Assignment-2

Name: Sushma Purella

H. No:2303A52188

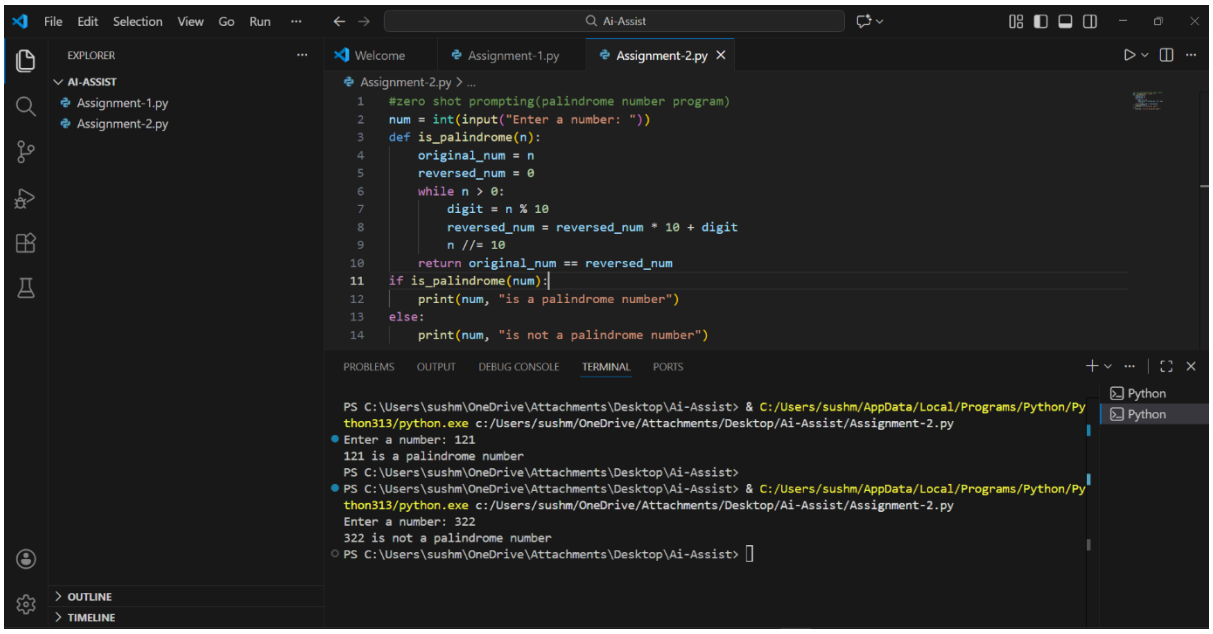
Batch:44

Question 1: Zero-Shot Prompting (Palindrome Number Program)

Prompt:

Write a Python program to check whether a given number is a palindrome or not and display the result.

Code:



Observation:

The model is able to generate a working palindrome program without any prior examples.

The generated code correctly reverses the number and compares it with the original.

However, the prompt does not specify input validation or optimization, so the solution may not handle edge cases (like negative numbers) efficiently.

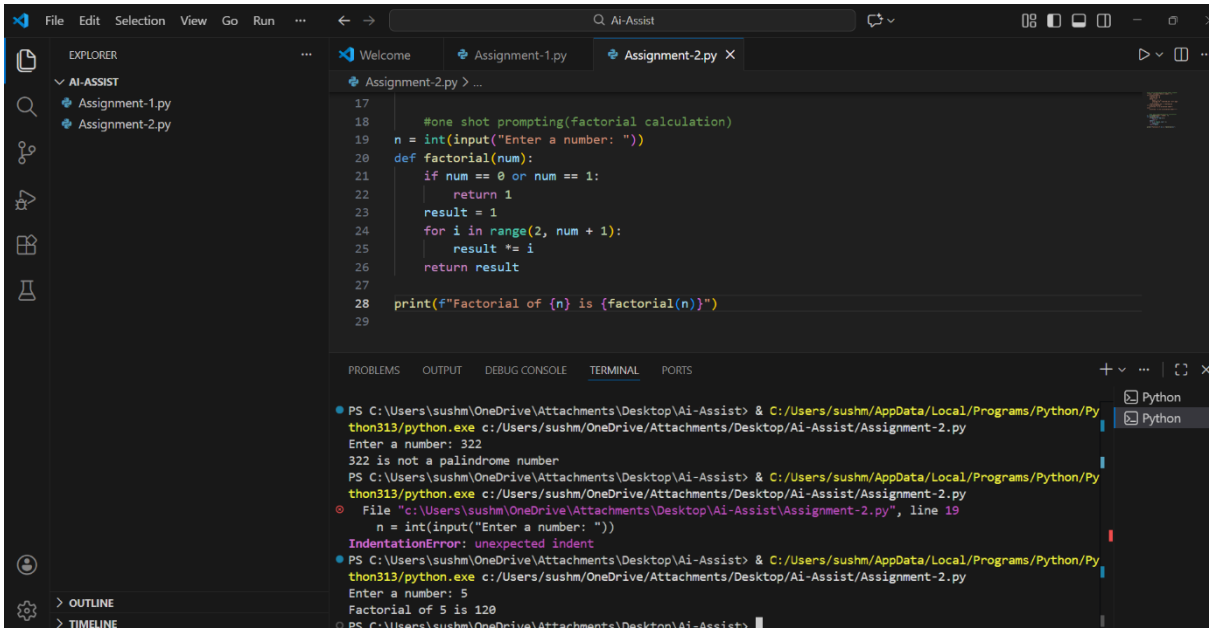
This shows that zero-shot prompting works for simple problems, but for better control and optimized output, structured or context-managed prompts are preferred.

Question 2: One-Shot Prompting (Factorial Calculation)

Prompt:

Write a Python program to calculate the factorial of a given number

Code:



Observation:

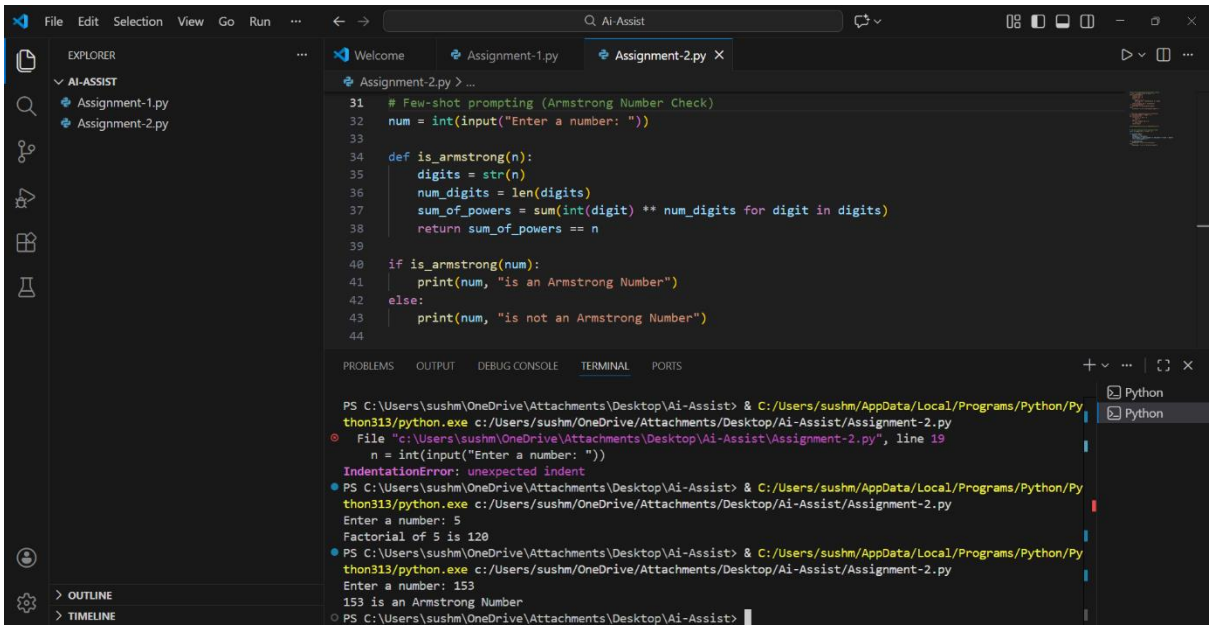
- The provided example helps the model understand the expected logic and output format.
- The generated code correctly handles base cases such as 0 and 1.
- The program uses an iterative approach, which is efficient and easy to understand.
- Compared to zero-shot prompting, one-shot prompting produces more accurate and structured output.

Question 3: Few-Shot Prompting (Armstrong Number Check)

Prompt:

Write a Python program to check whether a given number is an Armstrong Number

Code:



Observation:

- Providing multiple examples helps the model clearly understand the definition of an Armstrong number.
- The generated program correctly computes the sum of each digit raised to the power of the number of digits.
- The output format closely matches the examples.
- Few-shot prompting produces more accurate and consistent results than zero-shot and one-shot prompting.

Question 4: Context-Managed Prompting (Optimized Number Classification)

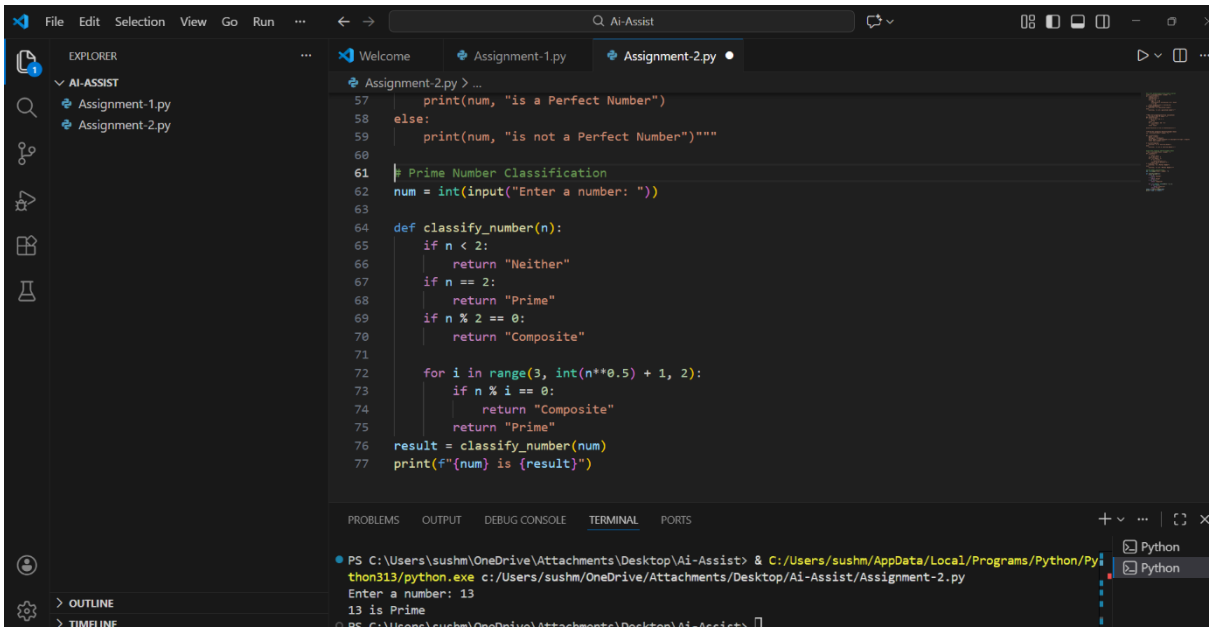
Prompt:

You are an experienced Python developer.
Generate an optimized Python program that classifies a given integer as Prime, Composite, or Neither.

Instructions & Constraints:

- Accept only valid integer input.
- Handle special cases such as negative numbers, 0, and 1 correctly.
- Use an efficient algorithm (check divisibility only up to √n).
- Avoid unnecessary loops or redundant checks.
- The program must clearly print whether the number is *Prime*, *Composite*, or *Neither*.
- Do not include explanations or comments—only clean, executable Python code.

Code:



Observation:

- The program efficiently classifies a number as Prime, Composite, or Neither.
- Special cases such as numbers less than 2 are correctly handled as Neither.

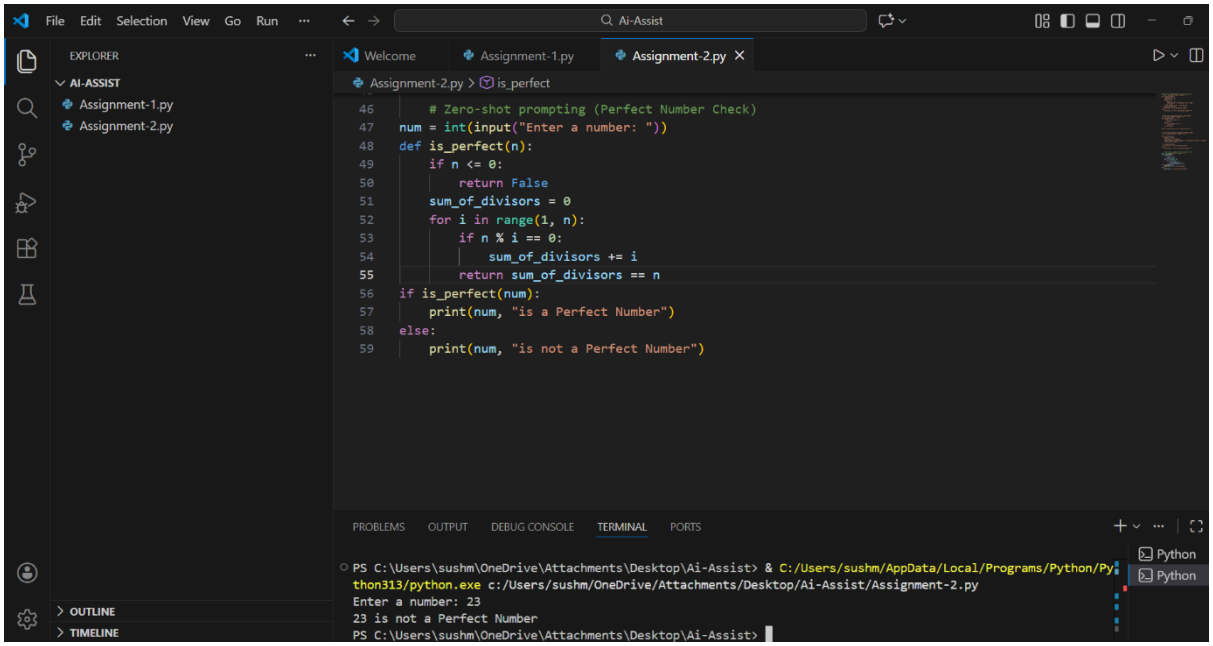
- The logic optimizes performance by checking divisibility only up to the square root of the number.
- Even numbers greater than 2 are immediately classified as Composite, reducing unnecessary iterations.
- This approach is more efficient and accurate compared to basic prime-checking methods that test all numbers up to n.

Question 5: Zero-Shot Prompting (Perfect Number Check)

Prompt:

Write a Python program to check whether a given number is a Perfect Number or not and display the result.

Code:



Observation:

- The program checks whether the sum of all proper divisors of a number is equal to the number itself.
- It correctly handles invalid inputs such as zero or negative numbers.
- The logic is simple and easy to understand, making it suitable for basic problem understanding.

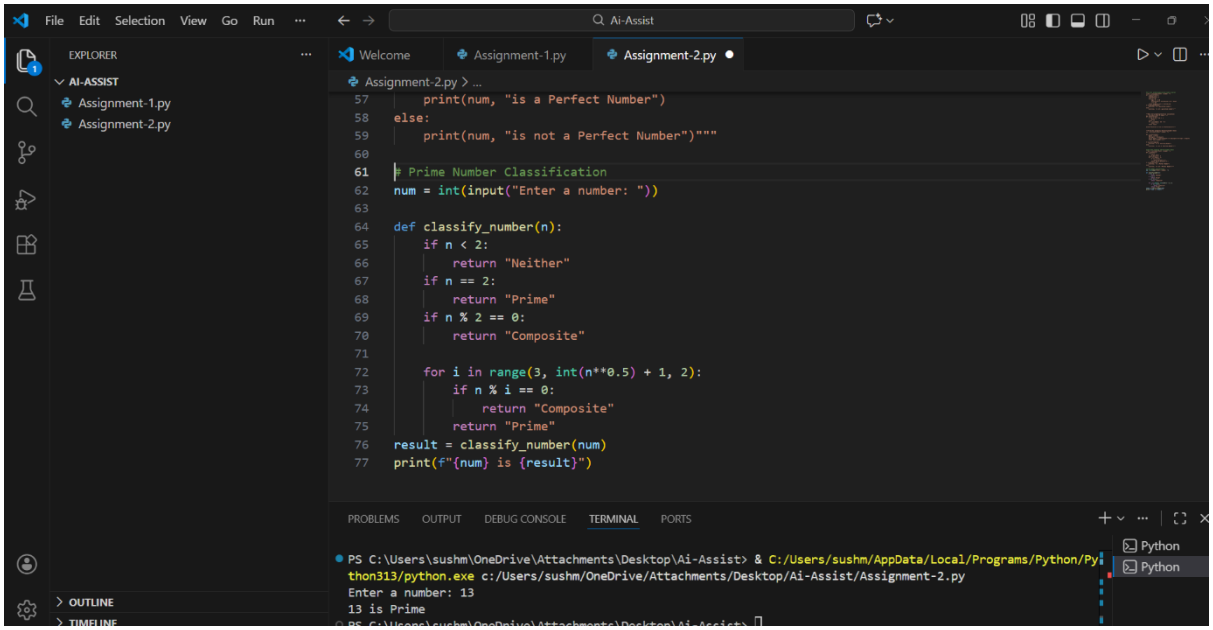
- However, since no optimization constraints were specified in the prompt, the program uses a brute-force approach, which may be inefficient for large numbers.
- This demonstrates that zero-shot prompting works well for simple tasks but may lack efficiency and control.

Question 6: Few-Shot Prompting (Even or Odd Classification with Validation)

Prompt:

Write a Python program to classify a given number as Prime, Composite, or Neither and display the result.

Code:



Observation:

- The program correctly classifies numbers less than 2 as Neither.
- It identifies 2 as a prime number and eliminates even numbers early to improve efficiency.
- The divisibility check is optimized by iterating only up to the square root of the number and skipping even values.
- This approach reduces time complexity compared to naive methods.

- The logic ensures accurate classification with improved performance.