AI ASSITED CODING

ASSIGNMENT-1.2
2303A52341
BATCH-32

Question 1:

Task 1:

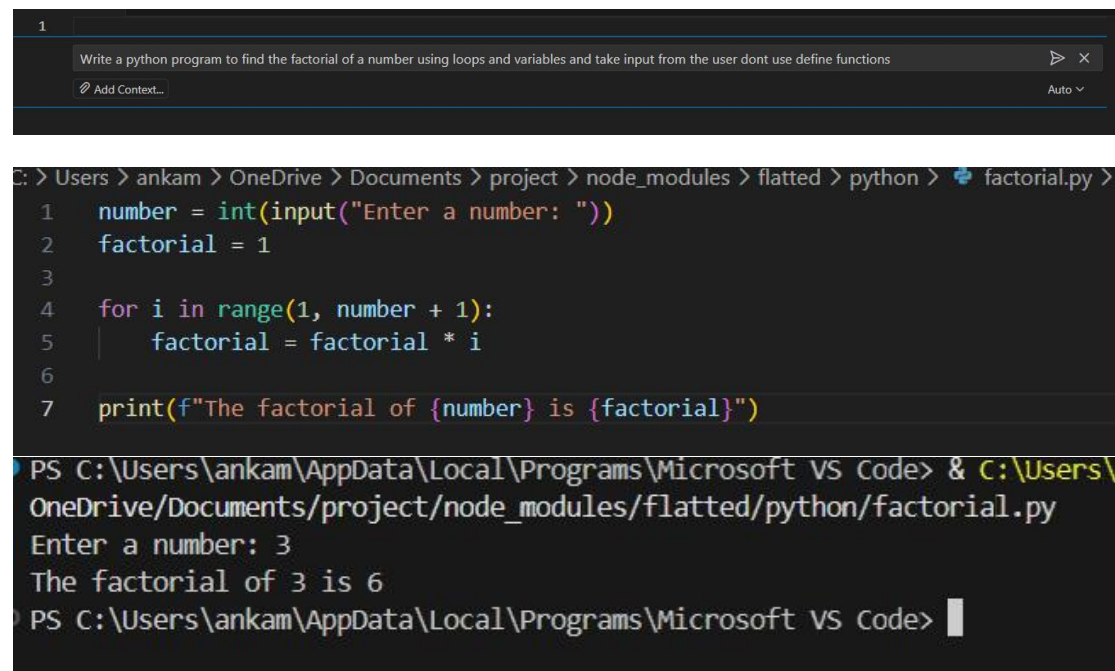AI-Generated Logic Without Modularization (Factorial without Functions)

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

```
1

Write a python program to find the factorial of a number using loops and variables and take input from the user dont use define functions
Add Context...                                                                                                    Auto ∨
```

```
C: > Users > ankam > OneDrive > Documents > project > node_modules > flatted > python > 🐍 factorial.py >
1    number = int(input("Enter a number: "))
2    factorial = 1
3
4    for i in range(1, number + 1):
5        factorial = factorial * i
6
7    print(f"The factorial of {number} is {factorial}")
```

```
PS C:\Users\ankam\AppData\Local\Programs\Microsoft VS Code> & C:\Users\.
OneDrive/Documents/project/node_modules/flatted/python/factorial.py
Enter a number: 3
The factorial of 3 is 6
PS C:\Users\ankam\AppData\Local\Programs\Microsoft VS Code>
```

Explanation:

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 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"

```python
#write a python code to remove unncessary variables, improve loops clearly,
# # enhance readability and efficiency

try:
    n = int(input("Enter a "))
    if n < 0:
        print("Factorial is not defined for negative numbers.")
    else:
        factorial = 1
        for i in range(2, n + 1):
            factorial *= i
        print(f"The factorial of {n} is: {factorial}")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

```
Enter a 5
The factorial of 5 is: 120
```

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)

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.

Add Context...   GPT-5 mini ∨

```python
1    def get_number_from_user():
2        """Get a positive integer from the user."""
3        return int(input("Enter a number: "))
4
5    def calculate_factorial(n):
6        """Calculate and return the factorial of n."""
7        if n < 0:
8            return None
9        if n == 0 or n == 1:
10           return 1
11       result = 1
12       for i in range(2, n + 1):
13           result *= i
14       return result
15
16   def display_result(number, factorial):
17       """Display the factorial result."""
18       print(f"Factorial of {number} is {factorial}")
19
20   def main():
21       """Main function to orchestrate the program."""
22       number = get_number_from_user()
23       factorial = calculate_factorial(number)
24       if factorial is not None:
25           display_result(number, factorial)
26       else:
27           print("Error: Please enter a non-negative number.")
28
29   if __name__ == "__main__":
30       main()
```

```
PS C:\Users\ankam\AppData\Local\Programs\Microsoft VS Code> C:\Users\a
eDrive/Documents/project/node_modules/flatted/python/convertedfact.py
Enter a number: 6
Factorial of 6 is 720
PS C:\Users\ankam\AppData\Local\Programs\Microsoft VS Code> []
```

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

| 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

```
write a python program in iterative version of the factorial program

Add Context...
```
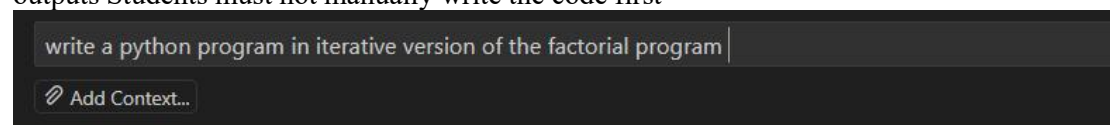
> Users > ankam > OneDrive > Documents > project > node_modules > flatted > python >

```python
1   # Iterative Factorial Program
2
3   def factorial(n):
4       if n < 0:
5           return "Factorial is not defined for negative numbers"
6       elif n == 0 or n == 1:
7           return 1
8       else:
9           result = 1
10          for i in range(2, n + 1):
11              result *= i
12          return result
13
14  # Take input from user
15  try:
16      num = int(input("Enter a number to find factorial: "))
17      print(f"Factorial of {num} is: {factorial(num)}")
18  except ValueError:
19      print("Please enter a valid integer")
```

```
ject/node_modules/flatted/python/iterative.py
Enter a number to find factorial: 5
Factorial of 5 is: 120
PS C:\Users\ankam\AppData\Local\Programs\Microsoft VS Code> & C:\Us
/project/node_modules/flatted/python/iterative.py
Enter a number to find factorial: -1
Factorial of -1 is: Factorial is not defined for negative numbers
PS C:\Users\ankam\AppData\Local\Programs\Microsoft VS Code>
```

generate a python program in recursive version for the factorial of a number by taking input user friendly

🔗 Add Context...

```
> Users > ankam > OneDrive > Documents > project > node_modules > flatted > python > 🐍 recursive.py > ...
1    def factorial(n):
2        """Calculate factorial of n using recursion"""
3        if n < 0:
4            return "Error: Factorial is not defined for negative numbers"
5        elif n == 0 or n == 1:
6            return 1
7        else:
8            return n * factorial(n - 1)
9
10   # Main program
11   try:
12       num = int(input("Enter a number to calculate its factorial: "))
13       result = factorial(num)
14       print(f"The factorial of {num} is: {result}")
15   except ValueError:
16       print("Error: Please enter a valid integer")
```

```
Enter a number to calculate its factorial: 5
The factorial of 5 is: 120
PS C:\Users\ankam\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ankam\
/project/node_modules/flatted/python/recursive.py
Enter a number to calculate its factorial: -2
The factorial of -2 is: Error: Factorial is not defined for negative numbers
PS C:\Users\ankam\AppData\Local\Programs\Microsoft VS Code>
```

Different between iterative and recursive

Aspect	Iterative	Recursive
Readability	Easy to follow	Cleaner but less obvious
Stack usage	No extra stack	Uses call stack
Performance	Faster and efficient	Slower due to overhead
When recursion not recommended	–	Large inputs, deep calls