

School of Computer Science and Artificial Intelligence

Lab Assignment # 1

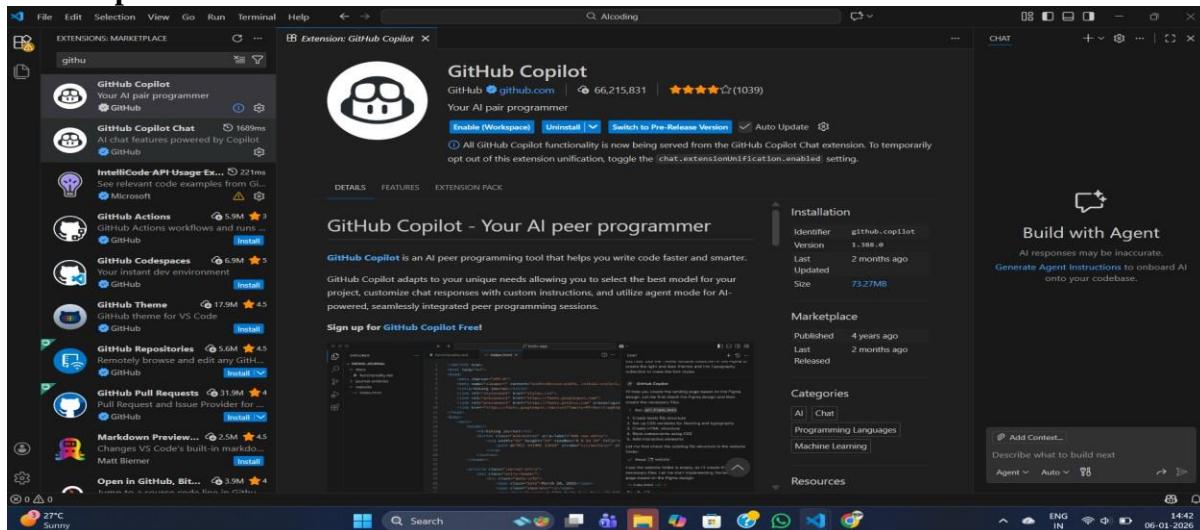
Program	: B. Tech (CSE)
Specialization	:
Course Title	: AI Assisted coding
Course Code	:
Semester	II
Academic Session	: 2025-2026
Name of Student	: R.Akash Reddy
Enrollment No.	: 2403A51L30
Batch No.	51
Date	:06-01-2026

Submission Starts here

OUTPUT :

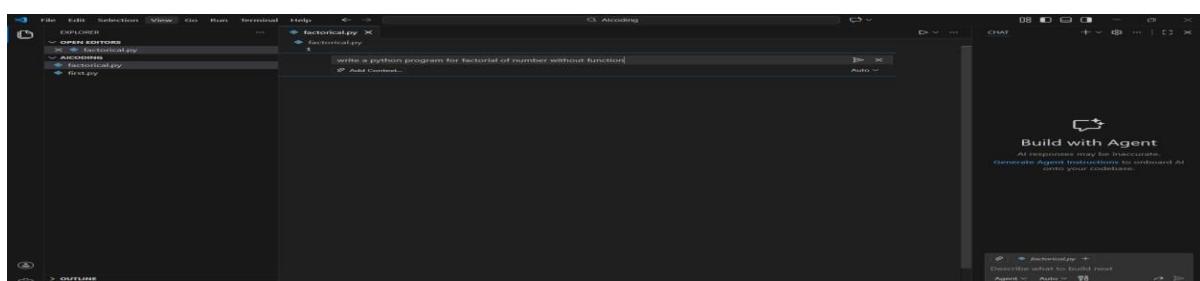
SCREENSHOTS:

Task 0: Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



Task1: 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.



The screenshot shows the VS Code interface with the AI Coding extension active. The Explorer sidebar shows files factorial.py, first.py, and AI CODING. The main editor window contains the following Python code:

```

factorial.py >_
1 num = int(input("Enter a number: "))
2 factorial = 1
3
4 if num < 0:
5     print("Factorial is not defined for negative numbers")
6 elif num == 0:
7     print("Factorial of 0 is 1")
8 else:
9     for i in range(1, num + 1):
10        factorial *= i
11    print(f"Factorial of {num} is {factorial}")

```

The Chat panel on the right has a message "Build with Agent" with the note "AI responses may be inaccurate." and a link "Generate Agent Instructions". A small AI icon is shown above the message.

This screenshot shows the same VS Code setup as the previous one, but the terminal tab at the bottom is active, displaying the output of the generated code:

```

PS C:\Users\nanip\OneDrive\Desktop\AIcoding> & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nanip/OneDrive/Desktop/AIcoding/factorial.py
Enter a number: 5
Factorial of 5 is 120

```

The Chat panel remains the same, showing the "Build with Agent" message and the AI icon.

- ❖ The Copilot is very helpful because we can generate code by just giving a prompt in Copilot Chat (**ctrl + I**)
- ❖ The code generated was as requested in the prompt

TASK - 2

Task Description

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

- ❖ Reduce unnecessary variables
- ❖ Improve loop clarity
- ❖ Enhance readability and efficiency

```

File Edit Selection View Go Run Terminal Help ⏎ ⏎ ...
EXPLORER factorial.py ...
OPEN EDITORS factorial.py ...
ACODING -Stch_51_Naniprasad_2403A51L11_SESD_A7...
-WRL0001.tmp Batch_51_Naniprasad_2403A51L11_SESD_A7...
factorial.py first.py
factorial.py > ...
1 n=int(input())
2 fact=1
3 for i in range(1,n+1):
4     fact*=i
5 print("the factorial of {} is {}".format(n,fact))
6

```

Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.

PROBLEMS PORTS TERMINAL OUTPUT DEBUG CONSOLE

PS C:\Users\nanip\OneDrive\Desktop\Acoding>

OUTLINE TIMELINE

18°C Search

Python + ...

Ln 2, Col 7 Spaces: 4 UTF-8 CRLF { } Python 3.13.7 ENG 19:28

```

File Edit Selection View Go Run Terminal Help ⏎ ⏎ ...
EXPLORER factorial.py ...
OPEN EDITORS factorial.py ...
ACODING -Stch_51_Naniprasad_2403A51L11_SESD_A7...
-WRL0001.tmp Batch_51_Naniprasad_2403A51L11_SESD_A7...
factorial.py first.py
factorial.py > ...
1 n=int(input())
2 fact=1
3 for i in range(1,n+1):
4     fact*=i
5 print("the factorial of {} is {}".format(n,fact))
6

```

Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.

PROBLEMS PORTS TERMINAL OUTPUT DEBUG CONSOLE

PS C:\Users\nanip\OneDrive\Desktop\Acoding> & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nanip/OneDrive/Desktop/Acoding/factorial.py
5
the factorial of 5 is 120
PS C:\Users\nanip\OneDrive\Desktop\Acoding>

OUTLINE TIMELINE

18°C Clear Search

Python Python

Python + ...

Ln 2, Col 7 Spaces: 4 UTF-8 CRLF { } Python 3.13.7 ENG 19:28 06-01-2026

What was improved?

- Shorter multiplication statement
- **factorial = factorial * i** → **factorial *= i**

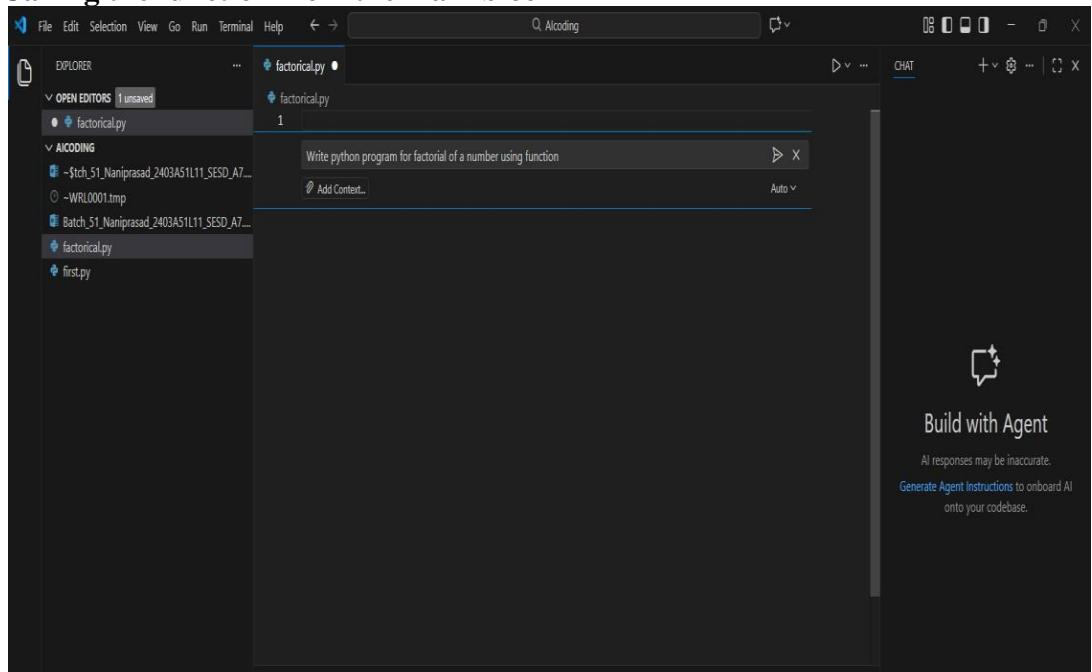
- ❖ The loop logic is self-explanatory, so the comment was removed.
- ❖ # Why the new version is better?
- ❖ Readability
 - ❖ *= is clearer and more concise.
 - Fewer lines and less clutter make the code easier to read.
- ❖ Maintainability
 - Cleaner code is easier to modify and debug.
 - Reduced redundancy lowers the chance of mistakes.
- ❖ Performance
 - Performance is effectively the same.
- ❖ *= is marginally optimized at the bytecode level, but the difference is negligible.

Task3

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



```

1 def factorial(n):
2     """Calculate the factorial of a number."""
3     if n < 0:
4         return "Factorial is not defined for negative numbers"
5     elif n == 0 or n == 1:
6         return 1
7     else:
8         result = 1
9         for i in range(2, n + 1):
10            result *= i
11    return result
12
13
14 # Get input from user
15 num = int(input("Enter a number: "))
16 print(f"Factorial of {num} is: {factorial(num)}")

```

The screenshot shows a code editor window with the following details:

- File Explorer:** Shows files like `factorial.py` and `first.py`.
- Code Editor:** Displays the Python code for calculating factorials.
- AI Coding Panel:** A sidebar titled "Build with Agent" with instructions to "Generate Agent Instructions to onboard AI onto your codebase."
- Terminal:** Shows the command `python factorial.py` being run and the output: "Factorial of 5 is: 120".
- System Tray:** Shows the date and time as 06-01-2026, 19:36.

❖ **Modularity improves reusability by:**

Allowing the factorial() function to be reused in multiple programs without rewriting code.

Making the program easier to test, update, and debug.

Improving code organization, where logic is separated from input/output handling.

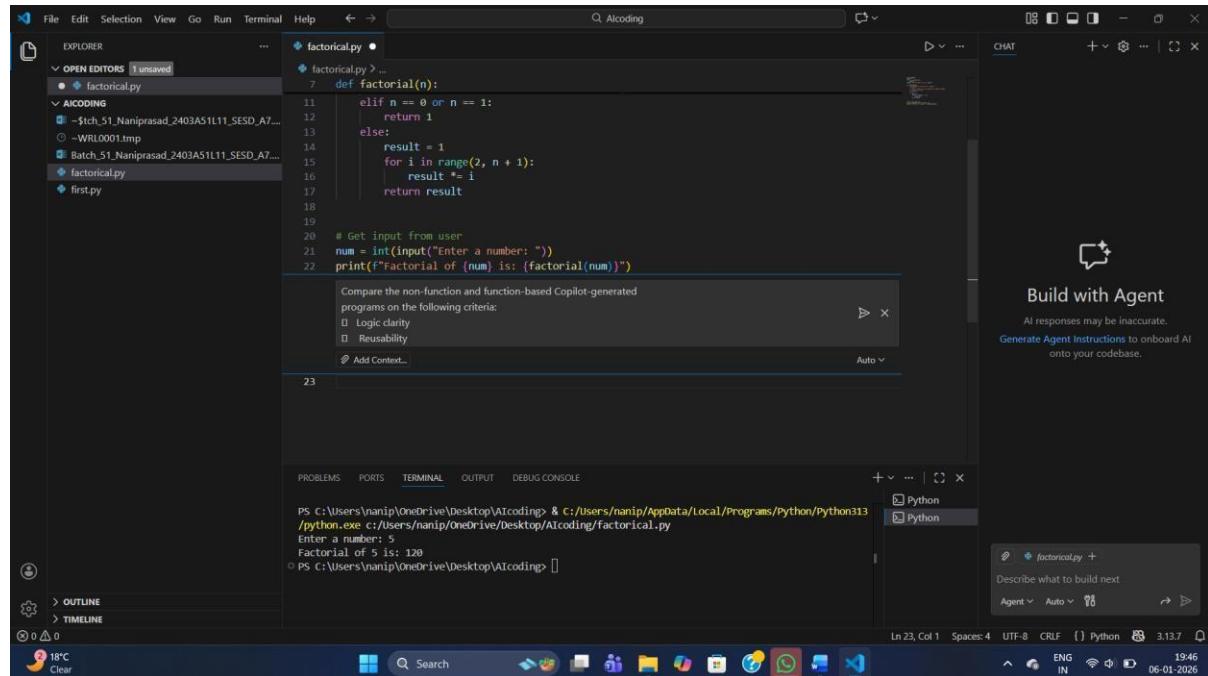
Supporting scalability, as the same function can be extended or integrated into larger projects.

#Task4

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



The screenshot shows two versions of a factorial program in the Explorer pane. The first version is a non-function-based approach:

```

factorial.py > ...
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
    
```

The second version is a function-based approach:

```

factorial.py > ...
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# Get input from user
num = int(input("Enter a number: "))
print("Factorial of {} is: {}".format(num, factorial(num)))
    
```

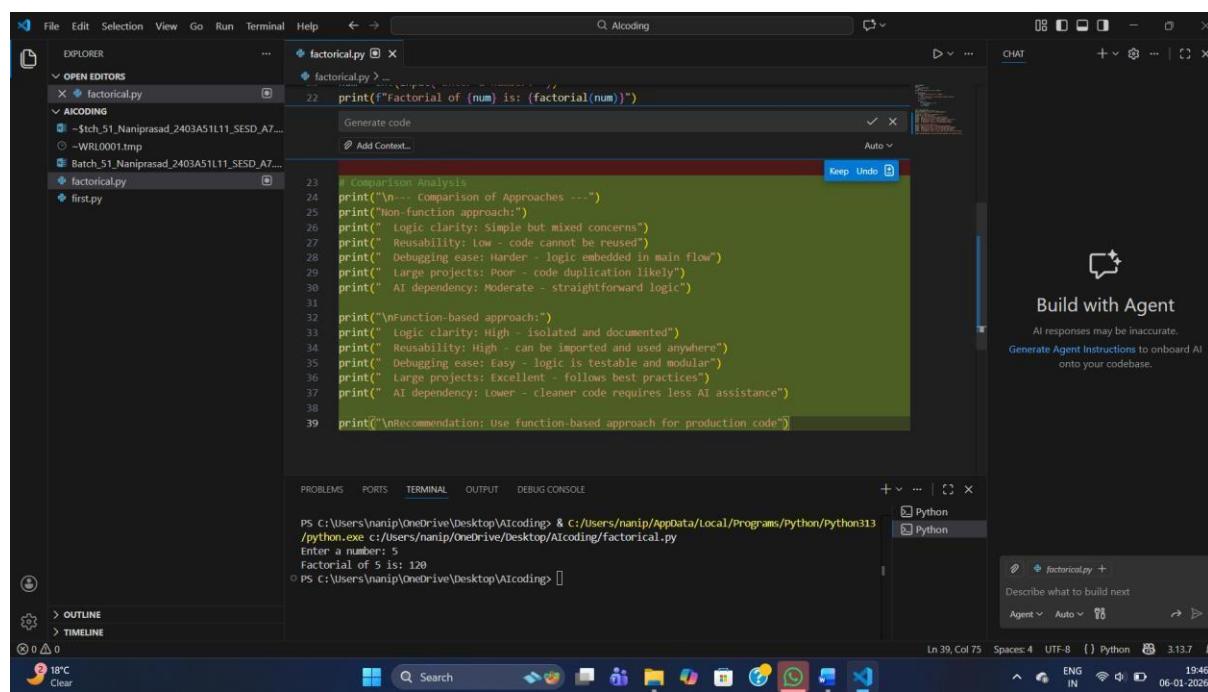
A tooltip in the center of the screen compares the two approaches based on the following criteria:

- Logic clarity
- Reusability
- Add Context...

The terminal pane shows the execution of both programs. The output for the function-based approach is:

```

PS C:\Users\nanip\OneDrive\Desktop\AIcoding> & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe C:/Users/nanip/OneDrive/Desktop/AIcoding/factorial.py
Enter a number: 5
Factorial of 5 is: 120
PS C:\Users\nanip\OneDrive\Desktop\AIcoding> []
    
```



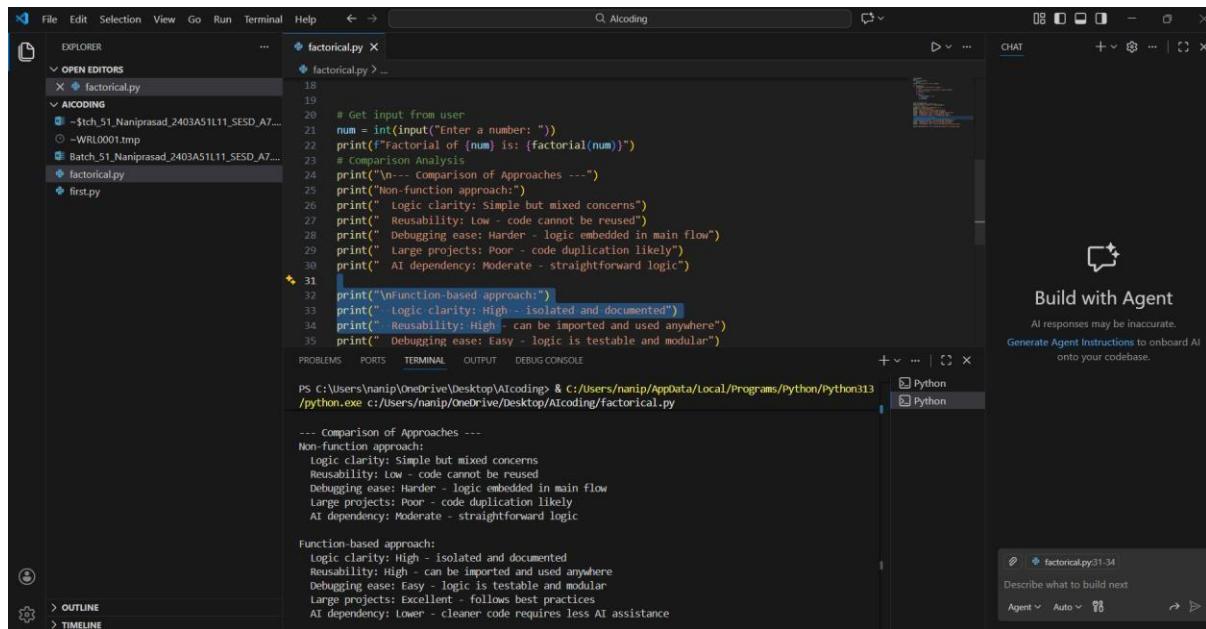
The screenshot shows the comparison analysis and recommendation for the factorial programs. The analysis highlights the following points:

- # Comparison Analysis
- 22 print("Factorial of {} is: {}".format(factorial(num)))
- 23 # Comparison Analysis
- 24 print("\n--- comparison of Approaches ---")
- 25 print("Non-function approach:")
- 26 print(" Logic clarity: Simple but mixed concerns")
- 27 print(" Reusability: low - code cannot be reused")
- 28 print(" Debugging ease: Harder - logic embedded in main flow")
- 29 print(" Large projects: Poor - code duplication likely")
- 30 print(" AI dependency: Moderate - straightforward logic")
- 31
- 32 print("\nFunction-based approach:")
- 33 print(" Logic clarity: High - isolated and documented")
- 34 print(" Reusability: High - can be imported and used anywhere")
- 35 print(" Debugging ease: Easy - logic is testable and modular")
- 36 print(" Large projects: Excellent - follows best practices")
- 37 print(" AI dependency: Lower - cleaner code requires less AI assistance")
- 38
- 39 print("\nRecommendation: Use function-based approach for production code")

The terminal pane shows the execution of the function-based approach. The output is:

```

PS C:\Users\nanip\OneDrive\Desktop\AIcoding> & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe C:/Users/nanip/OneDrive/Desktop/AIcoding/factorial.py
Enter a number: 5
Factorial of 5 is: 120
PS C:\Users\nanip\OneDrive\Desktop\AIcoding> []
    
```



The screenshot shows the VS Code interface with the 'AI CODING' extension active. The left sidebar shows files like 'factorial.py', 'first.py', and 'Batch_51_Naniprasad_2403A51L11_SESD_A7....'. The main editor window displays Python code for calculating factorials, with annotations for logic clarity, reusability, debugging ease, and AI dependency. The terminal below shows the command run: PS C:\Users\nanip\OneDrive\Desktop\AIcoding> & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nanip/OneDrive/Desktop/AIcoding/factorial.py. The right sidebar features a 'Build with Agent' panel.

```

18
19
20 # Get input from user
21 num = int(input("Enter a number: "))
22 print(f"Factorial of {num} is: {factorial(num)}")
23 # Comparison Analysis
24 print("\n--- Comparison of Approaches ---")
25 print("Non-function approach:")
26 print(" Logic clarity: Simple but mixed concerns")
27 print(" Reusability: Low - code cannot be reused")
28 print(" Debugging ease: Harder - logic embedded in main flow")
29 print(" Large projects: Poor - code duplication likely")
30 print(" AI dependency: Moderate - straightforward logic")
31
32 print("\nFunction-based approach:")
33 print(" Logic clarity: High - isolated and documented")
34 print(" Reusability: High - can be imported and used anywhere")
35 print(" Debugging ease: Easy - logic is testable and modular")
36
37 --- Comparison of Approaches ---
38 Non-function approach:
39   Logic clarity: Simple but mixed concerns
40   Reusability: Low - code cannot be reused
41   Debugging ease: Harder - logic embedded in main flow
42   Large projects: Poor - code duplication likely
43   AI dependency: Moderate - straightforward logic
44
45 Function-based approach:
46   Logic clarity: High - isolated and documented
47   Reusability: High - can be imported and used anywhere
48   Debugging ease: Easy - logic is testable and modular
49   Large projects: Excellent - follows best practices
50   AI dependency: Lower - cleaner code requires less AI assistance

```

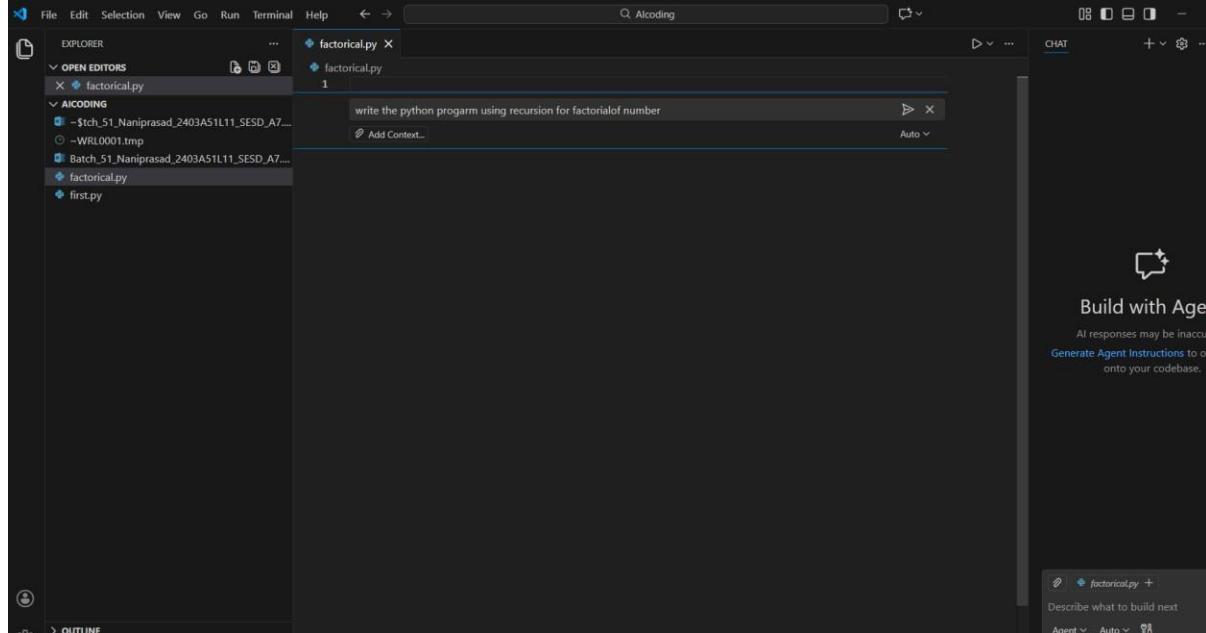
TASK - 5

Task Description

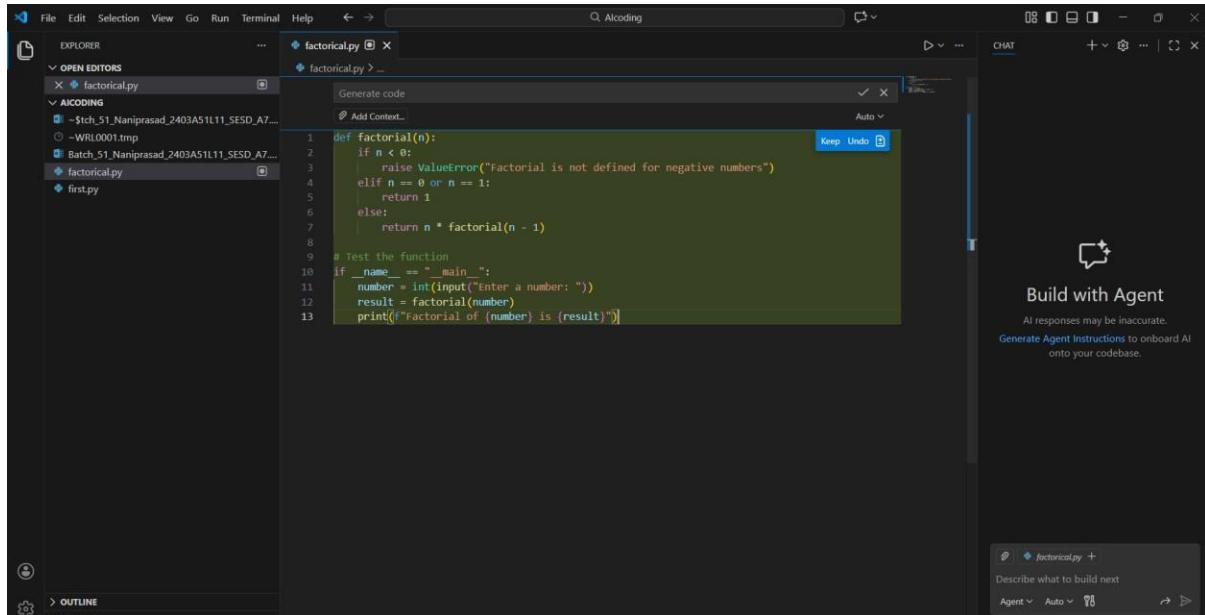
Prompt Copilot to generate:

An iterative version of the logic

A recursive version of the same logic



The screenshot shows the VS Code interface with the 'AI CODING' extension active. The left sidebar shows files like 'factorial.py', 'first.py', and 'Batch_51_Naniprasad_2403A51L11_SESD_A7....'. The main editor window has the text 'write the python program using recursion for factorialof number' entered. The right sidebar features a 'Build with Agent' panel.

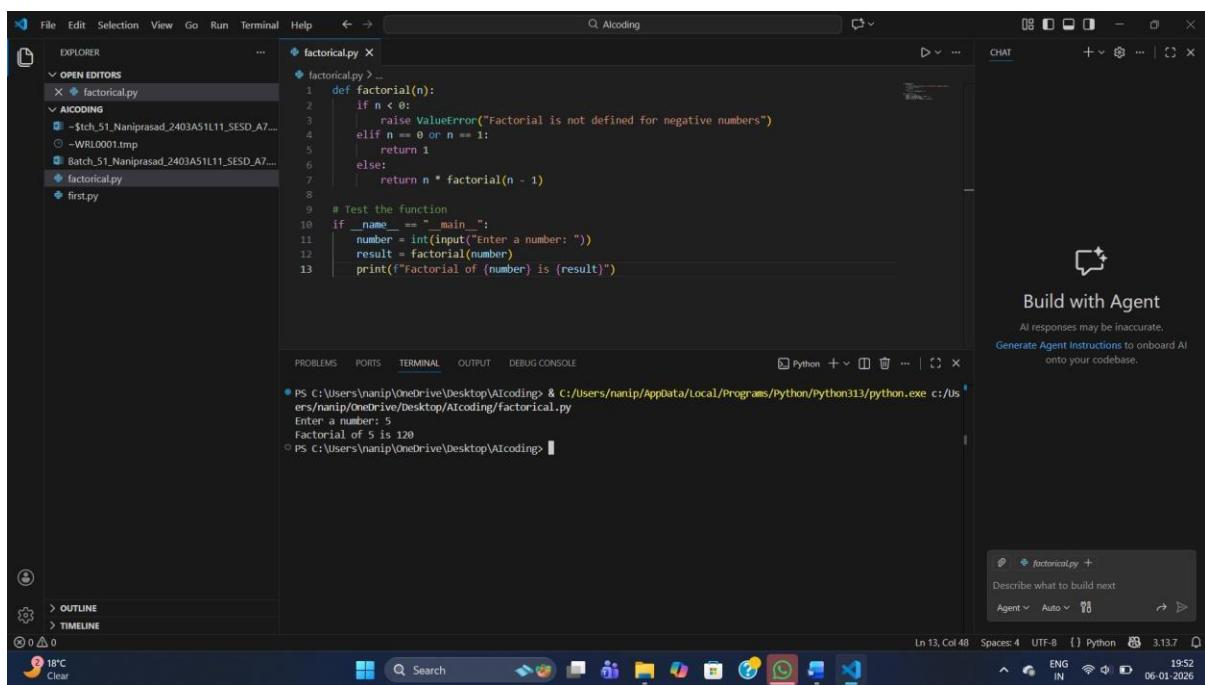


```

def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Test the function
if __name__ == "__main__":
    number = int(input("Enter a number: "))
    result = factorial(number)
    print(f"Factorial of {number} is {result}")

```



```

def factorial(n):
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Test the function
if __name__ == "__main__":
    number = int(input("Enter a number: "))
    result = factorial(number)
    print(f"Factorial of {number} is {result}")

```

PROBLEMS PORTS TERMINAL OUTPUT DEBUG CONSOLE

PS C:\Users\nanip\OneDrive\Desktop\Alcoding> & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nanip/OneDrive/Desktop/Alcoding/factorial.py
Enter a number: 5
Factorial of 5 is 120
PS C:\Users\nanip\OneDrive\Desktop\Alcoding>

Explanation:

How the Function Works

1. Negative number check

Factorials are not defined for negative numbers. If the input is negative, the program raises an error message.

2. Base cases

For 0 and 1, the factorial is defined as 1. This acts as the stopping condition for recursion.

3. Recursive case

For numbers greater than 1, the function calls itself with $n-1$. This recursive process continues until it reaches the base case.

Example:

- To compute $5!$, the function calculates $5 \times 4!$.

- Then $4!$ becomes $4 \times 3!$, and so on, until it reaches $1!$.
 - Main Program Flow
 - The program asks the user to enter a number.
 - It then calls the factorial function with that number.
 - Finally, it prints the result in a clear message.
 - Example Execution
- If the user enters 5:
- The recursive calls break it down step by step until reaching 1.
 - The final result is 120.
- So the program outputs: *Factorial of 5 is 120.*
- Summary
- This program demonstrates:
- Recursion (function calling itself).
 - Error handling (for negative inputs).
 - Base cases (to stop recursion).
 - User interaction (taking input and displaying output).

