

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	: S. Bharath
Enrollment No.	: 2403A51L17
Batch No.	: 51
Date	: 06-01-2026

Submission Starts here

OUTPUT :

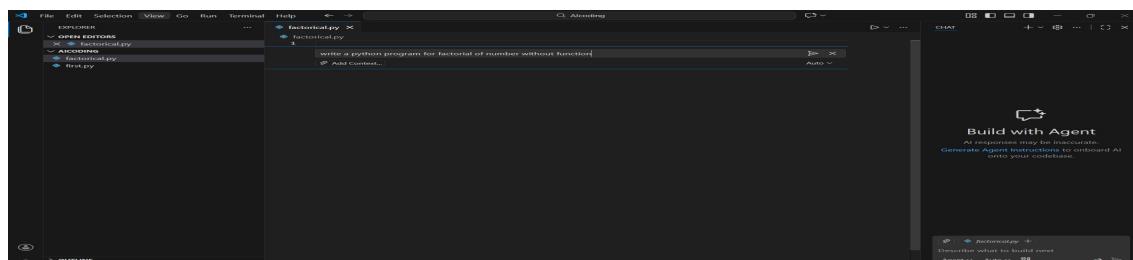
SCREENSHOTS:

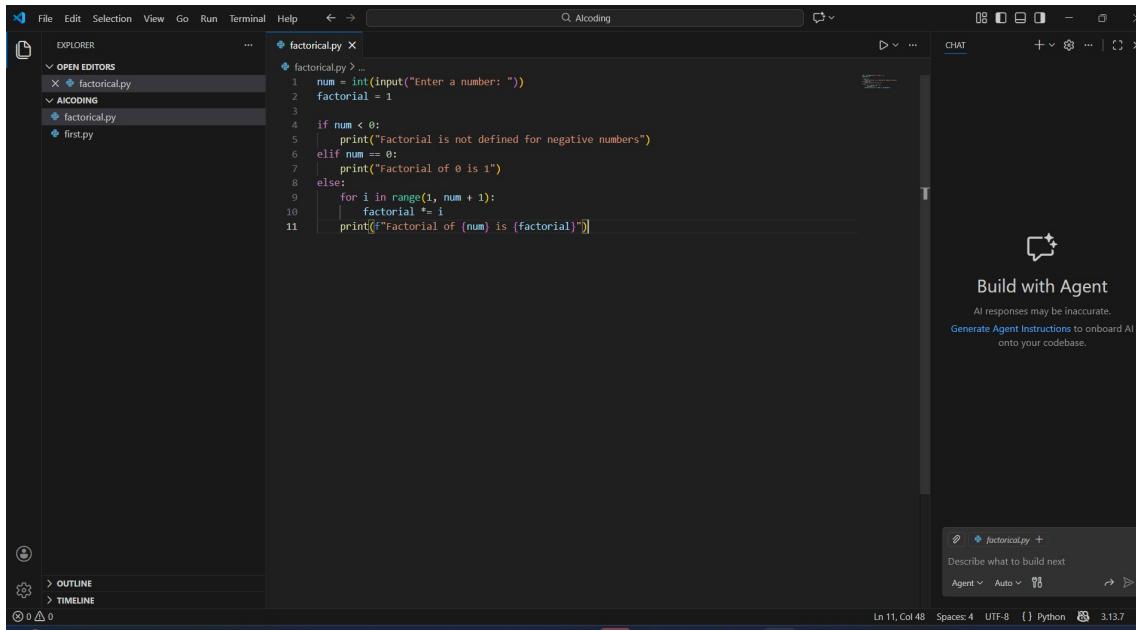
Task 0: Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



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.

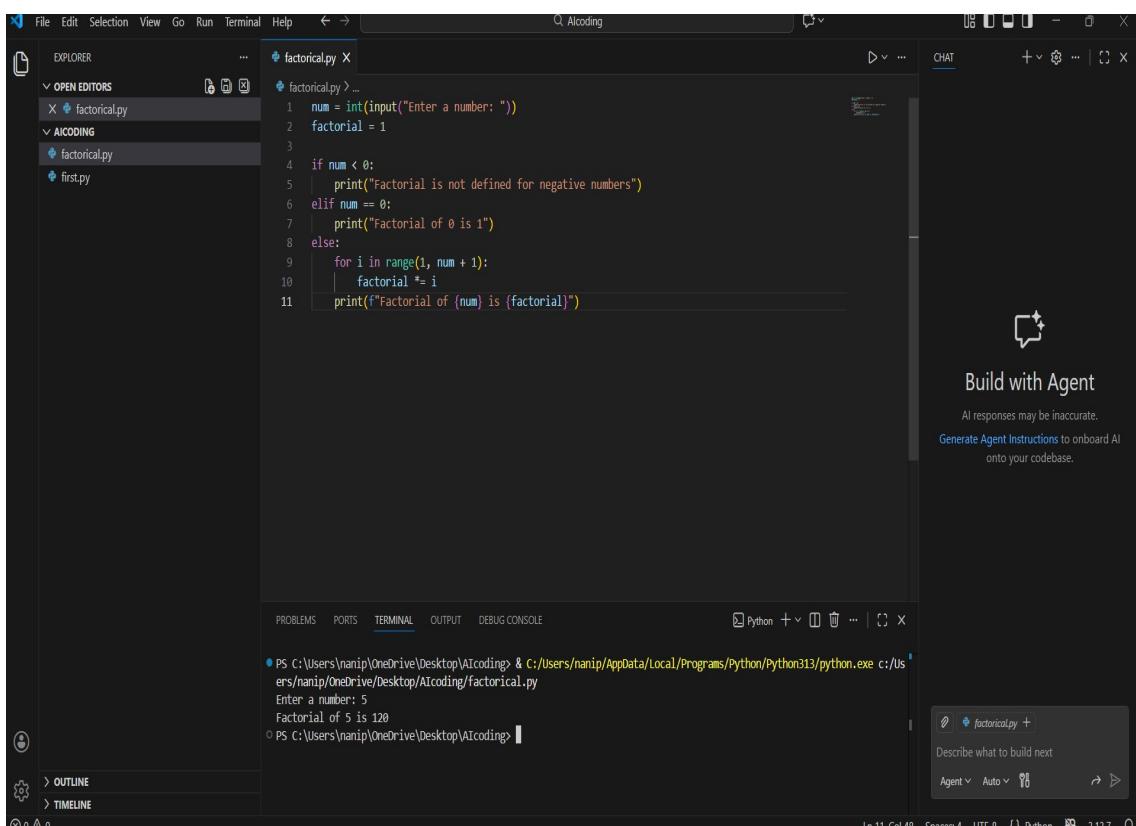




```

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}")

```



```

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}")

```

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 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

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

- Explorer View:** Shows files factorial.py, first.py, and a folder AI CODING containing several temporary files.
- Terminal:** Displays the command PS C:\Users\Nanip\OneDrive\Desktop\AIcoding>.
- AI Coding Extension Panel:** Shows the code for factorial.py and a "Build with Agent" button.
- Status Bar:** Shows the file path C:\Users\Nanip\OneDrive\Desktop\AIcoding\fatorial.py, Python 3.13.7, and the date 06-01-2026.

```
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))
```

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

- Explorer View:** Shows files factorial.py, first.py, and a folder AI CODING containing several temporary files.
- Terminal:** Displays the command PS C:\Users\Nanip\OneDrive\Desktop\AIcoding> & C:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/nanip/OneDrive/Desktop/AIcoding/fatorial.py
- Output Console:** Displays the output: the factorial of 5 is 120.
- AI Coding Extension Panel:** Shows the code for factorial.py and a "Build with Agent" button.
- Status Bar:** Shows the file path C:\Users\Nanip\OneDrive\Desktop\AIcoding\fatorial.py, Python 3.13.7, and the date 06-01-2026.

```
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))
```

What was improved?

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

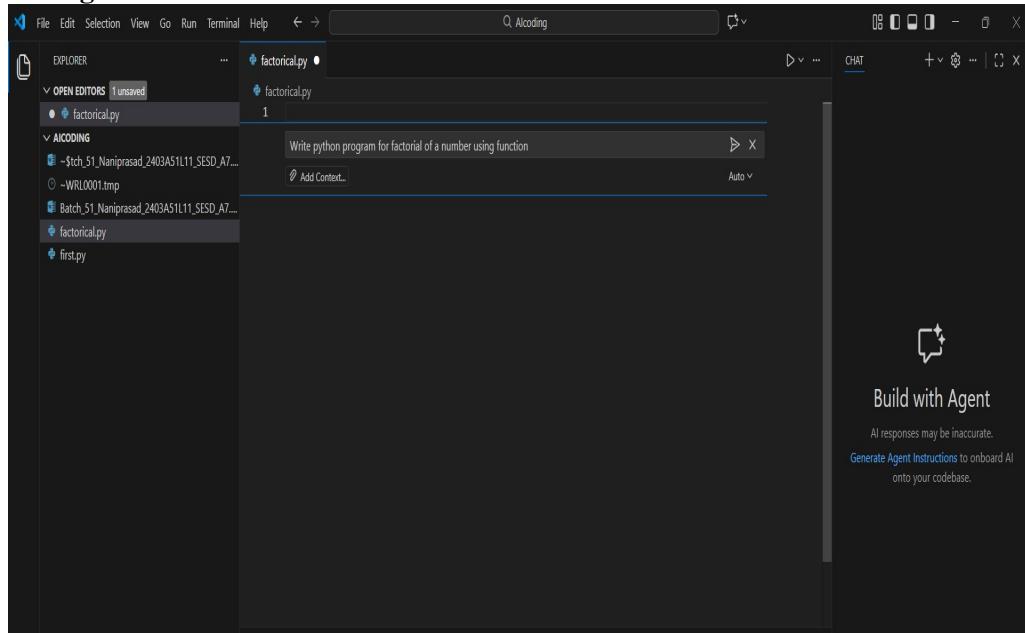
- ❖ The loop logic is self-explanatory, so the comment was removed.
- ❖ # Why the new version is better?
 - ❖ Readability
 - ❖ *= is clearer and more concise.
 - Fewer lines and less clutter make the code easier to read.
- ❖ Maintainability
 - Cleaner code is easier to modify and debug.
 - Reduced redundancy 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)}")

```

❖ 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.

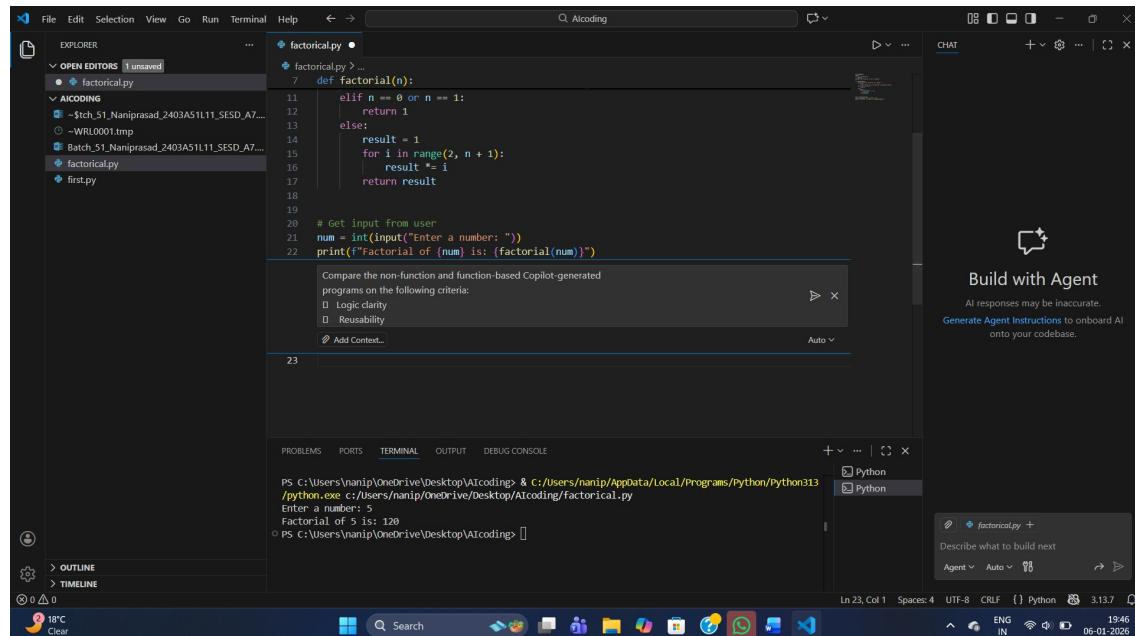
Supporting scalability, as the same function can be extended or integrated into larger projects.

#Task4

Task Description

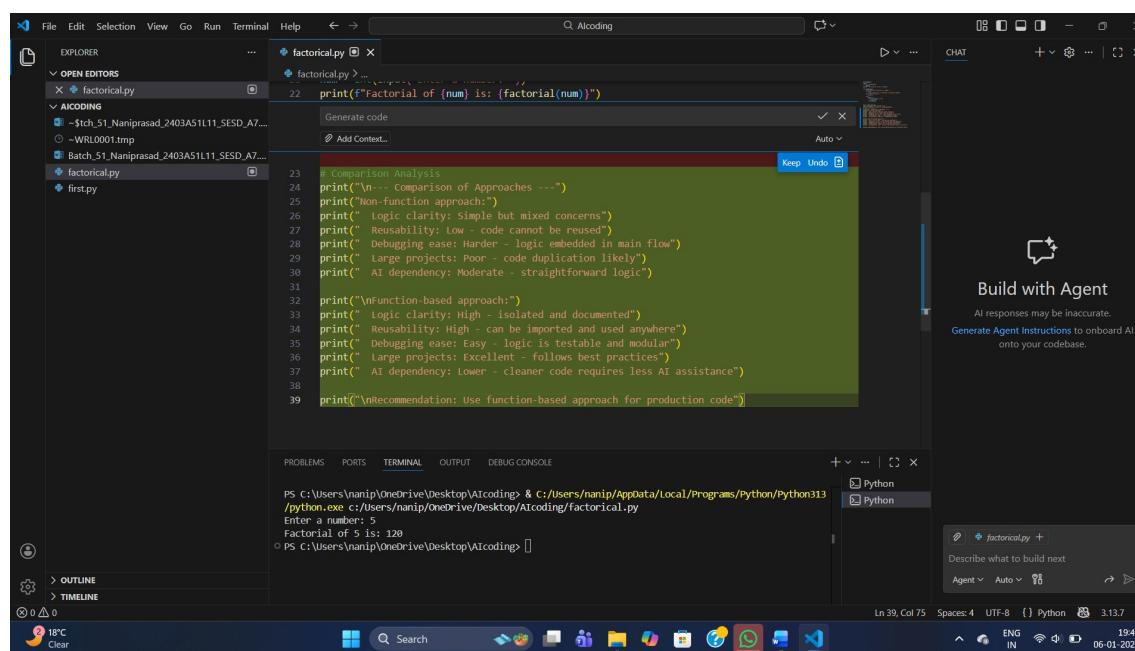
Compare the non-function and function-based Copilot-generated programs on the following criteria:

- ❖ Logic clarity
- ❖ Reusability
- ❖ Debugging ease
- ❖ Suitability for large projects
- ❖ AI dependency risk



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

- Explorer:** Shows files factorial.py, factorialcal.py, and first.py.
- Editor:** Displays the code for factorialcal.py, which contains a recursive factorial function and a main loop to get user input.
- Terminal:** Shows the command `python factorialcal.py` being run, with the output "Factorial of 5 is: 120".
- Chat Panel:** A tooltip from the AI Agent asks to "Compare the non-function and function-based Copilot-generated programs on the following criteria:" followed by a list of checkboxes: Logic clarity, Reusability, and Add Context... (disabled).
- Bottom Status Bar:** Shows the date and time (06-01-2026), battery level (18%), and system icons.



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

- Explorer:** Shows files factorial.py, factorialcal.py, and first.py.
- Editor:** Displays the code for factorialcal.py, which includes a comparison analysis section at the bottom.
- Terminal:** Shows the command `python factorialcal.py` being run, with the output "Factorial of 5 is: 120".
- Chat Panel:** A tooltip from the AI Agent asks to "Describe what to build next" and provides a link to "Generate Agent Instructions".
- Bottom Status Bar:** Shows the date and time (06-01-2026), battery level (18%), and system icons.

The screenshot shows the Visual Studio Code interface with the 'AI CODING' extension active. The left sidebar shows files like 'factorical.py', 'first.py', and 'Batch_51_Naniprasad_2403A51L11_SESD_A7...'. The main editor window displays Python code for calculating factorials, including a comparison of non-function and function-based approaches. The bottom right corner features a 'Build with Agent' panel where users can describe what to build next.

```

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

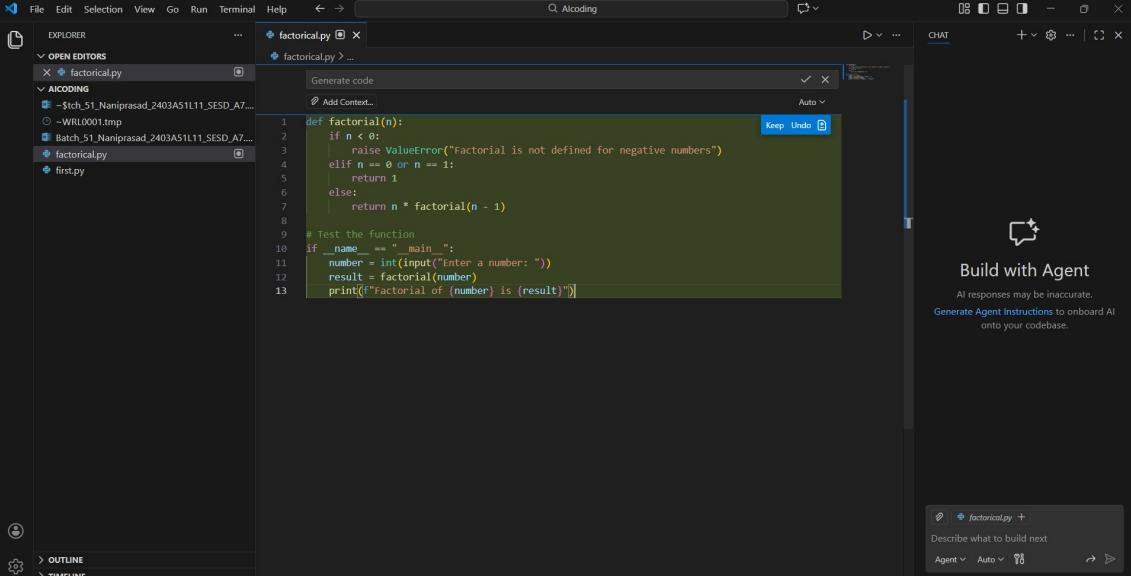
Task Description

Prompt Copilot to generate:

An iterative version of the logic

A recursive version of the same logic

This screenshot shows the same VS Code environment but with a different prompt in the AI panel: 'write the python program using recursion for factorialof number'. The rest of the interface and code are identical to the first screenshot.



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

- File Explorer:** Shows files factorial.py, first.py, and several temporary files.
- Editor:** Displays the Python code for factorial.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}")
      
```
- AI Panel:** Shows a "Build with Agent" section with a message: "All responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase."
- Terminal:** Shows the command PS C:\Users\nanip\OneDrive\Desktop\Alcoding & c:/Users/nanip/AppData/Local/Programs/Python/Python313/python.exe c:/users/nanip/Desktop/Alcoding/factorial.py and the output: Enter a number: 5 Factorial of 5 is 120.
- Bottom Bar:** Includes icons for search, file operations, and system status.

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).

