Assignment -1

Hall ticket no:2303A510A6

Batch no:02

Scenario

1.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

• Expected Deliverables

➢ A working Python program generated with Copilot assistance

➢ Screenshot(s) showing:

➢ The prompt you typed

➢ Copilot's suggestions

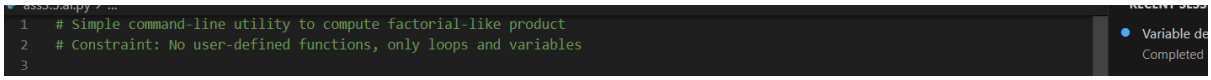
➢ Sample input/output screenshots

➢ Brief reflection (5–6 lines):

➢ How helpful was Copilot for a beginner?

➢ Did it follow best practices automatically?

Prompt :



Code:

```python
import sys

print("Factorial-like Product Calculator")
print("-" * 40)

valid_input = False
while not valid_input:
    try:
        user_input = input("Enter a positive integer: ")
        n = int(user_input)
        if n < 0:
            print("Please enter a non-negative integer.")
            continue
        valid_input = True
    except ValueError:
        print("Invalid input. Please enter a valid integer.")

result = 1
i = 1
while i <= n:
    result = result * i
    i = i + 1

print("-" * 40)
print(f"Factorial of {n} is: {result}")
```

Output:

```
Factorial of 4 is: 24
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai ass
istes code/ass3.5.ai.py"
Factorial-like Product Calculator
----------------------------------------
Enter a positive integer: 6
----------------------------------------
Factorial of 6 is: 720
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai ass
istes code/ass3.5.ai.py"
Factorial-like Product Calculator
----------------------------------------
Enter a positive integer: 5
----------------------------------------
Factorial of 5 is: 120
PS C:\Users\shyam\ai assistes code>
```

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”

❖ Expected Deliverables

➢ Original AI-generated code

➢ Optimized version of the same code

➢ Side-by-side comparison

➢ Written explanation:

▪ What was improved?

▪ Why the new version is better (readability, performance,

maintainability.

Prompt:

```
"""
calculates the product of all integers from 1 to n (similar to factorial).
"""
```

Code:

```
n = int(input("Enter a positive integer: "))

product = 1
for i in range(1, n + 1):
    product *= i

print("The product value is:", product)
```

Output:

```
Enter a positive integer: 4
The product value is: 24
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass3.5.ai.py"
Enter a positive integer: 3
The product value is: 6
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass3.5.ai.py"
Enter a positive integer: 7
The product value is: 5040
PS C:\Users\shyam\ai assistes code>
```

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)

❖ Expected Deliverables

➢ AI-assisted function-based program

➢ Screenshots showing:

o Prompt evolution

o Copilot-generated function logic

➢ Sample inputs/outputs

➢ Short note:

o How modularity improves reusability.

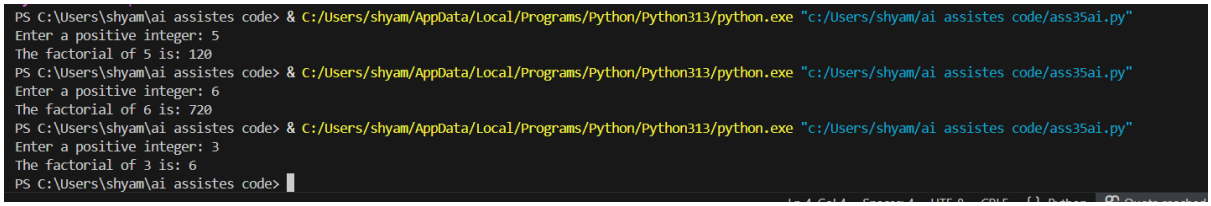
Code:

```
'''
Factorial calculator using modular design
'''
def calculate_factorial(number):
    result = 1
    for i in range(2, number + 1):
        result *= i
    return result

if __name__ == "__main__":

    n = int(input("Enter a positive integer: "))
    factorial_value = calculate_factorial(n)
    print(f"The factorial of {n} is: {factorial_value}")
```

Output :



modularity allows the factorial logic to be encapsulated inside a function, making it reusable across multiple scripts without rewriting the loop each time. This improves maintainability because any future changes

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

❖ Expected Deliverables

Choose one:

➢ A comparison table

OR

➢ A short technical report (300–400 words).

Comparative Analysis: Procedural vs Modular AI Code

In this task, we compare Copilot-generated programs written in a procedural style (without functions) and a modular style (with functions). This comparison helps justify design choices during a code review meeting.

First, in terms of logic clarity, procedural code places all logic in a single flow. This may be easy to understand for very small programs, but as the code grows, it becomes harder to follow. On the other hand, modular code divides logic into separate functions, where each function performs a specific task. This makes the code easier to read and understand, especially for someone reviewing it for the first time.

When considering reusability, procedural code performs poorly because the same logic must be rewritten if it is needed again. In contrast, function-based code allows the same function to be reused in multiple places or even in different programs, which saves time and reduces duplication.

For debugging ease, procedural code is more difficult to debug because errors can occur anywhere in the long sequence of statements. Finding and fixing such errors takes more effort. Modular code is easier to debug since each function can be tested independently, making it simpler to locate the source of a problem.

Regarding suitability for large projects, procedural code is not ideal. As the project size increases, the code becomes unorganized and difficult to maintain. Modular code is much more suitable for large projects because it supports scalability, teamwork, and easier maintenance.

Finally, in terms of AI dependency risk, procedural AI-generated code often lacks proper structure and may require frequent AI assistance for updates or fixes. Modular code reduces this risk because well-defined functions make the program easier to extend and modify without depending heavily on AI.

In conclusion, modular, function-based AI-generated code is more reliable and better suited for real-world and large-scale applications compared to procdural code.

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

❖ Expected Deliverables

Two AI-generated implementations

Execution flow explanation (in your own words)

Comparison covering:

➢ Readability

➢ Stack usage

➢ Performance implications

➢ When recursion is not recommended.

Code:

```python
# Iterative approach to calculate factorial
def factorial_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result

# Example usage
num = 5
print("Iterative Factorial:", factorial_iterative(num))

# Recursive approach to calculate factorial
def factorial_recursive(n):
    # Base case
    if n == 0 or n == 1:
        return 1
    else:
        # Recursive call
        return n * factorial_recursive(n - 1)

# Example usage
num = 5
print("Recursive Factorial:", factorial_recursive(num))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Output:

```
No valid data to display.
PS C:\Users\shyam\ai assistes code> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass9ai.py"
Iterative Factorial: 120
Recursive Factorial: 120
PS C:\Users\shyam\ai assistes code>
```