

AI Assisted Coding

ASSIGNMENT 1.2

Name: Madadi Uday

H.TNO: 2303A52038

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.

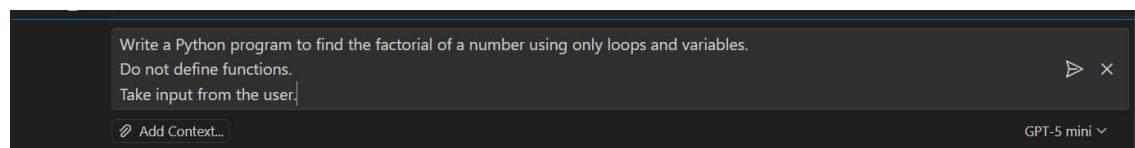
Task Description: Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.

- Constraint:

Do not define any custom function

Logic must be implemented using loops and variables only

Prompt:



Code Screenshot:

```
⚡ lab1_2.py > ...
1   n = int(input("Enter a number: "))
2
3   factorial = 1
4
5   i = 1
6   while i <= n:
7       factorial = factorial * i
8       i = i + 1
9
10  print(f"The factorial of {n} is {factorial}")
11
```

Output(O/P):

```
PS D:\6th semester\Python> python lab1_2.py
Enter a number: 23
The factorial of 23 is 25852016738884976640000
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

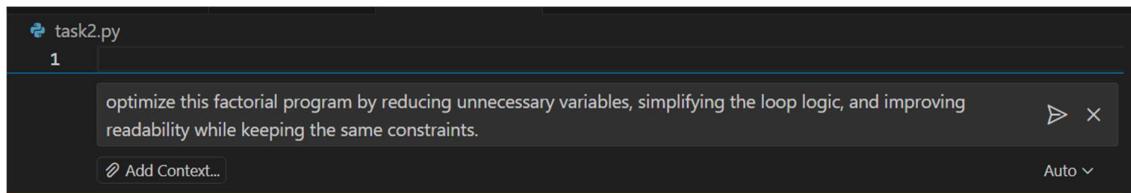
Hint: Prompt Copilot with phrases like

“optimise this code”,

“simplify logic”, or

“Make it more readable”

Prompt:



Code Screenshot:

```
task2.py > ...
1  def factorial(n: int) -> int:
2      """Return n! for non-negative integer n."""
3      if n < 0:
4          raise ValueError("n must be non-negative")
5      result = 1
6      for i in range(2, n + 1):
7          result *= i
8      return result
9
10
11 if __name__ == "__main__":
12     try:
13         n = int(input("Enter a non-negative integer: "))
14         print(f"{n}! =", factorial(n))
15     except ValueError as e:
16         print("Invalid input:", e)
17
```

Output(O/P):

```
Enter a non-negative integer: 9  
9! = 362880
```

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:

The screenshot shows the GitHub Copilot interface. A file named "task3.py" is open in the code editor. The first line of code is "1". Below the code editor is a prompt box containing the following text:
convert this factorial program into a modular version by creating a user-defined function with meaningful names. Call the function from the main block and add clear inline comments for readability.
At the bottom of the prompt box is a button labeled "Add Context...".

Code Screenshot:

```
1 # Function to calculate factorial of a given number
2 def calculate_factorial(number):
3     """
4     Calculate the factorial of a given number.
5
6     Args:
7         number: An integer for which factorial is to be calculated
8
9     Returns:
10        The factorial of the given number
11    """
12
13    # Initialize result to 1
14    result = 1
15
16    # Multiply result by each number from 1 to the given number
17    for i in range(1, number + 1):
18        result *= i
19
20    return result
21
22 # Main block to execute the program
23 if __name__ == "__main__":
24     # Prompt user to input a number
25     user_input = int(input("Enter a number to calculate its factorial: "))
26
27     # Call the factorial function with user input
28     factorial_result = calculate_factorial(user_input)
29
30     # Display the result to the user
31     print(f"The factorial of {user_input} is: {factorial_result}")
32
```

Output(O/P):

```
Enter a number to calculate its factorial: 8
The factorial of 8 is: 40320
```

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:

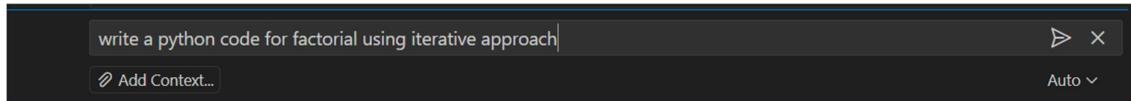
Prompt Copilot to generate:

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:



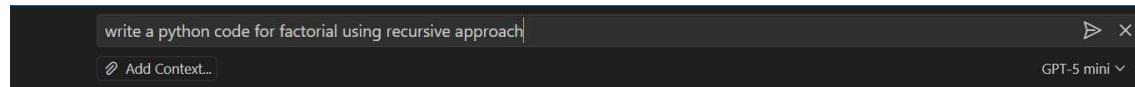
Code:

```
1  def factorial(n):
2      """
3          Calculate factorial of n using iterative approach.
4
5      Args:
6          n: A non-negative integer
7
8      Returns:
9          The factorial of n
10     """
11    if n < 0:
12        raise ValueError("Factorial is not defined for negative numbers")
13
14    result = 1
15    for i in range(2, n + 1):
16        result *= i
17
18    return result
19
20
21 # Test the function
22 if __name__ == "__main__":
23     num = 5
24     print(f"Factorial of {num} is {factorial(num)}")
25
26     num = 0
27     print(f"Factorial of {num} is {factorial(num)}")
28
29     num = 10
30     print(f"Factorial of {num} is {factorial(num)}")
31
```

Output:

```
Factorial of 4 is 24
Factorial of 7 is 5040
Factorial of 0 is 1
```

Prompt 2:



Code:

```
1  def factorial(n):
2      """
3          Calculate factorial of n using recursive approach.
4      """
5      Args:
6          n: A non-negative integer
7
8      Returns:
9          The factorial of n
10     """
11
12     # Base case
13     if n == 0 or n == 1:
14         return 1
15     # Recursive case
16     else:
17         return n * factorial(n - 1)
18
19
20     # Test the factorial function
21     if __name__ == "__main__":
22         # Get input from user
23         num = int(input("Enter a number to find its factorial: "))
24
25         # Validate input
26         if num < 0:
27             print("Factorial is not defined for negative numbers.")
28         else:
29             result = factorial(num)
30             print(f"The factorial of {num} is {result}")
```

Output:

```
Enter a number to find its factorial: 11
The factorial of 11 is 39916800
```

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