

School of Computer Science and Artificial Intelligence

Lab Assignment # 1

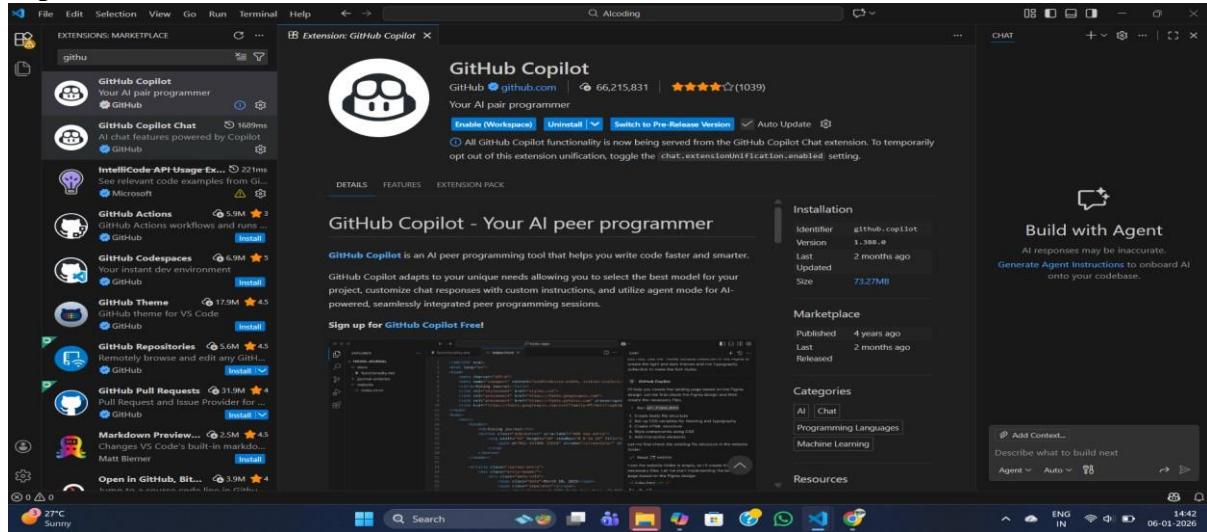
Program	: B. Tech (CSE)
Specialization	:
Course Title	: AI Assisted coding
Course Code	:
Semester	: II
Academic Session	: 2025-2026
Name of Student	: Nagasai Adepu
Enrollment No.	: 2403A51L18
Batch No.	: 51
Date	: 06-01-2026

Submission Starts here

OUTPUT :

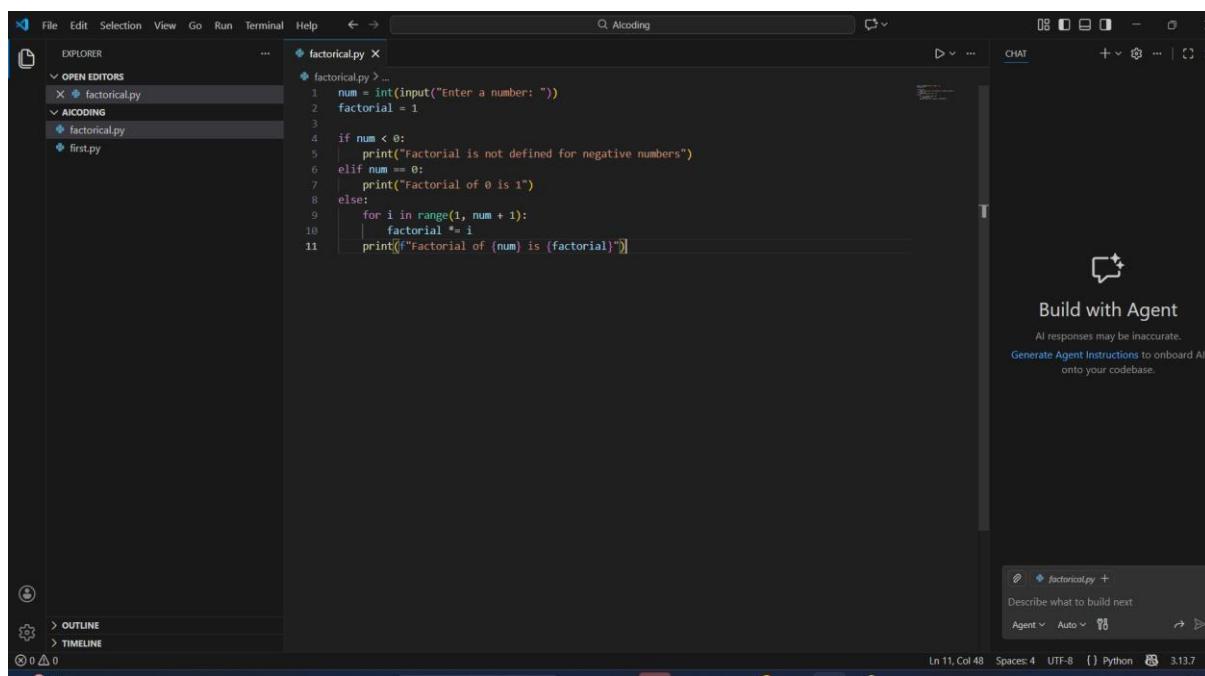
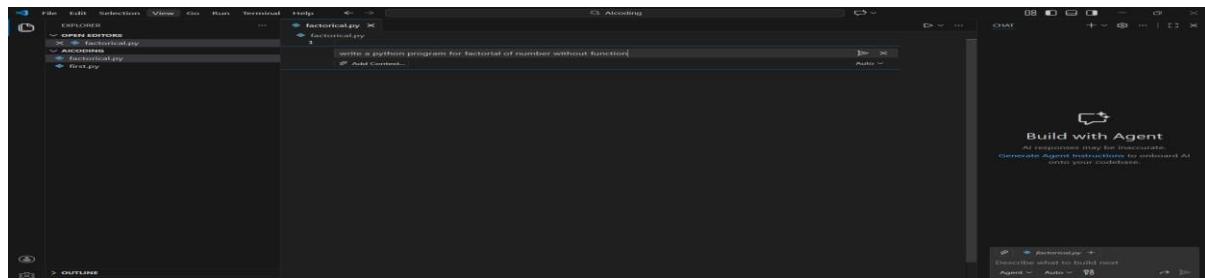
SCREENSHOTS:

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 following details:

- File Explorer:** Shows files: factorial.py, factorial.py > ..., first.py.
- Code Editor:** Displays Python code for calculating factorial:

```

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}")

```
- Terminal:** Shows command-line output for factorial.py:

```

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>

```
- AI Coding Panel:** A floating panel titled "Build with Agent" contains a text input field "Describe what to build next" and dropdowns for "Agent" and "Auto".
- Bottom Status Bar:** Shows file paths, encoding (UTF-8), and other system information.

- ❖ 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

```

factorial.py > ...
1 n=int(input())
2 fact=1
3 for i in range(1,n+1):
4     fact*=i
5 print(f"the factorial of {n} is {fact}")
6

```

The screenshot shows the VS Code interface with the factorial.py file open in the editor. The code calculates the factorial of a number input by the user. A 'Build with Agent' feature is visible in the bottom right corner.

```

factorial.py > ...
1 n=int(input())
2 fact=1
3 for i in range(1,n+1):
4     fact*=i
5 print(f"the factorial of {n} is {fact}")
6

```

The screenshot shows the VS Code interface with the same factorial.py file, but now it includes a terminal output window at the bottom. The terminal shows the command being run and the resulting output: "the factorial of 5 is 120". The 'Build with Agent' feature is also present.

What was improved?

- Shorter multiplication statement
- **factorial = factorial * i → factorial *= i**
- Removed unnecessary comment

❖ 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.

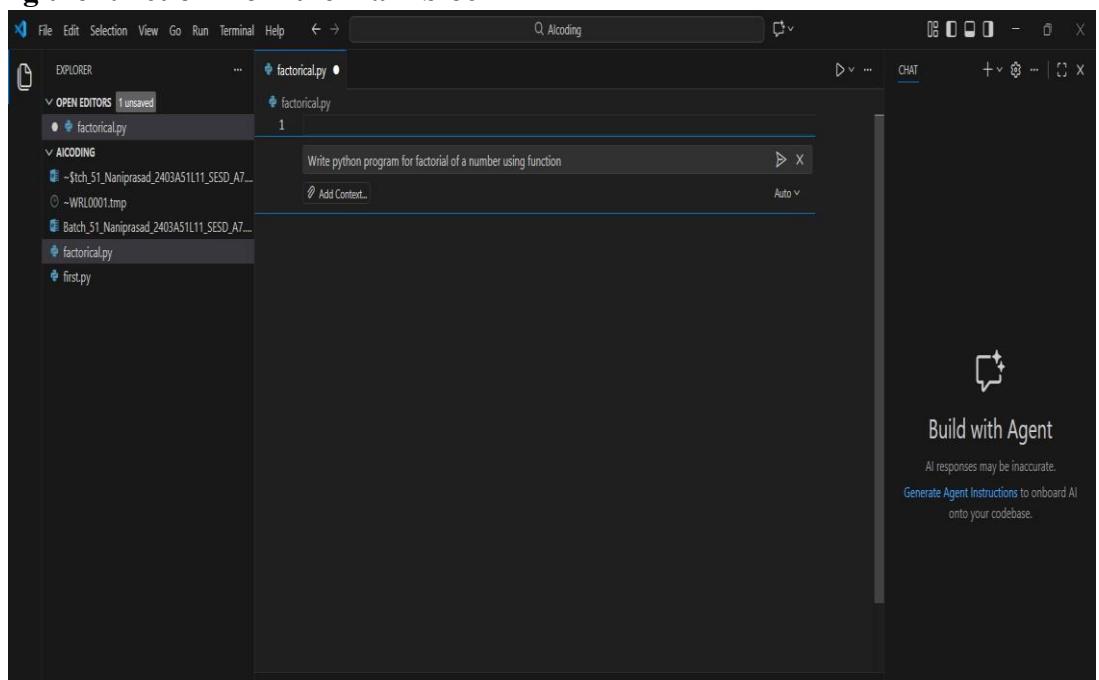
❖ *= is marginally optimized at the bytecode level, but the difference is negligible.

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)}")

```

❖ 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.

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

```

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(factorial(num)))

```

Compare the non-function and function-based Copilot-generated programs on the following criteria:

- Logic clarity
- Reusability
- Debugging ease
- Large projects
- AI dependency

```

factorial.py
-----
# Comparison Analysis
print("\n---- Comparison of Approaches ----")
print("Non-function approach:")
print(" Logic clarity: Simple but mixed concerns")
print(" Reusability: Low - code cannot be reused")
print(" Debugging ease: Harder - logic embedded in main flow")
print(" Large projects: Poor - code duplication likely")
print(" AI dependency: Moderate - straightforward logic")

print("\nFunction-based approach:")
print(" Logic clarity: High - isolated and documented")
print(" Reusability: High - can be imported and used anywhere")
print(" Debugging ease: Easy - logic is testable and modular")
print(" Large projects: Excellent - follows best practices")
print(" AI dependency: Lower - cleaner code requires less AI assistance")

print("\nRecommendation: Use function-based approach for production code")

```

Comparison Analysis

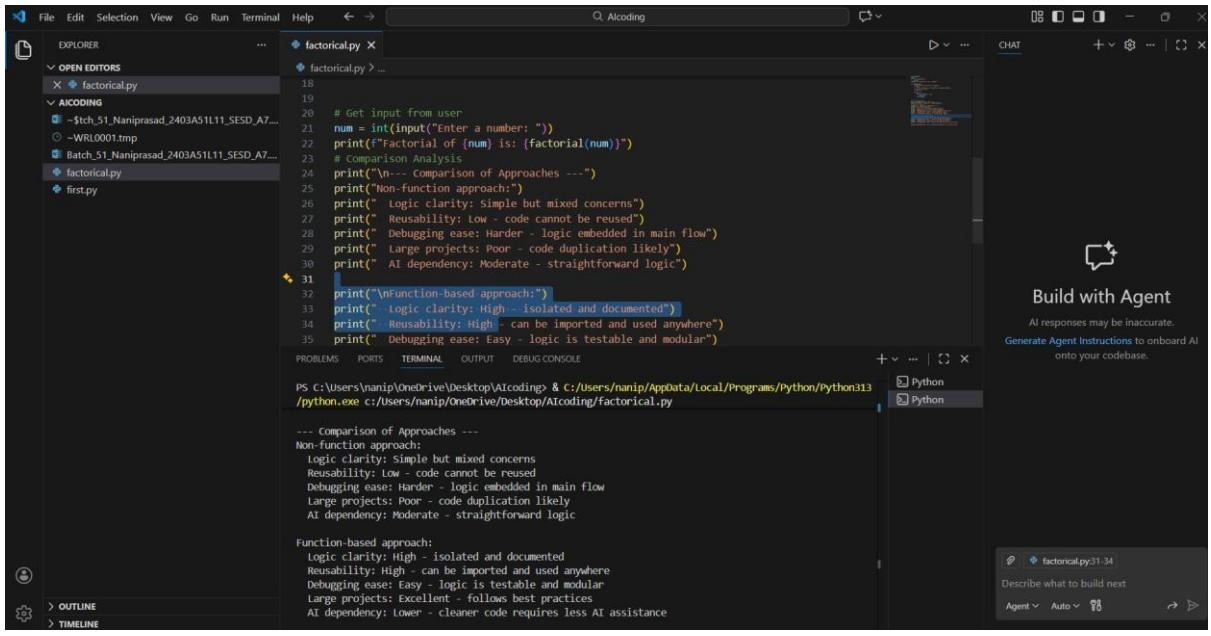
Non-function approach:

- Logic clarity: Simple but mixed concerns
- Reusability: Low - code cannot be reused
- Debugging ease: Harder - logic embedded in main flow
- Large projects: Poor - code duplication likely
- AI dependency: Moderate - straightforward logic

Function-based approach:

- Logic clarity: High - isolated and documented
- Reusability: High - can be imported and used anywhere
- Debugging ease: Easy - logic is testable and modular
- Large projects: Excellent - follows best practices
- AI dependency: Lower - cleaner code requires less AI assistance

Recommendation: Use function-based approach for production code



The screenshot shows the Alcoding IDE interface. In the center, there is a code editor window titled "factorical.py" with the following content:

```

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")

```

Below the code editor is a terminal window showing the command: PS C:\Users\nanip\OneDrive\Desktop\Alcoding & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nanip/OneDrive/Desktop/Alcoding/factorical.py

In the bottom right corner of the IDE, there is a "Build with Agent" panel. It has a message: "Build with Agent" and "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." Below this, it says "Describe what to build next" and "Agent Auto".

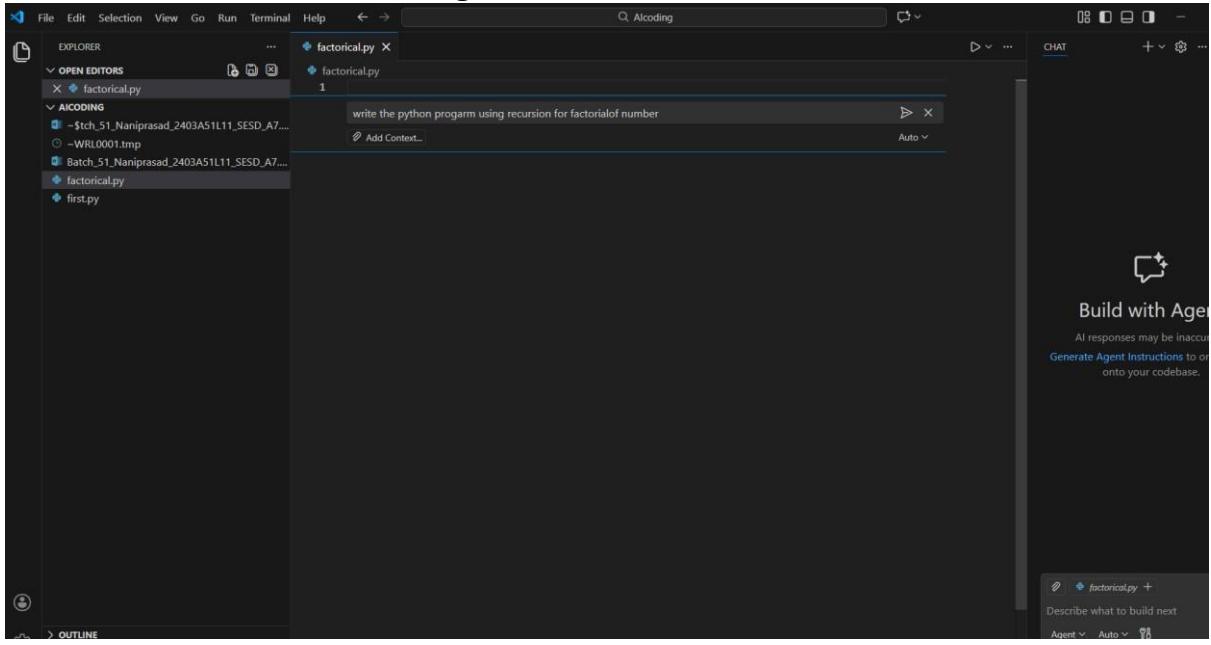
TASK - 5

Task Description

Prompt Copilot to generate:

An iterative version of the logic

A recursive version of the same logic



The screenshot shows the Alcoding IDE interface. In the center, there is a code editor window titled "factorical.py" with the following content:

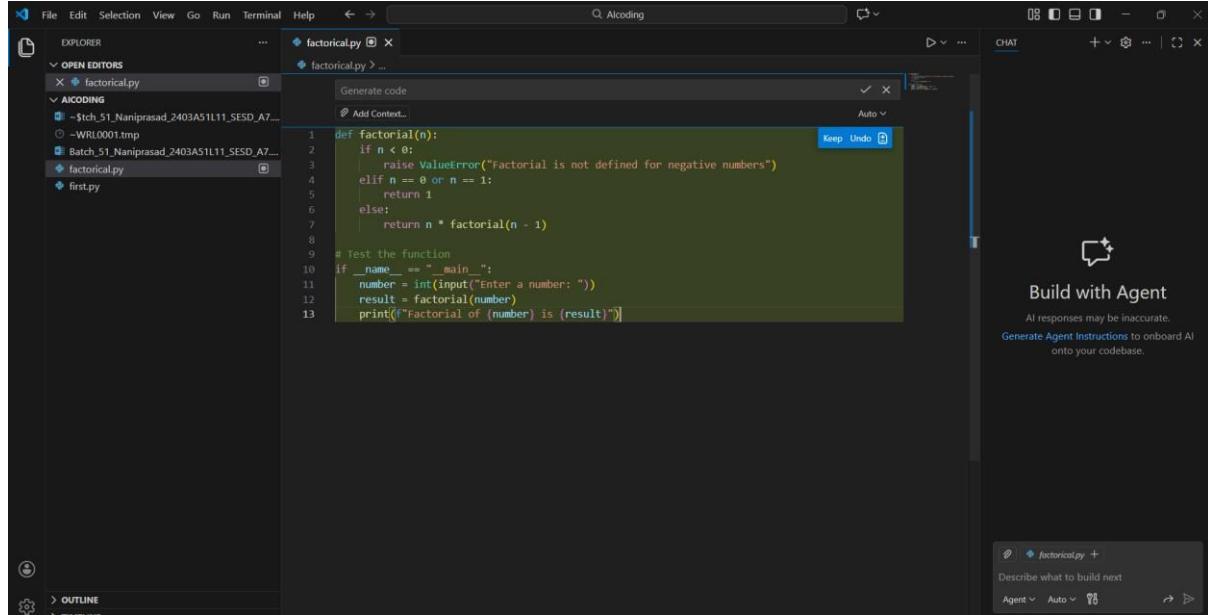
```

1
2 write the python program using recursion for factorial of number
3

```

Below the code editor is a terminal window showing the command: PS C:\Users\nanip\OneDrive\Desktop\Alcoding & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nanip/OneDrive/Desktop/Alcoding/factorical.py

In the bottom right corner of the IDE, there is a "Build with Agent" panel. It has a message: "Build with Agent" and "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." Below this, it says "Describe what to build next" and "Agent Auto".



```

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)

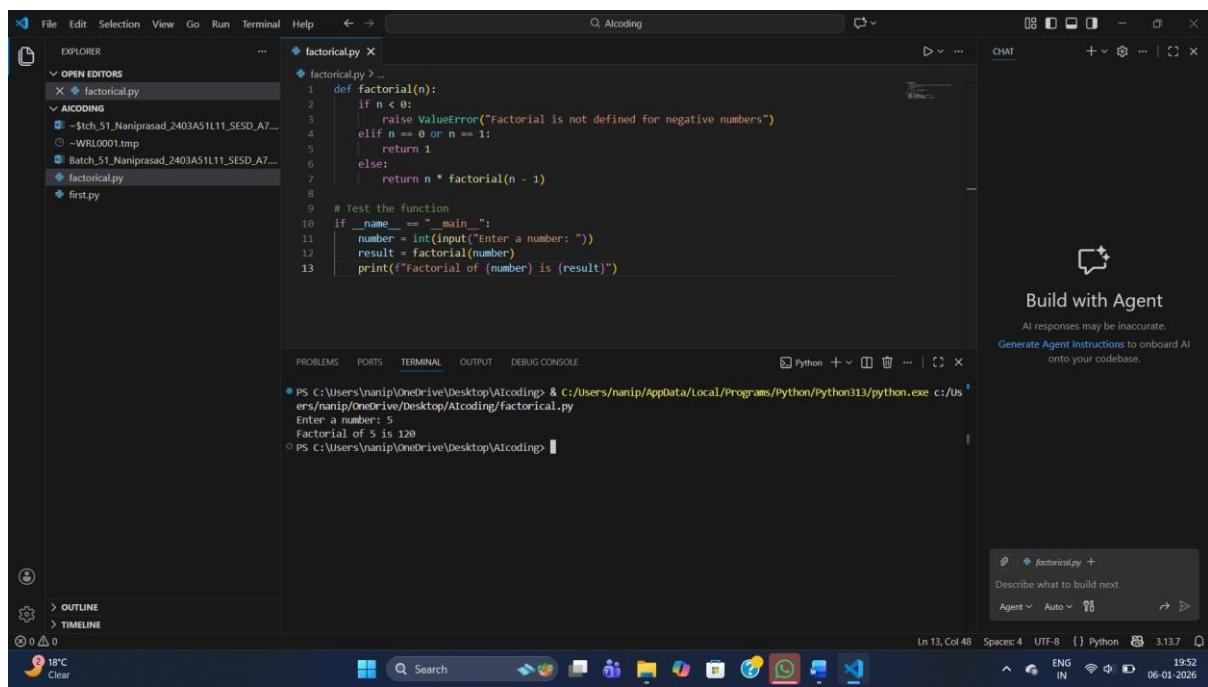
# Test the function
if __name__ == "__main__":
    number = int(input("Enter a number: "))
    result = factorial(number)
    print(f"Factorial of {number} is {result}")

```

Build with Agent

All responses may be inaccurate.

Generate Agent Instructions to onboard AI onto your codebase.



```

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}")

```

Build with Agent

All 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\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>

Ln 13, Col 48 Spaces: 4 UTF-8 Python 3.13.7 ENG IN 06-01 1952

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).**

