ASSIGNMENT – 6.1

**ZIAUDDIN**

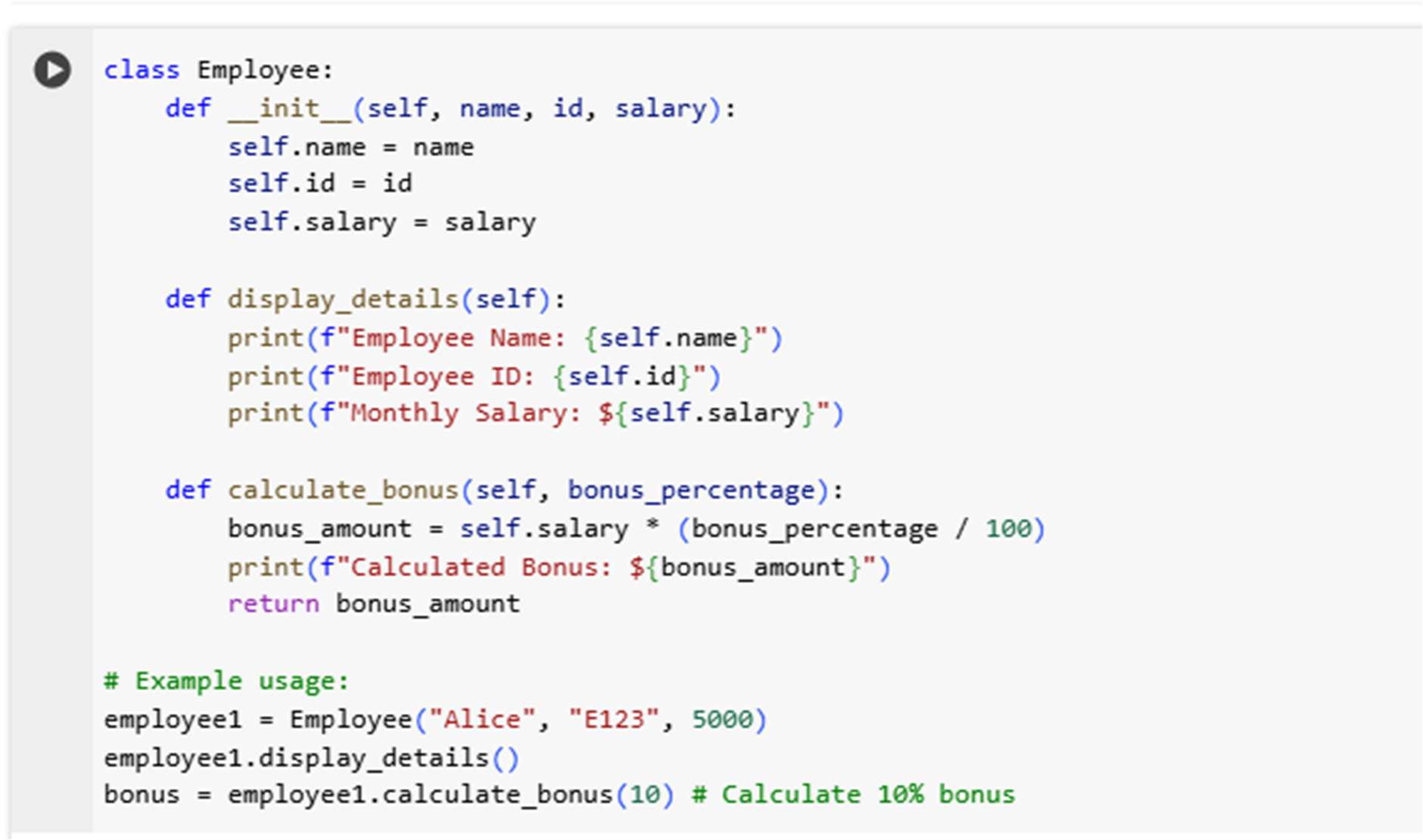
**2403A51271**

**Batch-12**

**Task 1:**

**(Classes – Employee Management)**

* Task: Use AI to create an Employee class with attributes (name, id, salary) and a method to calculate yearly salary.

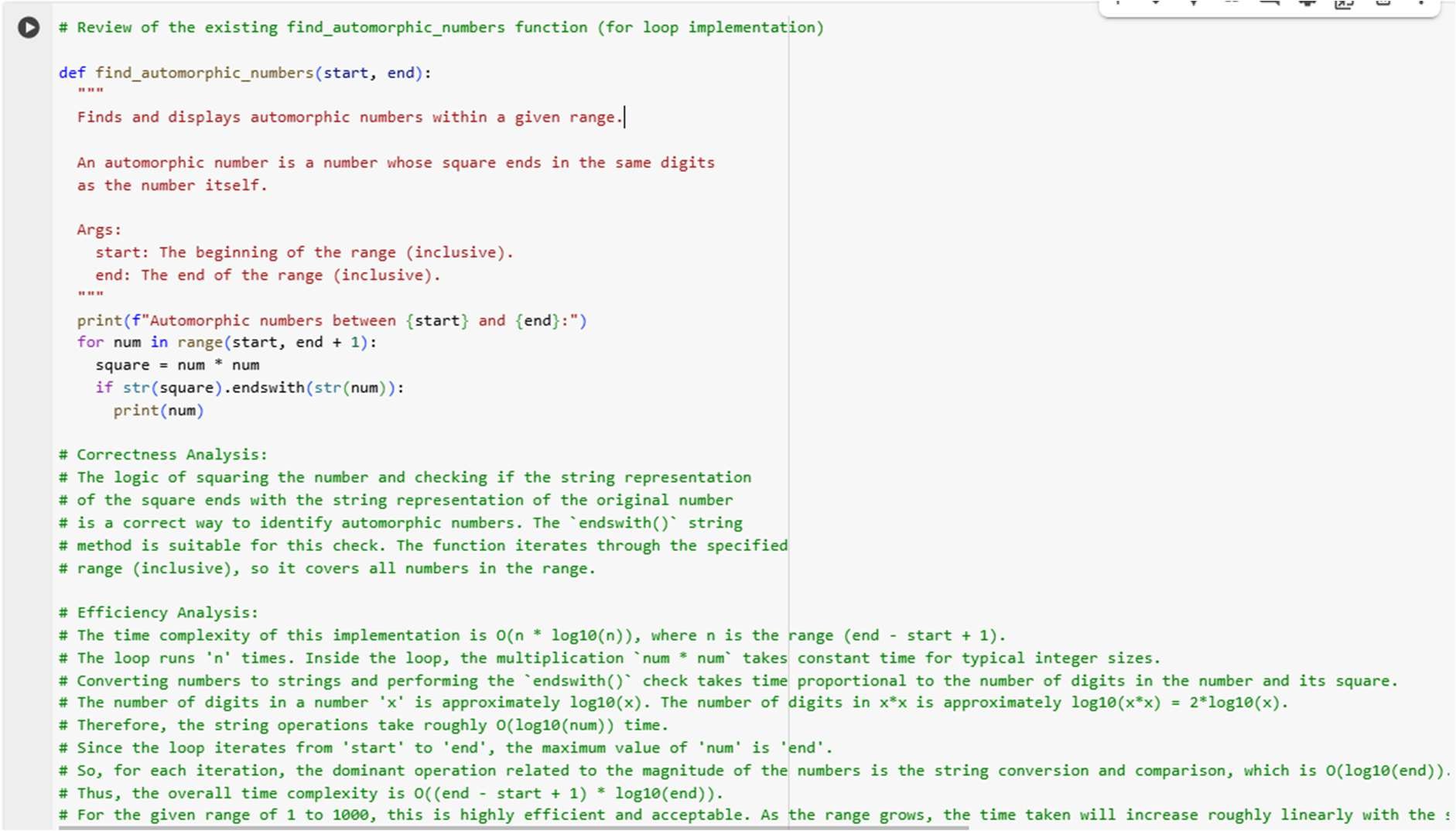
**CODE:**

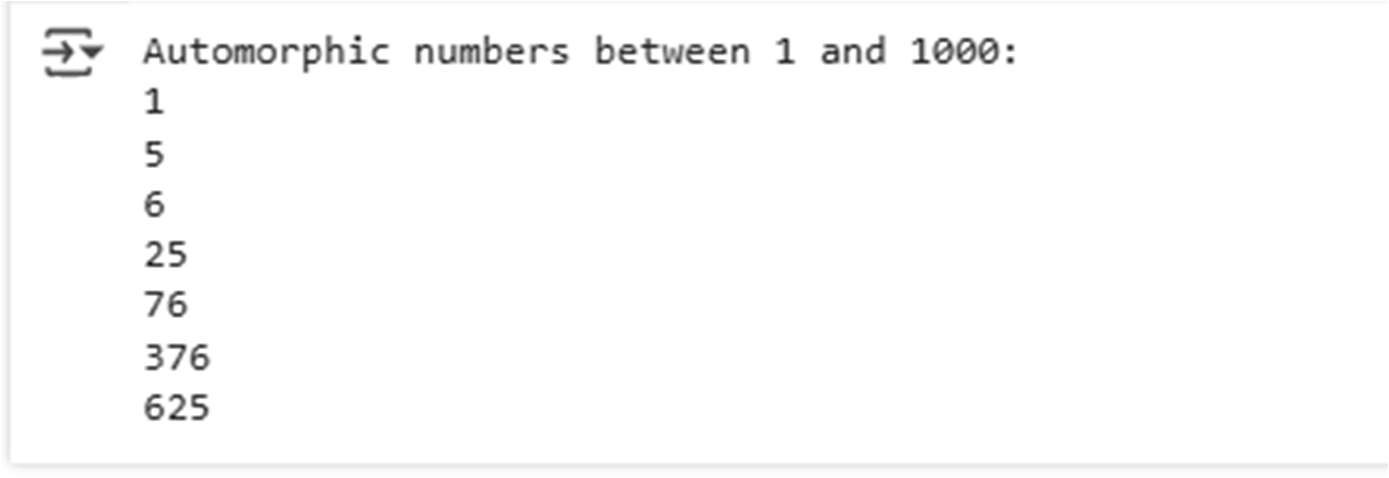
**OUTPUT:**

## Task 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.

**CODE:**

**OUTPUT:**

# Subtask:

Examine the existing code for correctness and discuss its efficiencency.

**WHILE LOOP:**

**COMPARISION:**

## Syntax and Structure:

* + For loop (find\_automorphic\_numbers): Uses `for num in range(start, end + 1):`
    - The loop initialization (`num = start`), condition (`num <= end`), and iteration step (`num += 1`) are implicitly handled by the `range()` function and the `for` statement.
  + While loop (find\_automorphic\_numbers\_while): Uses `while num <= end:`
    - Requires explicit initialization of the loop variable (`num = start`) before the loop.
    - Requires explicit updating of the loop variable (`num += 1`) within the loop body.
    - The loop condition (`num <= end`) is stated directly in the `while` statement.

## Readability and Conciseness:

* + For loop: Generally considered more readable and concise when iterating over a known sequence or range.
    - The structure `for item in sequence:` clearly indicates iteration over a collection.
    - For iterating through a range, the `range()` function makes the start and end points immediately clear.
  + **While loop:** Can be slightly less concise for simple range iteration as it requires managing the loop variable manually.
    - More suitable when the number of iterations is not known in advance or the loop termination depends on a complex condition.
    - For this specific task (iterating through a defined range), the for loop is arguably more idiomatic and slightly more readable.

## Performance and Efficiency:

* + For loop and While loop implementations for this task are expected to have virtually identical performance.
    - Both loops execute the same core logic (squaring, string conversion, endswith check) for each number in the range.
    - The underlying operations within the loops are the same.
    - Any minor differences in overhead are negligible for typical ranges like 1 to 1000.
    - Both implementations have a time complexity of O((end - start + 1) \* log10(end)) as discussed previously.

## Advantages and Disadvantages for this Task:

* + **For loop:**
    - Advantage: More natural and concise syntax for iterating over a fixed range. Less prone to infinite loops due to forgetting to update the loop variable.
    - Disadvantage: Less flexible than a while loop if the iteration logic becomes more complex and doesn't fit the simple 'iterate over a sequence' pattern.

## While loop:

* + Advantage: More flexible when the loop condition is not directly tied to iterating through a simple range or sequence. Useful for conditions that change dynamically.
  + Disadvantage: Requires more careful management of the loop variable (initialization and update), increasing the potential for off-by-one errors or infinite loops if not handled correctly.
  + For this specific task, the advantage of flexibility offered by the while loop is not particularly needed, making the for loop a slightly better fit in terms of simplicity and conciseness.

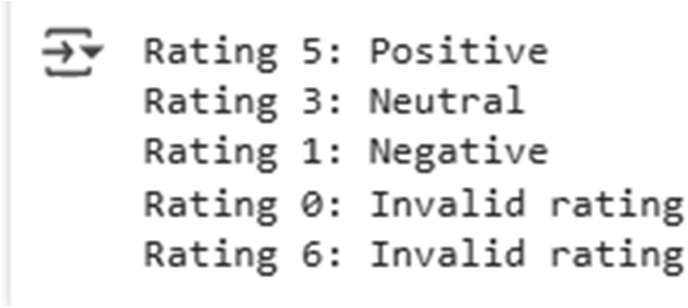
**Conclusion:** For finding automorphic numbers within a fixed range, the `for` loop implementation is generally preferred due to its cleaner syntax and reduced risk of errors related to loop variable management, although both implementations are equally efficient.

# Task 3 :

**(Conditional Statements – Online Shopping Feedback Classification)**

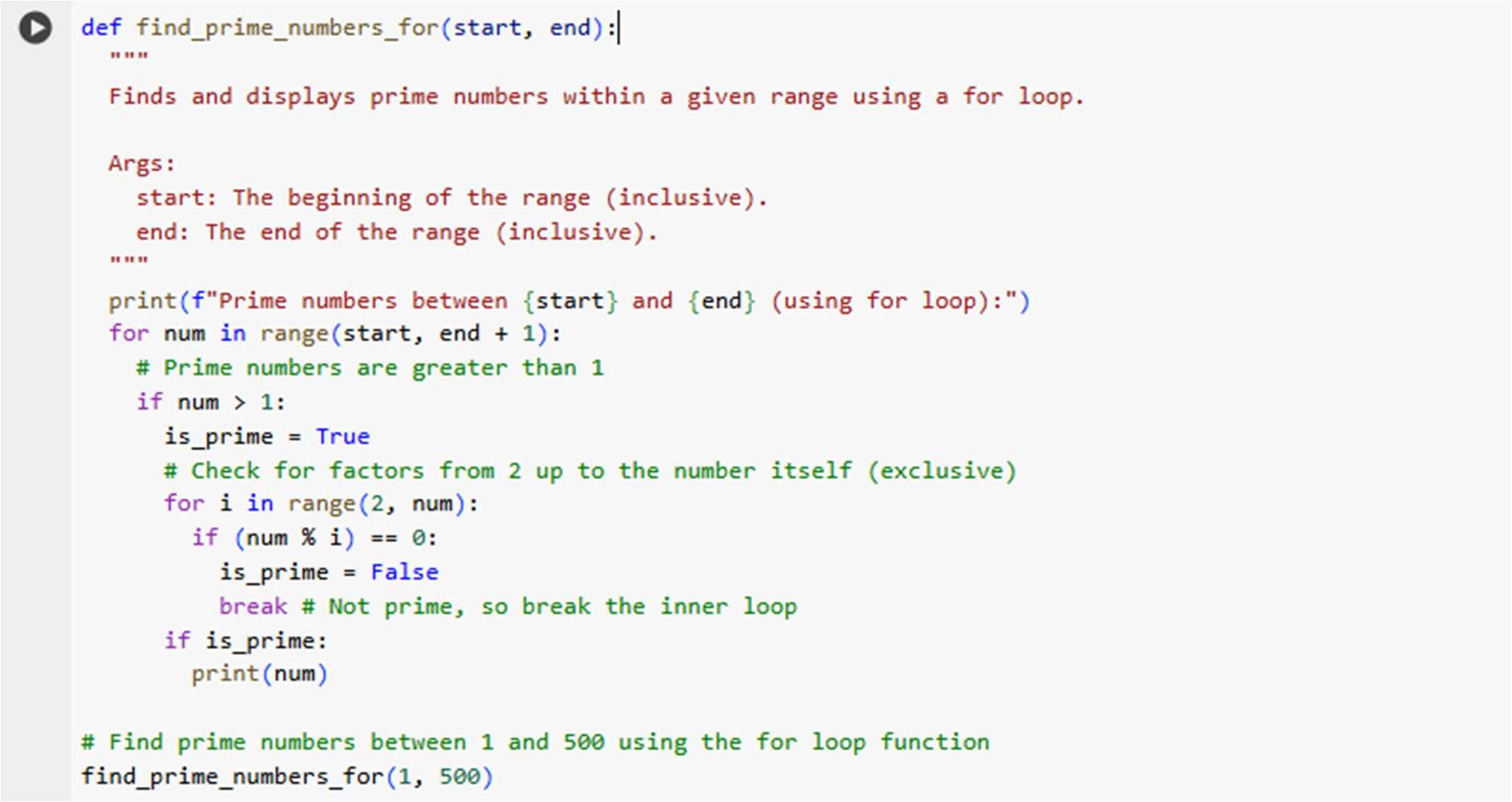
* Task: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).

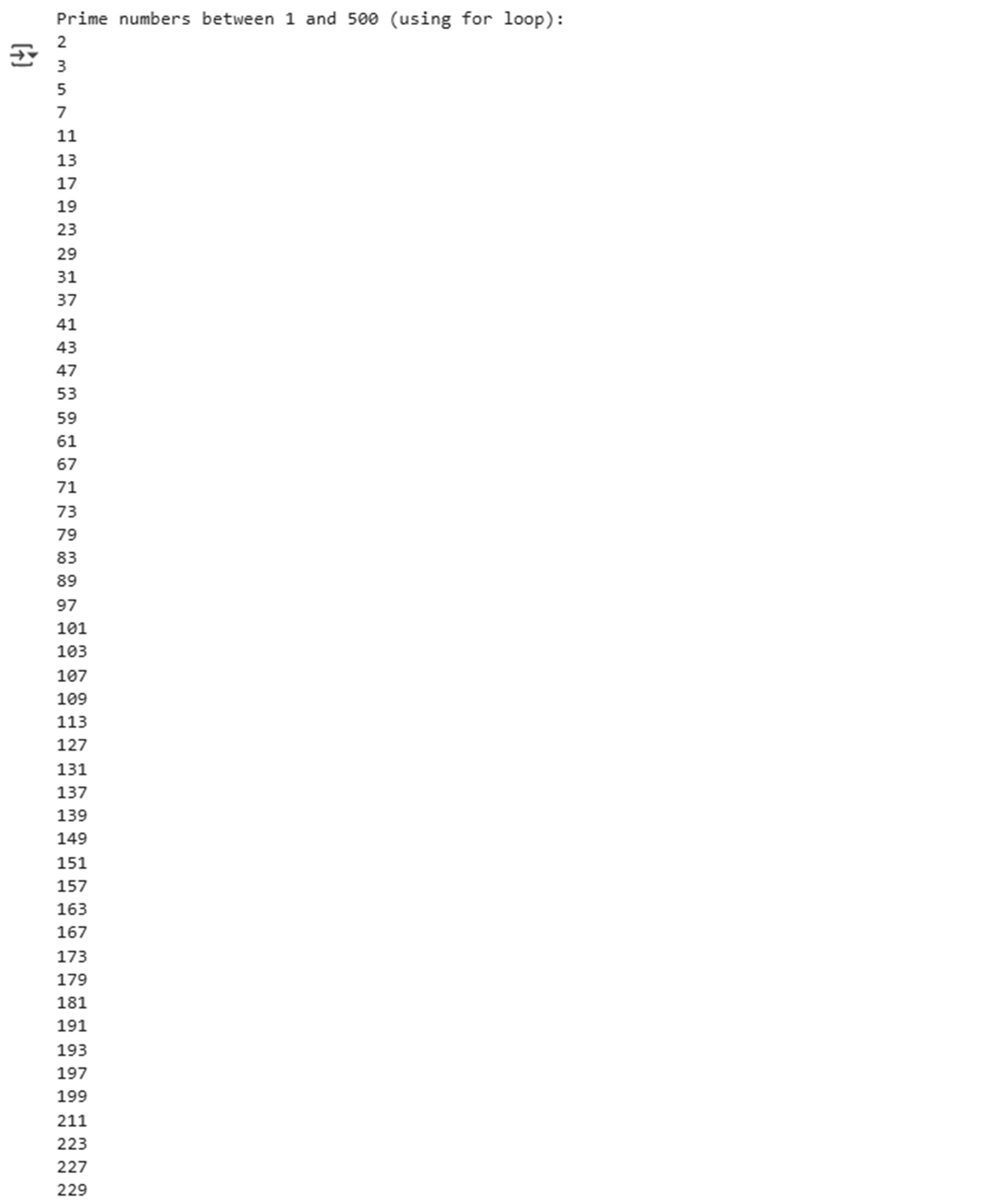
**CODE:**

**OUTPUT:**

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

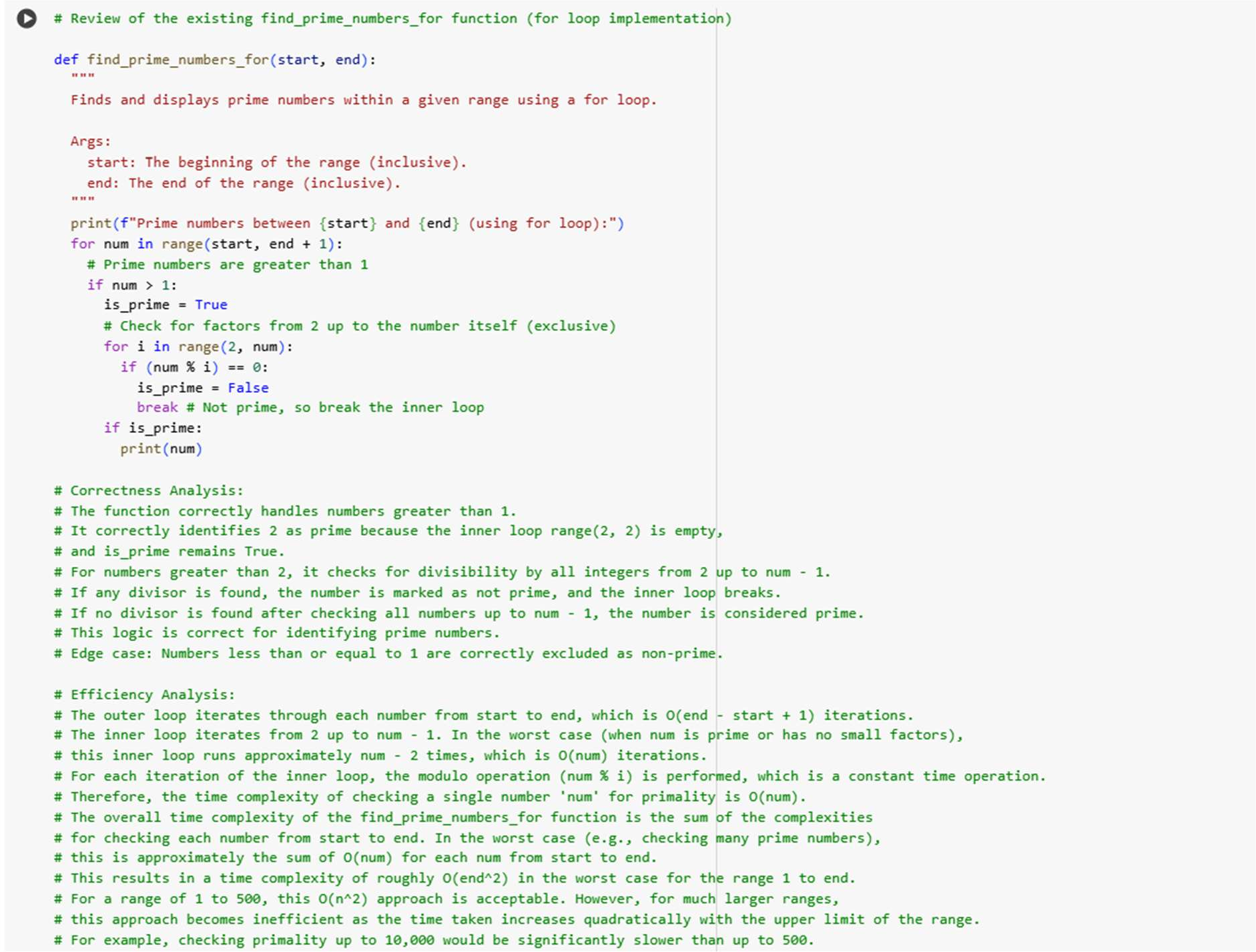
* Task: Generate a function using AI that displays all prime numbers within a user-specified range (e.g., 1 to 500).

CODE:

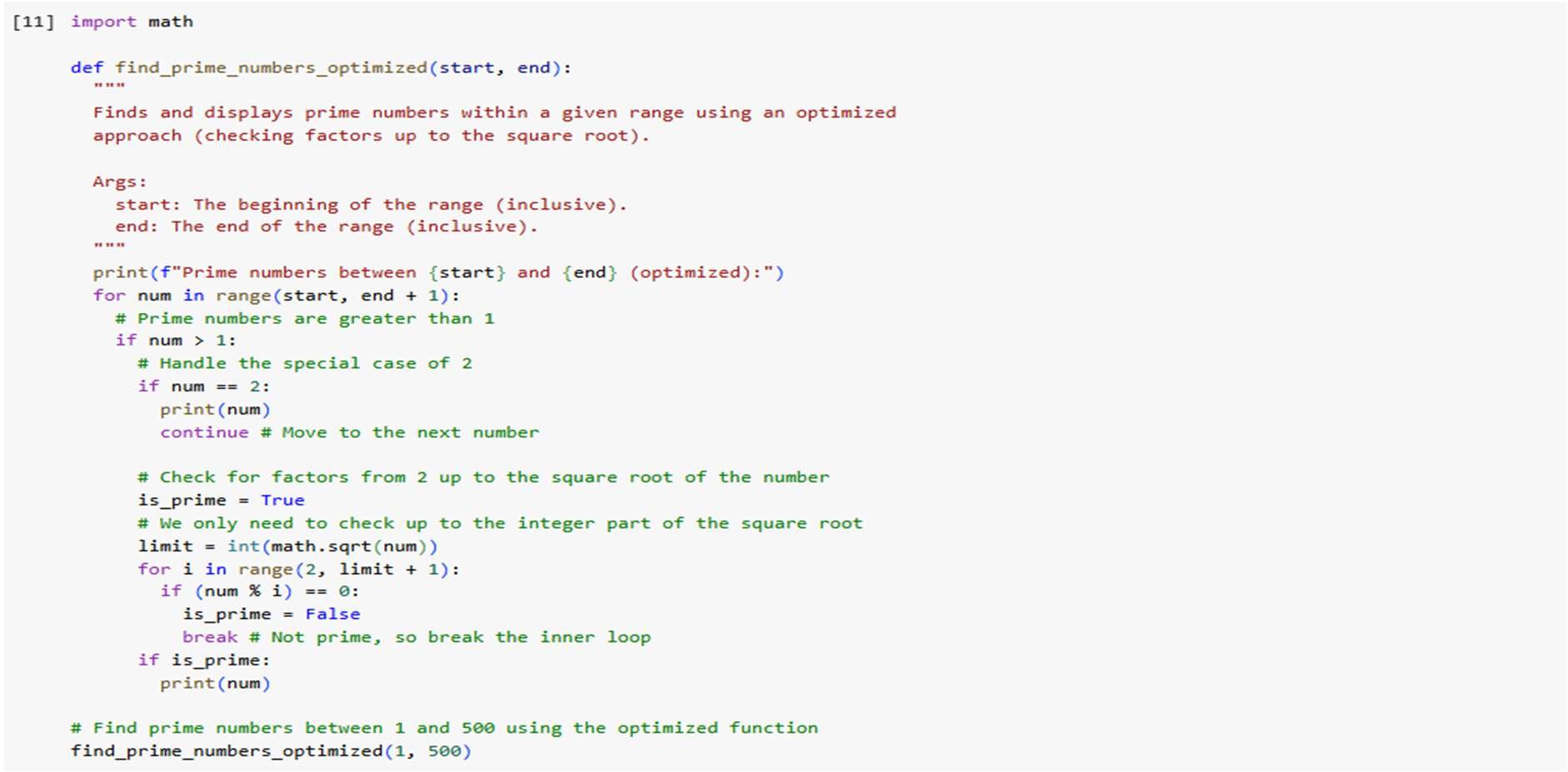
**OUTPUT:**

# Subtask:

Examine the generated code for correctness and discuss its efficiency, particularly focusing on the prime-checking logic.



**Subtask:**Create an optimized version of the function, for example, by using the square root method for prime checking.



## output:

**Subtask:**

Discuss the differences in efficiency between the initial and optimized versions. **Comparison of Efficiency: Initial (O(n^2)) vs. Optimized (O(n \* sqrt(n))) Prime Finding:**

## Time Complexity Comparison:

* + Initial (for loop): O(end^2)
    - The inner loop checks for divisibility up to `num - 1` for each number `num`, leading to a quadratic relationship with the upper limit of the range (`end`).
  + Optimized (square root method): O(end \* sqrt(end))
    - For each number `num`, the inner loop checks for divisibility up to

`sqrt(num)`. The maximum value of `sqrt(num)` is `sqrt(end)`.

* + - The overall complexity is the sum of checking each number, which is approximately proportional to `end \* sqrt(end)`.

## Why checking up to the square root is sufficient:

To determine if a number `num` is prime, we need to check if it has any divisors other than 1 and itself.

If `num` has a divisor `i` such that `1 < i < num`, then `num = i \* j` for some integer `j` where `1 < j < num`.

If `i` is a divisor, then `j = num / i` is also a divisor. Consider the relationship between `i` and `j`:

* If `i < sqrt(num)`, then `j = num / i > num / sqrt(num) = sqrt(num)`. So, if there's a divisor less than the square root, there's a corresponding divisor greater than the square root.
* If `i > sqrt(num)`, then `j = num / i < num / sqrt(num) = sqrt(num)`. So, if there's a divisor greater than the square root, there's a corresponding divisor less than the square root.
* If `i = sqrt(num)`, then `num` is a perfect square, and `i = j = sqrt(num)` is a divisor.

This means that if a number `num` has any divisor other than 1 and itself, it must have at least one divisor less than or equal to its square root.

Therefore, to check for primality, we only need to test for divisibility by integers from 2 up to the integer part of `sqrt(num)`.

## Contribution to overall performance improvement:

The reduction in the number of checks for each number significantly improves performance, especially for larger ranges.

For a number `num`, the initial method performs approximately `num` checks in the worst case.

The optimized method performs approximately `sqrt(num)` checks in the worst case.

The number of checks is reduced from linear (proportional to `num`) to sub- linear (proportional to `sqrt(num)`).

As the range (`end`) increases, the difference between `end^2` and `end \* sqrt(end)` grows substantially.

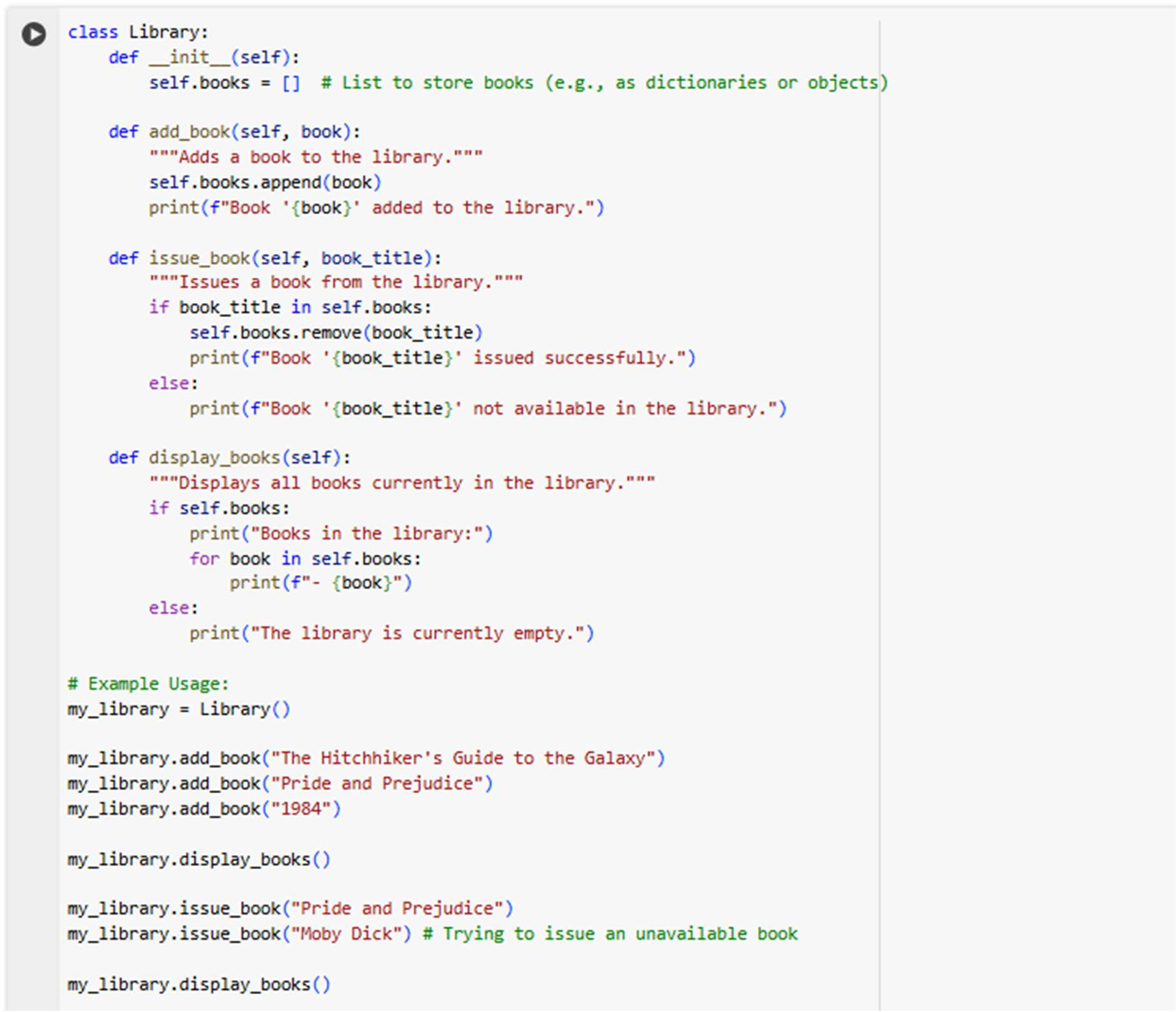
For example:

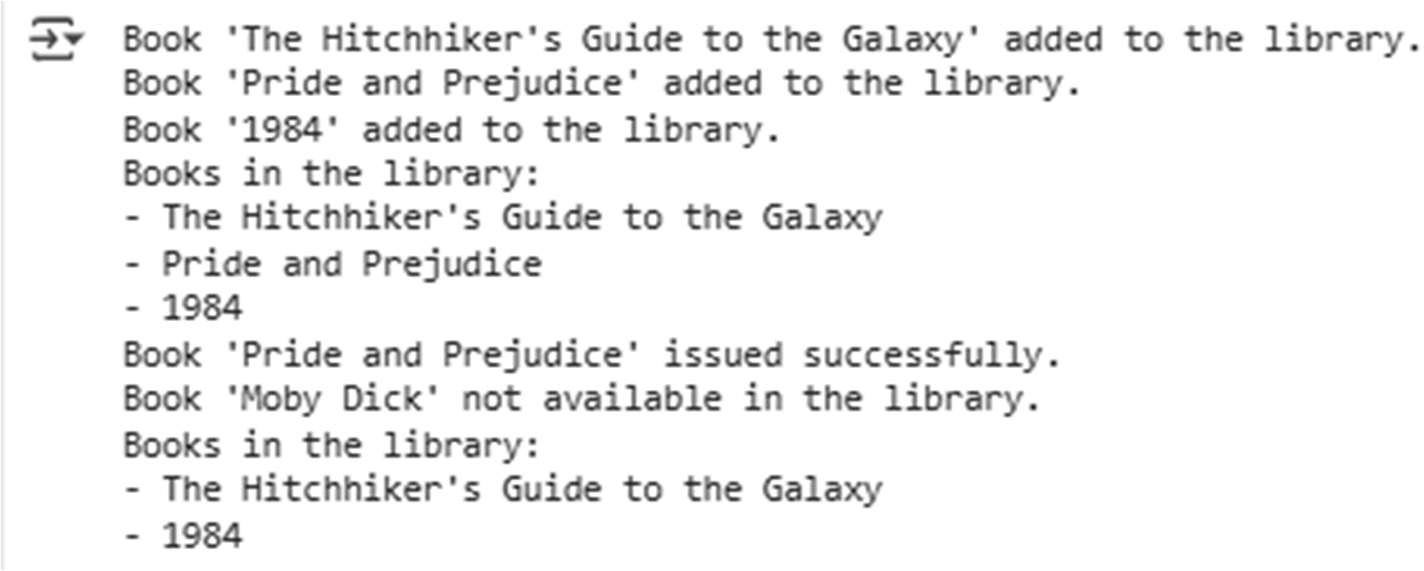
* If end = 100, initial checks ~ 100^2 = 10,000. Optimized checks ~ 100 \* sqrt(100) = 100 \* 10 = 1,000.
* If end = 1000, initial checks ~ 1000^2 = 1,000,000. Optimized checks ~ 1000 \* sqrt(1000) ≈ 1000 \* 31.6 ≈ 31,600.
* If end = 10000, initial checks ~ 10000^2 = 100,000,000. Optimized checks ~ 10000 \* sqrt(10000) = 10000 \* 100 = 1,000,000.

The optimized method performs significantly fewer checks for each number, and this saving accumulates over the entire range, leading to a much faster execution time for larger inputs.

## Task 5 (Classes – Library System)

* + Task: Use AI to build a Library class with methods to add\_book(), issue\_book(), and display\_books().

CODE:

**OUTPUT:**