Lab Assignment – 2

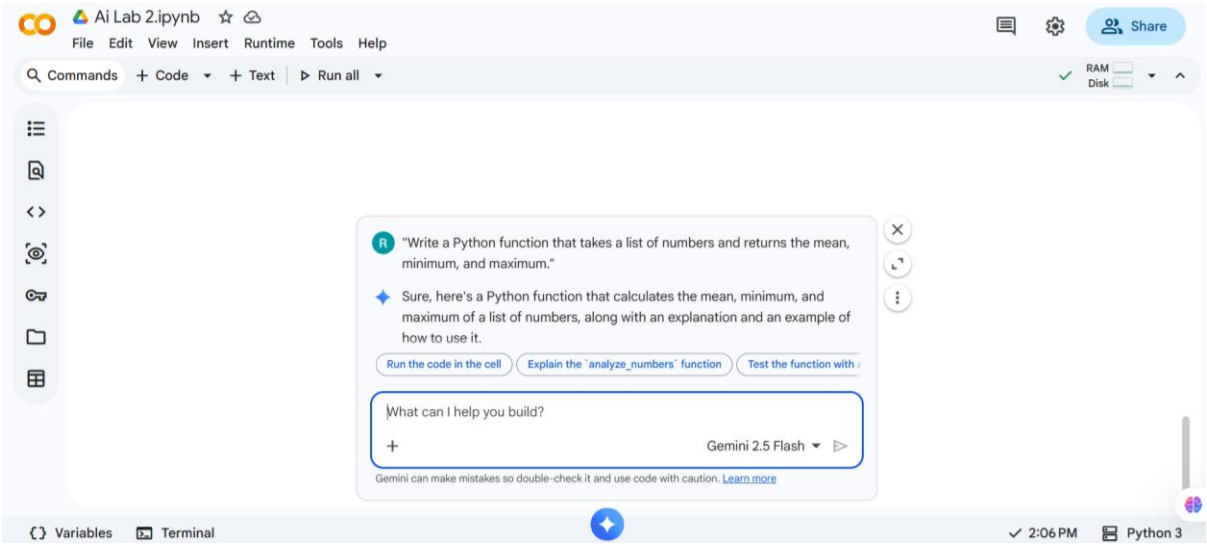
Name : L .Srinivas Reddy

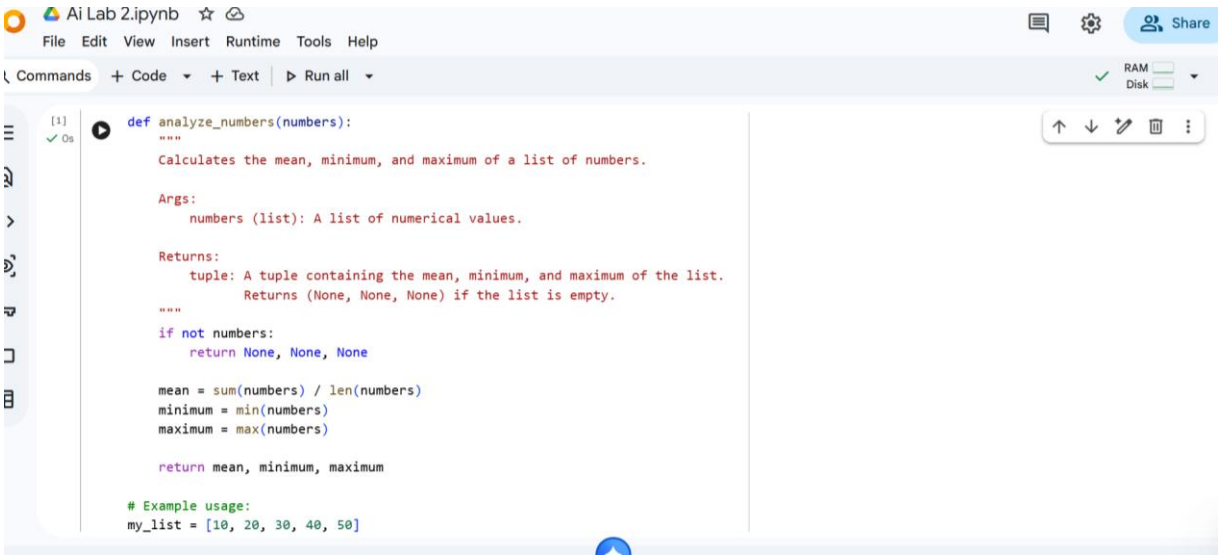
HT.No : 2303A52274

Batch : 36

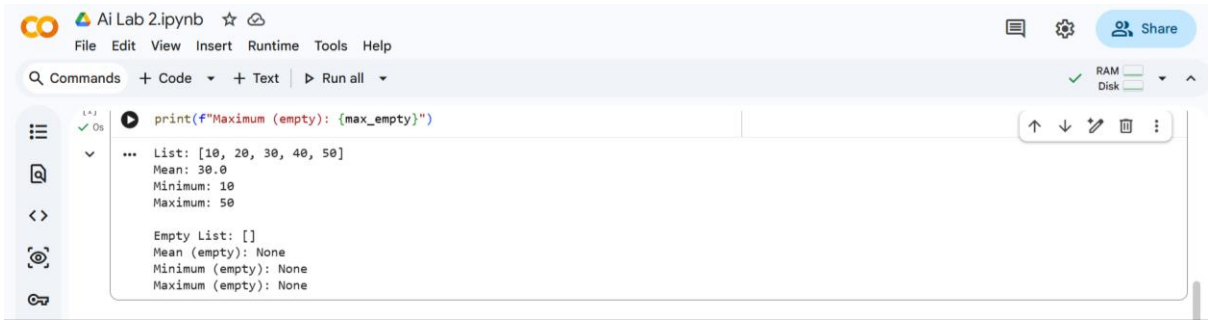
Task 1: Statistical Summary for Survey Data
Use Google Gemini in Colab to generate a Python function that reads a list of numbers and calculates the **mean, minimum, and maximum** values.

1 . Screenshot of Gemini prompt and result





OUTPUT :



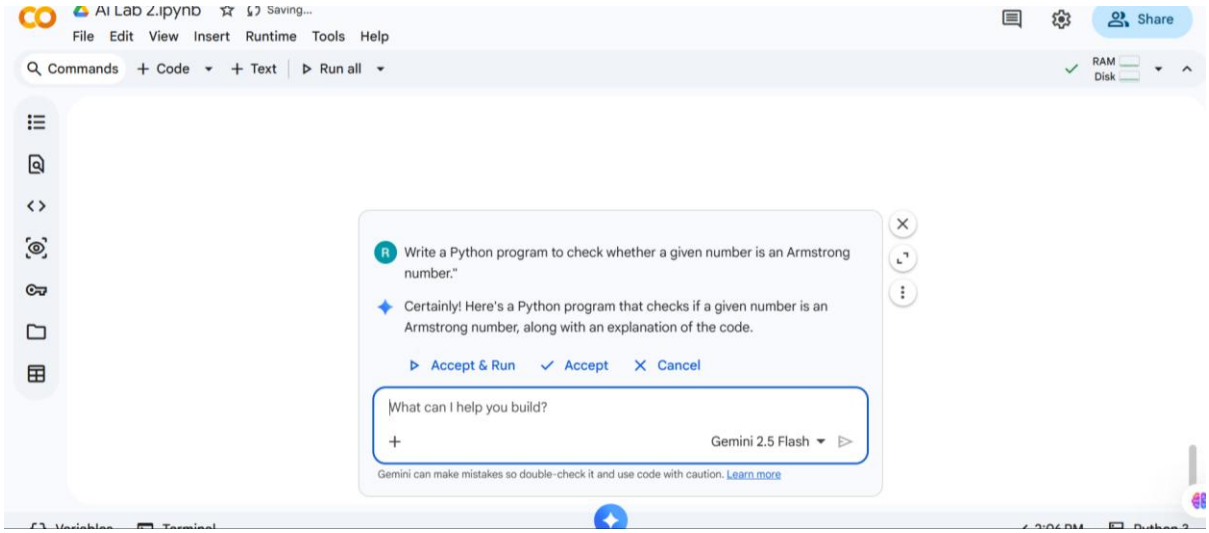
Explanation

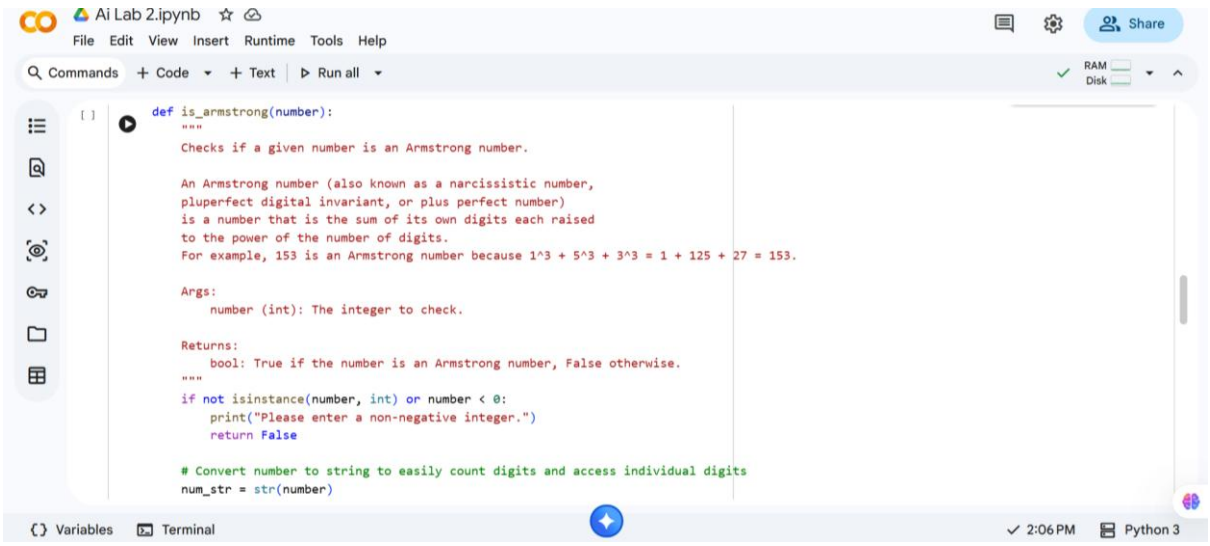
- **def analyze_numbers(numbers):**: This defines a function named analyze_numbers that takes one argument, numbers, which is expected to be a list.
- **Docstring**: The triple-quoted string explains what the function does, its arguments (Args), and what it returns (Returns). This is good practice for documenting your code.
- **if not numbers:**: This line checks if the input list numbers is empty. If it is, the function returns (None, None, None) to avoid errors (like division by zero for the mean or min()/max() on an empty list).
- **mean = sum(numbers) / len(numbers)**: Calculates the mean (average) by summing all numbers in the list using sum() and dividing by the count of numbers using len().
- **minimum = min(numbers)**: Finds the smallest number in the list using the built-in min() function.
- **maximum = max(numbers)**: Finds the largest number in the list using the built-in max() function.

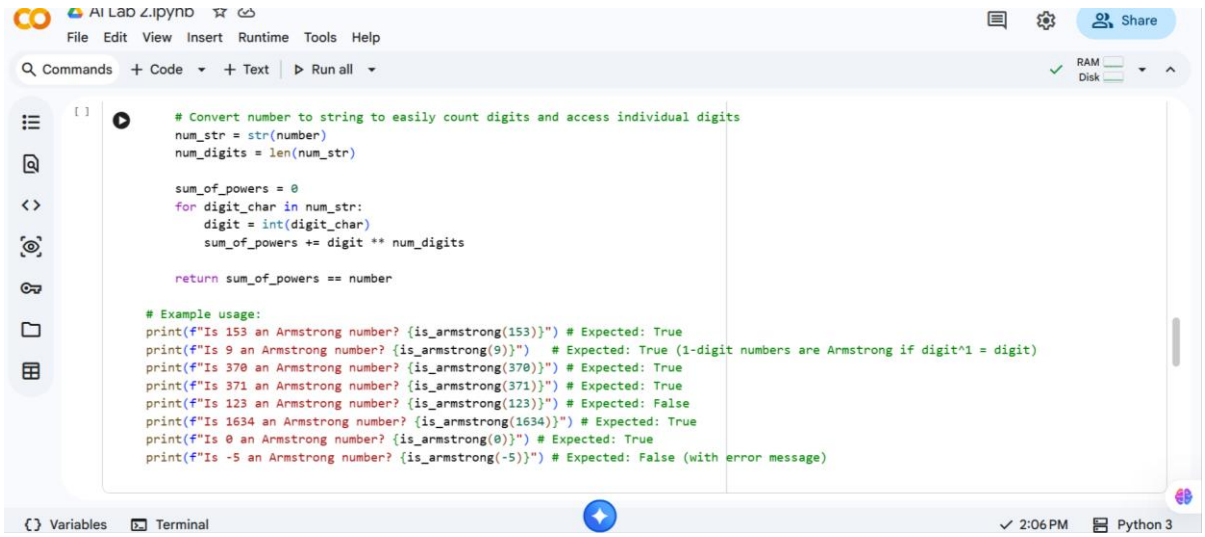
- **return mean, minimum, maximum**: The function returns these three calculated values as a tuple.

Task 2: Armstrong Number – AI Comparison

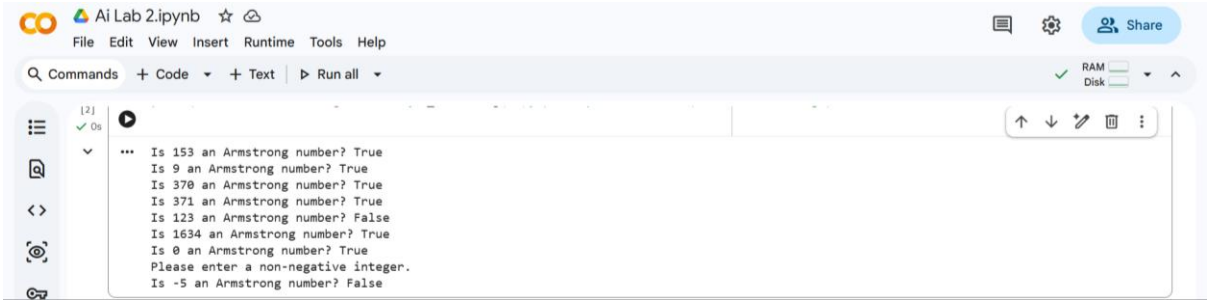
Generate an Armstrong number checker using Gemini and GitHub Copilot.

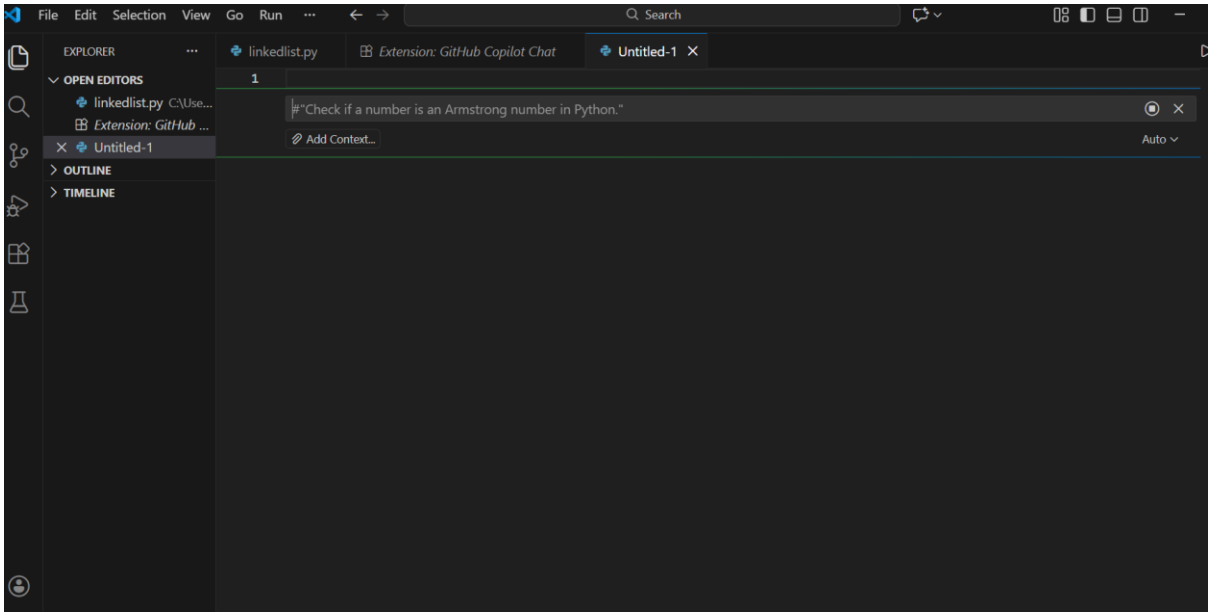


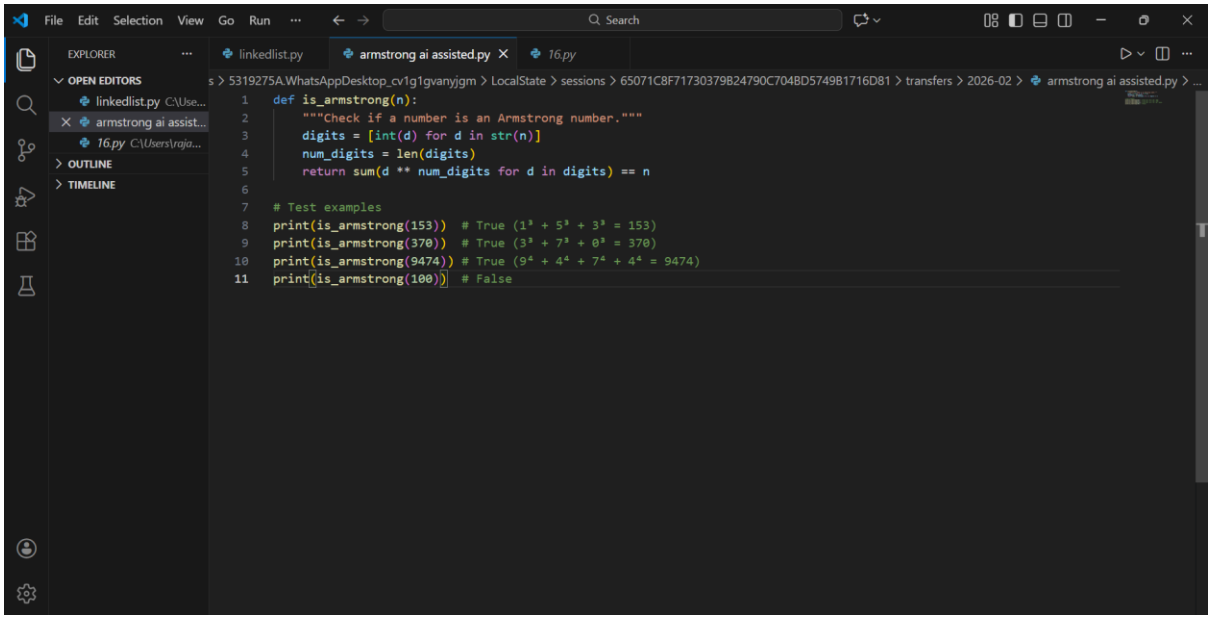




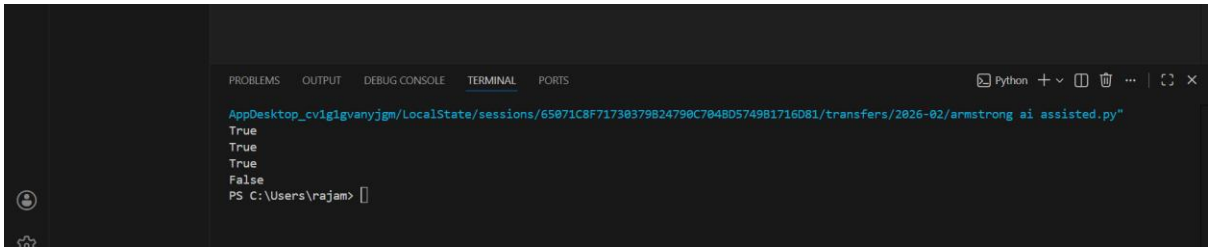
OUTPUT :







OUTPUT :



Explanation :

- **def is_armstrong(number):**: This defines a function named is_armstrong that takes one argument, number, which is expected to be an integer.
- **Docstring**: The triple-quoted string explains what the function does, its arguments (Args), and what it returns (Returns). It also clarifies the definition of an Armstrong number.
- **Input Validation**: if not isinstance(number, int) or number < 0: checks if the input is a non-negative integer. Armstrong numbers are typically defined for non-negative integers. If invalid, it prints a message and returns False.
- **Convert to String**: num_str = str(number) converts the input number to a string. This makes it easy to:
    - num_digits = len(num_str): Get the number of digits.
    - Iterate through each digit.
- **Calculate Sum of Powers**:
    - sum_of_powers = 0 initializes a variable to store the sum.
    - The for loop iterates through each character (digit_char) in the num_str.
    - digit = int(digit_char) converts the character back to an integer.
    - sum_of_powers += digit ** num_digits calculates the digit raised to the power of the total number of digits and adds it to the running sum.
- **Comparison**: Finally, return sum_of_powers == number compares the calculated sum with the original number. If they are equal, the number is an Armstrong number, and the function returns True; otherwise, it returns False.

Comparison Table

Feature	**Google Gemini**	**GitHub Copilot**
Logic Style	Uses string conversion	Uses mathematical operations
Code Structure	Function-based	Inline procedural code

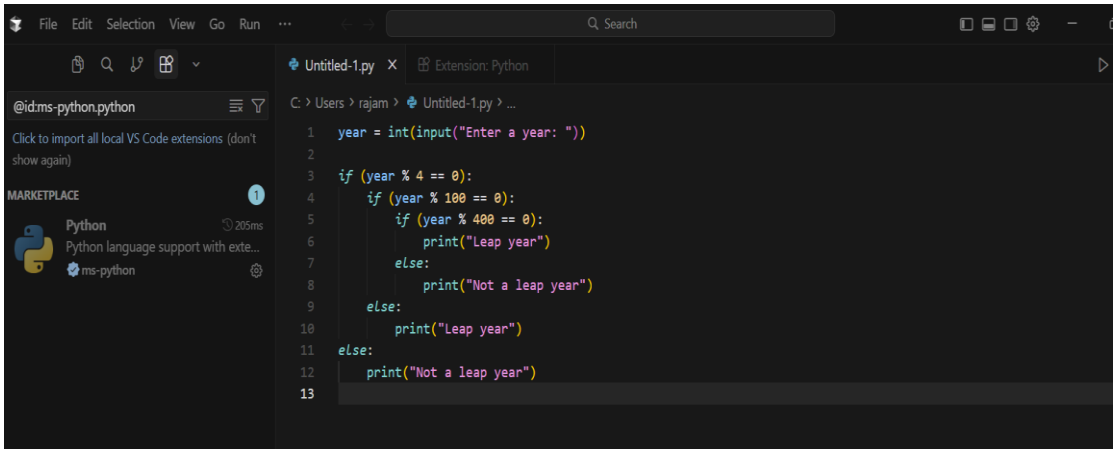
Feature	Google Gemini	GitHub Copilot
Readability	Very clear and beginner-friendly	Slightly complex but efficient
Lines of Code	More	Fewer
Ease of Understanding	High	Medium
Suitability for Beginners	Excellent	Good

Task 3: Leap Year Validation Using Cursor AI

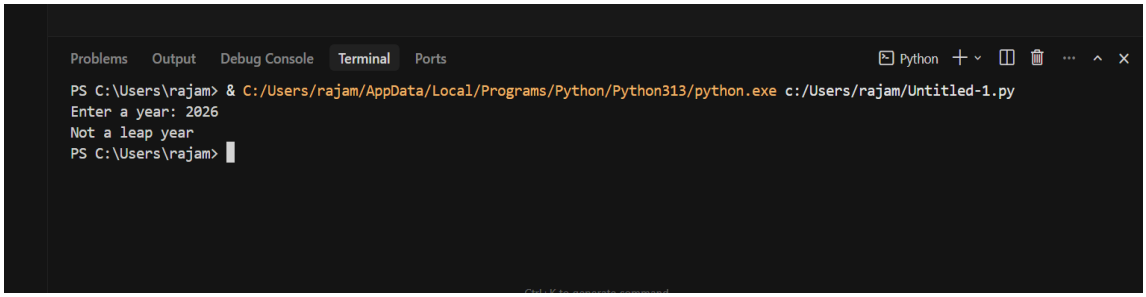
Use Cursor AI to generate a Python program that checks whether a given year is a leap year.
Use at least two different prompts and observe changes in code.

Prompt 1

Write a Python program to check whether a given year is a leap year

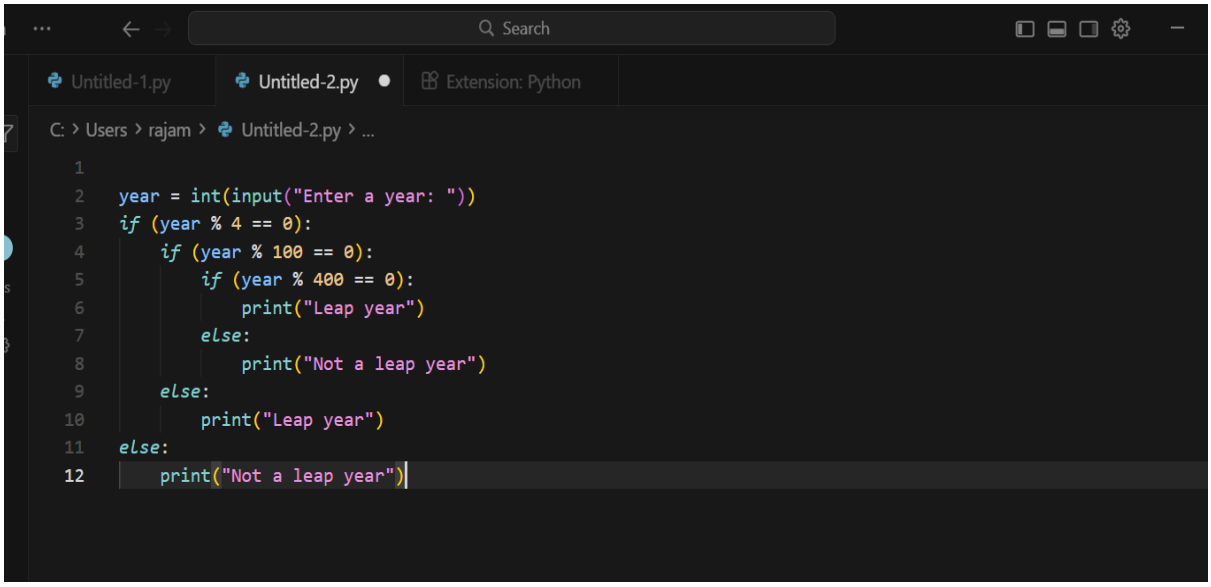


OUTPUT :

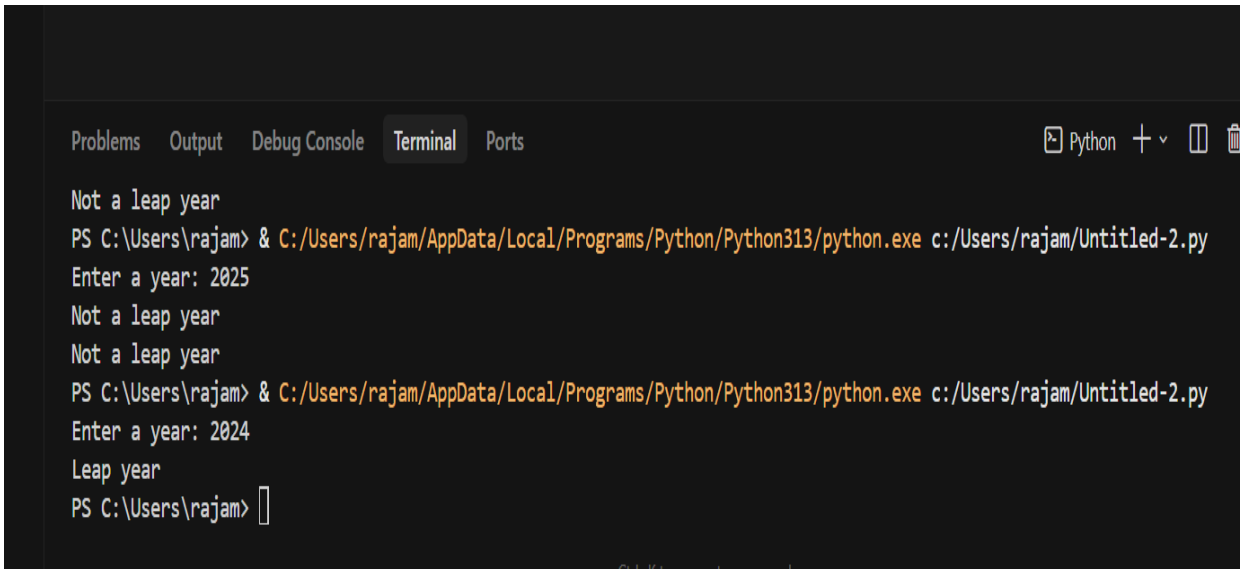


Prompt 2

Write an optimized and user-friendly Python program to validate leap year with proper comments



OUTPUT



Comparison Between the Two Versions

Feature	Version 1	Version 2
Prompt Type	Simple	Optimized
Code Structure	Procedural	Function-based

Feature	Version 1	Version 2
Readability	Good	Very Good
Reusability	Low	High
Comments & Documentation	No	Yes
Suitable for Backend Systems	Medium	High

Conclusion:

Using different prompts in Cursor AI results in improved code quality, better structure, and higher maintainability.

Task 4: Student Logic + AI Refactoring (Odd/Even Sum)

Write a Python program that calculates the **sum of odd and even numbers in a tuple**, then refactor it using any AI tool.

Original code

```
t = (1, 2, 3, 4, 5, 6)

even_sum = 0

odd_sum = 0

for i in t:

    if i % 2 == 0:

        even_sum = even_sum + i

    else:

        odd_sum = odd_sum + i

print("Sum of even numbers:", even_sum)

print("Sum of odd numbers:", odd_sum)
```

Output

```
Sum of even numbers: 12

Sum of odd numbers: 9
```

Refactored Code

```
t = (1, 2, 3, 4, 5, 6)
even_sum = sum(num for num in t if num % 2 == 0)
odd_sum = sum(num for num in t if num % 2 != 0)
print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

Output

```
Sum of even numbers: 12
Sum of odd numbers: 9
```

Explanation of Improvements

Aspect	Original Code	Refactored Code
Code Length	More lines	Fewer lines
Logic Style	Traditional loop	Pythonic (list comprehension)
Readability	Good	Very clear and concise
Performance	Normal	Slightly optimized
Maintainability	Medium	High