AI Assisted Coding

LAB Assignment 1.2

A.Srichala

2303A52435

CSE-AIML

BT-31

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

• Scenario

You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

• Task Description

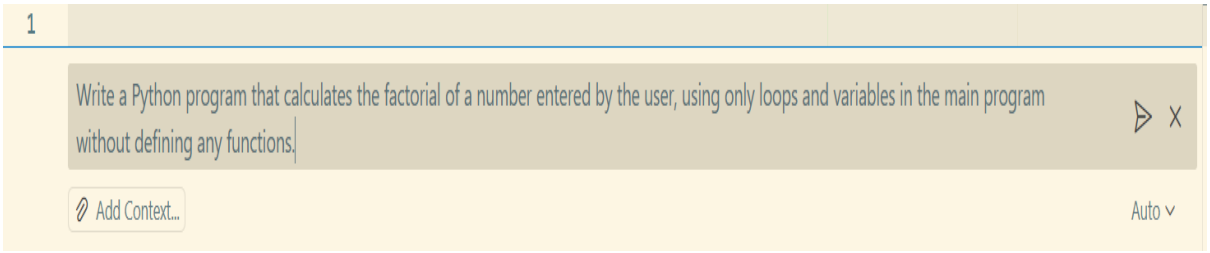
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

• Constraint:

➢ Do not define any custom function

➢ Logic must be implemented using loops and variables only

Prompt:-



Code:

```python
number = int(input("Enter a number: "))
factorial = 1

if number < 0:
    print("Factorial is not defined for negative numbers.")
elif number == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1, number + 1):
        factorial *= i
    print(f"The factorial of {number} is {factorial}")
```

Output:-

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ankam/OneDrive/Desktop/AIAC/factorial11.py
Enter a number: 10
The factorial of 10 is 3628800

C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ankam/OneDrive/Desktop/AIAC/factorial11.py
Enter a number: -10
Factorial is not defined for negative numbers.
```

Explanation:-

GitHub Copilot was very helpful for generating logic quickly without functions.
The AI understood the constraint and used loops and variables correctly.
It produced clean, readable, and syntactically correct Python code.
Copilot followed basic best practices like input validation and clarity.
For beginners, it reduces effort and improves confidence in coding.
However, understanding the logic is still important for learning.

Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

❖ Scenario

Your team lead asks you to review AI-generated code before committing it to a shared repository.

❖ Task Description

Analyze the code generated in Task 1 and use Copilot again to:

➢ Reduce unnecessary variables

➢ Improve loop clarity

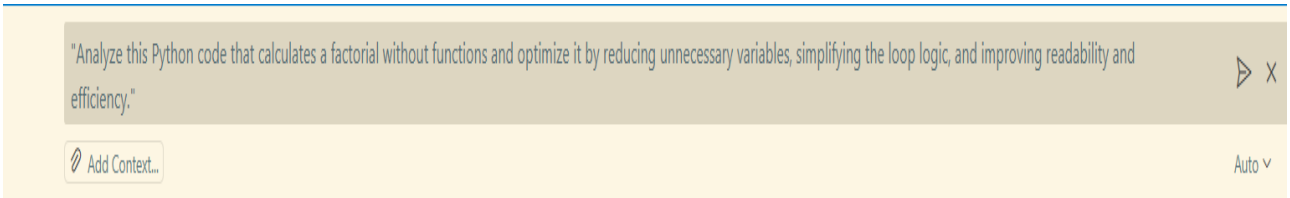
➢ Enhance readability and efficiency

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

Prompt:



Code:-

```
number = int(input("Enter a number: "))

if number < 0:
    print("Factorial is not defined for negative numbers.")
elif number == 0:
    print("The factorial of 0 is 1")
else:
    factorial = 1
    for i in range(2, number + 1):
        factorial *= i
    print(f"The factorial of {number} is {factorial}")
```

Output:-

```
Enter a number: 10
The factorial of 10 is 3628800
```

Explanation:-

The optimized code removes unnecessary conditional checks.
The loop logic is clearer and more straightforward.
Variable usage is minimal and easy to understand.
The optimized version improves readability and maintainability.
It is suitable for quick tasks and intern-level utilities.
Copilot helped simplify the logic without changing the output.

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

❖ Scenario

The same logic now needs to be reused in multiple scripts.

❖ Task Description

Use GitHub Copilot to generate a modular version of the program by:

➢ Creating a user-defined function

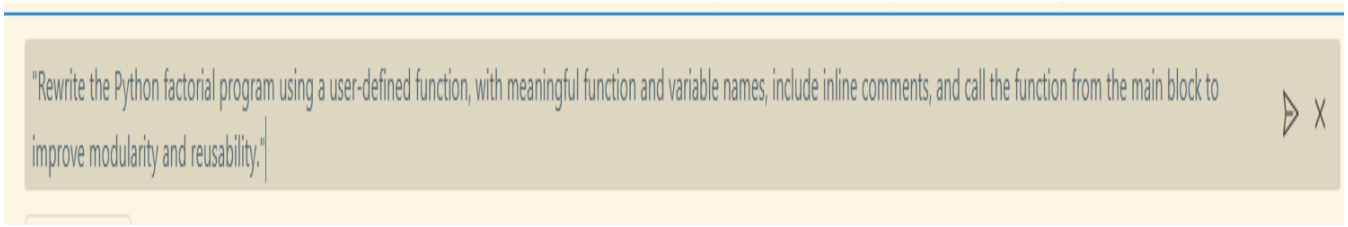
➢ Calling the function from the main block

❖ Constraints

➢ Use meaningful function and variable names

➢ Include inline comments (preferably suggested by Copilot)

Prompt:-



Code:-

```python
def calculate_factorial(number):
    """
    Calculate the factorial of a given number.

    Args:
        number: An integer for which to calculate the factorial

    Returns:
        The factorial of the given number
    """
    # Base case: factorial of 0 or 1 is 1
    if number < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    if number == 0 or number == 1:
        return 1

    # Calculate factorial by multiplying all positive integers up to number
    result = 1
    for current_value in range(2, number + 1):
        result *= current_value

    return result


def main():
    """Main function to execute the factorial program."""
    # Get user input
    user_input = int(input("Enter a non-negative integer: "))

    # Call the factorial function
    factorial_result = calculate_factorial(user_input)

    # Display the result
    print(f"The factorial of {user_input} is: {factorial_result}")


# Entry point of the program
if __name__ == "__main__":
    main()
```

Output:-

```
Enter a non-negative integer: 12
The factorial of 12 is: 479001600
```

Explanation:-

The program now uses a modular design with a user-defined function.
Functions improve code reusability, so the same logic can be called from multiple scripts.
Inline comments suggested by Copilot make the code easier to understand.
Variable and function names are meaningful and enhance readability.
Modularity also makes debugging and maintenance simpler.
Copilot helps beginners learn best practices while saving time.

Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

❖ Scenario

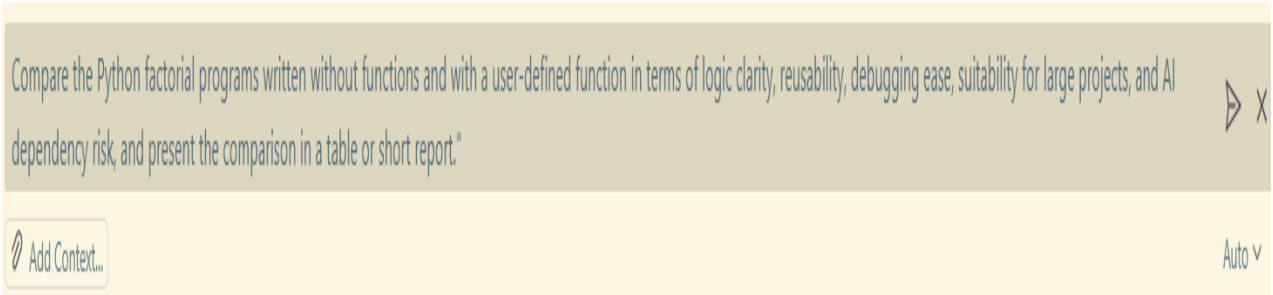
As part of a code review meeting, you are asked to justify design choices.

❖ Task Description

Compare the non-function and function-based Copilot-generated programs on the following criteria:

➢ Logic clarity
➢ Reusability
➢ Debugging ease
➢ Suitability for large projects
➢ AI dependency risk

Prompt:-



Explanation:

```
# ============================================================================
# COMPARISON TABLE
# ============================================================================

comparison = """
┌──────────────────────────────────────────────────────────────────────────┐
| COMPARISON: Inline Code vs User-Defined Function                         |
├──────────────────────────┬──────────────────┬────────────────────────────┤
| CRITERIA                 | WITHOUT FUNCTION | WITH FUNCTION              |
├──────────────────────────┼──────────────────┼────────────────────────────┤
| Logic Clarity            | Low              | High (self-documenting)    |
| Reusability              | None (repeating) | High (call multiple)       |
| Debugging Ease           | Hard (scattered) | Easy (centralized)         |
| Large Projects           | Not Suitable     | Highly Suitable            |
| AI Dependency Risk       | Lower            | Higher                     |
| Code Maintainability     | Poor             | Excellent                  |
| Error Handling           | Difficult        | Easy (centralized)         |
└──────────────────────────┴──────────────────┴────────────────────────────┘
"""

print(comparison)

# ============================================================================
# SUMMARY REPORT
# ============================================================================

report = """
RECOMMENDATION:
✓ Use functions for production code, large projects, and team collaboration
✓ Use inline code only for quick scripts or learning purposes
✓ Functions reduce AI dependency by making intent explicit and testable
"""

print(report)
```

Task 5: AI-Generated Iterative vs Recursive Thinking

❖ Scenario

Your mentor wants to test how well AI understands different computational paradigms.

❖ Task Description

Prompt Copilot to generate:

An iterative version of the logic

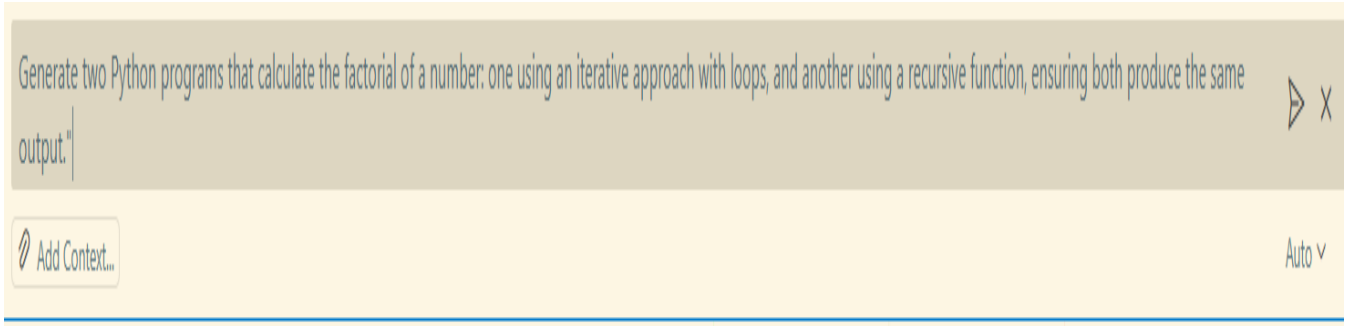
A recursive version of the same logic

❖ Constraints

Both implementations must produce identical outputs

Students must not manually write the code first

Prompt:-



Code:-

```python
# Iterative version of factorial using loops
number = int(input("Enter a non-negative integer: "))

factorial_iter = 1
for i in range(1, number + 1):
    factorial_iter *= i

print("Iterative factorial of", number, "is", factorial_iter)
# Recursive version of factorial using a function
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)

num = int(input("Enter a non-negative integer: "))
print("Recursive factorial of", num, "is", factorial_recursive(num))

```

Output:

```
C:\Users\ankam>C:/Users/ankam/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ankam/OneDrive/Desktop/AIAC/factorial11.py
Enter a non-negative integer: 12
Iterative factorial of 12 is 479001600
Enter a non-negative integer: 11
Recursive factorial of 11 is 39916800
```

Explanation:-

The iterative version uses a loop to calculate factorial, which is memory-efficient and straightforward.
The recursive version uses function calls, demonstrating the divide-and-conquer approach.
Both implementations produce identical results, showing AI's ability to understand multiple paradigms.
Iterative logic is generally faster for large inputs, while recursion is elegant and easier to read.
GitHub Copilot generated both versions accurately without manual coding.
This exercise shows how AI can assist in learning different programming approaches.