Name : N. Soumya

H NO:2303A51869

Batch-13

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

❖ **Expected Output**

➢ Correct prime / non-prime result

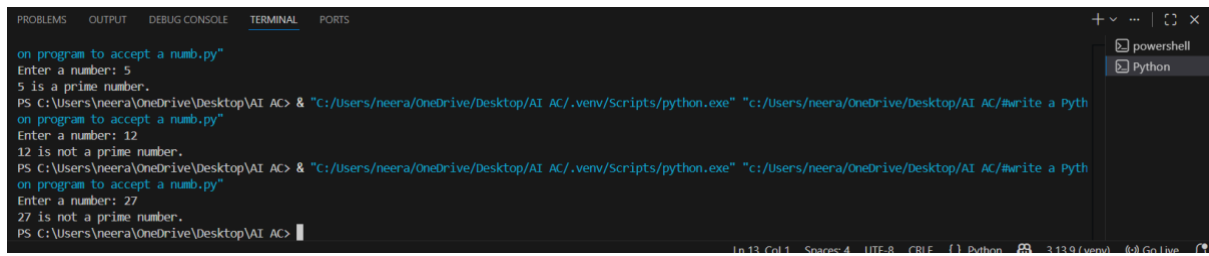➢ Screenshots showing Copilot-generated code suggestions

➢ Sample inputs and outputs

Prompt:

 # write a Python program to accept a number from the user and check whether it is a prime number or not, using logic only in the main code (no user-defined functions), which accepts user input and display the result.

#Code

```python
num = int(input("Enter a number: "))
if num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            print(f"{num} is not a prime number.")
            break
    else:
        print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

#Output:

#Explanation:

The program takes a number as input from the user.
It checks whether the number can be divided by any number other than 1 and itself.
If it is divisible, the number is not a prime number .If it is not divisible by any number, it is a prime number.

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"
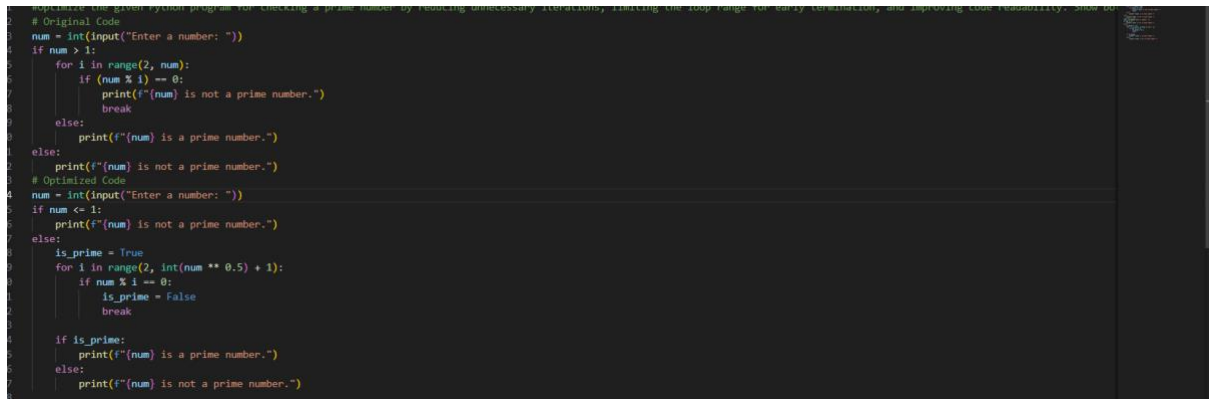
❖ **Expected Output**

➢ Original and optimized code versions

➢ Explanation of how the improvements reduce time complexity

#Optimize the given Python program for checking a prime number by reducing unnecessary iterations, limiting the loop range for early termination, and improving code readability. Show both the original code and the optimized code, and explain how the changes improve efficiency and reduce time complexity.
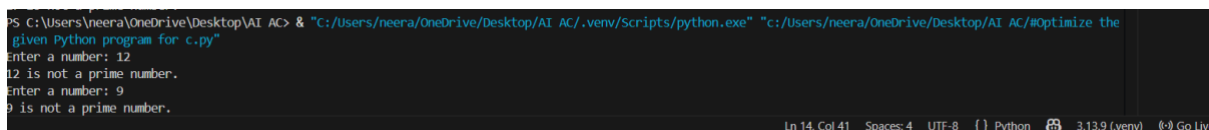
#Code

```
#Optimize the given Python program for checking a prime number by reducing unnecessary iterations, limiting the loop range for early termination, and improving code readability. Show bo
# Original Code
num = int(input("Enter a number: "))
if num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            print(f"{num} is not a prime number.")
            break
    else:
        print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
# Optimized Code
num = int(input("Enter a number: "))
if num <= 1:
    print(f"{num} is not a prime number.")
else:
    is_prime = True
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

    if is_prime:
        print(f"{num} is a prime number.")
    else:
        print(f"{num} is not a prime number.")
```

#OUTPUT:

```
PS C:\Users\neera\OneDrive\Desktop\AI AC> & "C:/Users/neera/OneDrive/Desktop/AI AC/.venv/Scripts/python.exe" "c:/Users/neera/OneDrive/Desktop/AI AC/#Optimize the
 given Python program for c.py"
Enter a number: 12
12 is not a prime number.
Enter a number: 9
9 is not a prime number.
```
Ln 14, Col 41   Spaces: 4   UTF-8   {} Python   3.13.9 (.venv)   Go Live

#EXPLANATION:

The optimized code reduces unnecessary iterations by checking divisibility only up to the square root of the number instead of all numbers up to n .Early termination using break stops the loop as soon as a factor is found, saving time .Using a boolean variable improves code readability and clarity.

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)

❖ **Expected Output**

➢ Correctly working prime-checking function

➢ Screenshots documenting Copilot's function generation

➢ Sample test cases and outputs

#Prompt

#Write a Python program using a user-defined function to check whether a given number is a prime number. The function should return a Boolean value (True or False). Accept input from the user, call the function, and display the result. Include clear and meaningful comments to explain the logic

#Code

```python
#Write a Python program using a user-defined function to check whether a given number is a prime number. The function should return a Boolean value
def is_prime(num):
    # Check if the number is less than or equal to 1
    if num <= 1:
        return False
    # Check for factors from 2 to the square root of the number
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True
# Accept input from the user
number = int(input("Enter a number: "))
# Call the function and display the result
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

#output

```
PS C:\Users\neera\OneDrive\Desktop\AI AC> & "C:/Users/neera/OneDrive/Desktop/AI AC/.venv/Scripts/python.exe" "c:/Users/neera/OneDrive/Desktop/AI AC/##W
rite a Python program using a user-de.py"
Enter a number: 27
27 is not a prime number.
PS C:\Users\neera\OneDrive\Desktop\AI AC> & "C:/Users/neera/OneDrive/Desktop/AI AC/.venv/Scripts/python.exe" "c:/Users/neera/OneDrive/Desktop/AI AC/##W
rite a Python program using a user-de.py"
Enter a number: 17
17 is a prime number.
```

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

❖ **Expected Output**

Comparison table or short analytical report

#prompt

Compare two Python programs for checking prime numbers: one written without user-defined functions and one written using a function. Analyze them based on code clarity, reusability, ease of debugging, and suitability for large-scale applications.

#Code

#Output:

#JUSTIFICATION

The first program without functions works correctly but has all the logic inline, making it less clear and harder to reuse or debug .The second program uses a user-defined function, which separates the prime-checking logic from the main program, improving readability and maintainability.

Using a function allows reusability, so the same logic can be called multiple times without rewriting code.

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)

❖ **Expected Output**

➢ Two correct implementations

➢ Comparison discussing:

▪ Execution flow

▪ Time complexity

▪ Performance for large inputs

▪ When each approach is appropriate

#Prompt

#Write two Python programs to check if a number is prime: one using a basic check from 2 to n-1, and one optimized by checking only up to √n.

#Code

```python
#Write two Python programs to check if a number is prime: one using a basic check from 2 to n-1, and one optimized by checking only up to √n. Include comments and display the result.
# Program 1: Basic check from 2 to n-1
num = int(input("Enter a number: "))
is_prime = True
if num > 1:

    for i in range(2, num):
        if (num % i) == 0:
            is_prime = False
            break
    if is_prime:
        print(f"{num} is a prime number (Basic Check).")
    else:
        print(f"{num} is not a prime number (Basic Check).")
else:
    print(f"{num} is not a prime number (Basic Check).")
# Program 2: Optimized check up to √n
num = int(input("Enter a number: "))
is_prime = True
if num > 1:
    for i in range(2, int(num**0.5) + 1):
        if (num % i) == 0:
            is_prime = False
            break
    if is_prime:
        print(f"{num} is a prime number (Optimized Check).")
    else:
        print(f"{num} is not a prime number (Optimized Check).")
else:
    print(f"{num} is not a prime number (Optimized Check).")
```

#Output

#Justification

The first program uses a basic divisibility check from 2 to n-1. While it is simple and easy to understand, it performs many unnecessary iterations, making it inefficient for large numbers.

The second program uses an optimized approach, checking divisibility only up to √n. This reduces the number of iterations significantly, improving efficiency while giving the same correct result.

The optimized version is faster, easier to scale, and more suitable for larger input values compared to the basic approach.