

ASSENMENT-1

NAME: A. Vamshi Krishna

H.T.N : 2303A52249

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 :

```
# Accept user input
number = int(input("Enter a number: "))

# Assume the number is prime
is_prime = True

# Check for factors
if number <= 1:
    is_prime = False
else:
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            is_prime = False
            break

# Output the result
if is_prime:
    print(f'{number} is a prime number.')
else:
    print(f'{number} is not a prime number.)
```

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 :

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a tree view with "EXPLORER" expanded, displaying "AI ASSISTED CODE" and "AI-ASS.py".
- Code Editor:** Displays the file "AI-ASS.py" with the following code:

```
24
25
26
27     # Optimization: Add special case handling for even numbers
28     if number < 2:
29         is_prime = False
30     elif number == 2:
31         is_prime = True
32     elif number % 2 == 0:
33         is_prime = False
34     else:
35         # Check only odd divisors from 3 onwards
36         for i in range(3, int(number**0.5) + 1, 2):
37             if number % i == 0:
38                 is_prime = False
39                 break
40
```
- Terminal:** Shows command-line output for running the script.

```
PS C:\Users\adapula.vamsi.krish\Desktop\AI Assisted Code> &"C:/Users/adapula.vamsi.krish/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/adapula.vamsi.krish/Desktop/AI Assisted Code/AI-ASS.py"
Enter a prime number:
23 is a prime number.
```
- Right Panel:** An "AI ASSISTED CODE" panel is open, showing an "OPTIMIZATION REPORT" for the file "AI-ASS.py". The report includes:
 - Original and optimized code versions
 - Explanation of how the improvements reduce time complexity
 - A "Read" button to review the changes.
- Bottom Status Bar:** Shows file path (AI-ASS.py), line 26, column 5, spaces 4, UTF-8, CRLF, Python 3.11.7, Go Live, ENG IN, and a timestamp (08/01/2024).

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 :

```
# Function-based approach for reusability
def is_prime_number(num):
    """
    Check if a number is prime.

    Args:
        num: Integer to check for primality

    Returns:
        Boolean: True if prime, False otherwise
    """
    if num < 2:
        return False
    elif num == 2:
        return True
    elif num % 2 == 0:
        return False
    else:
        for i in range(3, int(num**0.5) + 1, 2):
            if num % i == 0:
                return False
        return True

# Get user input
number = int(input("Enter a number: "))

# Call the function and display result
if is_prime_number(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

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 :

The screenshot shows a code editor window with a Python file named 'task1.py'. The code is as follows:

```
def is_prime():
    # Prime number check without using function
    1. Accept user input
    2. number = int(input("Enter a number: "))
    3. If number <= 1:
        4. return False
    5. Else:
        6. If number is prime:
            7. Is_prime = True
        8. Else:
            9. If check for factors:
                10. If number <= 3:
                    11. Is_prime = True
                12. Else:
                    13. For i in range(2, int(number**0.5) + 1):
                        14. If number % i == 0:
                            15. Is_prime = False
                            16. Break
                    17. End loop
            18. End if
            19. If Is_prime == True:
                20. Is_prime = True
            21. Else:
                22. Print("Number is not a prime number.")
```

The terminal below the code editor shows the command 'python task1.py' being run and the output '55 is not a prime number.'

fig: TASK - 1

The screenshot shows a code editor window with a Python file named 'task3.py'. The code is as follows:

```
def is_prime():
    # Function based approach for reusability
    1. If number <= 1:
        2. return False
    3. Else:
        4. If number is prime:
            5. Is_prime = True
        6. Else:
            7. For i in range(2, int(number**0.5) + 1):
                8. If number % i == 0:
                    9. Is_prime = False
                    10. Break
            11. End loop
        12. End if
    13. If Is_prime == True:
        14. Print("Number is a prime number.")
    15. Else:
        16. Print("Number is not a prime number.")
```

The terminal below the code editor shows the command 'python task3.py' being run and the output '55 is not a prime number.'

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 user-defined 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 \sqrt{n})

PROMPT :

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

CODE :

The screenshot shows the VS Code interface with the "AI ASSISTED CODE" extension active. The "EXPLORER" view shows a file named "AI-ASS.py". The "CODE OPTIMIZATION" panel on the right displays two versions of the code: "Original and optimized code versions" and "Explanation of how the improvements reduce time complexity". The "EXPLORER" view also shows the terminal output where the user enters a number and the script prints whether it is prime or not using both basic and optimized methods.

```
# Basic divisibility check approach
def is_prime_basic(num):
    """
    Basic approach: Check divisibility by all numbers from 2 to n-1
    """
    if num < 2:
        return False
    for i in range(2, num):
        if num % i == 0:
            return False
    return True

# Optimized approach: Check divisibility up to √n
def is_prime_optimized(num):
    """
    Optimized approach: Check divisibility only up to square root of n
    Reduces iterations significantly for large numbers
    """
    if num < 2:
        return False
    if num == 2:
        return True
    if num % 2 == 0:
        return False
    for i in range(3, int(num**0.5) + 1, 2):
        if num % i == 0:
            return False
    return True

# Performance comparison
test_number = int(input("Enter a number to test: "))
print(f"\nBasic approach: {is_prime_basic(test_number)}")
print(f"Optimized approach: {is_prime_optimized(test_number)}")
```

Code/AI-ASS.py
Enter a number: 66
66 is not a prime number.
PS C:\Users\adapala vamshi krish\Desktop\AI Assisted Code> & "C:/Users/adapala vamshi krish/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/adapala vamshi krish/Desktop/AI-ASS.py"
Enter a number: 77
77 is not a prime number.
PS C:\Users\adapala vamshi krish\Desktop\AI Assisted 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.