

## 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** : Harshavardhan  
**Enrollment No.** : 2403A51L37  
**Batch No.** : 52  
**Date** : 06-01-2026

### Submission Starts here

**OUTPUT :**

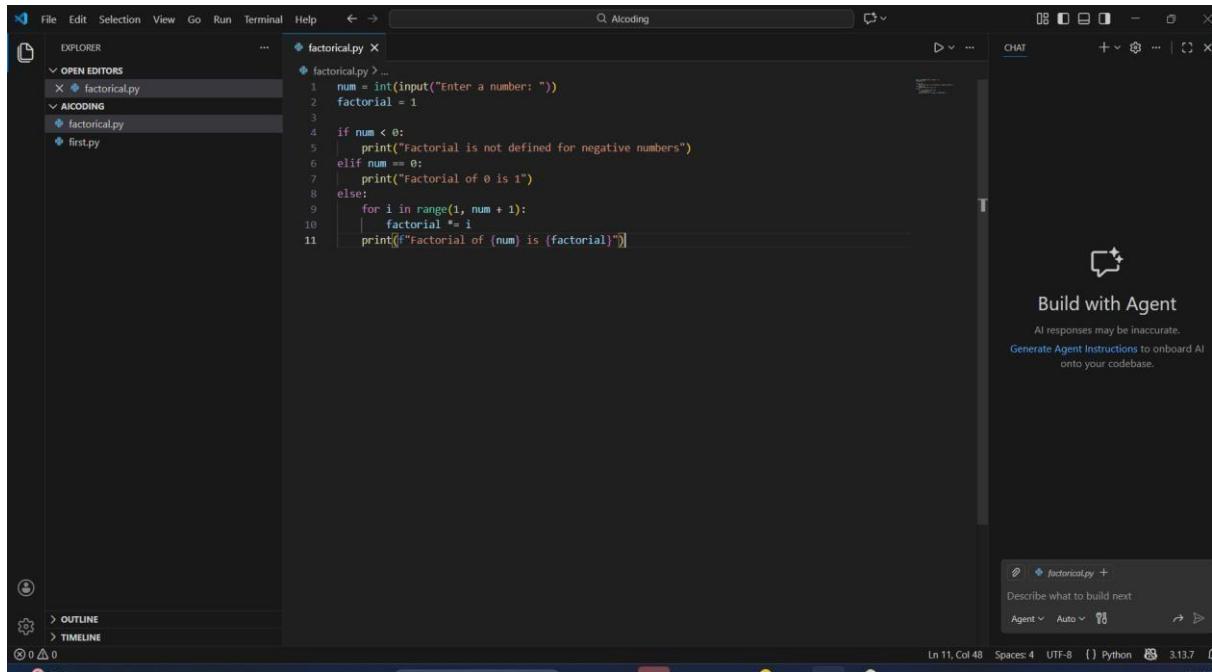
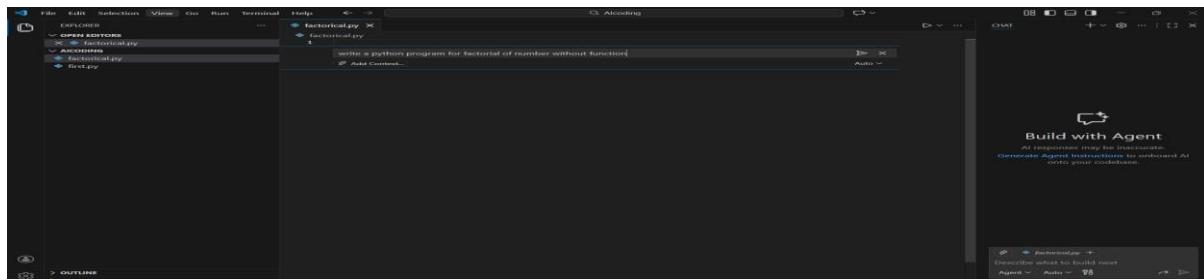
**SCREENSHOTS:**

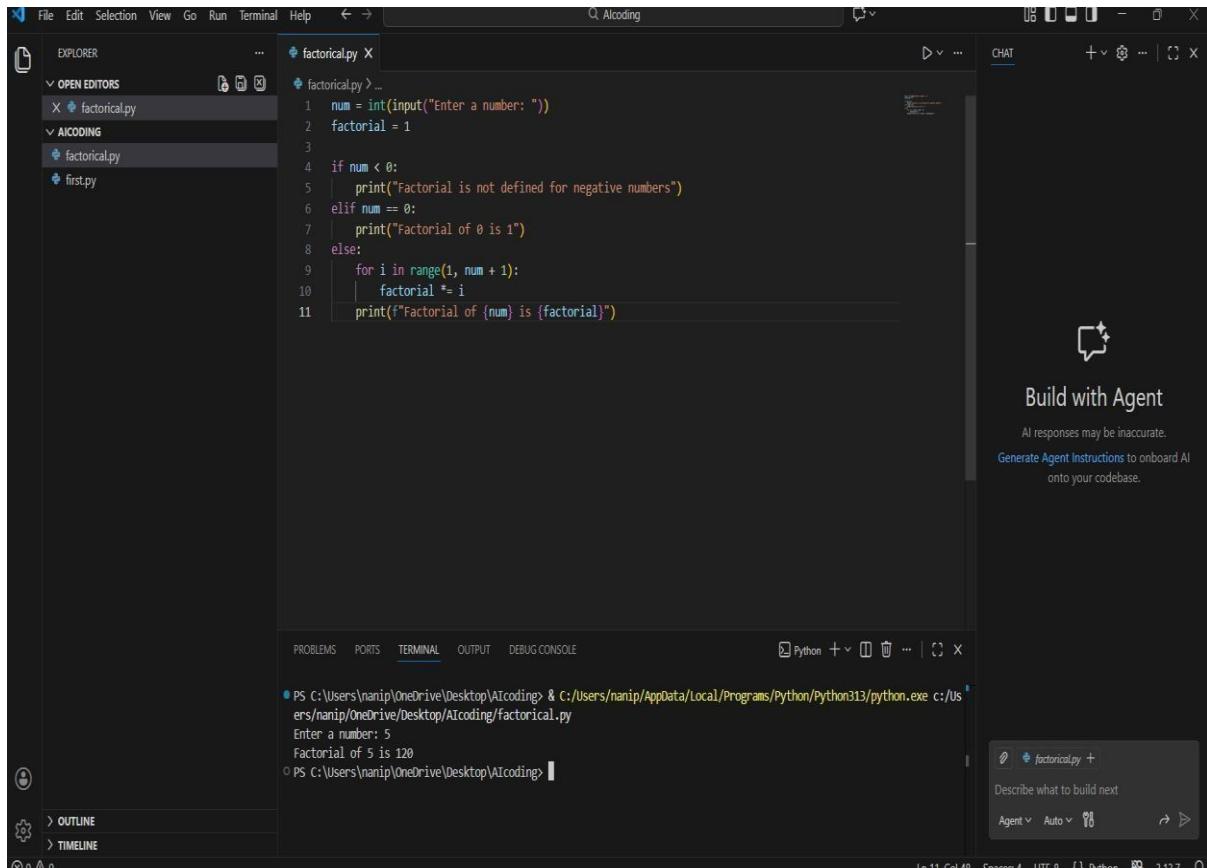
**Task 0: Install and configure GitHub Copilot in VS Code. Take screenshots of each step.**



### 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 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:

- File Explorer:** Shows files like factorial.py, first.py, and several temporary files.
- Code Editor:** Displays the factorial.py code:

```

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
    
```
- Terminal:** Shows the command PS C:\Users\nanip\OneDrive\Desktop\AIcoding> followed by the output of the script execution.
- AI Coding Panel:** A floating panel titled "Build with Agent" with the sub-instruction "Describe what to build next". It includes dropdowns for "Agent" (set to "Auto") and "Python" (set to "C:\Users\nanip\AppData\Local\Programs\Python\python313\python.exe").
- Bottom Status Bar:** Shows the file path, line number (Ln 2, Col 7), spaces (Spaces: 4), encoding (UTF-8), and other system information.

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

- File Explorer:** Shows files like factorial.py, first.py, and several temporary files.
- Code Editor:** Displays the factorial.py code, identical to the previous screenshot.
- Terminal:** Shows 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/factorial.py followed by the output of the script execution.
- AI Coding Panel:** A floating panel titled "Build with Agent" with the sub-instruction "Describe what to build next". It includes dropdowns for "Agent" (set to "Auto") and "Python" (set to "C:\Users\nanip\AppData\Local\Programs\Python\python313\python.exe").
- Bottom Status Bar:** Shows the file path, line number (Ln 2, Col 7), spaces (Spaces: 4), encoding (UTF-8), and other system information.

## # 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

- 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.

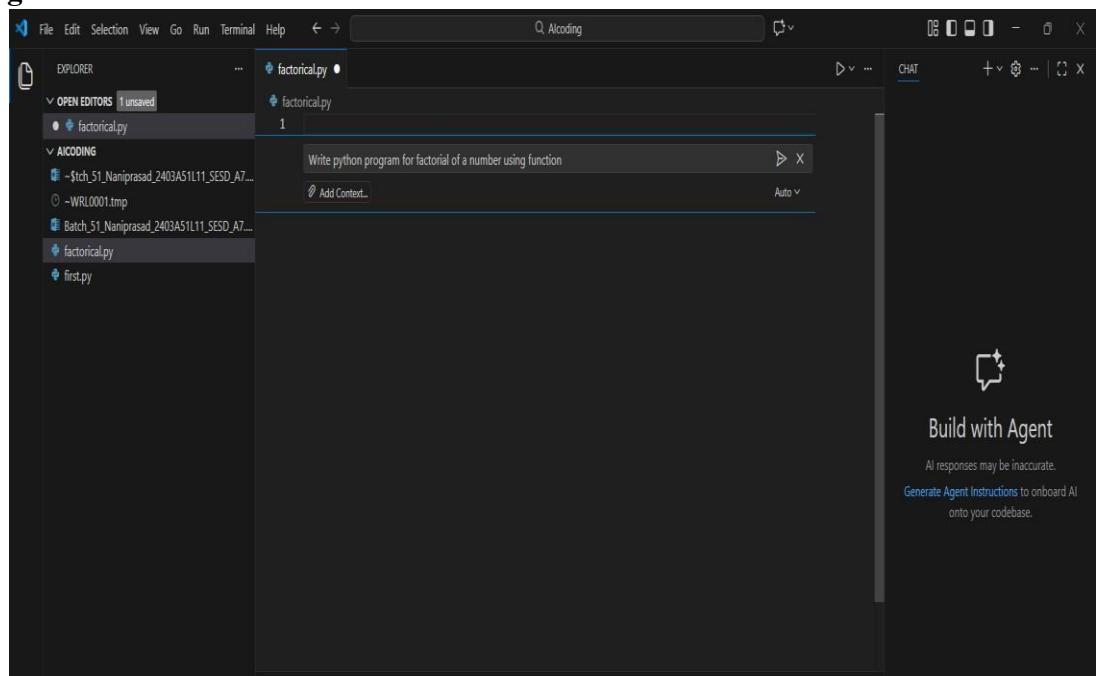
† \*= is marginally optimized at the bytecode level, but the difference is negligible.

### # 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)}")

```

† 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
- Add Context...

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>

Build with Agent

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

```

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")

```

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>

Build with Agent

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

The screenshot shows the VS Code interface with the 'factorical.py' file open in the editor. The code implements a factorial function using recursion and includes a comparison of its logic with a non-function-based approach. The AI Coding extension is active, providing analysis and suggestions. The terminal shows the command used to run the script.

```

1 # Get input from user
2 num = int(input("Enter a number: "))
3 print(f"Factorial of {num} is: {factorial(num)}")
4
5 # Comparison Analysis
6 print("\n--- Comparison of Approaches ---")
7 print("Non-function approach:")
8 print(" Logic clarity: Simple but mixed concerns")
9 print(" Reusability: Low - code cannot be reused")
10 print(" Debugging ease: Harder - logic embedded in main flow")
11 print(" Large projects: Poor - code duplication likely")
12 print(" AI dependency: Moderate - straightforward logic")
13
14 print("\nFunction-based approach:")
15 print(" Logic clarity: High - isolated and documented")
16 print(" Reusability: High - can be imported and used anywhere")
17 print(" Debugging ease: Easy - logic is testable and modular")
18 print(" Large projects: Excellent - follows best practices")
19 print(" AI dependency: Lower - cleaner code requires less AI assistance")
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

```

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

## # TASK - 5

### Task Description

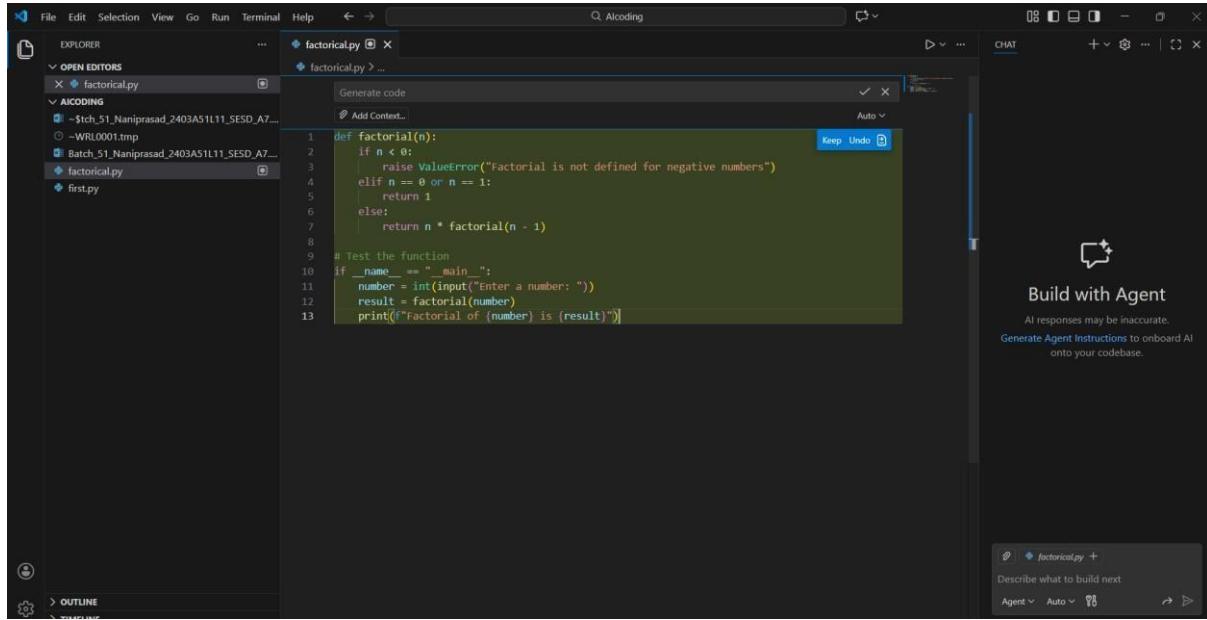
Prompt Copilot to generate:

An iterative version of the logic

A recursive version of the same logic

The screenshot shows the VS Code interface with the 'factorical.py' file open in the editor. A prompt is displayed in the AI Coding panel asking for a recursive version of the factorial logic. The AI Coding extension is active, providing suggestions and context for the code generation process.

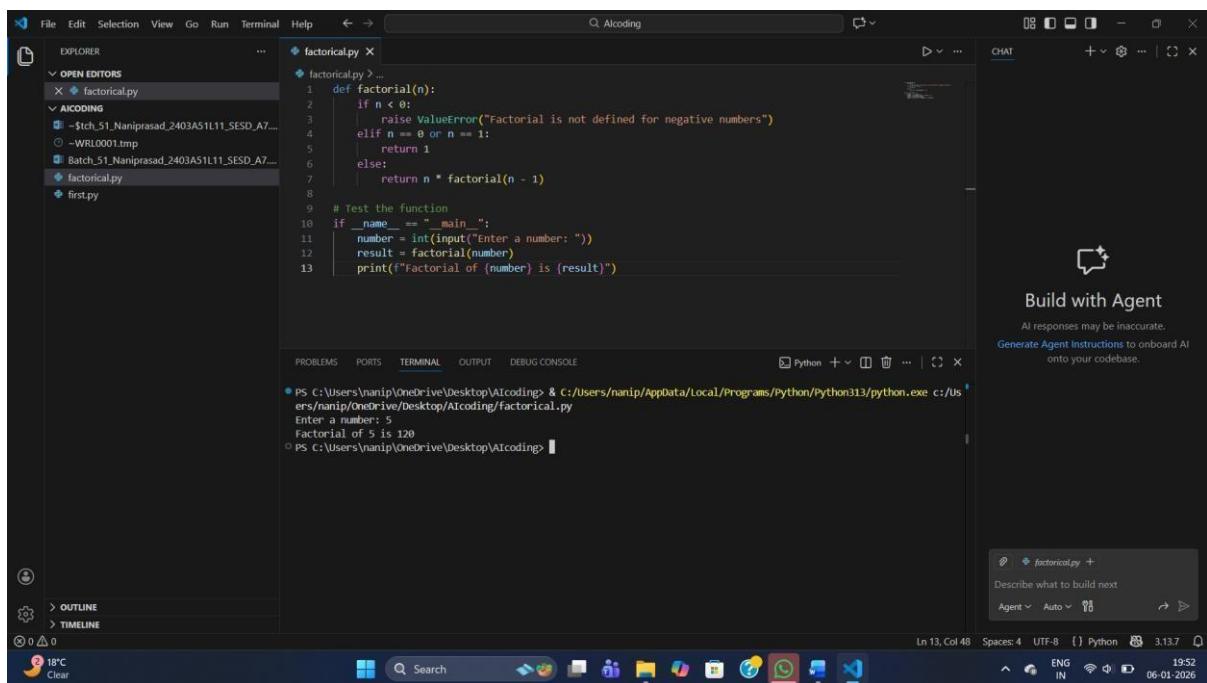
write the python program using recursion for factorialof number



```

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}")

```



```

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}")

```

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>

```

Ln 13, Col 48 Spaces: 4 UTF-8 Python 3.13.7 06-01-2026

## 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).

