

AIAC ASSIGNMENT 1.2

Name: P.RITHWICK
HT.NO: 2303A52484
Batch -31

Question 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

Prompt:

write a python program to print the factorial of a given number by using loops and variables and dont use any functions and the input should be dynamic

Code screenshot:

```
1 num = int(input("Enter a number: "))
2 factorial = 1
3
4 for i in range(1, num + 1):
5     factorial = factorial * i
6
7 print(f"The factorial of {num} is {factorial}")
```

Output:

```
Enter a number: 6
The factorial of 6 is 720
PS C:\Users\burra\OneDrive\문서\WPS Cloud Files\Desktop\sem 6\AIAC>
```

Copilot suggestion:

```
...
Docstring for factorial
this program computes the factorial of a non-negative integer
...  
...
```

Explanation:

Copilot was very helpful for beginners by quickly generating correct logic from prompts. It reduced coding time, followed best practices, and helped avoid syntax and logical errors. Overall, it made coding easier & faster.

Question 2:

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

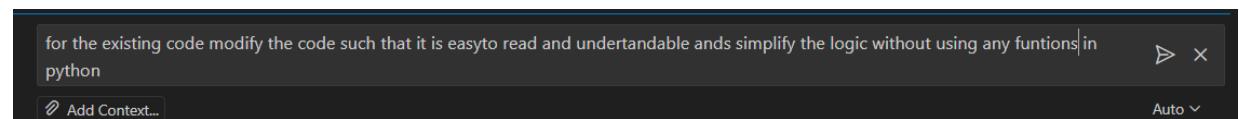
Hint: Prompt Copilot with phrases like

“optimize this code”,

“simplify logic”, or

“make it more readable”

Prompt:



Code screenshot:

```
# Get user input
num = int(input("Enter a number: "))

# Initialize factorial result
factorial = 1

# Calculate factorial by multiplying numbers from 1 to num
for i in range(1, num + 1):
    factorial *= i

# Display the result
print(f"The factorial of {num} is {factorial}")
```

Output:

```
Enter a number: 6
The factorial of 6 is 720
PS C:\Users\burra\OneDrive\문서\WPS Cloud Files\Desktop\sem 6\AIAC>
```

Explanation:

Optimized the code in minimal number of lines and short variable names

question 3

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

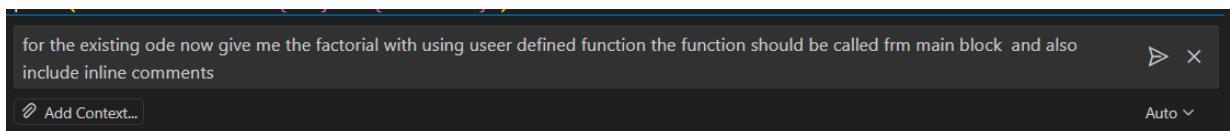
Task Description Use GitHub Copilot to generate a modular version of the program by:

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)

Promt:



Code screenshot:

```

# Get user input
num = int(input("Enter a number: "))

# Initialize factorial result
factorial = 1

# Calculate factorial by multiplying numbers from 1 to num
for i in range(1, num + 1):
    factorial *= i

# Display the result
print(f"The factorial of {num} is {factorial}")

def calculate_factorial(n):
    # Initialize factorial to 1
    result = 1
    # Multiply all numbers from 1 to n
    for i in range(1, n + 1):
        result *= i
    # Return the calculated factorial
    return result

# Main block
if __name__ == "__main__":
    # Get user input
    num = int(input("Enter a number: "))
    # Call the function to calculate factorial
    factorial = calculate_factorial(num)
    # Display the result
    print(f"The factorial of {num} is {factorial}")

```

Output:

```

& 'c:\Users\burra\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\burra\OneDrive\문서\WPS Cloud Files\Desktop\sem 6\AIAC>
4\bundled\libs\debugpy\launcher' '57526' '--' 'C:\Users\burra\OneDrive\문서\WPS Cloud Files\Desktop\sem 6\AIAC>
Enter a number: 7
The factorial of 7 is 5040
PS C:\Users\burra\OneDrive\문서\WPS Cloud Files\Desktop\sem 6\AIAC>

```

Explanation:

Copilot Modified The Code To Function Type
 & main block is created

question 4

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

Explanation:

Aspect	Non-function based	Function-based
Logic clarity	Logic is in one block. Harder to understand	Logic is separated. Easy to understand
Reusability	Code cannot be reused	Code can be reused easily
Debugging ease	Debugging is difficult	Debugging is easy
Suitability for large projects	Not suitable for large projects	Suitable for large projects
AI dependency risk	High risk of blind copying	Lower risk due to clear structure

Question 5

Scenario:

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

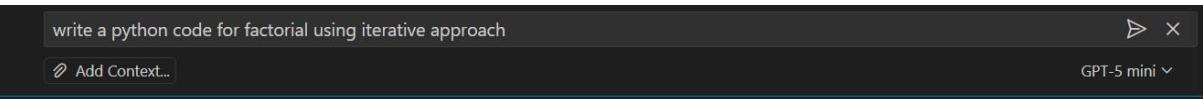
Task Description:

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

Promt 1



code:

```
# Get user input
num = int(input("Enter a number: "))

# Initialize factorial result
factorial = 1

# Calculate factorial by multiplying numbers from 1 to num
for i in range(1, num + 1):
    factorial *= i

# Display the result
print(f"The factorial of {num} is {factorial}")

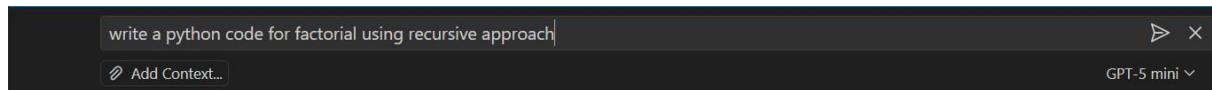
def calculate_factorial(n):
    # Initialize factorial to 1
    result = 1
    # Multiply all numbers from 1 to n
    for i in range(1, n + 1):
        result *= i
    # Return the calculated factorial
    return result


# Main block
if __name__ == "__main__":
    # Get user input
    num = int(input("Enter a number: "))
    # Call the function to calculate factorial
    factorial = calculate_factorial(num)
    # Display the result
    print(f"The factorial of {num} is {factorial}")
```

output:

```
4\bundled\libs\debugpy\launcher' '58210' '--' 'C:\Users\burra\OneDrive\문서\WPS Cloud Files\Desktop
Enter a number: 6
The factorial of 6 is 720
Enter a number: 
```

prompt 2



code

```
# Get user input
num = int(input("Enter a number: "))
# Call the function to calculate factorial iteratively
factorial = calculate_factorial(num)
# Initialize factorial result
factorial = 1

# Calculate factorial by multiplying numbers from 1 to num
for i in range(1, num + 1):
    factorial *= i

# Display the result
print(f"The factorial of {num} is {factorial}")

def calculate_factorial(n):
    # Initialize factorial to 1
    result = 1
    # Multiply all numbers from 1 to n
    for i in range(1, n + 1):
        result *= i
    # Return the calculated factorial
    return result

# Main block
if __name__ == "__main__":
    # Get user input
    num = int(input("Enter a number: "))
    # Call the function to calculate factorial
    factorial = calculate_factorial(num)
    # Display the result
    print(f"The factorial of {num} is {factorial}")
```

output

```
Enter a non-negative integer: 5  
120
```

Iterative version

The program starts with an initial value and repeatedly updates it using a loop until the condition is satisfied. The control flow stays within a single function and finishes once the loop ends.

Recursive version

The program calls itself with a smaller input each time. Each call waits on the stack until the base condition is reached. After that, results are returned step by step.

Aspect	Iterative	Recursive
Readability	Easy to follow for beginners	Slightly harder due to function calls
Stack usage	Uses constant memory	Uses call stack for each function call
Performance implications	Faster and memory efficient	Slower for large inputs due to stack overhead
When recursion is not recommended	Always safe for large inputs	Not recommended when input size is large or stack overflow is possible