G.ADITHYA

2303A51596

BATCH 25

AI ASSISTANT ASSIGNMENT 1

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

- Expected Deliverables

➢ A working Python program generated with Copilot assistance

➢ Screenshot(s) showing:

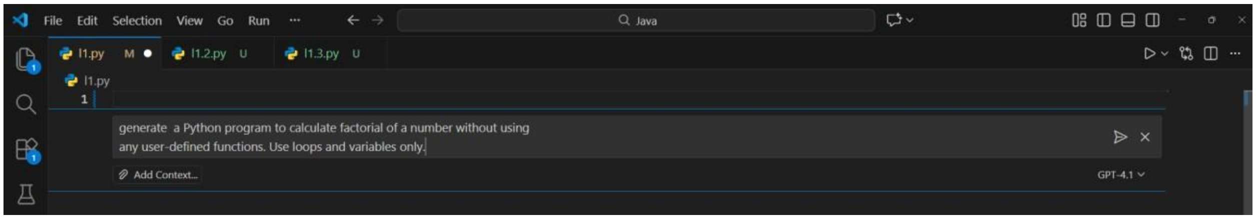
➢ The prompt you typed

➢ Copilot's suggestions

➢ Sample input/output screenshots

➢ Brief reflection (5–6 lines):

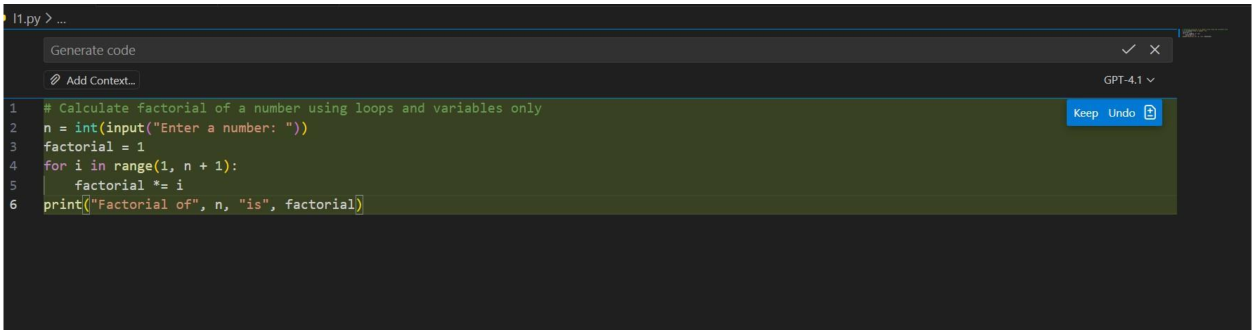
- ➢ How helpful was Copilot for a beginner?
- ➢ Did it follow best practices automatically?



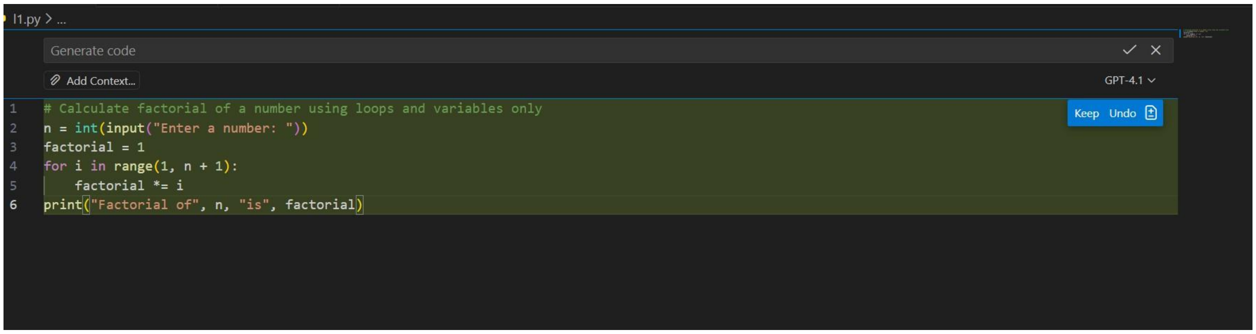
generate a Python program to calculate factorial of a number without using any user-defined functions. Use loops and variables only.

Generate a Python program to calculate factorial of a number without using any user-defined functions. Use loops and variables only.

```
# Calculate factorial of a number using loops and variables
only n = int(input("Enter a number: ")) factorial = 1 for i in
range(1, n + 1):
    factorial *= i print("Factorial of",
n, "is", factorial)
```



```python
# Calculate factorial of a number using loops and variables only
n = int(input("Enter a number: "))
factorial = 1
for i in range(1, n + 1):
    factorial *= i
print("Factorial of", n, "is", factorial)
```



```python
# Calculate factorial of a number using loops and variables only
n = int(input("Enter a number: "))
factorial = 1
for i in range(1, n + 1):
    factorial *= i
print("Factorial of", n, "is", factorial)
```

This task focuses on creating a simple Python program to calculate the factorial of a number without using any user-defined functions.

The scenario simulates a quick command-line utility required for startup intern onboarding, where speed and simplicity are prioritized over modular design.

GitHub Copilot is used to automatically generate the factorial logic based on a natural language prompt.

The entire logic is implemented directly in the main execution flow using loops and variables only.

User input is accepted from the command line, and the factorial value is computed iteratively.

This task helps demonstrate how AI tools assist beginners in generating correct procedural code.

It also highlights the limitations of non-modular code in terms of reusability and structure.

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

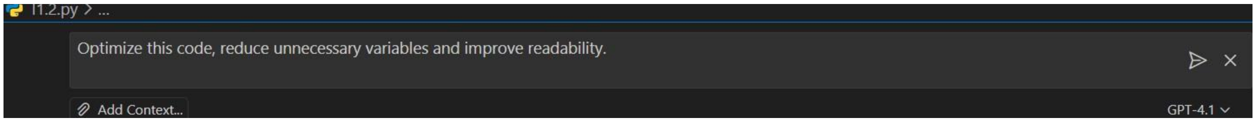
- ➢ Original AI-generated code
- ➢ Optimized version of the same code
- ➢ Side-by-side comparison ➢ Written explanation:

- ▪ What was improved?
- ▪ Why the new version is better (readability, performance,

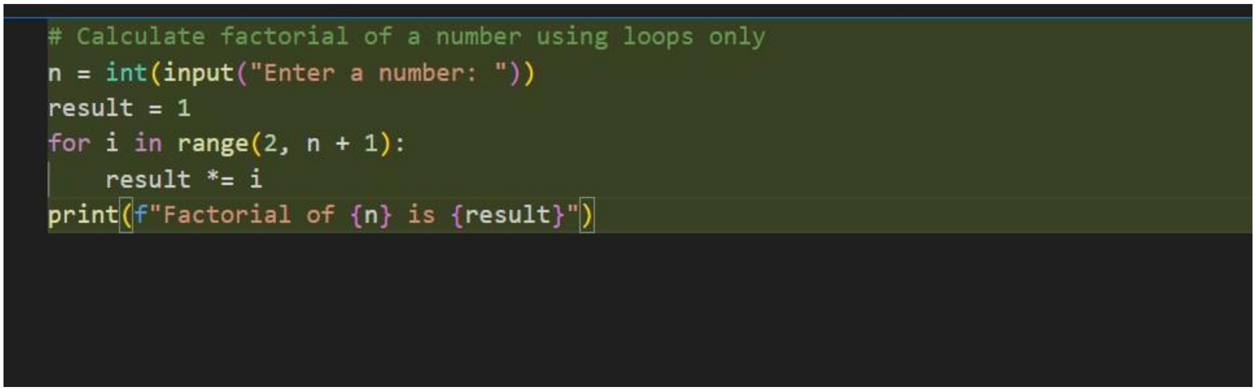
Maintainability.

Prompt

Generate a Python program to calculate factorial of a number without using any user-defined functions. Use loops and variables only.



Optimize this code, reduce unnecessary variables and improve readability.



```python
# Calculate factorial of a number using loops only
n = int(input("Enter a number: "))
result = 1
for i in range(2, n + 1):
    result *= i
print(f"Factorial of {n} is {result}")
```

In this task, the factorial program generated in Task 1 is reviewed and improved before committing it to a shared repository.
The original AI-generated code is analyzed for unnecessary variables and unclear logic.
GitHub Copilot is prompted again to optimize the code and improve readability.
The optimized version simplifies loop logic and improves variable naming.
Unnecessary statements are removed to make the code cleaner and more efficient.
This task emphasizes the importance of reviewing AI-generated code instead of using it blindly.
It also demonstrates how optimization improves maintainability and code quality.

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

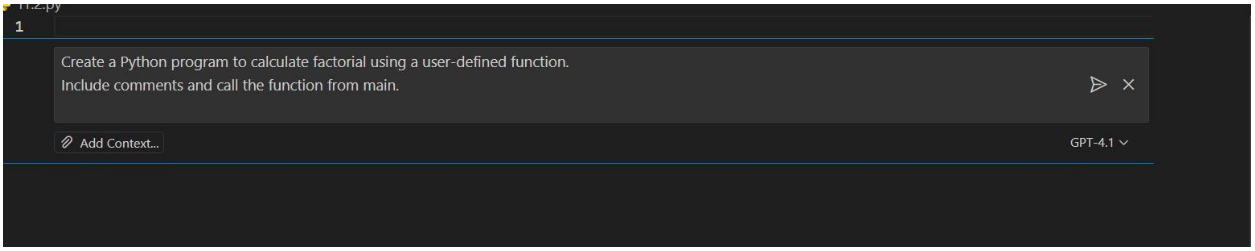
➢ Use meaningful function and variable names

➢ Include inline comments (preferably suggested by Copilot)

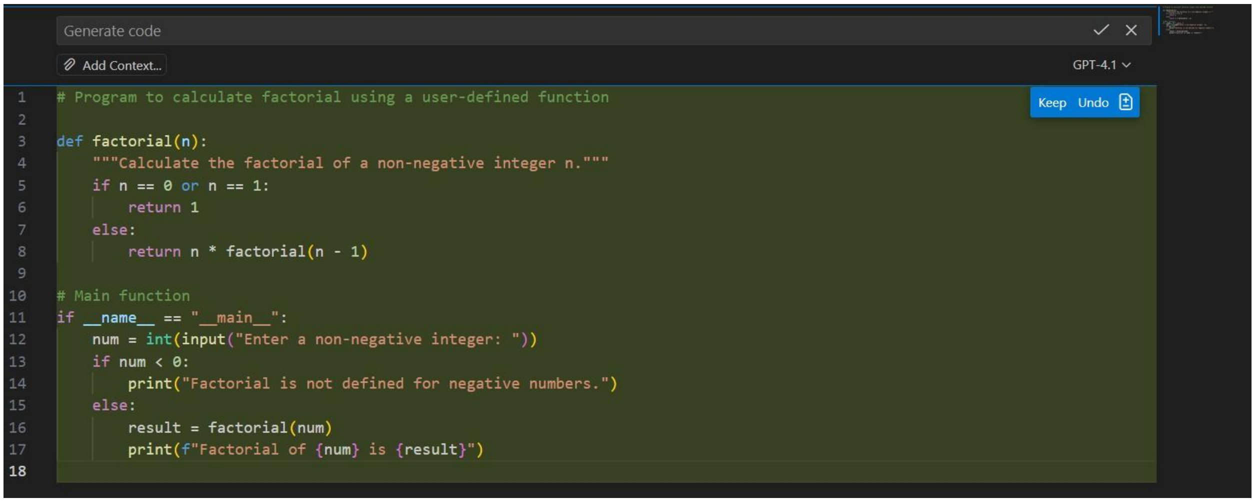
❖ Expected Deliverables

➢ 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.



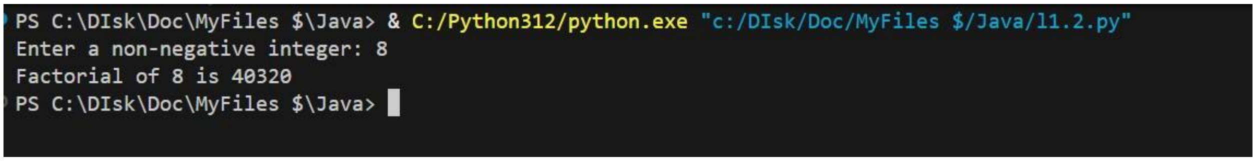
Create a Python program to calculate factorial using a user-defined function.
Include comments and call the function from main.



```python
# Program to calculate factorial using a user-defined function

def factorial(n):
    """Calculate the factorial of a non-negative integer n."""
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Main function
if __name__ == "__main__":
    num = int(input("Enter a non-negative integer: "))
    if num < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        result = factorial(num)
        print(f"Factorial of {num} is {result}")
```



```
PS C:\DIsk\Doc\MyFiles $\Java> & C:/Python312/python.exe "c:/DIsk/Doc/MyFiles $/Java/l1.2.py"
Enter a non-negative integer: 8
Factorial of 8 is 40320
PS C:\DIsk\Doc\MyFiles $\Java>
```

This task introduces modular programming by converting the factorial logic into a user-defined function.

The scenario assumes that the same factorial logic must be reused across multiple scripts or applications.

GitHub Copilot is used to generate a function that computes the factorial of a given number.

The function is called from the main block of the program to display the result.

Meaningful function and variable names are used to improve code clarity.

Inline comments, preferably suggested by Copilot, explain the logic inside the function.

This task highlights how modular design improves reusability, readability, and maintainability.

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

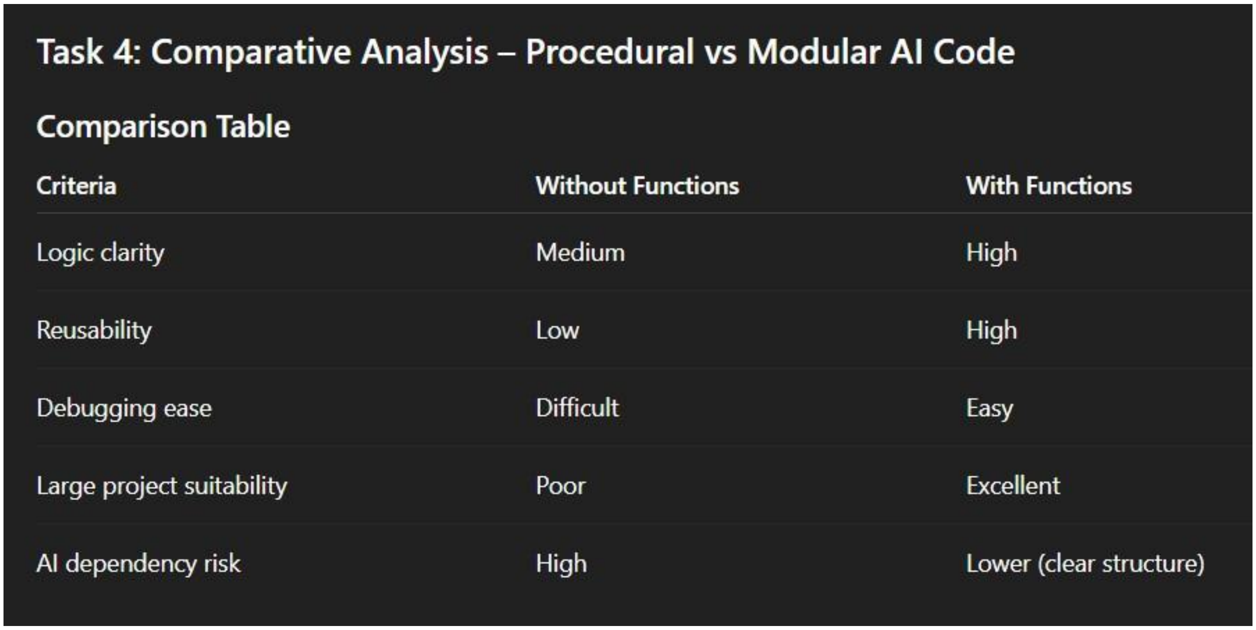
➢ Suitability for large projects

➢ AI dependency risk ❖ Expected Deliverables Choose one:

➢ A comparison table

OR

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



Task 4: Comparative Analysis – Procedural vs Modular AI Code

Comparison Table

Criteria	Without Functions	With Functions
Logic clarity	Medium	High
Reusability	Low	High
Debugging ease	Difficult	Easy
Large project suitability	Poor	Excellent
AI dependency risk	High	Lower (clear structure)

This task involves a detailed comparison between procedural factorial code and modular factorial code generated by GitHub Copilot.
The comparison is conducted as part of a technical code review meeting.
Both versions are evaluated based on logic clarity, reusability, and ease of debugging.
Procedural code is found to be simple but difficult to scale for large projects.
Modular code offers better structure and easier maintenance.
The analysis also considers AI dependency risks when relying on generated code.
This task helps justify design decisions in professional software development environments.

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

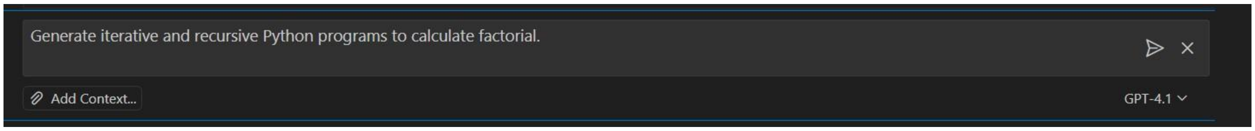
➢ Stack usage

➢ Performance implications

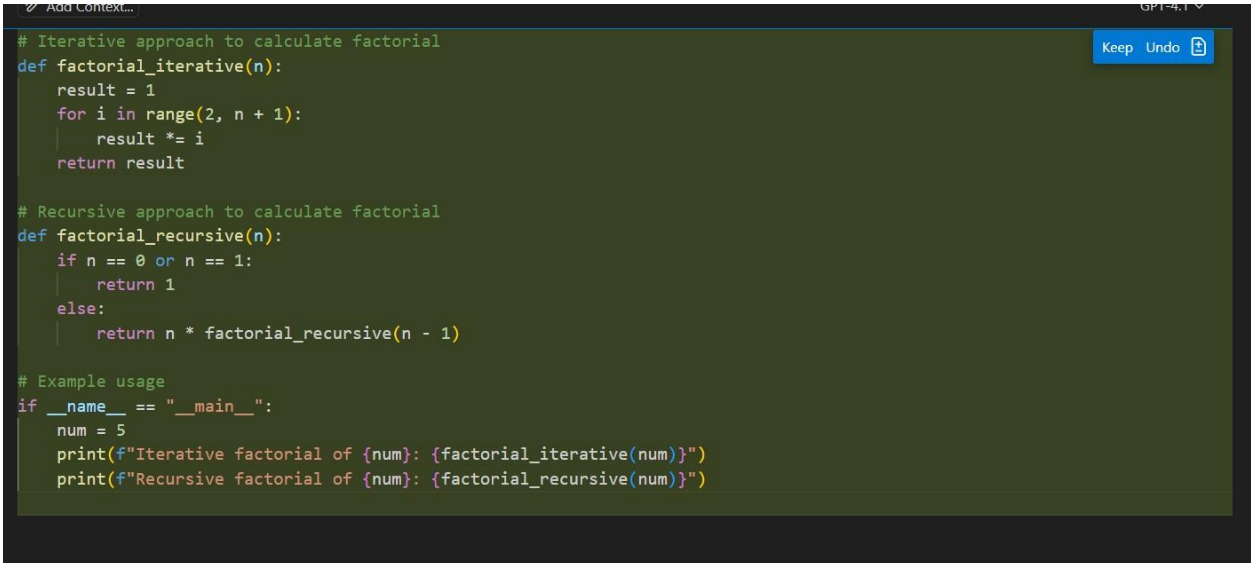
➢ When recursion is not re

Commended.

In this task, GitHub Copilot is used to generate two different computational approaches for calculating factorial.
One approach uses an iterative loop-based method, while the other uses a recursive method.
Both implementations are expected to produce identical outputs for the same input.
The execution flow of each approach is analyzed and explained in simple terms.
The task compares readability, stack usage, and performance implications of both methods.
Special attention is given to situations where recursion is not recommended, such as large input values.
This task demonstrates how AI handles different algorithmic paradigms for the same problem.



Generate iterative and recursive Python programs to calculate factorial.



```python
# Iterative approach to calculate factorial
def factorial_iterative(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

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

# Example usage
if __name__ == "__main__":
    num = 5
    print(f"Iterative factorial of {num}: {factorial_iterative(num)}")
    print(f"Recursive factorial of {num}: {factorial_recursive(num)}")
```