ASSIGNMENT - 3.5

Name: M. Sprusheeth Rao

Roll Number : 2303A51206

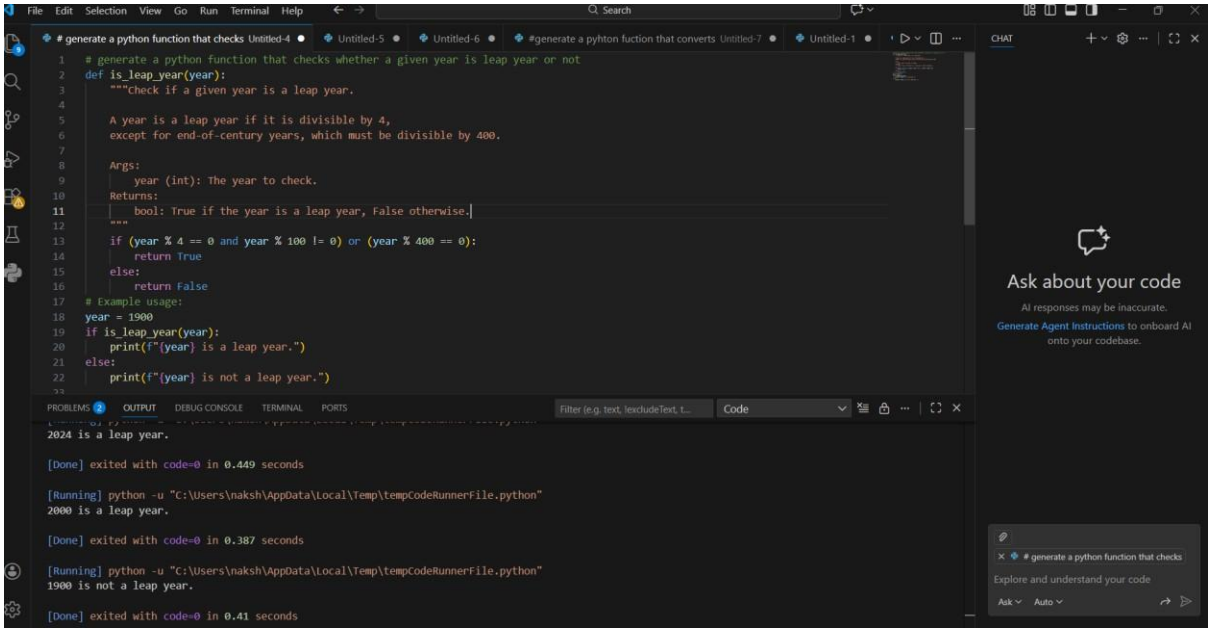
Batch - 04

Question 1: Zero-Shot Prompting (Leap Year Check)

Write a zero-shot prompt to generate a Python function that checks whether a given year is a leap year.

Week2 - Task:

- **Record the AI-generated code.**
- **Test with years like 1900, 2000, 2024.**
- **Identify logical flaws or missing conditions.**



Logical Flaws and Missing Conditions:

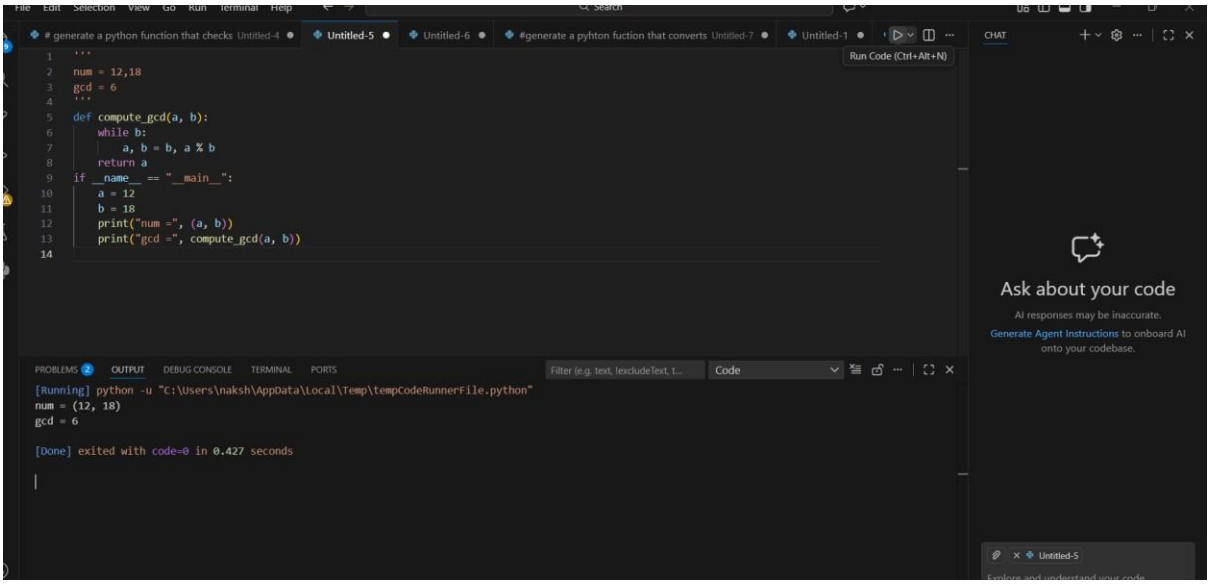
The zero-shot generated leap year function correctly implements the Gregorian leap year rules and produces correct results for years such as 1900, 2000, and 2024. However, it lacks input validation, as it assumes the year is always a valid integer. The function does not handle negative years, zero, or non-integer inputs, and it includes example print statements instead of returning a clean, reusable function. These missing conditions reduce the robustness of the solution.

Question 2: One-Shot Prompting (GCD of Two Numbers) Write a one-shot prompt with one example to generate a Python function that finds the Greatest Common Divisor (GCD) of two numbers.

Example:

Input: 12, 18 → Output: 6 Task:

- **Compare with a zero-shot solution.**
- **Analyze algorithm efficiency**.



Comparison and Efficiency Analysis:

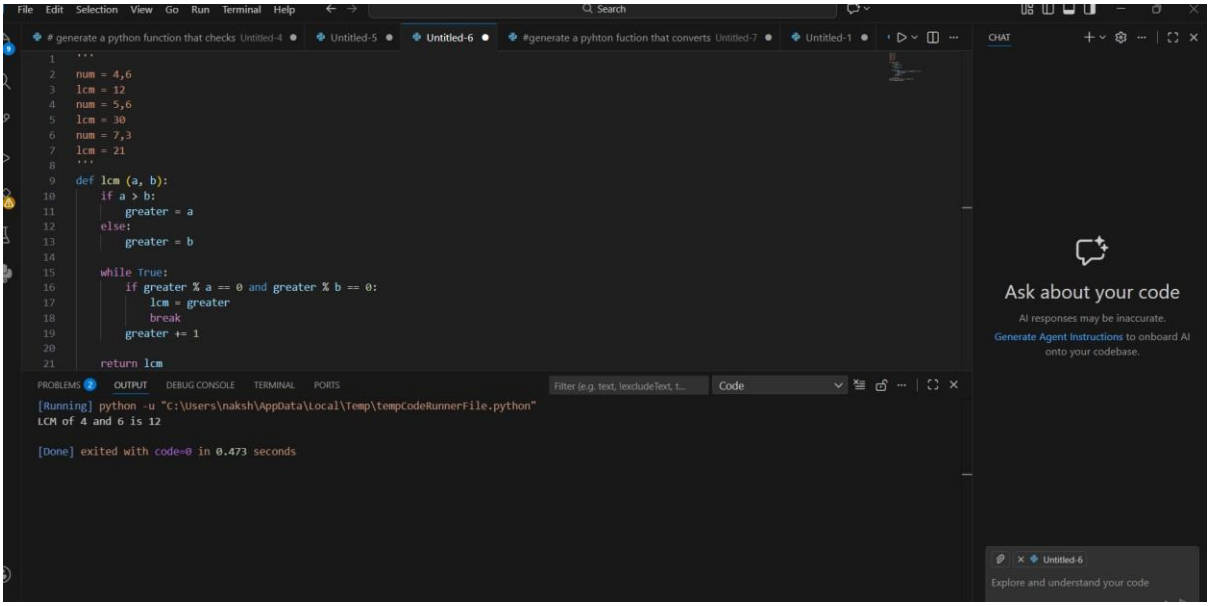
The one-shot prompted solution produces clearer and more efficient code compared to a zero-shot solution. By providing an example, the AI selects the Euclidean algorithm, which significantly improves performance. The algorithm runs in logarithmic time O(log n), making it efficient even for large inputs, whereas a zero-shot solution may use a slower brute-force method.

Question 3: Few-Shot Prompting (LCM Calculation)

Write a few-shot prompt with multiple examples to generate a Python function that computes the Least Common Multiple (LCM).

Examples:

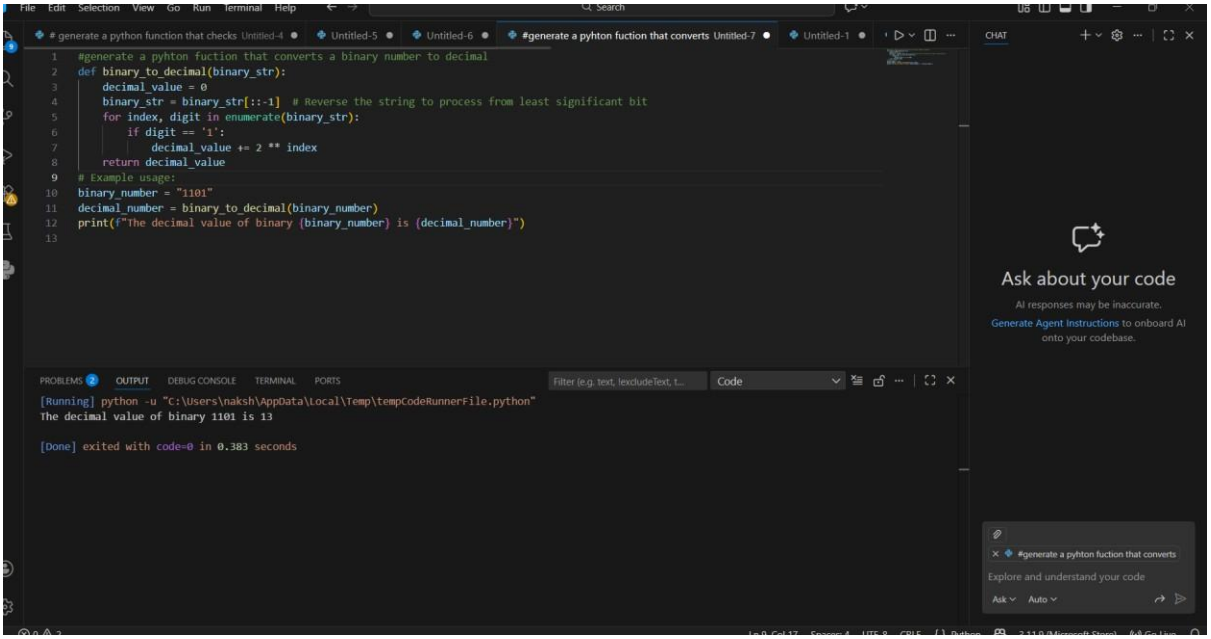
- **Input: 4, 6 → Output: 12**
- **Input: 5, 10 → Output: 10**
- **Input: 7, 3 → Output: 21 Task:**
- **Examine how examples guide formula selection.**
- **Test edge cases.**



Question 4: Zero-Shot Prompting (Binary to Decimal Conversion) Write a zero-shot prompt to generate a Python function that converts a binary number to decimal.

Task:

- **Test with valid and invalid binary inputs.**
- **Identify missing validation logic.**



Missing Validation Logic:
The zero-shot generated function correctly converts valid binary numbers to decimal but lacks proper input validation. It does not check whether the input contains only binary digits (0 and 1), does not handle empty strings, and fails to raise errors for invalid inputs. As a result, incorrect or non-binary inputs may produce misleading outputs, reducing the robustness of the solution.

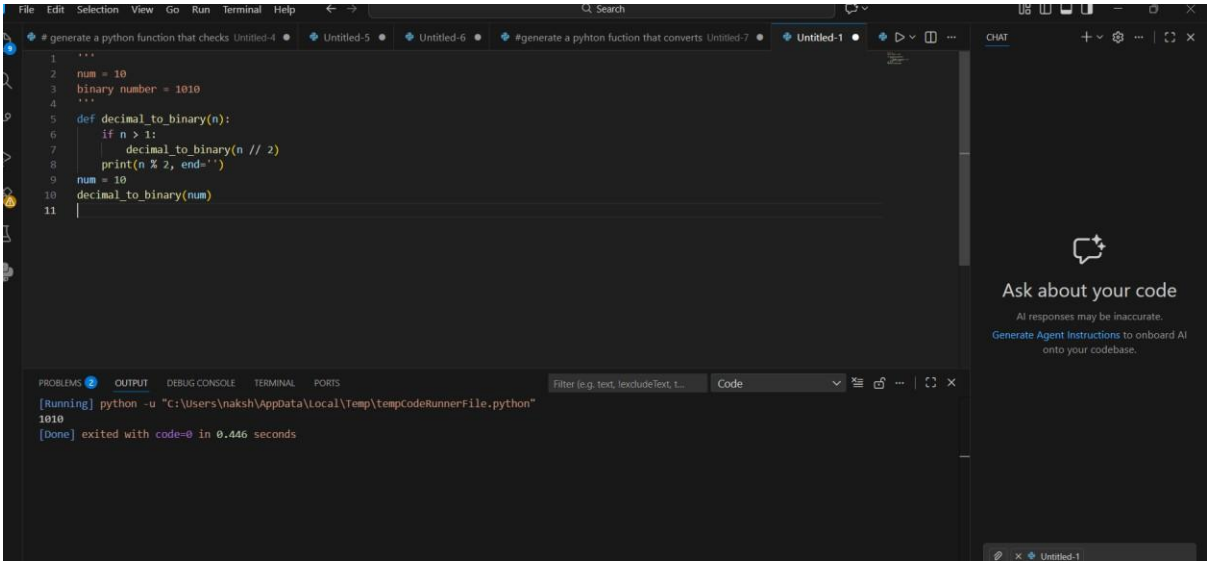
Question 5: One-Shot Prompting (Decimal to Binary Conversion)

Write a one-shot prompt with an example to generate a Python function that converts a decimal number to binary.

Example:

Input: 10 → Output: 1010 Task:

- **Compare clarity with zero-shot output.**
- **Analyze handling of zero and negative numbers.**



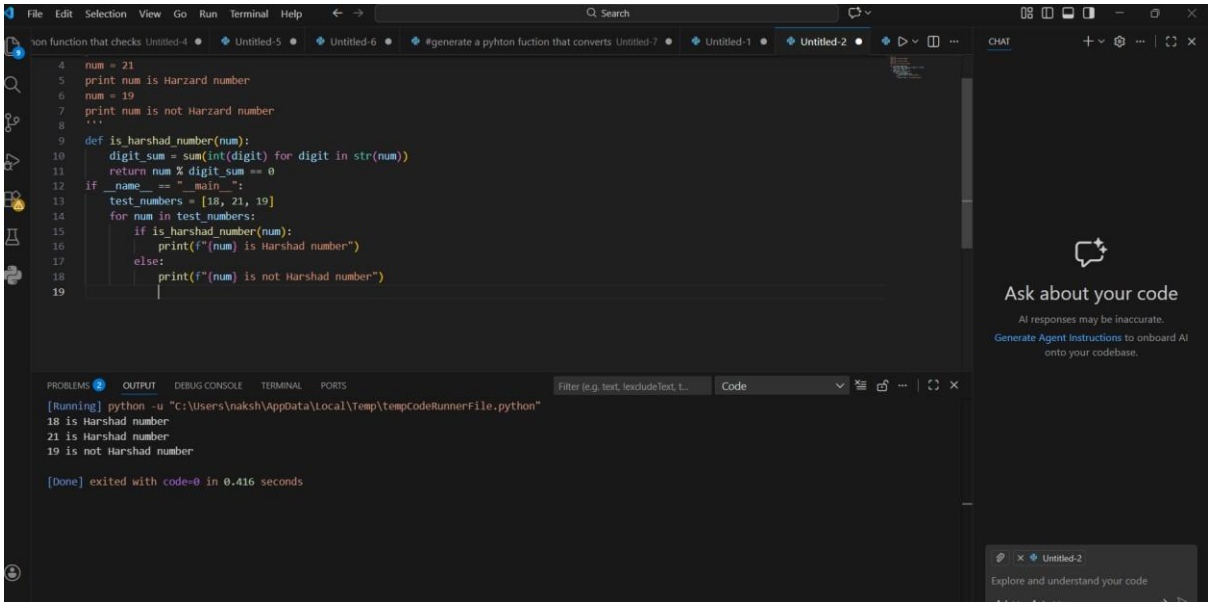
Comparison of Clarity:

The one-shot prompting output is more understandable than the zero-shot output since the given example explicitly states the desired input-output behavior. This assists the AI in organizing the logic properly, preventing ambiguity, and creating more readable and well-documented code. Conversely, zero-shot prompting tends to produce shorter and less descriptive code without edge-case coverage.

Question 6: Few-Shot Prompting (Harshad Number Check) Write a few-shot prompt to generate a Python function that checks whether a number is a Harshad (Niven) number.

Examples:

- **Input: 18 → Output: Harshad Number**
- **Input: 21 → Output: Harshad Number • Input: 19 → Output: Not a Harshad Number Task:**
- **Test boundary conditions.**
- **Evaluate robustness**



Robustness Evaluation:

The Harshad number function is robust if it correctly handles valid inputs, boundary cases such as 0 and single-digit numbers, and prevents division by zero errors. A robust implementation also explicitly handles negative values and restricts inputs to positive integers, ensuring consistent and reliable behavior across all test cases.