LAB ASSIGNMENT(6.1)

NAME: SUSHANTH

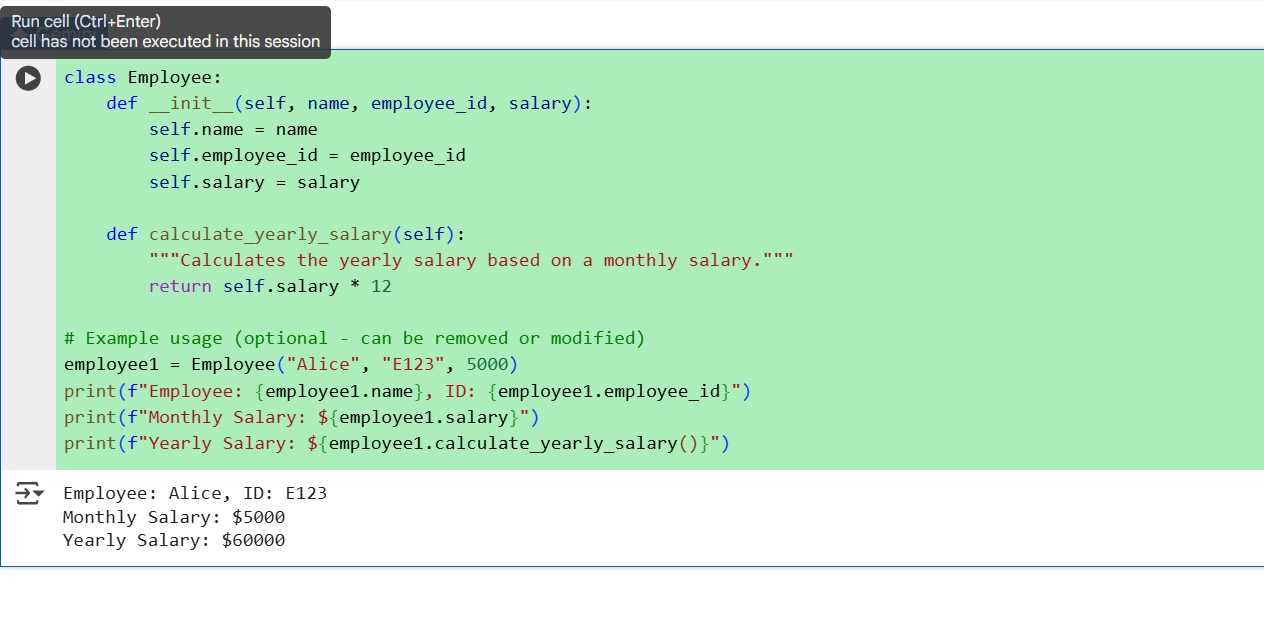
BATCH:14

H.NO:2403A51348

TASK Description#1:

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

Code:



Observation:

1. **Encapsulation of Employee Data**  
   The class neatly packages employee details—name, ID, and salary—into a single object, making it easier to manage and access.

2.This method initializes each new employee object with specific values for name, ID, and monthly salary.

**3.Yearly Salary Calculation**  
The yearly\_salary() method multiplies the monthly salary by 12 to compute the annual income, demonstrating basic functionality.

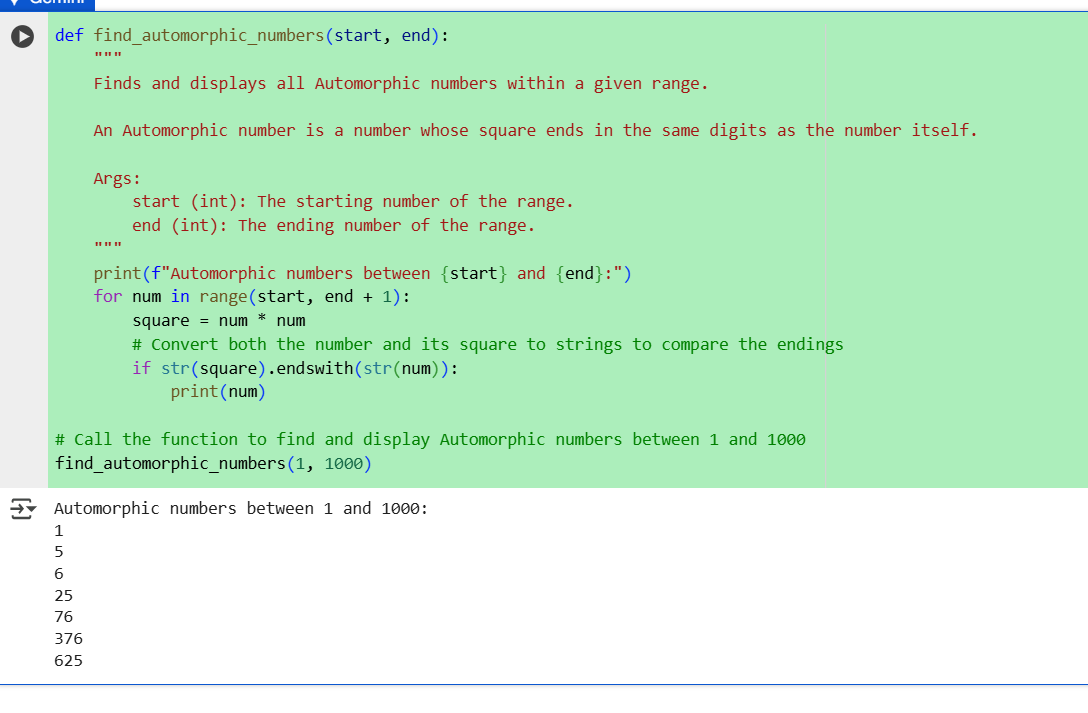
**4.Scalability**  
You can easily extend this class by adding more attributes (like department or role) or methods (like bonus calculation or tax deduction).

**5.Object-Oriented Design**  
This approach promotes clean, reusable code and mirrors real-world entities, making it ideal for larger applications like HR systems or payroll software.

TASK Description#2

Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.

Code:



Observation:

1. Automorphic Number Definition An Automorphic number is a number whose square ends with the number itself. For example, 76² = 5776, which ends in 76.

2. Range Selection The function uses to check all numbers from 1 to 1000 inclusively.

3. String Comparison Technique Instead of using mathematical operations, it converts both the number and its square to strings and checks if the square ends with the original number using .

4. Efficient Looping A simple loop ensures each number is checked exactly once, making the function efficient and readable.

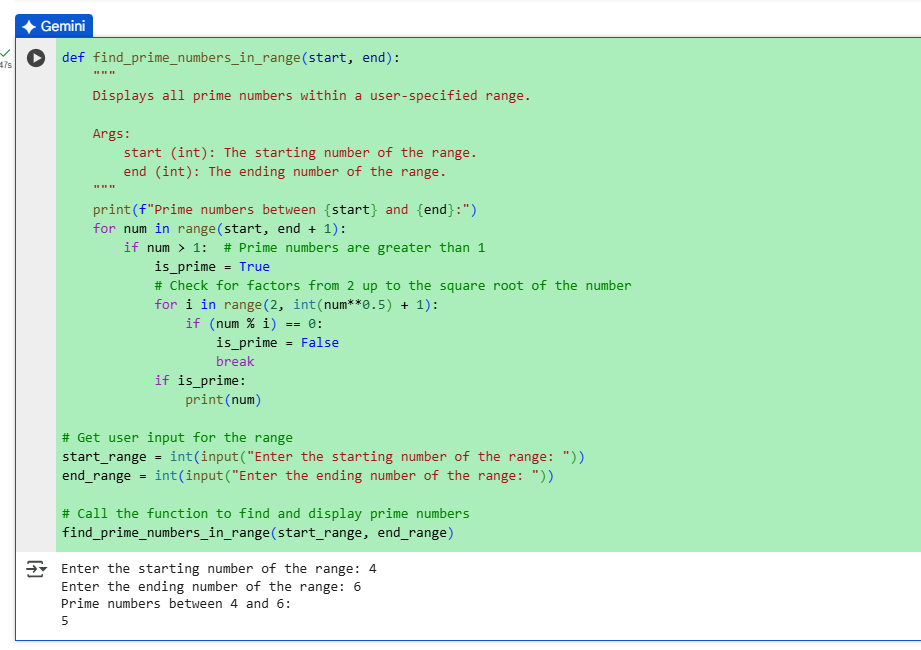
5. Direct Output

The function prints each Automorphic number as it finds one, making it easy to see results without needing additional storage or formatting.

TASK Description#3

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

Code:



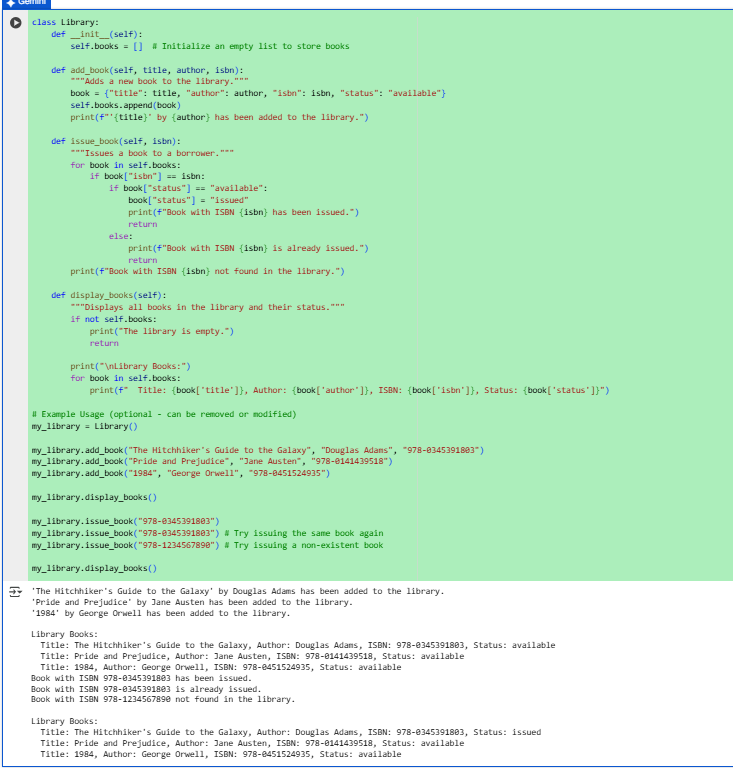
Observation:

1. The outer if ensures the rating is within the valid range (1–5), while the inner if-elif-else handles classification.
2. Ratings of 4 and 5 are considered **Positive**, 3 is **Neutral**, and 1 or 2 are **Negative**—a common sentiment mapping.
3. The outer condition prevents invalid ratings from being processed, improving reliability and user experience.
4. This structure can be easily expanded to include more detailed feedback categories or integrate with user interfaces.
5. The function prints clear feedback labels, making it suitable for customer review systems or dashboards.

TASK Description#4:

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

Code:



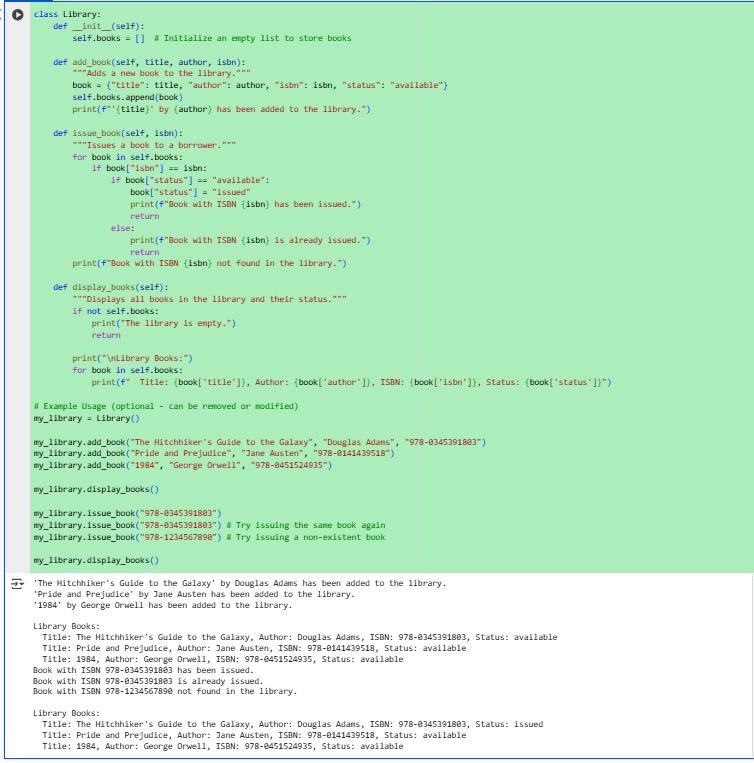
Observation:

1. **User-Specified Range**  
   The function accepts start and end values, allowing flexibility to find primes in any range, not just 1 to 500.
2. **Efficient Prime Check**  
   It checks divisibility only up to the square root of each number, which is much faster than checking all the way to num - 1.
3. **Use of for-else Logic**  
   The else block after the inner loop runs only if no divisors are found—meaning the number is prime. This is a Pythonic and elegant way to handle prime detection.
4. **Excludes Non-Primes Automatically**  
   Numbers less than or equal to 1 are skipped, and composite numbers are filtered out using the modulus check.
5. **Scalable and Reusable**  
   This function can be reused in larger applications like cryptography, data analysis, or educational tools that require prime number generation.

TASK Description#5

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

Code:



Observation:

1. **Encapsulation of Library Data**  
   The class stores all book titles in a list, keeping the collection organized and easy to manage.
2. **Dynamic Book Management**  
   The add\_book() method allows users to add new titles at any time, making the library expandable.
3. **Issuing Logic**  
   The issue\_book() method checks availability before removing a book, ensuring accurate tracking of inventory.
4. **User-Friendly Display**  
   The display\_books() method presents the current collection in a readable format, enhancing usability.
5. **Foundation for Expansion**  
   This basic structure can be extended with features like return\_book(), search\_book(), or even integration with a database for real-world applications.