AI Assisted Coding

ASSIGNMENT 1.2

Name: A. Tejasri

H.TNO: 2303A52455

BATCH – 31

Question 1:

Task 1:

AI-Generated Logic Without Modularisation (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 a modular design.

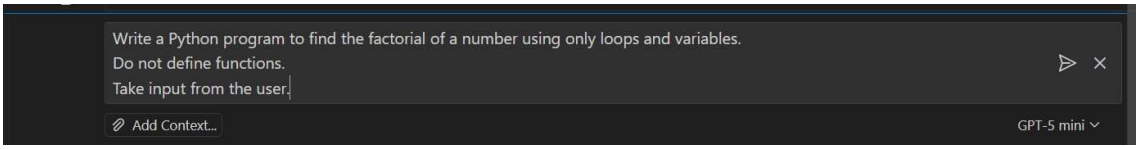
Task Description: Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

• Constraint:

Do not define any custom function

Logic must be implemented using loops and variables only

Prompt:



Code Screenshot:

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

factorial = 1

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

print(f"The factorial of {n} is {factorial}")
```

Output(O/P):

```
Enter a number: 15
The factorial of 15 is 1307674368000
PS D:\6th semester\Python>
```

Explanation:

Copilot was very helpful for a beginner as it quickly generated correct logic based on the prompt. It reduced the time required to write the code by suggesting a complete solution.

The generated code followed best practices such as input validation and edge case handling. Copilot also helped in avoiding syntax errors and logical mistakes.

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

Reduce unnecessary variables

Improve loop clarity

Enhance readability and efficiency

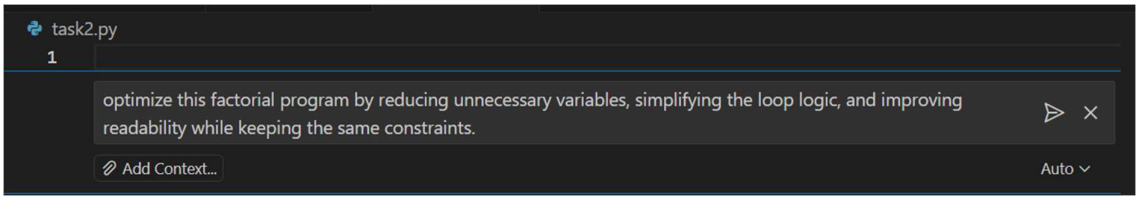
Hint: Prompt Copilot with phrases like

“optimise this code”,

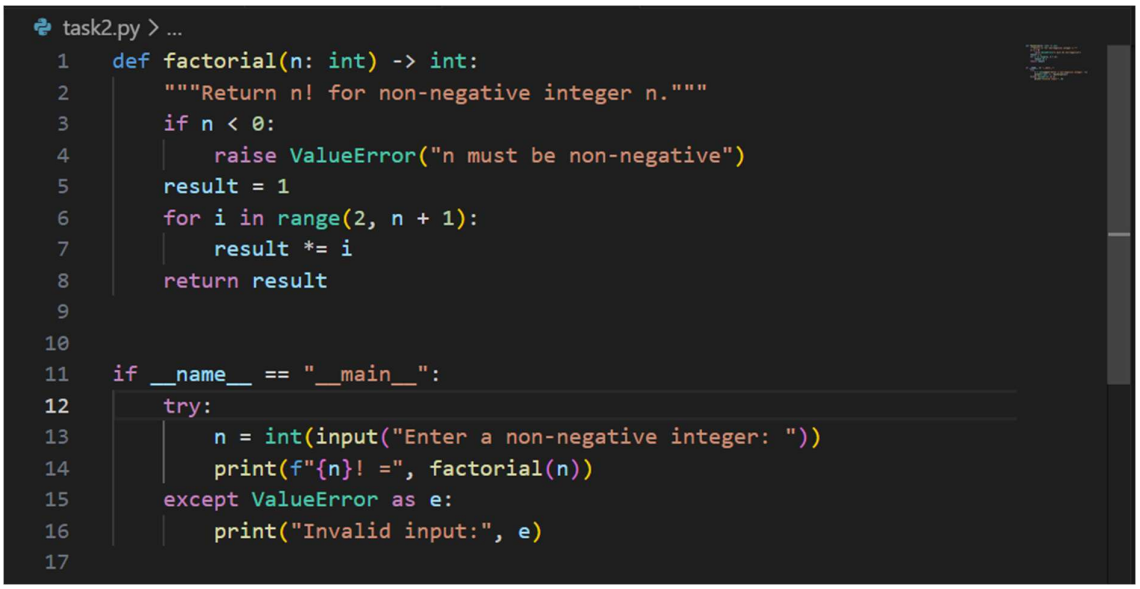
“simplify logic”, or

“Make it more readable”

Prompt:



Code Screenshot:



Output(O/P):

```
Enter a non-negative integer: 12
12! = 479001600
PS D:\6th semester\Python>
```

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.

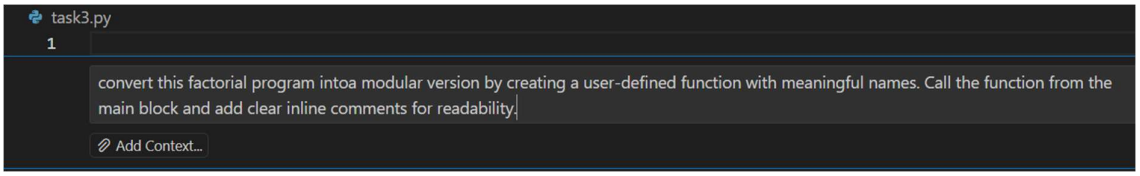
Task Description Use GitHub Copilot to generate a modular version of the program by:

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)

Prompt:



Code Screenshot:

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

    Args:
        number: An integer for which factorial is to be calculated

    Returns:
        The factorial of the given number
    """
    # Initialize result to 1
    result = 1

    # Multiply result by each number from 1 to the given number
    for i in range(1, number + 1):
        result *= i

    return result


# Main block to execute the program
if __name__ == "__main__":
    # Prompt user to input a number
    user_input = int(input("Enter a number to calculate its factorial: "))

    # Call the factorial function with user input
    factorial_result = calculate_factorial(user_input)

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

```

Output(O/P):

```
PS D:\6th semester\Python> python task3.py
Enter a number to calculate its factorial: 16
The factorial of 16 is: 20922789888000
```

Explanation:

Copilot Modified The Code To Function Type

& main block is created

Handled Edge Cases Well

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:

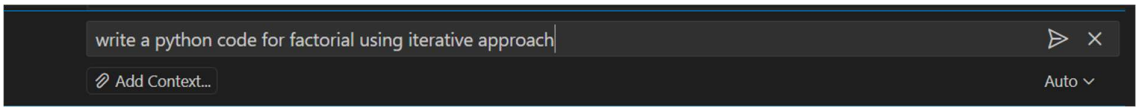
Your mentor wants to test how well AI understands different computational paradigms.

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

Prompt 1:



Code:

```python
def factorial(n):
    """
    Calculate factorial of n using iterative approach.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    result = 1
    for i in range(2, n + 1):
        result *= i

    return result


# Test the function
if __name__ == "__main__":
    num = 5
    print(f"Factorial of {num} is {factorial(num)}")

    num = 0
    print(f"Factorial of {num} is {factorial(num)}")

    num = 10
    print(f"Factorial of {num} is {factorial(num)}")

```

Output:

```
PS D:\6th semester\Python> python task4.py
Factorial of 5 is 120
Factorial of 0 is 1
Factorial of 10 is 3628800
```

Prompt 2:

Code:

```python
def factorial(n):
    """
    Calculate factorial of n using recursive approach.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n
    """
    # Base case
    if n == 0 or n == 1:
        return 1
    # Recursive case
    else:
        return n * factorial(n - 1)


# Test the factorial function
if __name__ == "__main__":
    # Get input from user
    num = int(input("Enter a number to find its factorial: "))

    # Validate input
    if num < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        result = factorial(num)
        print(f"The factorial of {num} is {result}")

```

Output:

```
PS D:\6th semester\Python> python task5.py
Enter a number to find its factorial: 8
The factorial of 8 is 40320
```

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