

## School of Computer Science and Artificial Intelligence

### Lab Assignment # 1

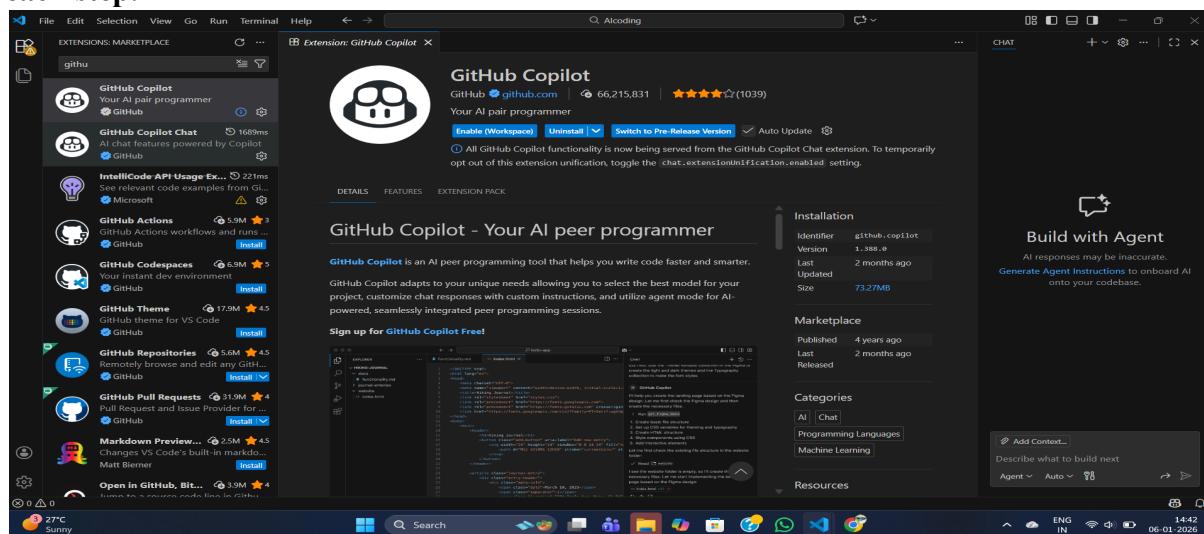
<b>Program</b>	: B. Tech (CSE)
<b>Specialization</b>	:
<b>Course Title</b>	: AI Assisted coding
<b>Course Code</b>	: 23CS201PC302
<b>Semester</b>	: II
<b>Academic Session</b>	: 2025-2026
<b>Name of Student</b>	: Mohammed Hashir Sayam Hussain
<b>Enrollment No.</b>	: 2403A51L27
<b>Batch No.</b>	: 51
<b>Date</b>	: 06-01-2026

### Submission Starts here

**OUTPUT :**

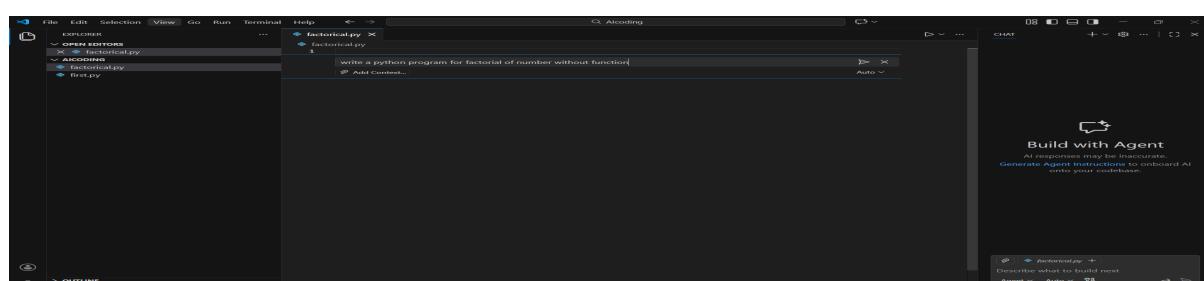
**SCREENSHOTS:**

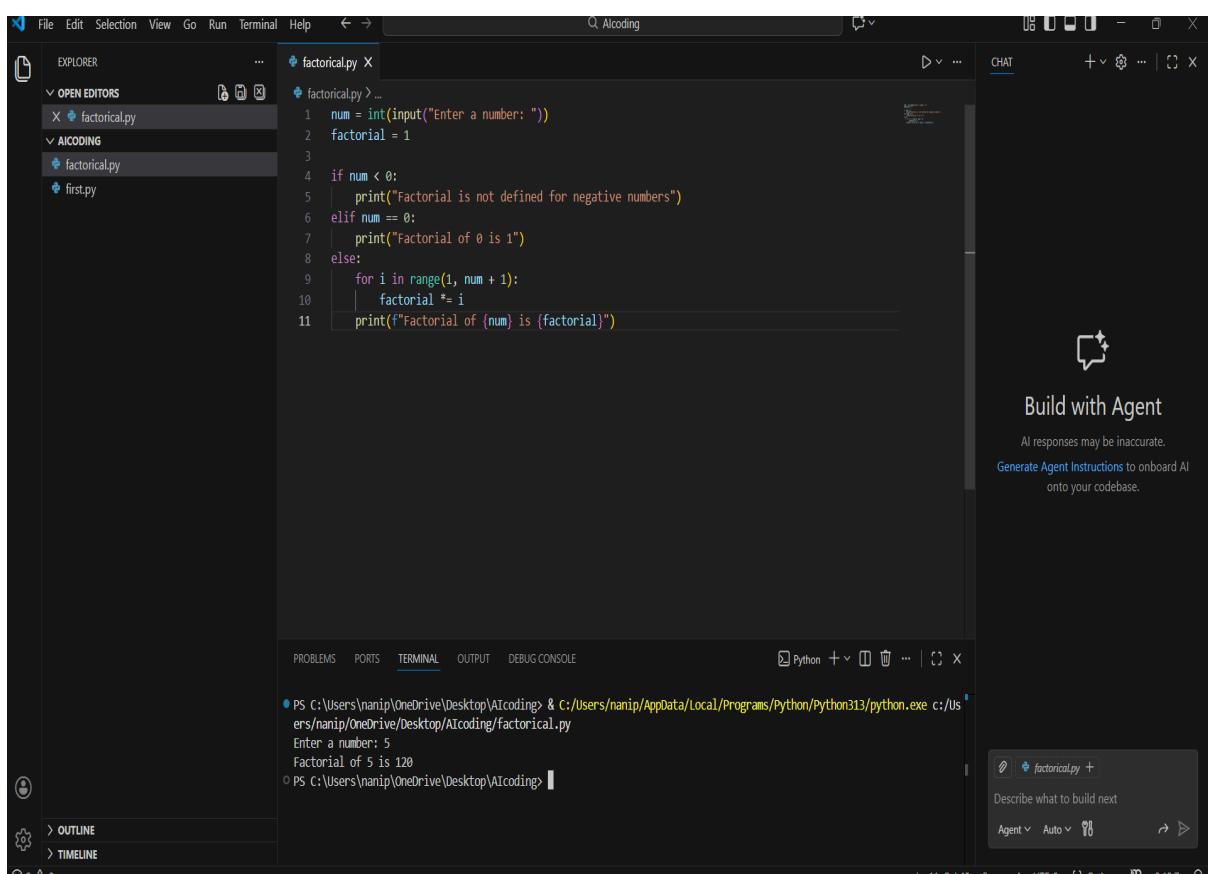
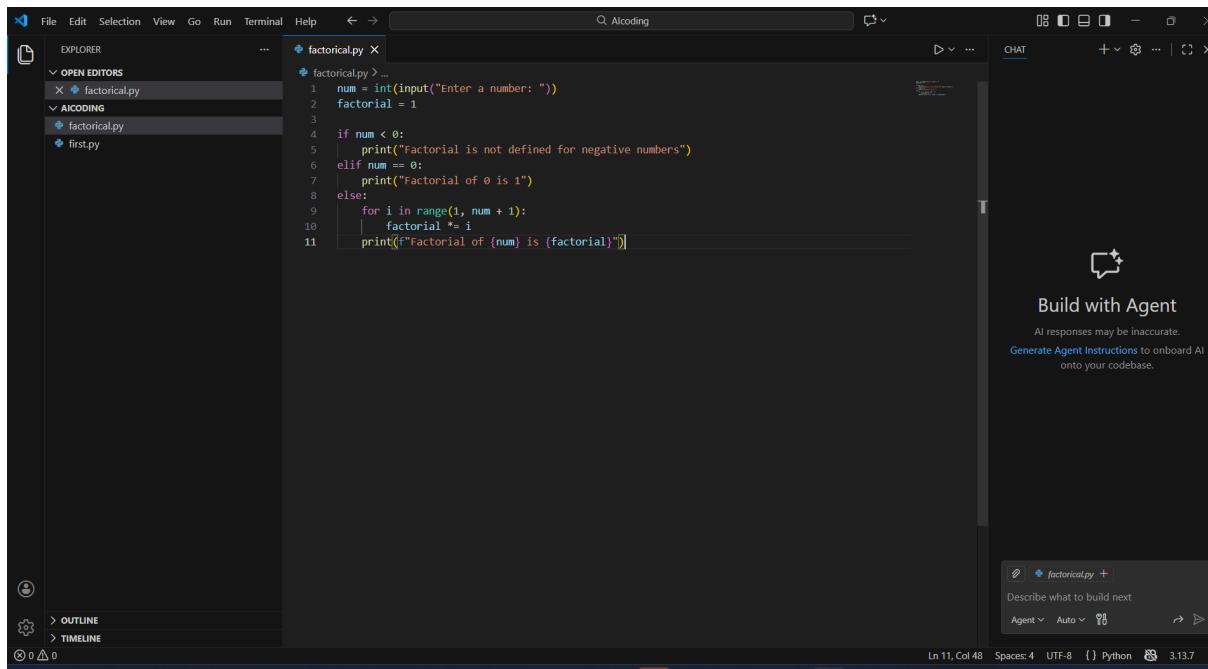
**Task 0: Install and configure GitHub Copilot in VS Code. Take screenshots of each step.**



#### 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 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

A screenshot of the Visual Studio Code (VS Code) interface. The Explorer sidebar shows files like 'factorial.py', 'first.py', and some temporary files. The terminal at the bottom shows a command prompt in a Windows environment (PS C:\Users\nanip\OneDrive\Desktop\AIcoding>). The main editor area contains the following Python code:

```
n=int(input())
fact=1
for i in range(1,n+1):
    fact*=i
print(f"the factorial of {n} is {fact}")
```

The interface includes a 'CHAT' panel on the right with a 'Build with Agent' button and a status bar at the bottom.

```

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

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

```

## # 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.

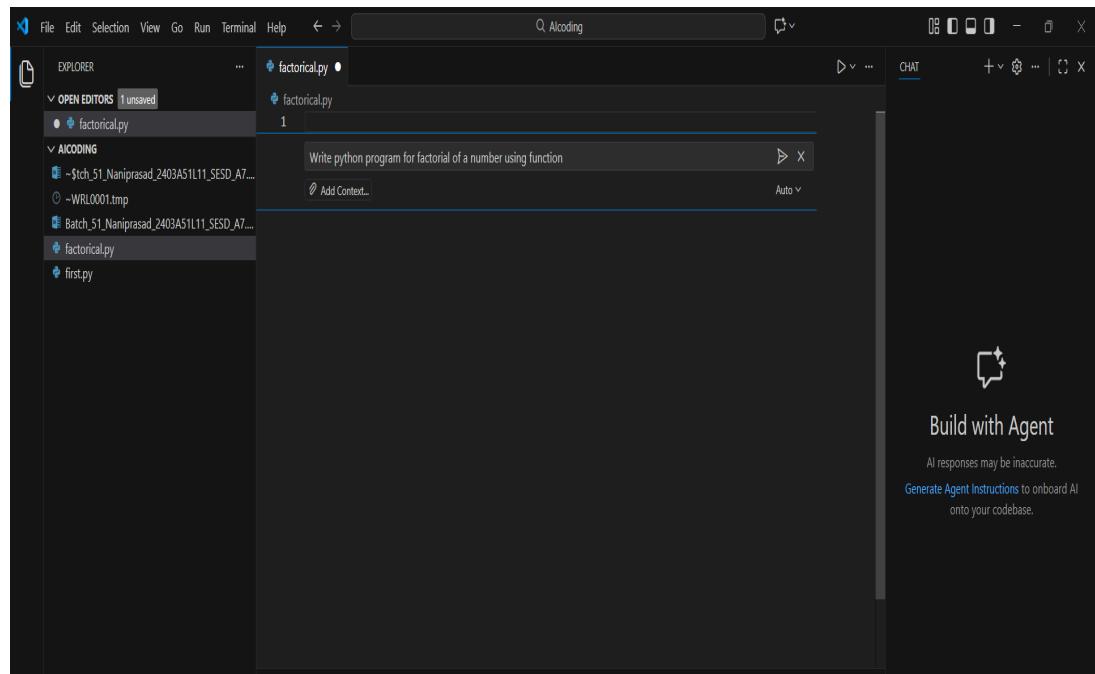
❖ \*= is marginally optimized at the bytecode level, but the difference is negligible.

## # 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



A screenshot of the Visual Studio Code (VS Code) interface. The Explorer sidebar shows files like 'factorical.py' and 'first.py'. In the center, there's a code editor with the following Python code:

```
def factorial(n):
    """Calculate the factorial of a number."""
    if n < 0:
        return "Factorial is not defined for negative numbers"
    elif 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(f"Factorial of {num} is: {factorial(num)}")
```

The status bar at the bottom right says 'Build with Agent'.

```

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 with a dark theme. In the center is a code editor window titled 'factorial.py' containing the provided Python code. To the right is a 'CHAT' window with a 'Build with Agent' section. Below the code editor is a terminal window showing the command 'python.exe c:/Users/nanip/OneDrive/Desktop/AlCoding/factorial.py' and the output 'Factorial of 5 is: 120'. At the bottom is a taskbar with various icons.

❖ **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 the VS Code interface with the following details:

- File Explorer:** Shows files like `factorical.py` (unsaved), `factorical.py` (ACI Coding), and `first.py`.
- Editor:** Displays Python code for calculating factorial. A tooltip from the AI Copilot provides comparison analysis between non-function and function-based approaches.
- Terminal:** Shows the command `python factorial.py` running in Python 3.13.7, outputting "Factorial of 5 is: 120".
- Output Panel:** Shows the Python extension status.
- Status Bar:** Includes weather information (18°C) and system details (Windows 10, 06-01-2026).

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files like `factorical.py` (unsaved), `factorical.py` (ACI Coding), and `first.py`.
- Editor:** Displays Python code for calculating factorial. A tooltip from the AI Copilot provides comparison analysis between non-function and function-based approaches.
- Terminal:** Shows the command `python factorial.py` running in Python 3.13.7, outputting "Factorial of 5 is: 120".
- Output Panel:** Shows the Python extension status.
- Status Bar:** Includes weather information (18°C) and system details (Windows 10, 06-01-2026).

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like `factorial.py`, `first.py`, and several log files.
- Code Editor:** Displays the `factorial.py` file 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")
```
- Terminal:** Shows the command to run the script: `python.exe c:/Users/nanip/OneDrive/Desktop/Aicoding/factorial.py`.
- Output:** Shows the execution results of the script.
- Problems:** Shows no errors or warnings.
- Status Bar:** Shows the current file path: `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`.
- Bottom Right:** A floating window titled "Build with Agent" with the message "Al responses may be inaccurate." It includes a "Generate Agent Instructions" button and a note about onboarding AI onto your codebase.

# TASK - 5

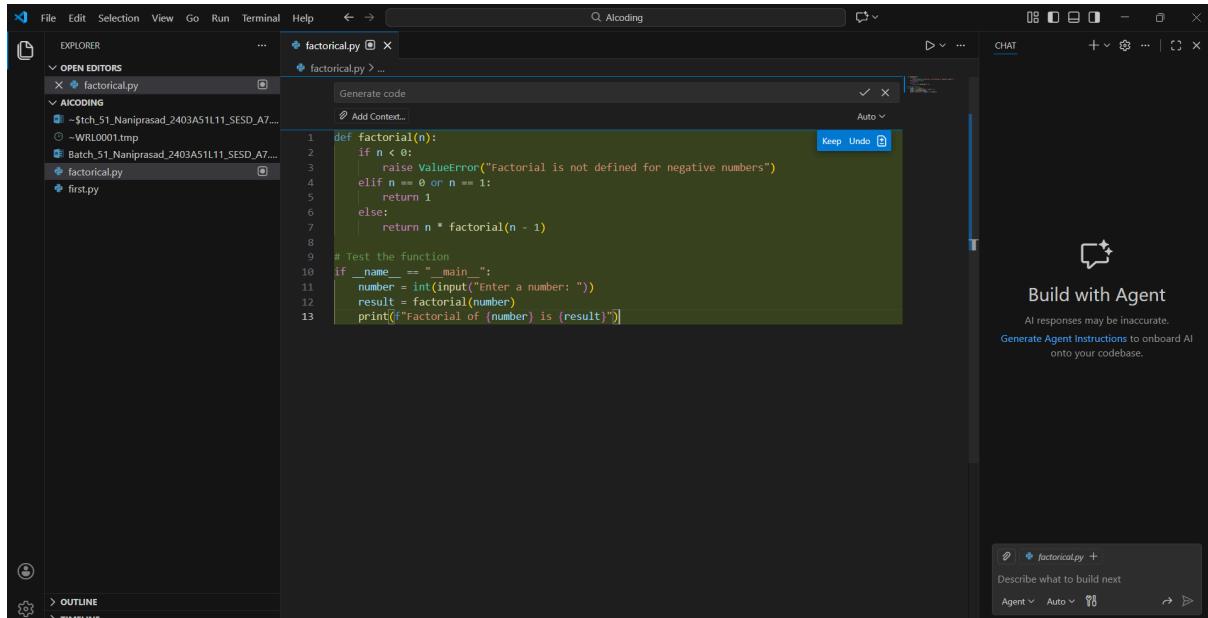
## Task Description

## Prompt Copilot to generate:

### An iterative version of the logic

### A recursive version of the same logic

A screenshot of the Visual Studio Code (VS Code) interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar containing 'Alcoding'. The left sidebar shows the Explorer view with several open editors, including 'factorical.py' and 'first.py'. The main editor area displays the code for 'factorical.py'. A floating AI assistant window titled 'factorical.py' contains the instruction 'write the python program using recursion for factorialof number' and an 'Add Context...' button. The bottom right corner features a dark-themed AI interface with a speech bubble icon, the text 'Build with Age', a note about inaccurate AI responses, and a 'Generate Agent Instructions to on' button. The status bar at the bottom shows the file path 'factorical.py' and a 'Describe what to build next' input field.

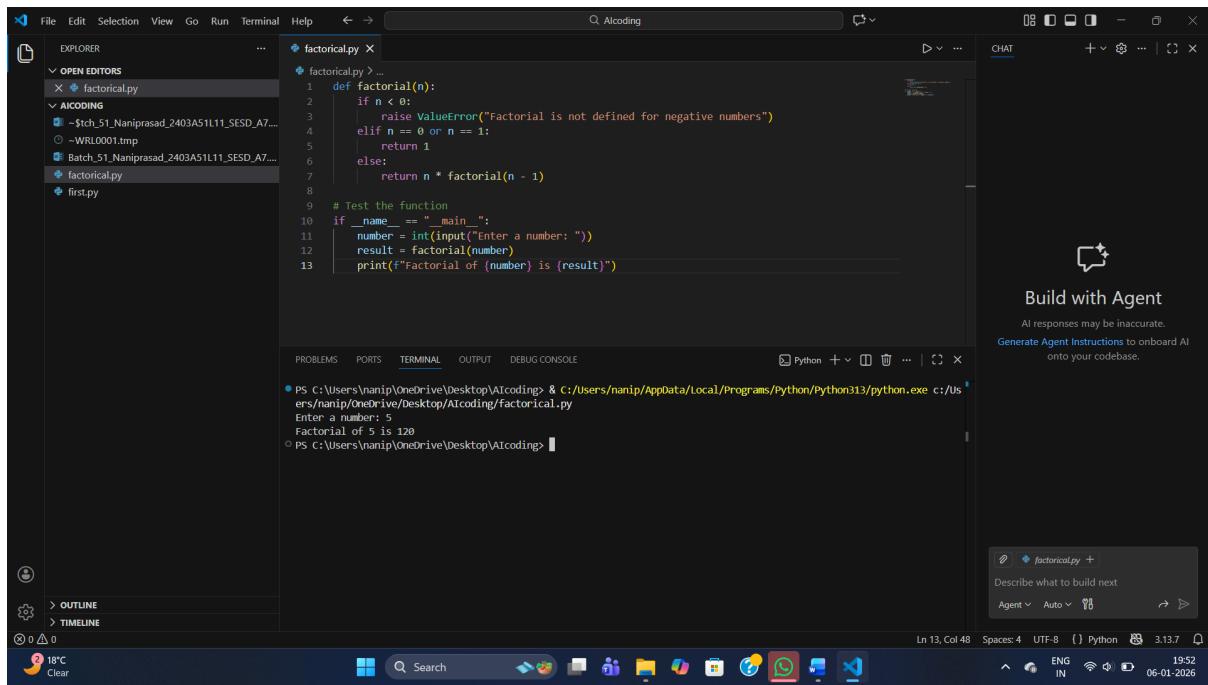


The screenshot shows the VS Code interface with the "factorial.py" file open in the editor. The code defines a factorial function that handles negative numbers, returns 1 for 0 or 1, and uses recursion for other values. A tooltip from the "AI CODING" extension is displayed over the code, suggesting the addition of a ValueError for negative numbers. The "CHAT" sidebar on the right shows a message from an AI agent about building the code.

```

1 def factorial(n):
2     if n < 0:
3         raise ValueError("Factorial is not defined for negative numbers")
4     elif n == 0 or n == 1:
5         return 1
6     else:
7         return n * factorial(n - 1)
8
9 # Test the function
10 if __name__ == "__main__":
11     number = int(input("Enter a number: "))
12     result = factorial(number)
13     print(f"Factorial of {number} is {result}")

```



This screenshot shows the same VS Code setup as above, but with the terminal tab active at the bottom. It displays the command `python factorial.py` being run and the resulting output: "Factorial of 5 is 120". The "CHAT" sidebar still shows the AI agent's message.

```

PS C:\Users\nanip\OneDrive\Desktop\AIcoding> python factorial.py
Enter a number: 5
Factorial of 5 is 120

```

## 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).

