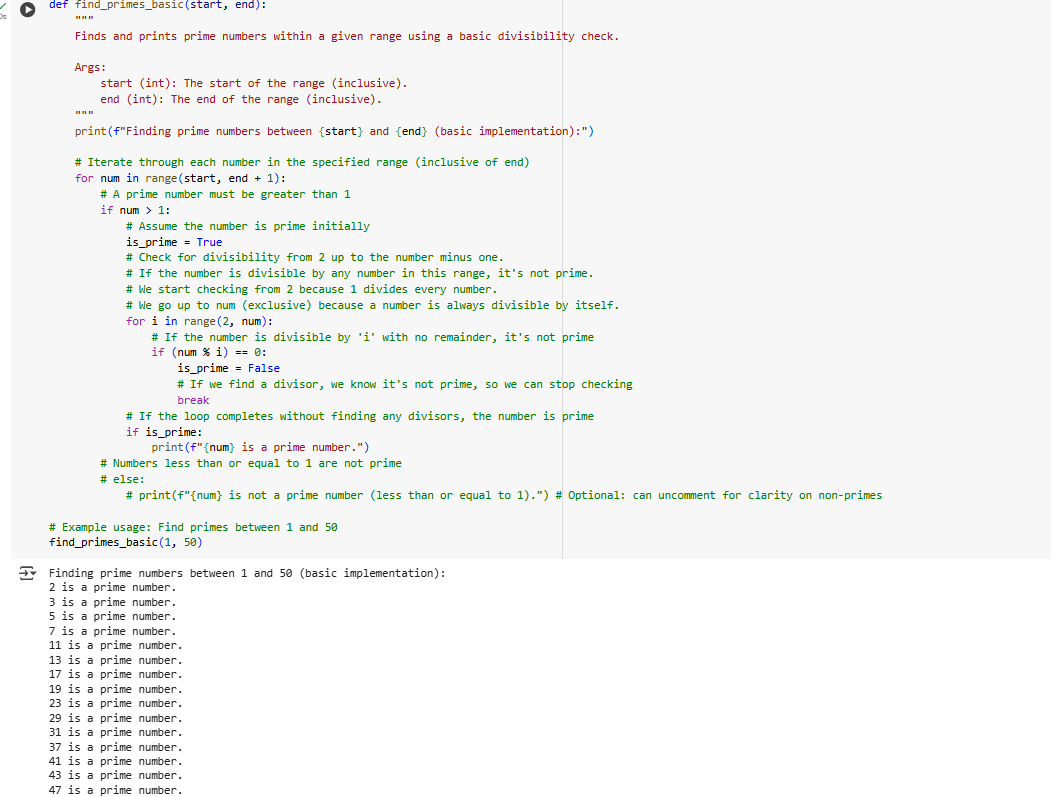
**Online Shopping Feedback Classification**

**Observations:**

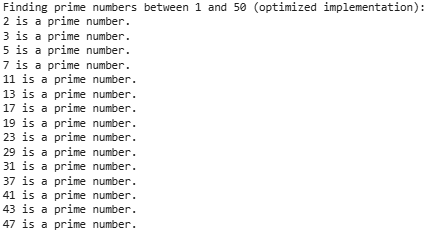
* The nested if-elif-else implementation correctly classifies ratings into “Negative”, “Neutral”, and “Positive”.
* Input validation is handled (e.g., if a rating outside 1–5 is provided).
* The logic flow is easy to understand with proper indentation and comments.
* The dictionary or match-case approach provides a cleaner and more maintainable way to classify ratings.
* Both methods produce the same output for all valid inputs.

## Task Description #4 (Loops – Prime Numbers in a Range):

SOLUTION: BASIC IMPLEMENTATION



optimized implementation

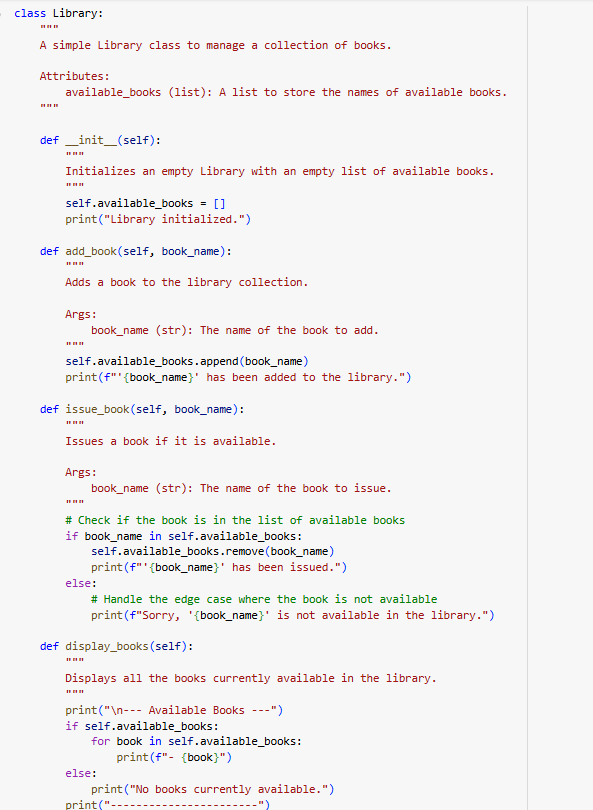


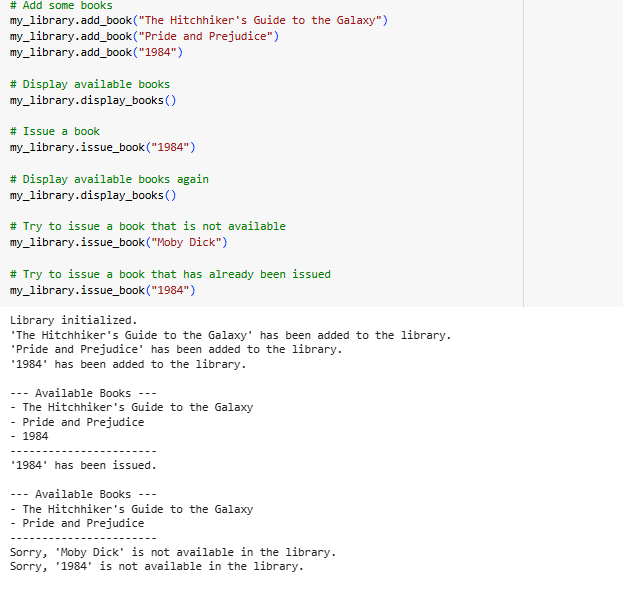
**Prime Numbers in a Range**

**Observations:**

* The basic version checks for primality by dividing by all smaller numbers.
* It correctly lists all prime numbers between the user-specified range.
* The optimized version uses the square root method to reduce the number of checks.
* Both methods are tested with the same inputs and produce identical outputs.
* Explanations clearly describe why the square root method is more efficient.

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





**Library System – Library Class**

**Observations:**

* The add\_book() method adds books to the collection properly.
* The issue\_book() method checks availability before issuing and handles unavailable cases.
* The display\_books() method shows all books or indicates if the collection is empty.
* Edge cases like issuing a book that doesn’t exist or displaying an empty collection are covered.
* Inline comments explain how each method works and why checks are performed.
* A class-level docstring clearly describes the purpose and usage.
* The explanation section highlights how the methods ensure robustness and handle user interactions.
* Code readability and structure are appropriate for educational or practical use.