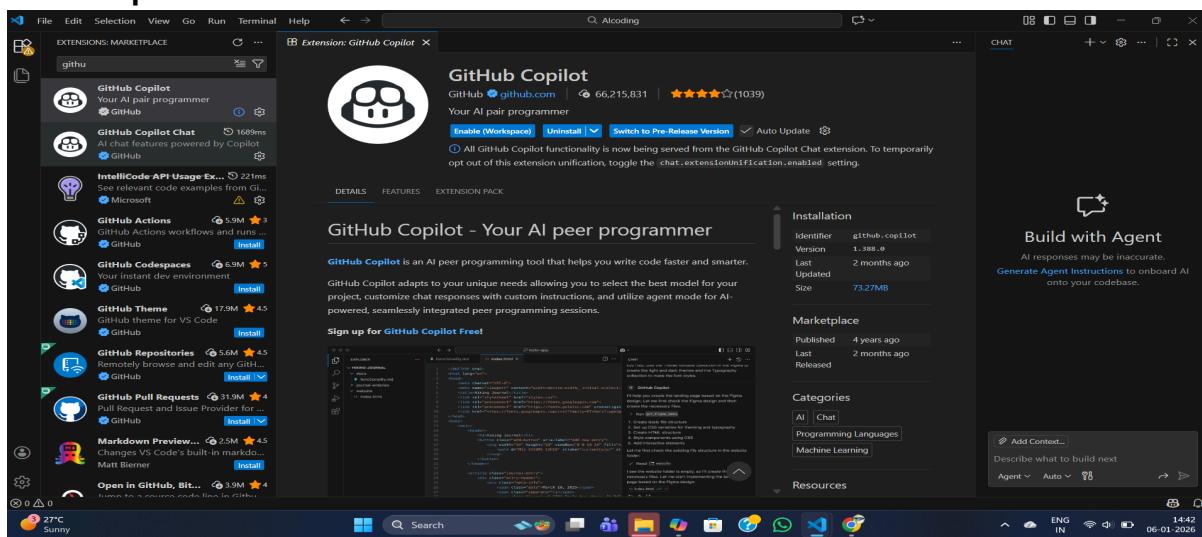


School of Computer Science and Artificial Intelligence

Lab Assignment # 1.2

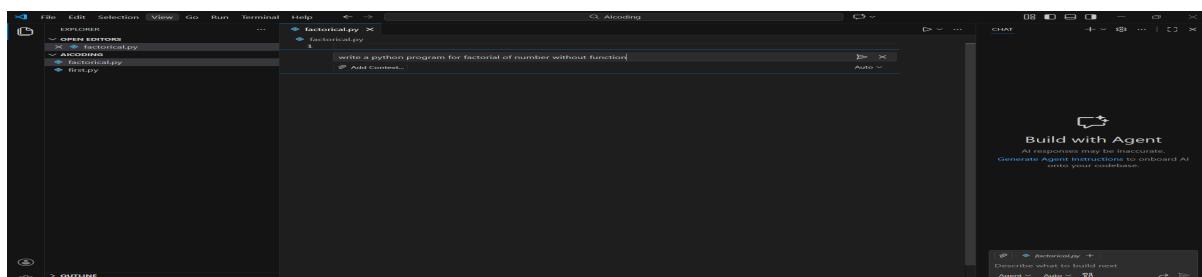
Program	: B. Tech (CSE)
Specialization	:
Course Title	: AI Assisted coding
Course Code	:
Semester	: II
Academic Session	: 2025-2026
Name of Student	: M.Srianth
Enrollment No.	: 2403A51L29
Batch No.	: 51
Date	: 06-01-2026

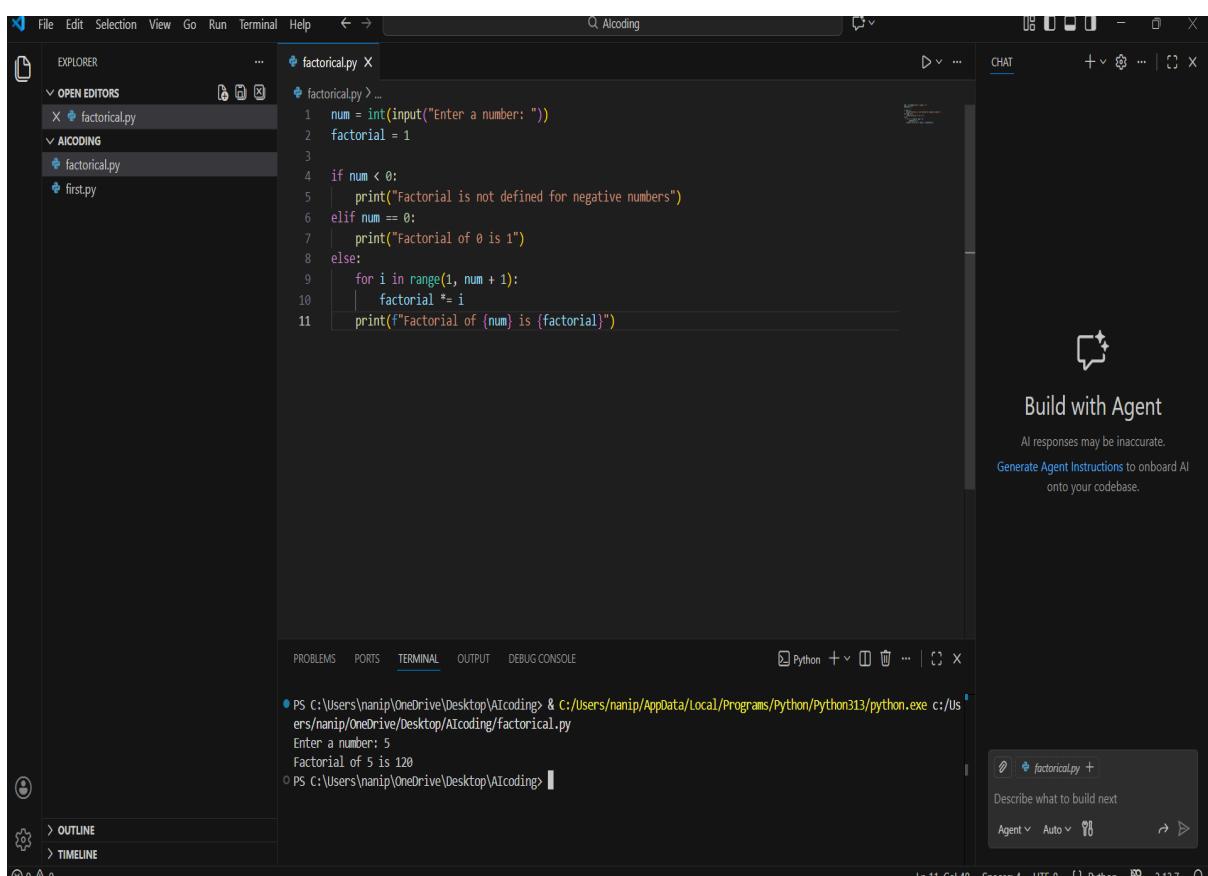
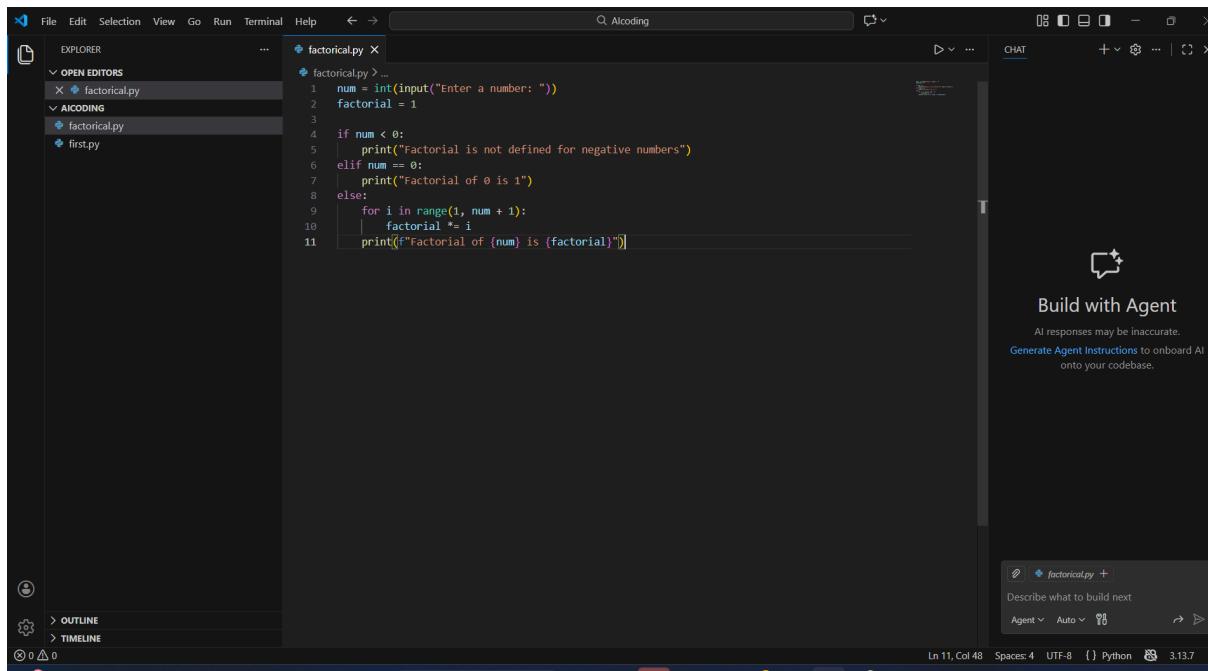
Task 0: Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



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**)

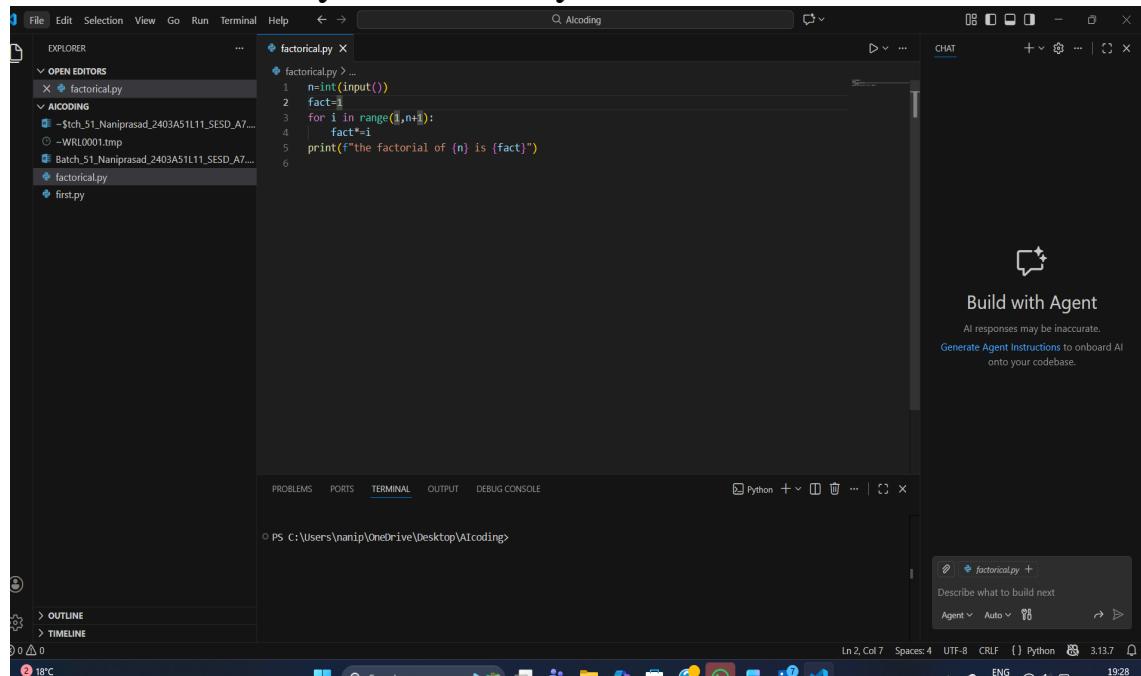
❖ The code generated was as requested
in the prompt

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



```

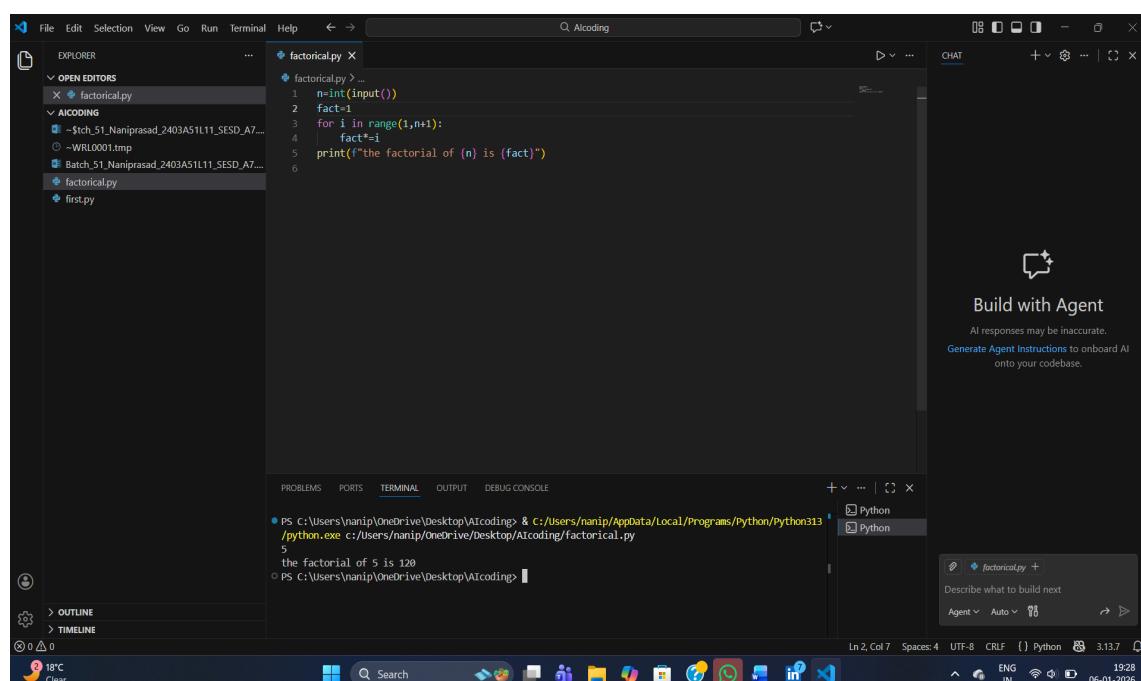
File Edit Selection View Go Run Terminal Help ⏎ ↻ 🔍 Alcoding
EXPLORER ... factorial.py
OPEN EDITORS factorial.py ...
AICODING
- $tch_51_Naniprasad_2403A51L11_SESD_A7...
-WRL0001.tmp
Batch_51_Naniprasad_2403A51L11_SESD_A7...
factorical.py
first.py

factorical.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

CHAT + ✎ 🔍 ... ×

PROBLEMS PORTS TERMINAL OUTPUT DEBUG CONSOLE
PS C:\Users\nanip\OneDrive\Desktop\AICoding>
Ln 2, Col 7 Spaces: 4 UTF-8 CRLF ( Python ENG 19:28

```



```

File Edit Selection View Go Run Terminal Help ⏎ ↻ 🔍 Alcoding
EXPLORER ... factorial.py
OPEN EDITORS factorial.py ...
AICODING
- $tch_51_Naniprasad_2403A51L11_SESD_A7...
-WRL0001.tmp
Batch_51_Naniprasad_2403A51L11_SESD_A7...
factorical.py
first.py

factorical.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

CHAT + ✎ 🔍 ... ×

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/factorical.py
5
the factorial of 5 is 120
PS C:\Users\nanip\OneDrive\Desktop\AICoding>
Ln 2, Col 7 Spaces: 4 UTF-8 CRLF ( Python ENG 19:28

```

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.

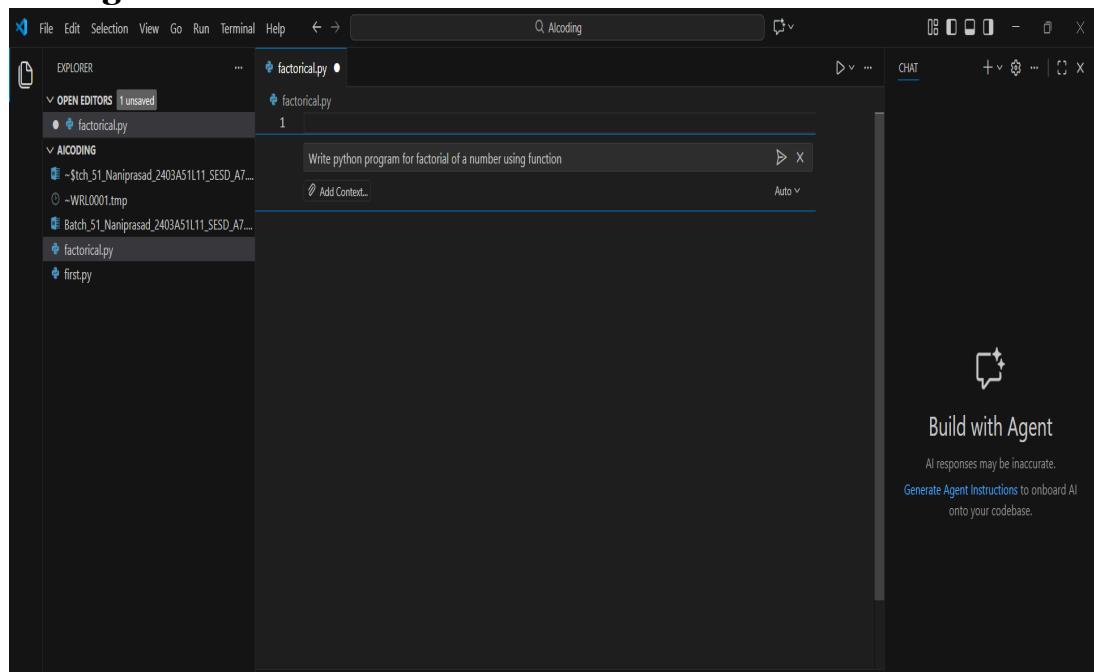
- ❖ Maintainability
 - Cleaner code is easier to modify and debug.
 - Reduced redundancy lowers the chance of mistakes.
- ❖ Performance
 - Performance is effectively the same.
- ❖ *= is marginally optimized at the bytecode level, but the difference is negligible.

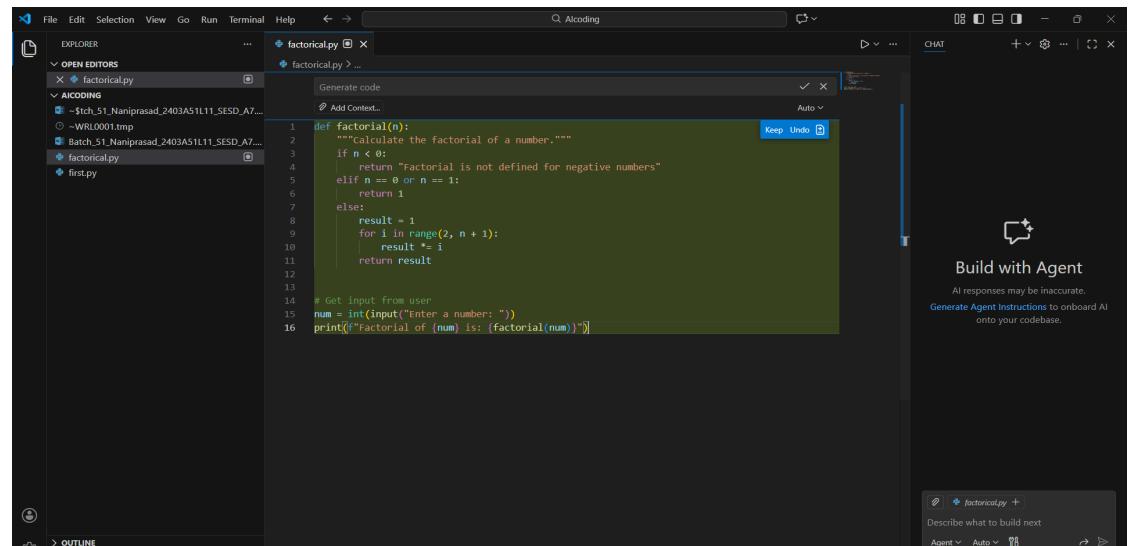
Task3

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

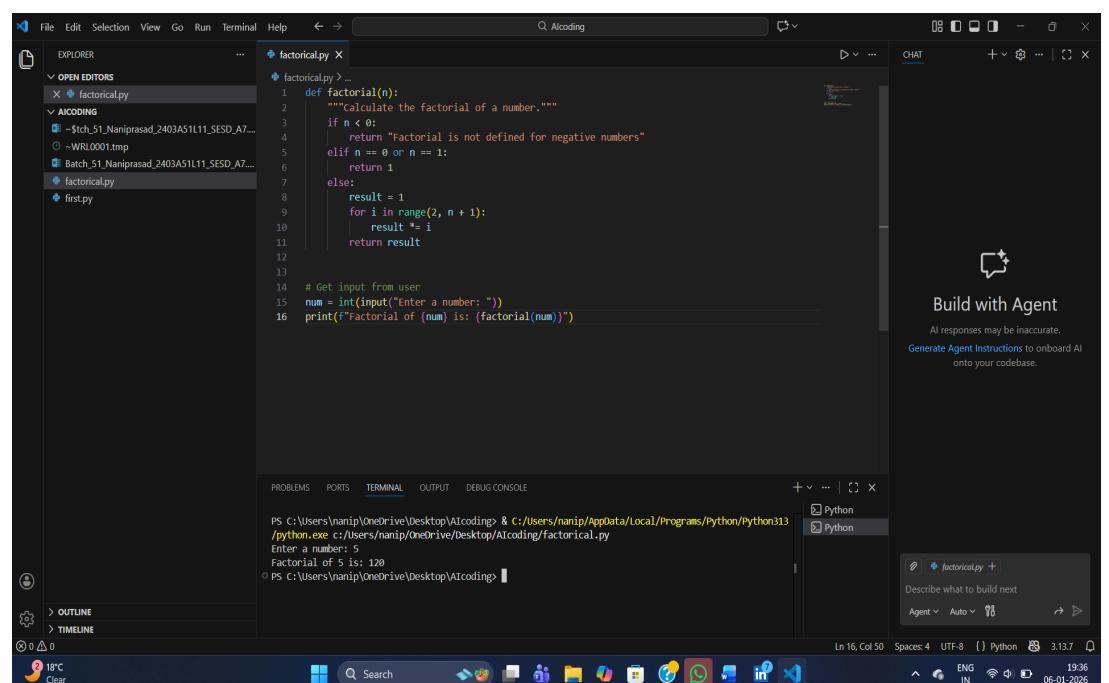




```

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)}")

```



```

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)}")

```

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>

```

❖ 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 a code editor interface with a dark theme. In the center, there is an editor window displaying a Python script named `factorical.py`. The script contains a function definition for calculating the factorial of a number. Below the code, a terminal window shows the script being run and outputting the result for n=5. A floating panel titled "Build with Agent" is visible, asking to compare the non-function and function-based Copilot-generated programs on specific criteria. The bottom right corner of the screen shows a system tray with icons for battery, signal, and date/time.

```

File Edit Selection View Go Run Terminal Help ← →
Q Alcoding
EXPLORER OPEN EDITORS 1 unsaved
AICODING ~$tch_51_Naniprasad_2403A51L11_SESD_A7...
~WRL0001.tmp Batch_51_Naniprasad_2403A51L11_SESD_A7...
factorical.py first.py
factorical.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(f"Factorial of {num} is: {factorial(num)}")

```

Compare the non-function and function-based Copilot-generated programs on the following criteria:
 Logic clarity
 Reusability
 Add Context...

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/AICoding/factorical.py
Enter a number: 5
Factorial of 5 is: 120
PS C:\Users\nanip\OneDrive\Desktop\Alcoding>

factorical.py + Describe what to build next
Agent Auto ✓ 19:46 06-01-2026

The screenshot shows the AI Coding interface with the following details:

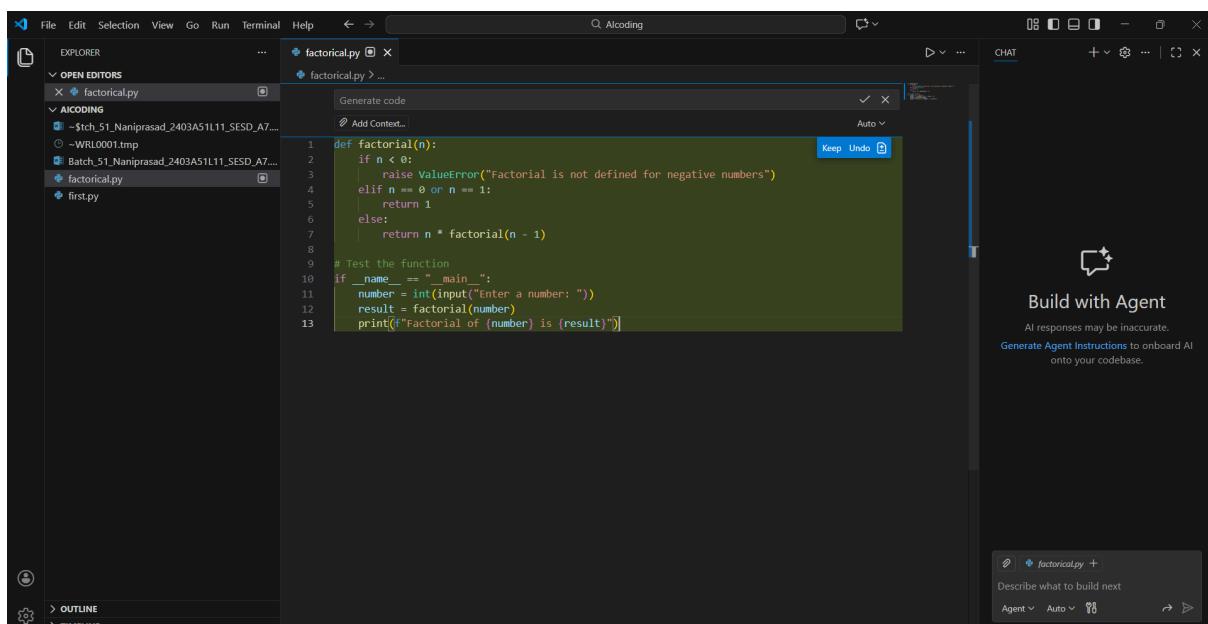
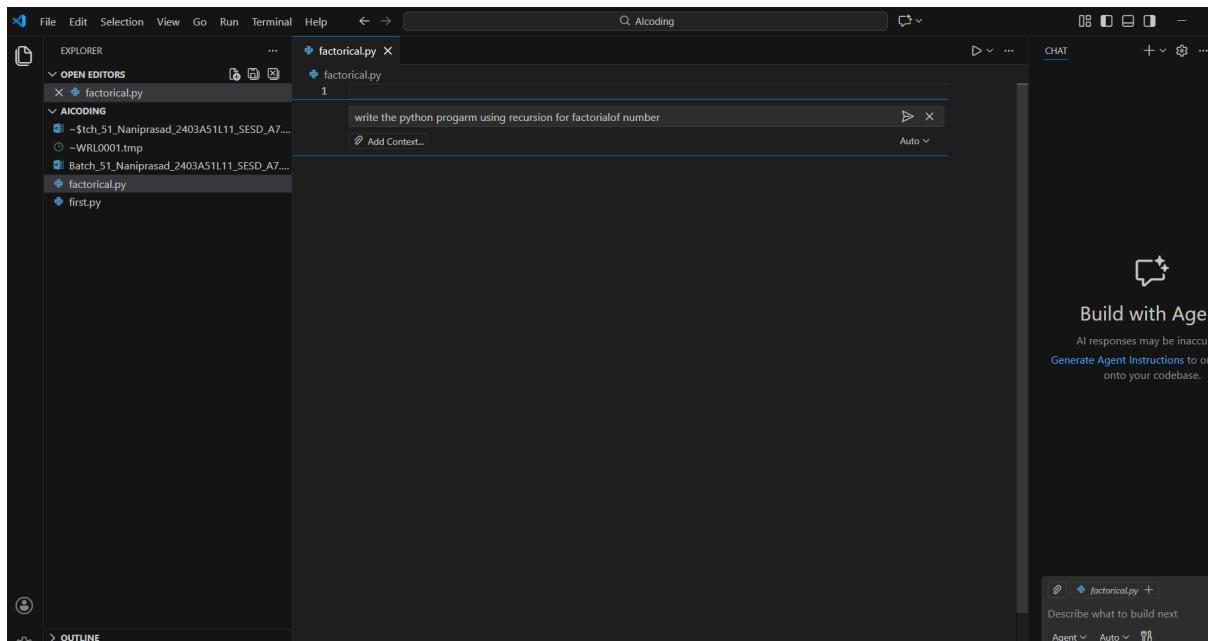
- File Explorer:** Shows files like factorial.py, factorial.py (AI), and first.py.
- Editor:** Displays code for factorial.py with a callout box highlighting a section of the code. The callout box contains "Comparison Analysis" and lists pros and cons for function-based and non-function-based approaches.
- Terminal:** Shows the command `python factorial.py` being run, with the output "Factorial of 5 is: 120".
- Output:** Shows the results of the comparison analysis.
- Bottom Status Bar:** Includes system icons, a weather icon (18°C), and a date/time stamp (06-01-2026).

The screenshot shows the AI Coding interface with the following details:

- File Explorer:** Shows files like factorial.py, factorial.py (AI), and first.py.
- Editor:** Displays code for factorial.py with a callout box highlighting a section of the code. The callout box contains "Comparison Analysis" and lists pros and cons for function-based and non-function-based approaches.
- Terminal:** Shows the command `python factorial.py` being run, with the output "Factorial of 5 is: 120".
- Output:** Shows the results of the comparison analysis.
- Bottom Status Bar:** Includes system icons, a weather icon (18°C), and a date/time stamp (06-01-2026).

TASK - 5

Task Description
Prompt Copilot to generate:
An iterative version of the logic
A recursive version of the same logic



The screenshot shows a dark-themed instance of Visual Studio Code. In the Explorer sidebar, there are several files listed under 'OPEN EDITORS' and 'AICODING'. The 'factorical.py' file is open in the main editor area. The code defines a recursive factorial function that handles negative numbers by raising a ValueError and returns 1 for base cases (0 or 1). It also includes a test block at the bottom to demonstrate the function's behavior. Below the editor is the terminal, which shows the command being run: 'PS C:\Users\nanip\OneDrive\Desktop\AICoding> & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nanip/OneDrive/Desktop/AICoding/factorical.py'. The terminal output shows the user entering '5' and the program printing 'Factorial of 5 is 120'. A floating 'Build with Agent' panel is visible on the right side of the interface.

```

factorical.py > ...
factorical.py > ...
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}")

```

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).**