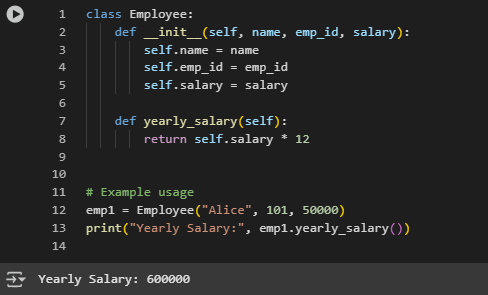
**AI LAB ASSIGNMENT 6.1**

*Hall ticket: 2403A51323*

*Batch: 13*

Task Description #1 (Classes – Employee Management)

AI Generated Code:

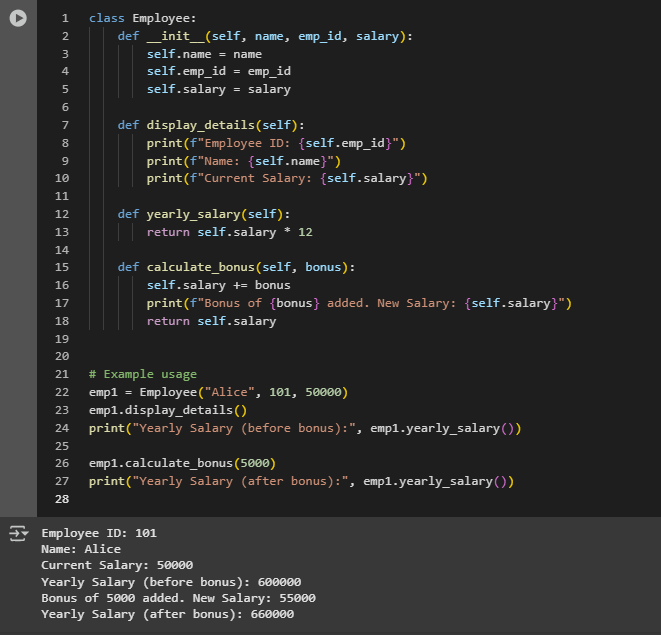


The generated code is correct in syntax and logic for calculating yearly salary. However, its structure is incomplete since it does not include methods for displaying employee details or calculating bonuses, which are required by the task.

The code is **minimal but incomplete**:

* Missing display\_details() method.
* Missing calculate\_bonus() method.
* Lacks input validation (e.g., negative salaries).

Updated Code:

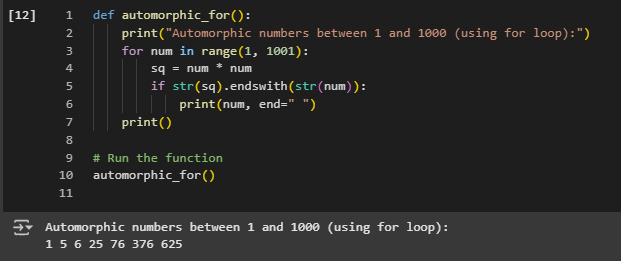


Observation:

The basic code of the Employee class correctly calculated the yearly salary using the given monthly salary, but it was limited as it did not display details or handle bonuses. In the updated version, the class was improved with methods to display employee details and calculate bonuses, making it more complete and functional.

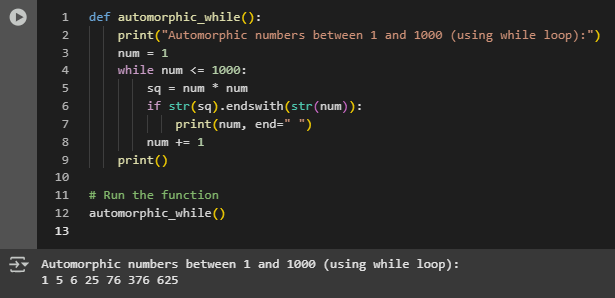
Task Description #2 (Loops – Automorphic Numbers in a Range)

AI-generated code:(Using for loop)

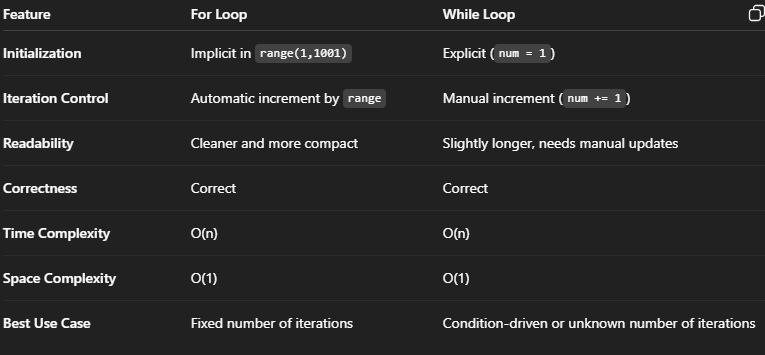


The for loop correctly identifies all Automorphic numbers between 1 and 1000 by checking if the square ends with the number.  
**Time Complexity:** O(n) for n numbers, with negligible overhead from string conversion.  
**Space Complexity:** O(1), as it only uses variables for the number and its square.

AI-generated code:(Using While loop)



Comparison of **for loop vs while loop:**

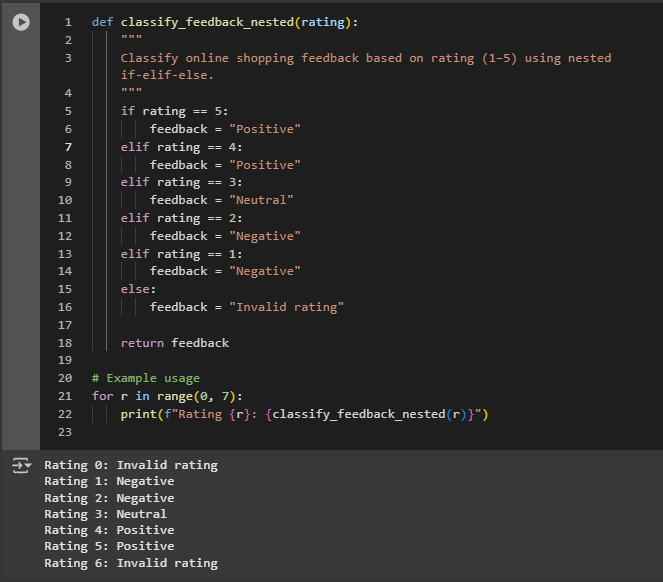
****

**Observation:**

Both the for loop and while loop implementations correctly identify all Automorphic numbers between 1 and 1000, which are 1, 5, 6, 25, 76, 376, and 625. The for loop is more concise and readable since it handles iteration automatically, whereas the while loop requires manual initialization and increment. Both methods have the same time complexity O(n) and space complexity O(1). The logic of checking if the square of a number ends with the number itself using string comparison is correct and efficient for the given range.

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

Initial Code:(ested if-elif-else)



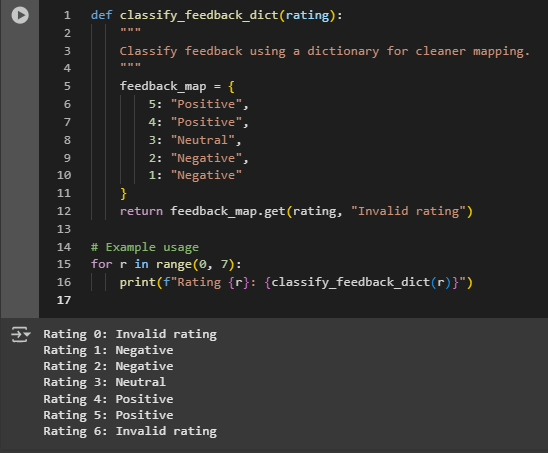
**Correctness:**

* The function correctly maps numerical ratings to feedback categories:
  + Ratings **5 and 4 → Positive**
  + Rating **3 → Neutral**
  + Ratings **2 and 1 → Negative**
* It also handles invalid inputs (ratings outside 1–5) by returning **“Invalid rating”**.
* Each condition is mutually exclusive and covers all possible valid inputs.

**Readability:**

* The nested if-elif-else structure is straightforward and easy to understand for beginners.
* However, there is **repetition** for similar categories (Positive for 5 and 4, Negative for 1 and 2), which makes the code slightly verbose.
* For small rating scales, readability is acceptable, but for larger scales or more categories, the code can become **hard to maintain**.

Alternative Approach using Dictionary:

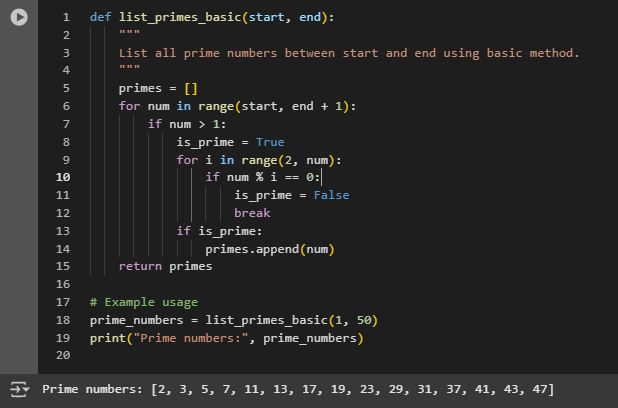


Observation:

The feedback classification function using nested if-elif-else correctly maps numerical ratings (1–5) to categories: **Positive (4–5), Neutral (3), Negative (1–2)**, and handles invalid ratings appropriately. The nested structure is easy to understand for small rating scales but contains some repetition, making it slightly verbose. The alternative dictionary-based approach achieves the same classification in a cleaner and more maintainable way, allowing quick updates to rating mappings without changing the core logic. Both approaches are correct, efficient (O(1) per lookup), and suitable for practical feedback classification tasks.

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

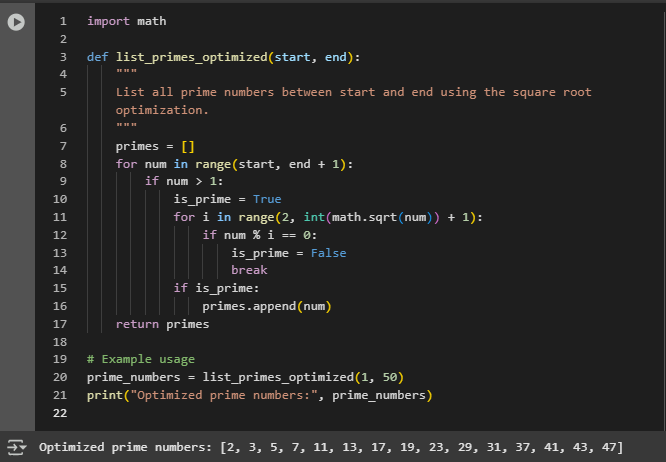
Simple Prime Number Function Using a For Loop:



**Correctness & Efficiency Analysis (Basic Version)**

* **Correctness:**
  + Checks all numbers > 1.
  + Iterates from 2 to num-1 to see if num has a divisor.
  + Works correctly for any small range.
* **Efficiency:**
  + Time complexity: O(n²) for range n (because for each number we check up to num-1).
  + Space complexity: O(n) (for storing prime numbers).
  + Inefficient for large ranges due to repeated unnecessary checks.

Optimized Prime Checking Using Square Root Method:

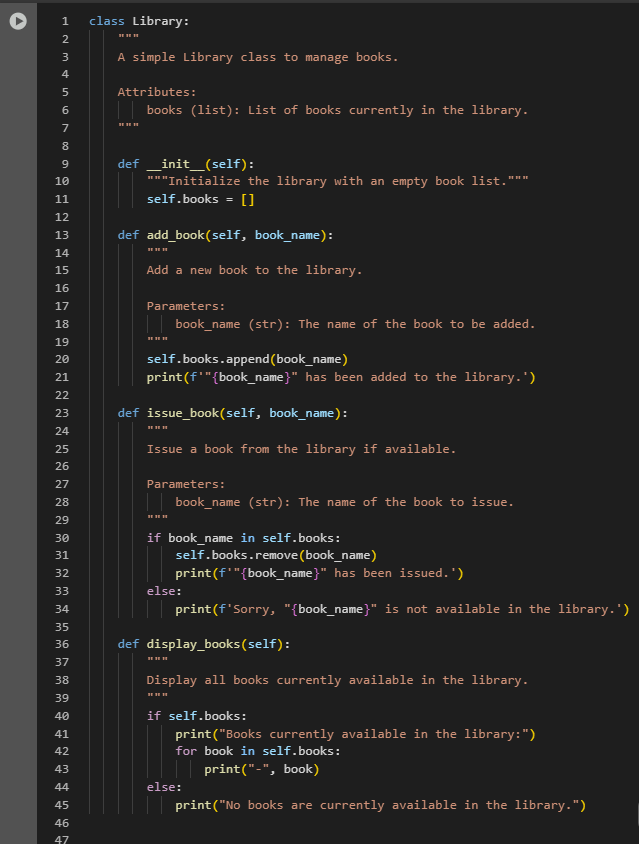
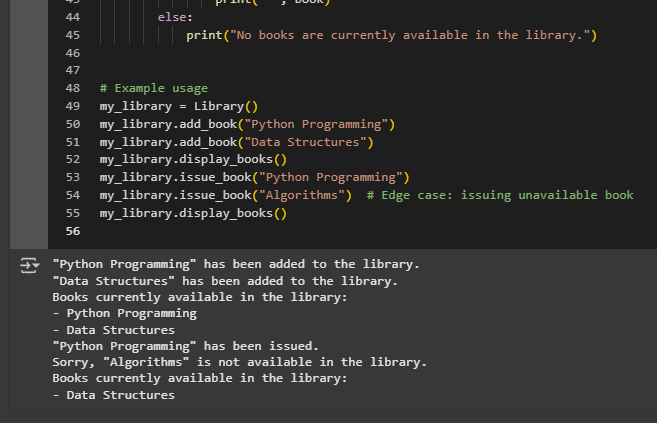


**Observation:**

The program successfully lists all prime numbers within a user-specified range. The basic for-loop implementation correctly identifies primes by checking divisibility from 2 up to the number minus one, but it is inefficient for large ranges. The optimized version using the square root method significantly reduces the number of checks while maintaining correctness, making it more efficient for larger ranges. Both methods correctly output prime numbers and demonstrate the effectiveness of algorithmic optimization in improving performance.

Task Description #5 (Classes – Library System)

Generated Code:

**Issuing unavailable books**

* Handled by checking if book\_name in self.books.
* If the book is not in the library, a message is printed instead of causing an error.

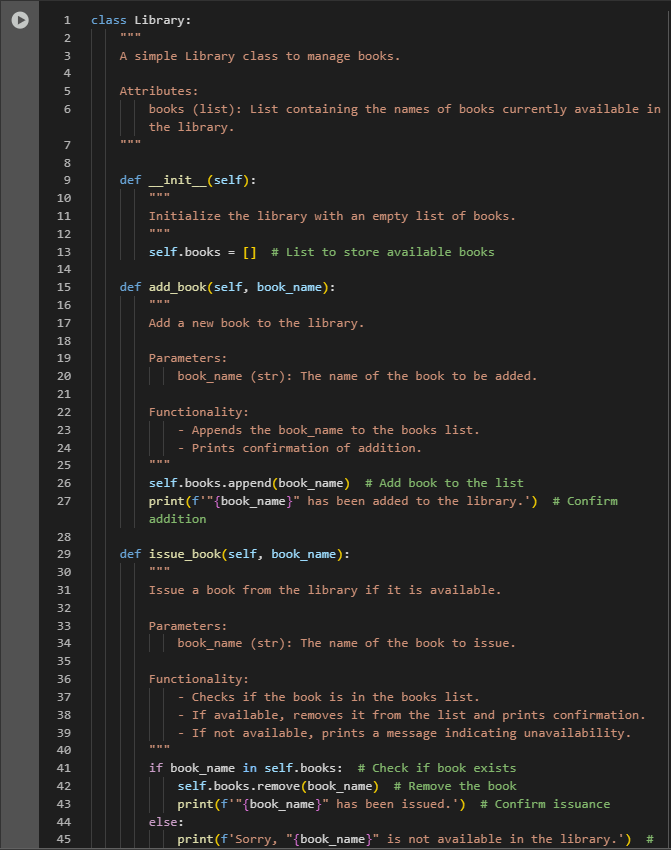
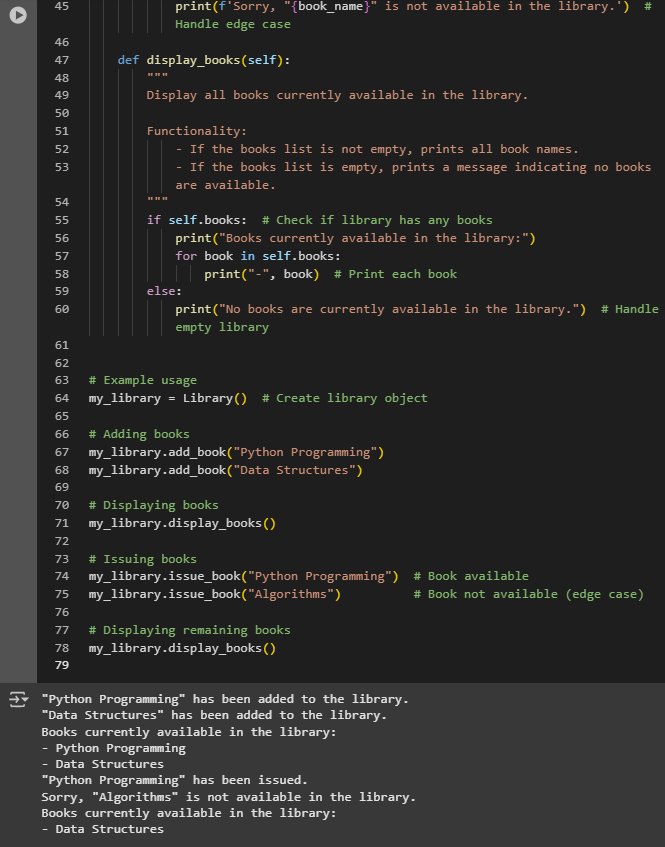
**Empty library**

* display\_books() checks if self.books is empty and prints a message accordingly.

**Adding duplicate books**

* Currently, duplicates are allowed.
* Could be improved by checking if book\_name not in self.books before adding.

After Adding Comments:

Observation:

The Library class successfully manages a collection of books with methods to add, issue, and display books. The add\_book() method correctly adds new books, while issue\_book() properly handles issuing available books and gracefully manages cases where a book is unavailable. The display\_books() method shows all current books and appropriately handles an empty library. Inline comments and documentation make the class easy to understand and maintain. Overall, the class demonstrates correct functionality, good readability, and effective handling of edge cases.