Assiginment -3.1

Hall tictket no:2303A510A6

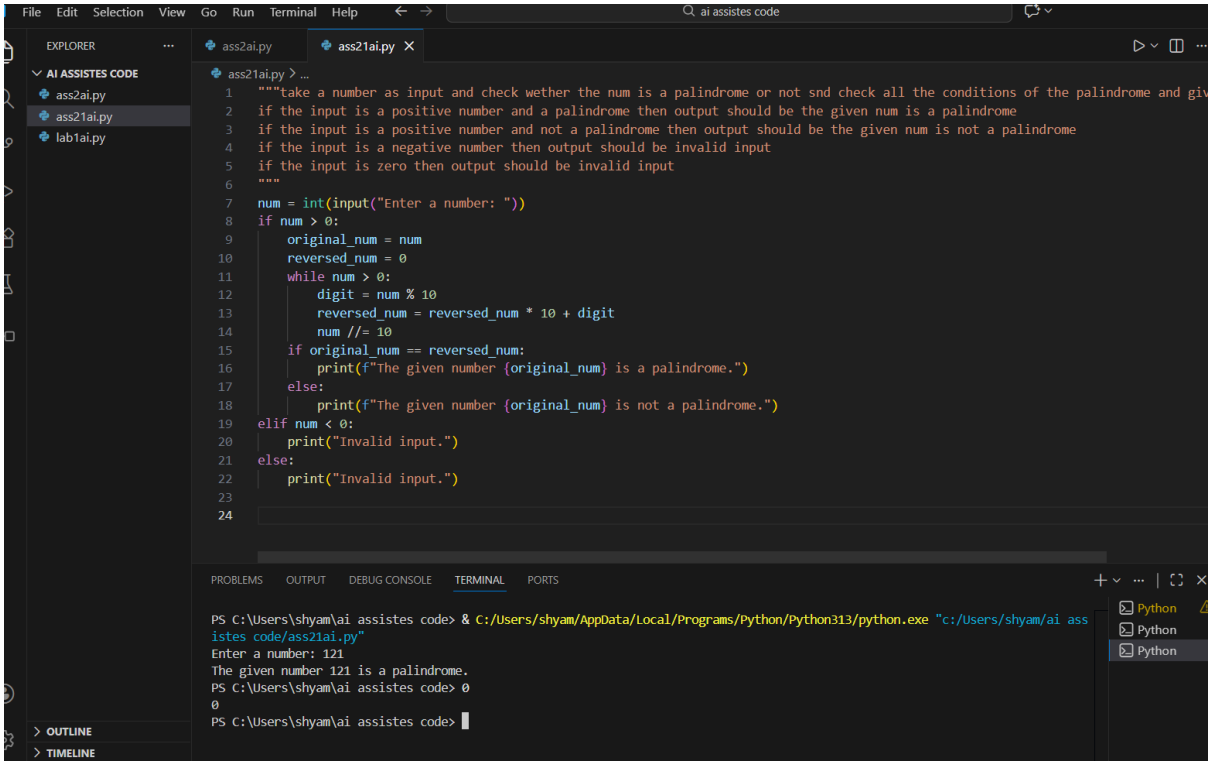
Batch no:02

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

Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.

Task:

- **Record the AI-generated code.**
- **Test the code with multiple inputs.**
- **Identify any logical errors or missing edge-case handling.**



write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

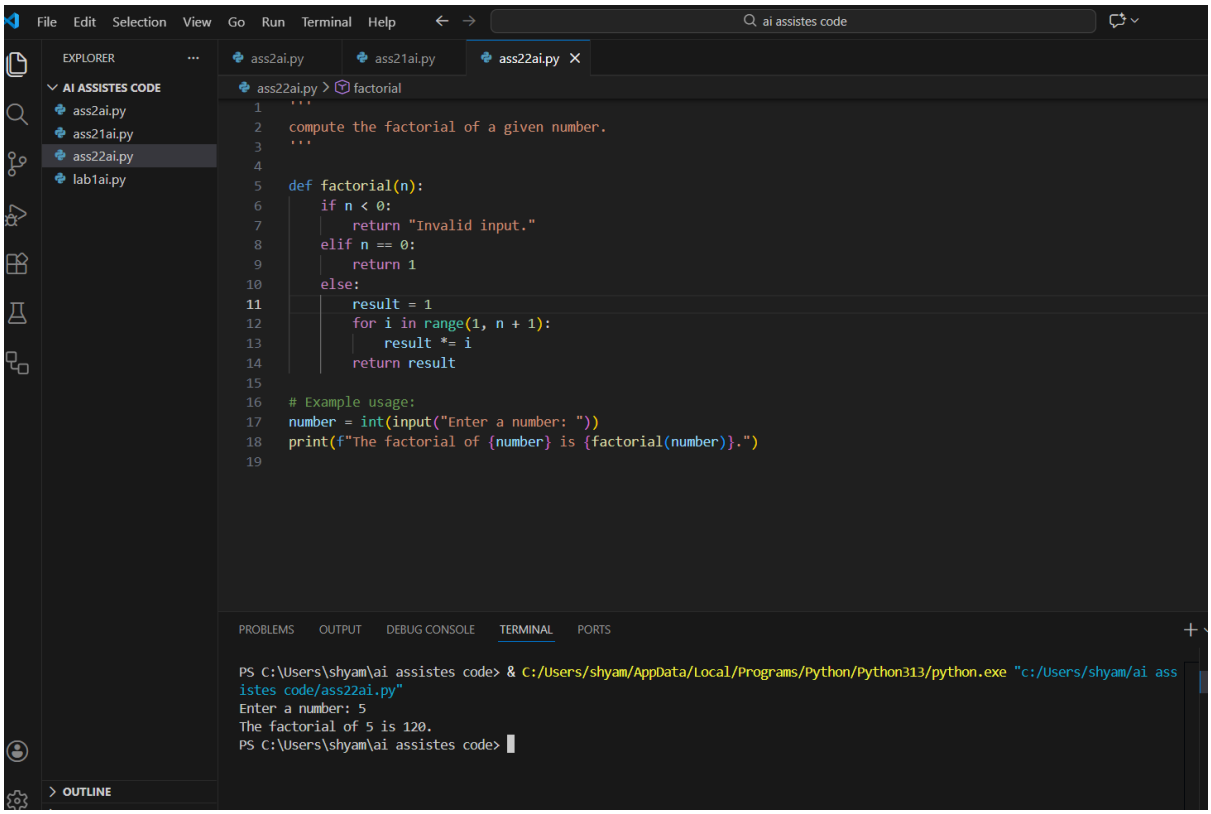
Example:

Input: 5 → Output: 120

Task:

- **Compare the generated code with a zero-shot solution.**

- **Examine improvements in clarity and correctness.**



Aspect	One-Shot Prompting	Zero-Shot Prompting
Guidance	Includes example (5 → 120)	No example provided
Clarity	More explicit task understanding	Relies on model's assumption
Approach	Iterative solution	Recursive solution
Readability	Easier for beginners	Slightly more complex
Correctness	Correct for non-negative integers	Correct but risk of recursion depth

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

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.

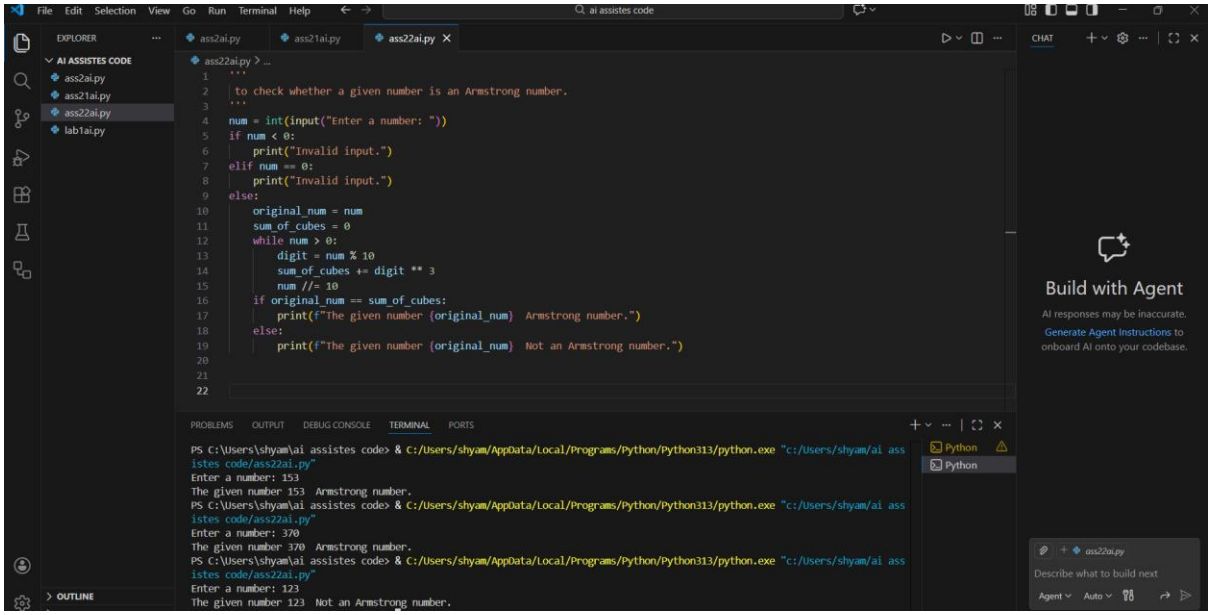
Examples:

- **Input: 153 → Output: Armstrong Number**
- **Input: 370 → Output: Armstrong Number**
- **Input: 123 → Output: Not an Armstrong Number**

Task:

• Analyze how multiple examples influence code structure and accuracy.

• Test the function with boundary values and invalid inputs.



Influence of Multiple Examples on Code Structure and Accuracy

- Multiple examples clearly demonstrate **both positive and negative cases**, reducing ambiguity.
- The AI correctly identifies that:
    - Each digit must be raised to the **number of digits**.
    - The sum must be compared with the original number.

Few-shot prompting helps the AI infer:

- The definition of an Armstrong number
- The expected output wording
- The correct handling of digit extraction and exponentiation

(Optional Extension)

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

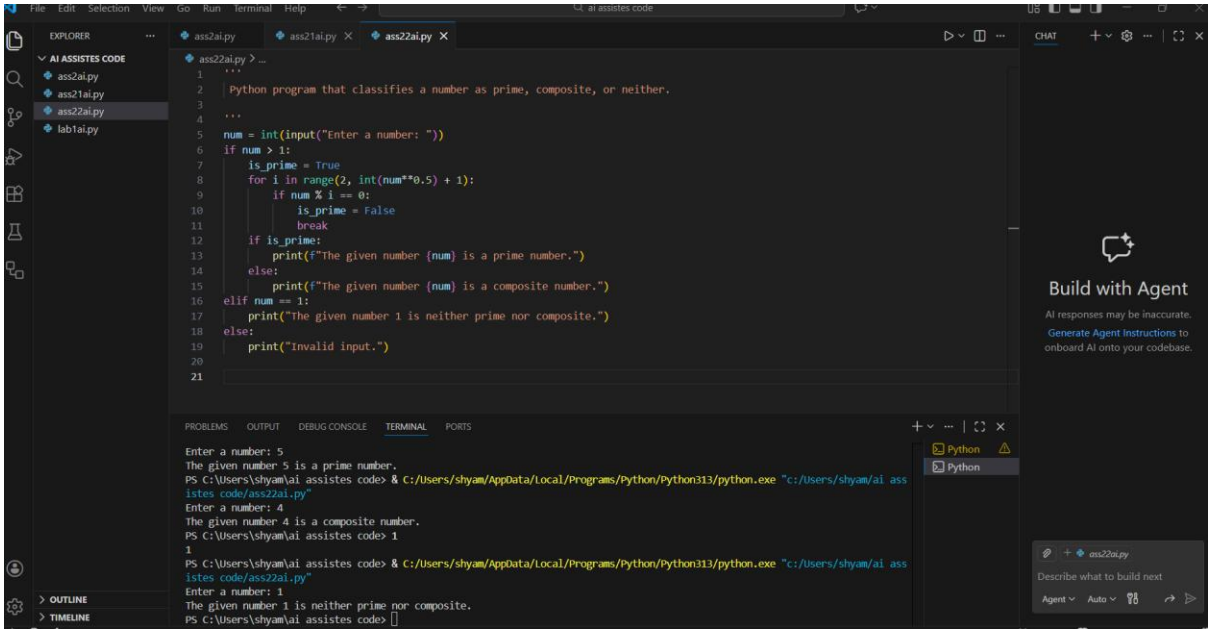
Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither.

ask:

• Ensure proper input validation.

• Optimize the logic for efficiency.

• Compare the output with earlier prompting strategies.



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

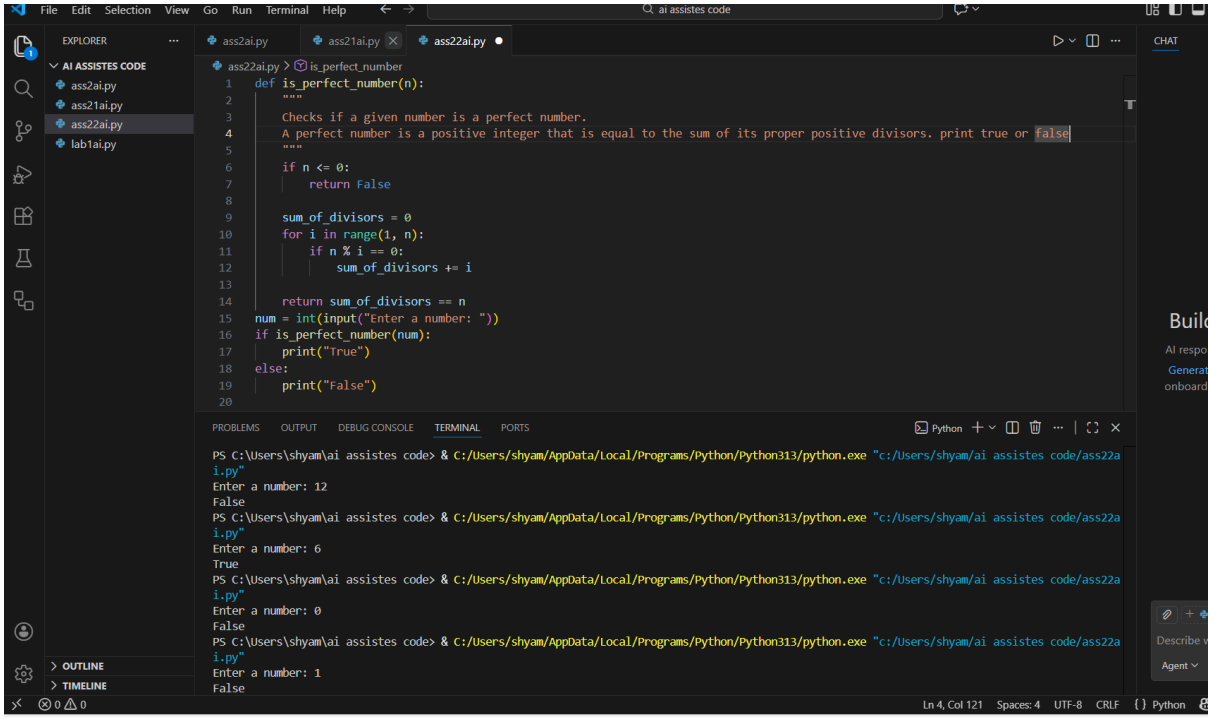
Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

• Record the AI-generated code.

• Test the program with multiple inputs.

• Identify any missing conditions or inefficiencies in the logic.



Missing Conditions and Inefficiencies

- No validation for negative numbers or zero.
- Loop runs up to n-1, which is inefficient for large values.
- Function does not handle non-integer inputs.

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

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

• Input: 8 → Output: Even

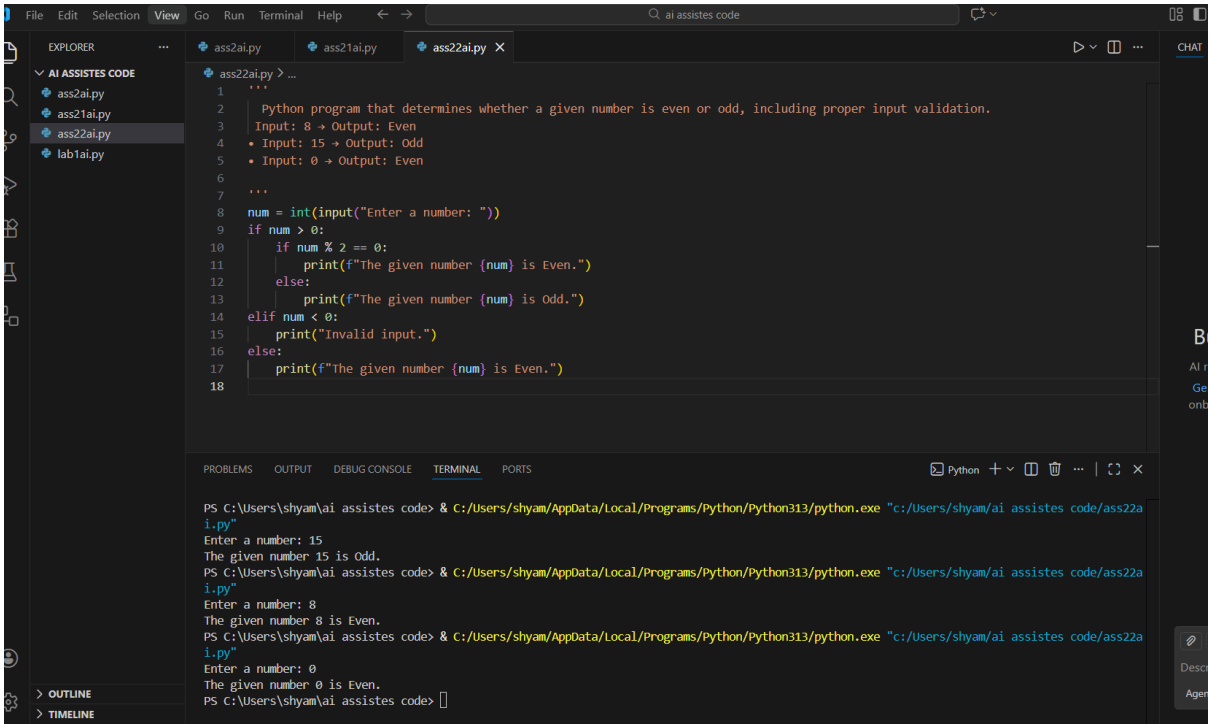
• Input: 15 → Output: Odd

• Input: 0 → Output: Even

Task:

• Analyze how examples improve input handling and output clarity.

• Test the program with negative numbers and non-integer inputs.



Analysis: Effect of Examples on Input Handling and Output Clarity

- Examples guide the program to return only “Even” or “Odd”.
- Including 0 → Even ensures correct handling of zero.
- Input validation is added to prevent non-integer inputs.