AI Assistant Coding

Assignment-1.2

Name:K.Trinay Prasad

HT No:2303A52070

Batch:32

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

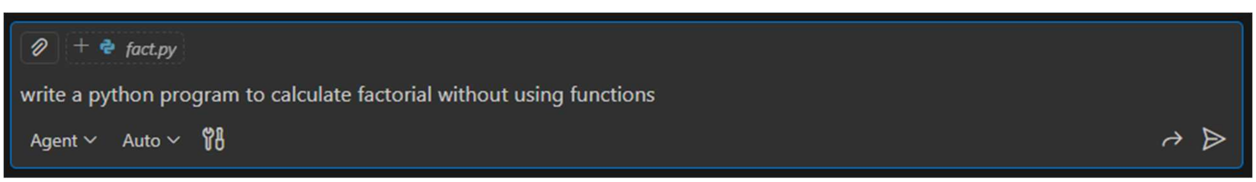
Scenario: You are building a **small command-line utility** for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

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
n = int(input("Enter a number: "))
fact = 1

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

print("Factorial is:", fact)
```

Output:

```
PS D:\AI Assicoding> & C:/Python313/python.exe "d:/AI Assicoding/fact.py"
Enter a number: 5
Factorial is: 120
PS D:\AI Assicoding>
```

Explanation:

The code generated from compile was well structured and for beginner it was very helpful. It reduces coding time and gives code without and logical or syntax error.

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:

fact.py

Write a Python program to find the factorial of a given number using only loops and variables. Do not define or use any functions. The code should be simple, optimized, easy to understand, and reliable.

Agent Auto

Code:

```python
# Input from user
n = int(input("Enter a non-negative integer: "))

# Validate input
if n < 0:
    print("Factorial is not defined for negative numbers.")
elif n == 0 or n == 1:
    print("Factorial is: 1")
else:
    # Calculate factorial using loop
    factorial = 1
    for i in range(2, n + 1):
        factorial = factorial * i
    print("Factorial is:", factorial)
```

Output:

```
PS D:\AI Assicoding> & C:/Python313/python.exe "d:/AI Assicoding/fact.py"
Enter a non-negative integer: 5
Factorial is: 120
PS D:\AI Assicoding>
```

Explanation:

Compared to previous code this code was most efficient as it handles the edge case for negative and zero input.

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

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

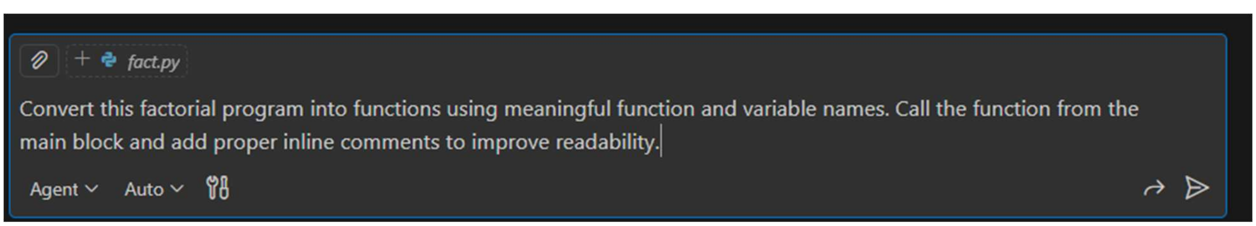
Task Description: Use GitHub Copilot to generate a **modular version** of the program by:

- ➢ 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:

```
def get_number() -> int:
    """Prompt the user to enter a number and return it as an integer."""
    return int(input("Enter a number: "))


def calculate_factorial(n: int) -> int:
    """Calculate the factorial of n using a loop. Assumes n >= 0."""
    factorial = 1  # Initialize factorial to 1

    for i in range(1, n + 1):  # Loop from 1 to n inclusive
        factorial *= i  # Multiply factorial by current i

    return factorial


def main():
    """Main function to handle the program logic."""
    n = get_number()  # Get the number from user input

    if n < 0:
        print("Factorial is not defined for negative numbers.")  # Handle negative input
    elif n == 0:  (function) def calculate_factorial(n: int) -> int
        print("Th
                  Calculate the factorial of n using a loop. Assumes n >= 0.
    else:
        result = calculate_factorial(n)  # Calculate factorial
        print(f"The factorial of {n} is {result}")  # Print the result


if __name__ == "__main__":
    main()  # Call main function when script is run directly
```

Output:

```
PS D:\AI Assicoding> & C:/Python313/python.exe "d:/AI Assicoding/fact.py"
Enter a number: 6
The factorial of 6 is 720
```

Explanation:

The code is converted into different functions with main block calling and improving readability.

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

	Without Functions	**With Functions**
Logic Clarity	Logic is written in a single flow,easy to understand but becomes unreadable as lines increases.	Logic is divided into functions, making the code to read & understand easily.
Reusability	Code cannot be reused	Functions can be reused in other programs without writing the logic again.
Debugging Ease	Debugging becomes hard as all the logic at one place	Debugging becomes easy as all the logic written multiple functions
Suitability for Large Projects	Not suitable for large projects	Suitable for large projects due to proper structure

AI Dependency Risk	Higher risk for long procedural code, hard to review or modify.	Lower risk generated in functions, easy to review & Modify.

- ➢ Suitability for large projects
- ➢ AI dependency risk
- ➢

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

Iterative:

Prompt:

```
fact.py
write code of factroial in itrative aproach
Agent    Auto
```

Code:

```python
#write code to compute the factorial of a number without using recursion and handle invalid inputs
def compute_factorial(n):
    '''
    Docstring for compute_factorial function

    :param n: non-negative integer to compute factorial
    :return: factorial of n or error message for invalid input
    '''
    if not isinstance(n, int) or n < 0:
        return "Error: Input must be a non-negative integer."

    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
# Example usage
number = 5
print(f"The factorial of {number} is {compute_factorial(number)}")
```

Output:

```
PS D:\AI Assicoding> & C:/Python313/python.exe "d:/AI Assicoding/fact.py"
The factorial of 5 is 120
PS D:\AI Assicoding>
```

Recursive:

Prompt:

```
fact.py
write code of factroial in recursive aproach
Agent  Auto
```

Code:

```python
def factorial_recursive(n: int) -> int:
    """Calculate the factorial of n using recursion."""
    if n == 0 or n == 1:
        return 1  # Base case: factorial of 0 or 1 is 1
    else:
        return n * factorial_recursive(n - 1)  # Recursive case


def main():
    """Main function to handle the program logic."""
    n = int(input("Enter a number: "))  # Get the number from user input

    if n < 0:
        print("Factorial is not defined for negative numbers.")  # Handle negative input
    else:
        result = factorial_recursive(n)  # Calculate factorial recursively
        print(f"The factorial of {n} is {result}")  # Print the result


if __name__ == "__main__":
    main()  # Call main function when script is run directly
```

Output:

```
PS D:\AI Assicoding> & C:/Python313/python.exe "d:/AI Assicoding/fact.py"
Enter a number: 7
The factorial of 7 is 5040
```

	Iterative Approach	**Recursive Approach**

Readability	Easy to understand for beginners	harder for beginners due to function calls
Stack Usage	Uses constant memory	Uses call stack for each function call
Performance	Faster and more memory-efficient	slower due to function call overhead