

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**

:

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and several extension icons. The left sidebar has sections for Explorer, AI ASSISTED CODE, and AI ASSISTED CODE. The main editor area contains the following Python code:

```
1 # Prime Number Check with out using Function
2
3 # Accept user input
4 number = int(input("Enter a number: "))
5
6 # Assume the number is prime
7 is_prime = True
8
9 # Check for factors
10 if number <= 1:
11     is_prime = False
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
18 # Output the result
19 if is_prime:
20     print(f"{number} is a prime number.")
21 else:
22     print(f"{number} is not a prime number.)
```

The terminal at the bottom shows the command 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 Assisted Code/AI-ASS.py" entered, followed by the output "5 is not a prime number." and the prompt PS C:\Users\Adapala Vamshi Krish\Desktop\AI Assisted 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**

:

```
# Optimization: Add special case handling for even numbers
if number < 2:
    is_prime = False
elif number == 2:
    is_prime = True
elif number % 2 == 0:
    is_prime = False
else:
    # Check only odd divisors from 3 onwards
    for i in range(3, int(number**0.5) + 1, 2):
        if number % i == 0:
            is_prime = False
            break
    else:
        is_prime = True
print(is_prime)

2 is a prime number.
```

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 the PyCharm IDE interface with the following details:

- Title Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Toolbars:** Standard, Editor, Navigation, Status Bar
- Code Area:** A Python script titled "prime.py" containing the following code:

```
1 # Prime number check with not using function
2
3 # Accept user limit
4 number = int(input("Enter a number: "))
5
6 # Assume the number is prime
7 prime = True
8
9 # Check for factors
10 if number < 2:
11     prime = False
12 else:
13     for i in range(2, int(number**0.5) + 1):
14         if number % i == 0:
15             prime = False
16             break
17
18 # Output the result
19 if prime:
20     print(prime, "is a prime number.")
21 else:
22     print(prime, "is not a prime number.")
```
- Run Tab:** RUN, STOP, DEBUG, RELOAD, RESTART, VIRTUAL ENVIRONMENTS, LOGS, STATUS, HELP
- Output Tab:** PyCharm output window showing the execution results:

```
PS C:\Users\lalita\PycharmProjects\PythonTutorials\Day-1> python calculate_prime.py
Enter a number: 13
13 is a prime number.
```
- Bottom Bar:** Search, Recent Files, Icons for various tools like GitHub, Git, Docker, etc.
- Right Sidebar:** Build with Agent, Run Configuration, and other project-related settings.

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 \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 following details:

- File Explorer:** Shows a file named "AI-ASS.py".
- Code Editor:** Displays Python code for prime number checking. It includes three main sections:
 - # Basic divisibility check approach
 - # Optimized approach: Check divisibility up to \sqrt{n}
 - # Performance comparison
- Terminal:** Shows command-line output for testing the code with numbers 66 and 77.
- CodeLens:** An AI-generated code lens provides optimization suggestions for the code.
- Status Bar:** Shows the current file is "AI-ASS.py", the line count is 106, and the date is 08-01-2026.

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.