ASSENMENT-1

NAME: B.MANUDEEP REDDY

H.T.N : 2303A52380

BATCH : 43

TASK 1 :

AI-Generated Logic Without Modularization (Prime Number Check Without Functions)

❖ Scenario

➢ You are developing a basic validation script for a numerical learning application.

❖ Task Description: Use GitHub Copilot to generate a Python program that:

➢ Checks whether a given number is prime

➢ Accepts user input

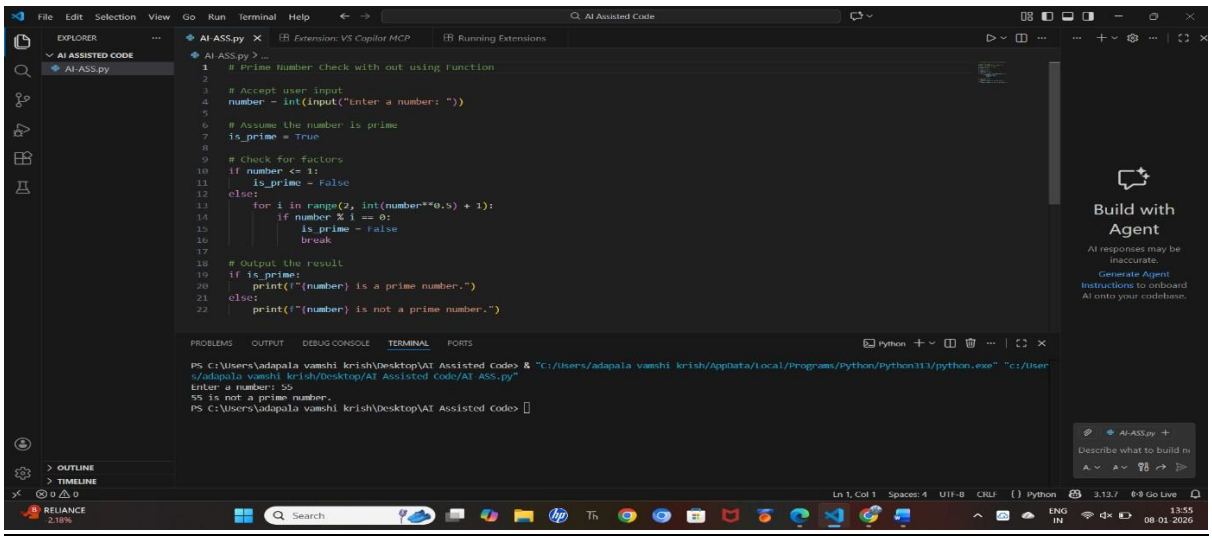
➢ Implements logic directly in the main code

➢ Does not use any user-defined functions

PROMPT :

AI-Generated Logic Without Modularization (Prime Number Check With out Functions) **CODE**

:



OBSERVATION :

The program demonstrates that a prime-checking algorithm can be implemented in a simple, linear, non-modular script while still being efficient and logically clear.

TASK 2 :

Efficiency & Logic Optimization (Cleanup)

❖ Scenario

The script must handle larger input values efficiently.

❖ Task Description

Review the Copilot-generated code from Task 1 and improve it by:

➢ Reducing unnecessary iterations

➢ Optimizing the loop range (e.g., early termination)

➢ Improving readability

➢ Use Copilot prompts like:

▪ "Optimize prime number checking logic"

▪ "Improve efficiency of this code" Hint:

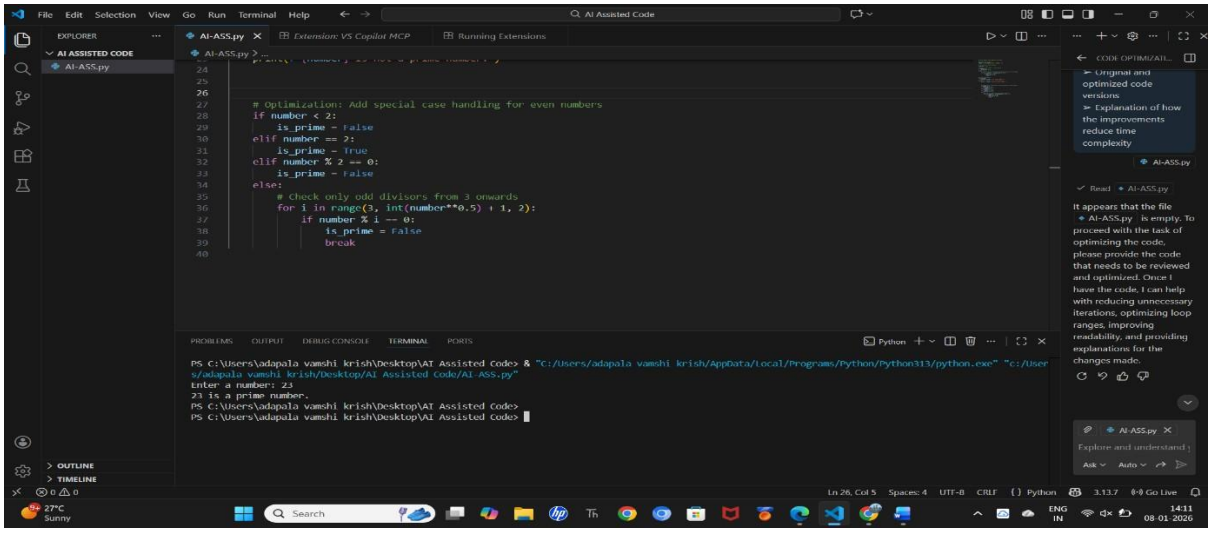
Prompt Copilot with phrases like

"optimize this code", "simplify logic", or "make it more readable"

PROMPT :

Prime Number Check Without Functions & Efficiency & Logic Optimization (Cleanup) **CODE**

:



OBSERVATION :

The optimized version demonstrates how algorithmic improvements and smart looping strategies can drastically improve performance while also making the code cleaner, faster, and more professional.

TASK – 3

Modular Design Using AI Assistance (Prime Number Check Using Functions)

❖ Scenario

The prime-checking logic will be reused across multiple modules.

❖ Task Description

Use GitHub Copilot to generate a function-based Python program that:

➢ Uses a user-defined function to check primality

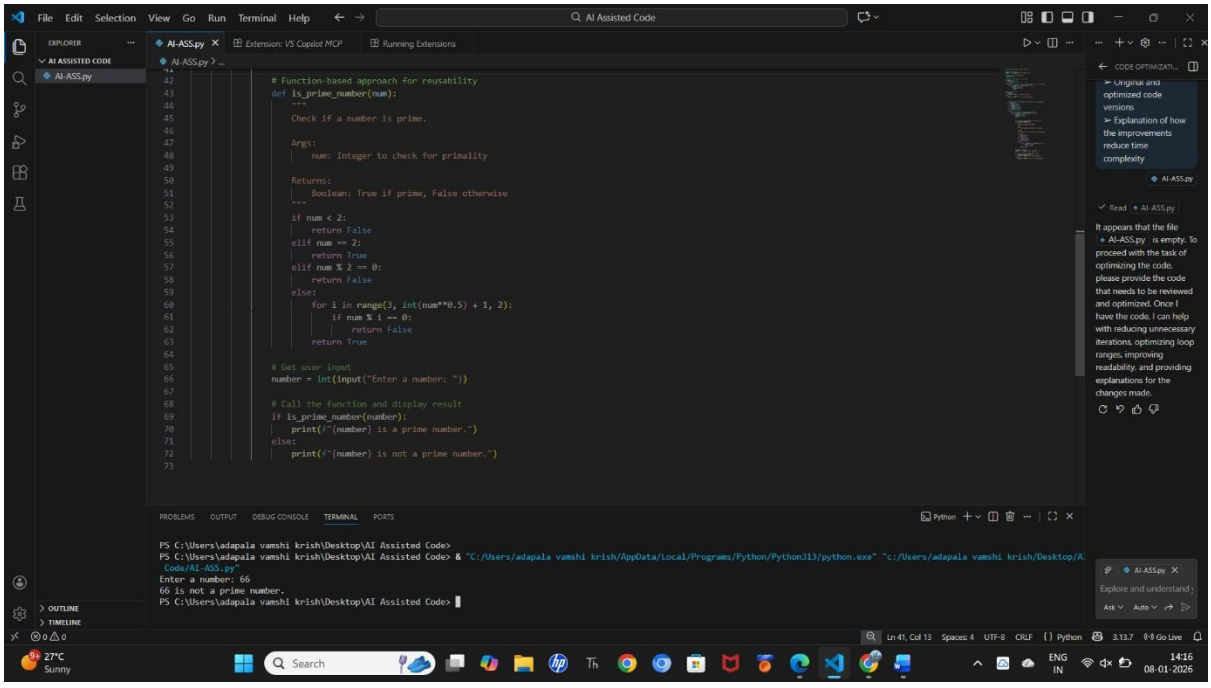
➢ Returns a Boolean value

➢ Includes meaningful comments (AI-assisted)

PROMPT :

Modular Design Using AI Assistance (Prime Number Check Using Functions **CODE**

:



OBSERVATION :

It shows using functions improves code organization, reusability, maintainability, and professionalism while keeping the logic efficient and clean.

TASK-4

Comparative Analysis –With vs Without Functions

❖ Scenario: You are participating in a technical review discussion.

❖ Task Description: Compare the Copilot-generated programs:

➢ Without functions (Task 1)

➢ With functions (Task 3)

➢ Analyze them based on:

➢ Code clarity

➢ Reusability

➢ Debugging ease

➢ Suitability for large-scale applications

CODE :

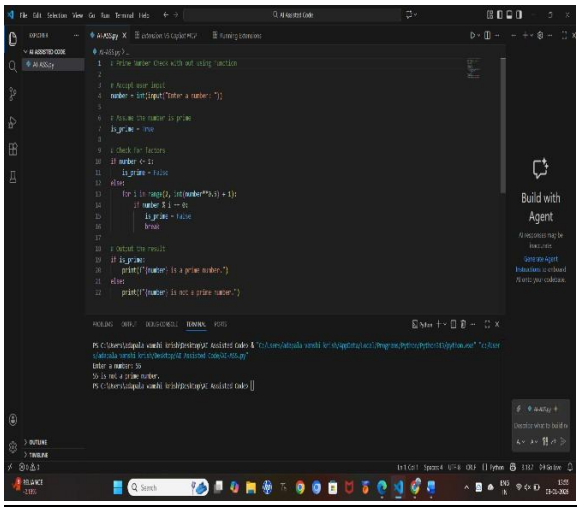


fig: TASK - 1

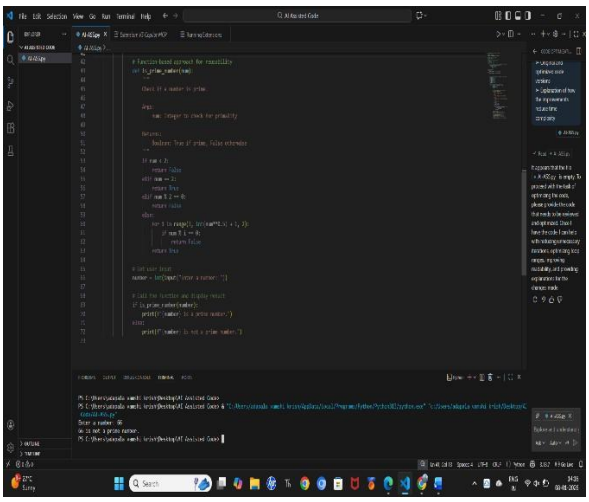


fig: TASK-2

OBSERVATION :

Task 1 demonstrates basic procedural programming, while Task 3 demonstrates structured, modular, and professional programming.
Using functions makes the code cleaner, reusable, scalable, and easier to maintain.

Feature	Task 1: Without Functions	Task 3: With Functions

Program Structure	**Entire logic is written in the main script**	**Logic is separated into a userdefined function**
Modularity	**Not modular**	**Modular design**
Reusability	**Cannot easily reuse the logic**	**Function can be reused in multiple programs**
Code Organization	**Mixed input, logic, and output in one place**	**Clear separation: main handles I/O, function handles logic**
Readability	**Acceptable for small programs, but gets messy for large ones**	**Much cleaner and easier to understand**
Maintainability	**Any change requires editing the whole script**	**Only the function needs to be updated**
Testing	**Cannot test logic independently**	**Function can be tested separately**
Scalability	**Not suitable for large projects**	**Suitable for large and multi-module projects**
Return Value	**Uses flags/print statements**	**Returns a Boolean (True / False)**
Professional Coding Style	**Basic / beginner style**	**Industry-standard structured approach**

TASK – 5

AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking)

❖ Scenario: Your mentor wants to evaluate how AI handles alternative logical strategies.

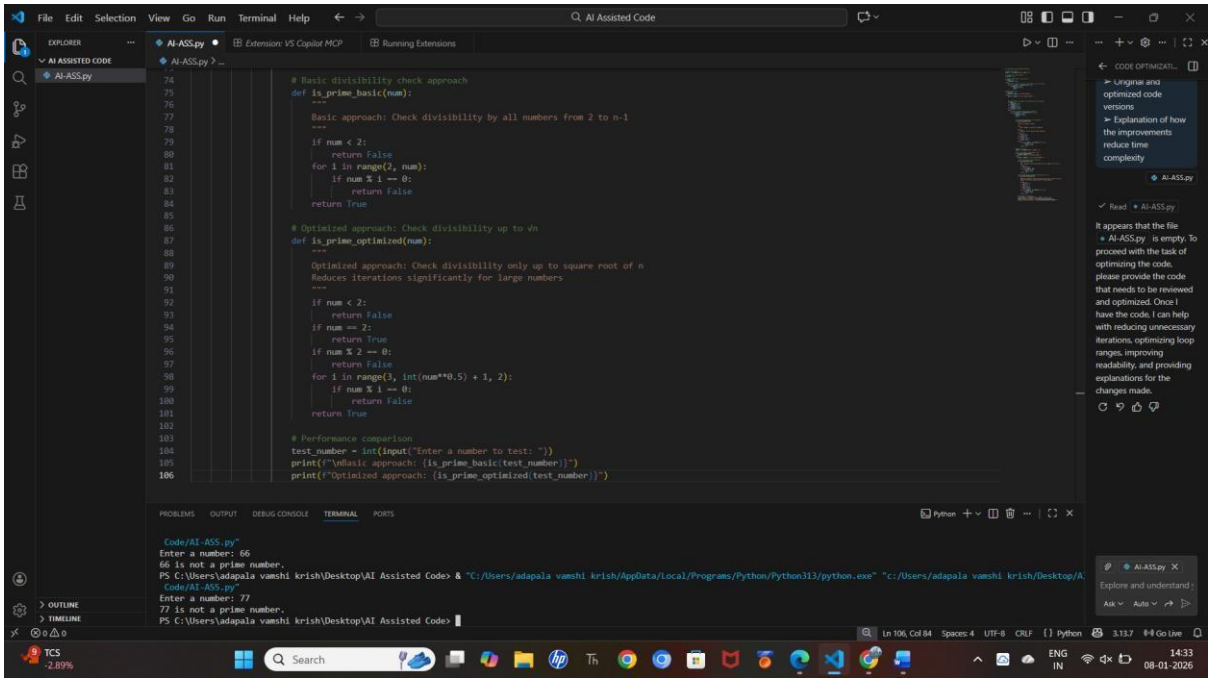
❖ Task Description: Prompt GitHub Copilot to generate:

➢ A basic divisibility check approach

➢ An optimized approach (e.g., checking up to √n)

PROMPT :

AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking) **CODE :**



OBSERVATION :

It clearly shows how different logical strategies for the same problem lead to huge differences in efficiency, and how AI can assist in generating and comparing both naive and optimized solutions.