

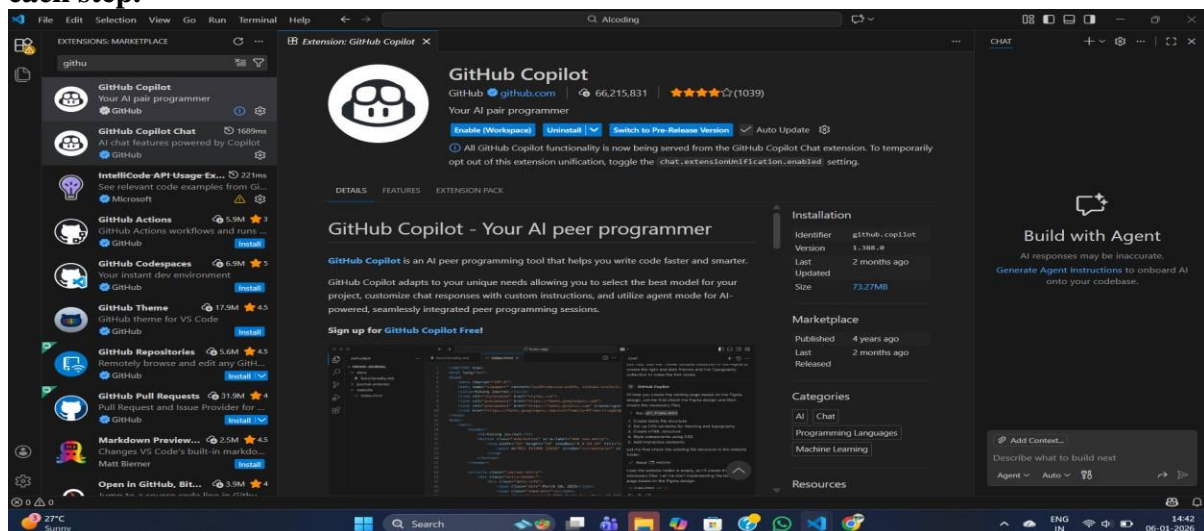
Lab Assignment # 1

Program : B. Tech (CSE)
Specialization :
Course Title : AI Assisted coding
Course Code :
Semester II
Academic Session : 2025-2026
Name of Student : R.Akash Reddy
Enrollment No. : 2403A51L30
Batch No. 51
Date :06-01-2026

Submission Starts here

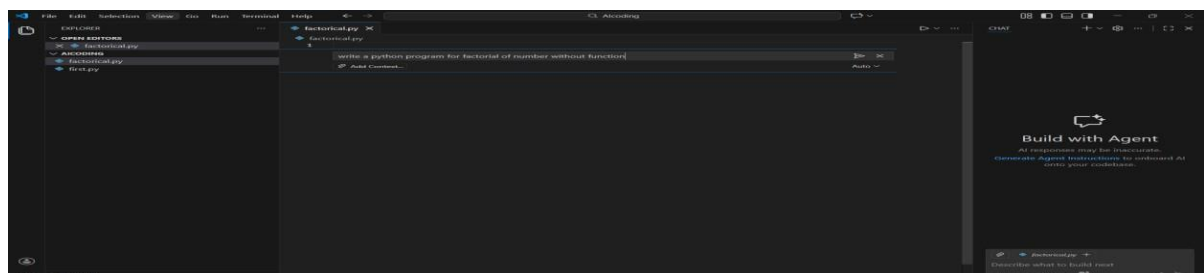
OUTPUT :
SCREENSHOTS:

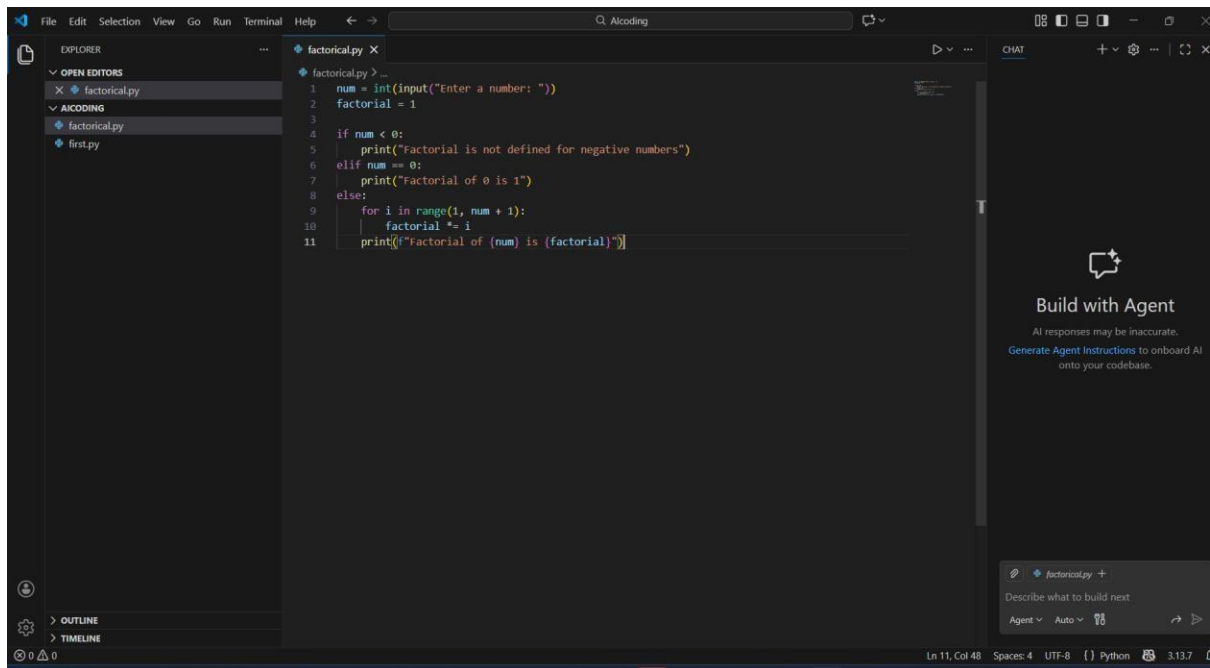
Task 0: Install and configure GitHub Copilot in VS Code. Take screenshots of each step.



Task1: Task Description

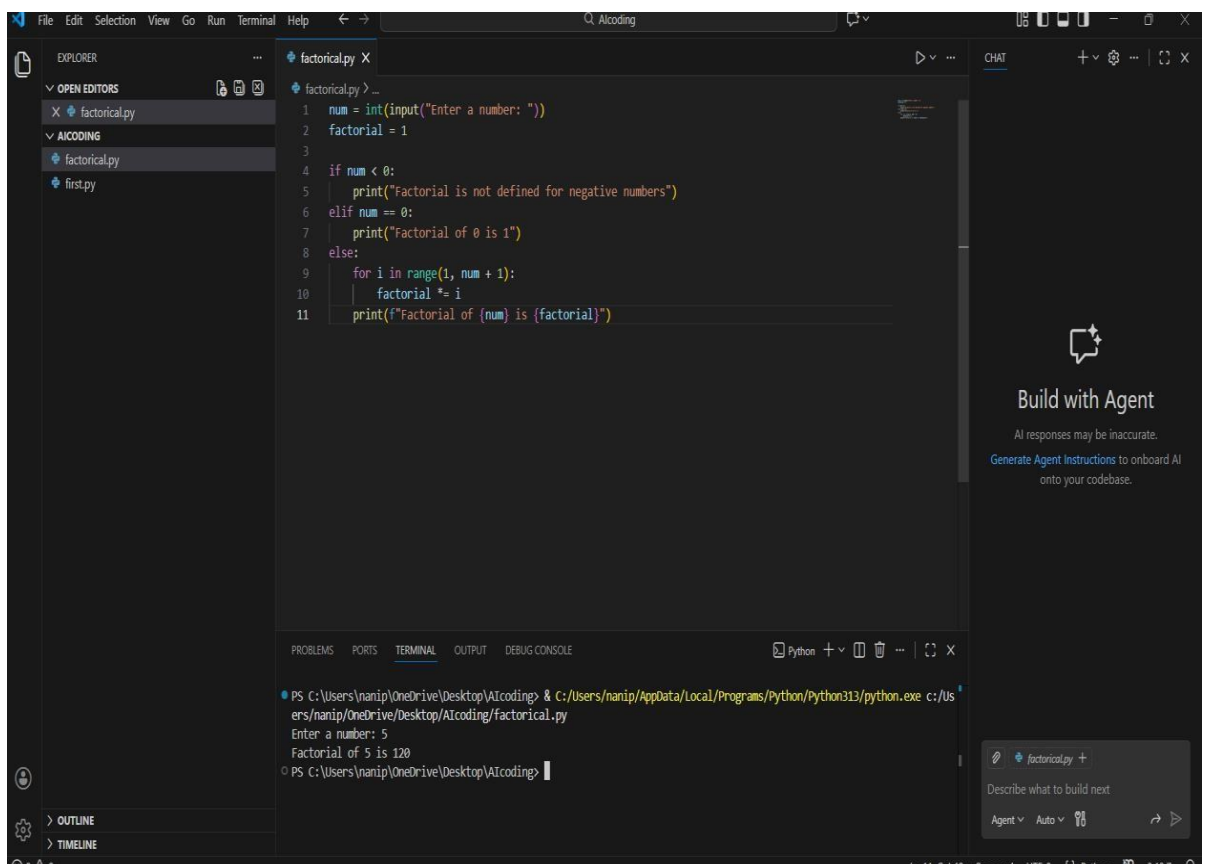
Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.





This screenshot shows the Visual Studio Code editor with a Python file named `factorial.py` open. The code implements a factorial function with error handling for negative numbers and zero. The interface includes the Explorer sidebar on the left, the main editor window, and the Chat sidebar on the right with a 'Build with Agent' prompt.

```
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}")
```



This screenshot shows the same Visual Studio Code editor with the `factorial.py` file. The terminal window at the bottom is active, showing the command to run the script and its output. The Chat sidebar on the right remains visible.

```
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 Visual Studio Code editor with a file named `factorial.py` open. The code is as follows:

```
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 Explorer sidebar on the left shows the project structure with files like `factorial.py` and `first.py`. The bottom status bar indicates the file is at Line 2, Column 7.

The screenshot shows the same Visual Studio Code editor with the `factorial.py` file. The code has been improved to use a more concise multiplication statement:

```
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 terminal at the bottom shows the command to run the script: `python.exe c:/Users/nanip/OneDrive/Desktop/AIcoding/factorial.py`, and the output: `the factorial of 5 is 120`. The status bar now shows Line 2, Column 8.

What was improved?

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

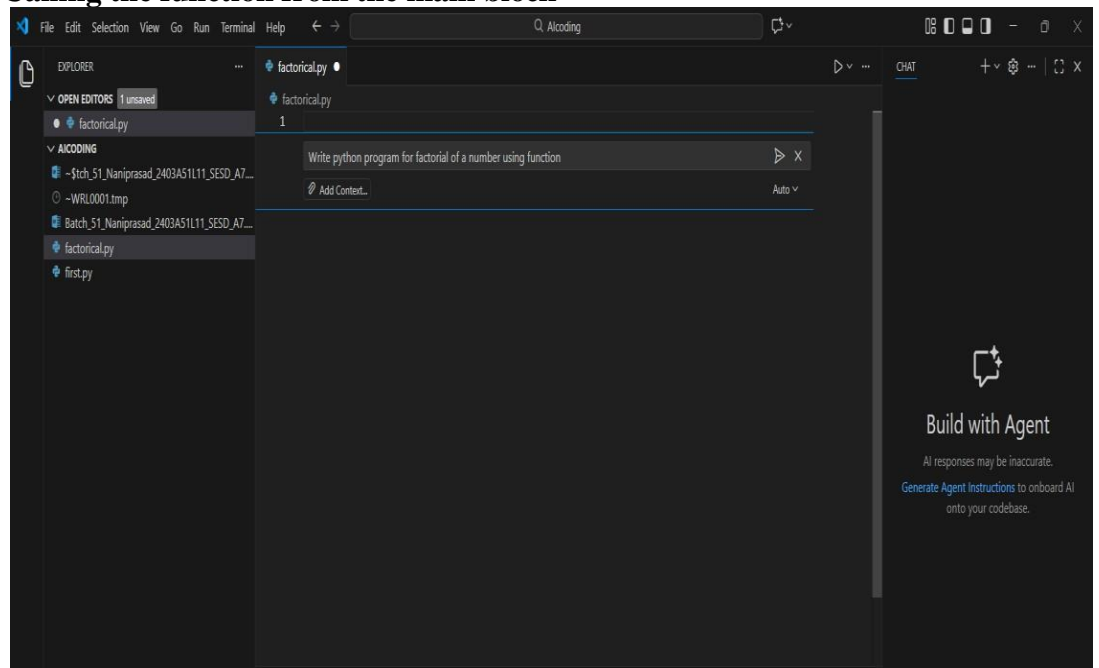
- ❖ The loop logic is self-explanatory, so the comment was removed.
- ❖ # Why the new version is better?
- ❖ Readability
 - ❖ *= is clearer and more concise.
 - Fewer lines and less clutter make the code easier to read.
- ❖ Maintainability
 - Cleaner code is easier to modify and debug.
 - Reduced redundancy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



The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left shows the file structure with 'factorial.py' selected. The main editor area displays the Python code for a factorial function. A 'Generate code' button is visible in the top right of the editor. The right sidebar contains a 'Build with Agent' section with a chat interface.

```

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 # Get input from user
14 num = int(input("Enter a number: "))
15 print(f"Factorial of {num} is: {factorial(num)}")

```

This screenshot shows the same VS Code editor interface, but with the terminal window open at the bottom. The terminal displays the command to run the script and its output.

```

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

```

❖ **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 Visual Studio Code editor with a file named `factorial.py` open. The code defines a `factorial` function and uses it to calculate the factorial of a user input. A chat window on the right shows a prompt to compare non-function and function-based Copilot-generated programs. The terminal shows the execution of the program, which outputs 'Factorial of 5 is: 120'.

```

7 def factorial(n):
11     elif n == 0 or n == 1:
12         return 1
13     else:
14         result = 1
15         for i in range(2, n + 1):
16             result *= i
17         return result
18
19 # Get input from user
20 num = int(input("Enter a number: "))
21 print(f"Factorial of {num} is: {factorial(num)}")
22
23

```

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

- ☐ Logic clarity
- ☐ Reusability

Terminal output:

```

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

```

The screenshot shows the Visual Studio Code editor with a file named `factorial.py` open. The code defines a `factorial` function and uses it to calculate the factorial of a user input. A chat window on the right shows a prompt to compare non-function and function-based Copilot-generated programs. The terminal shows the execution of the program, which outputs 'Factorial of 5 is: 120'.

```

22 print(f"Factorial of {num} is: {factorial(num)}")
23
24 # Comparison Analysis
25 print("\n-- Comparison of Approaches ----")
26 print("Non-function approach:")
27 print("  Logic clarity: Simple but mixed concerns")
28 print("  Reusability: Low - code cannot be reused")
29 print("  Debugging ease: Harder - logic embedded in main flow")
30 print("  Large projects: Poor - code duplication likely")
31 print("  AI dependency: Moderate - straightforward logic")
32
33 print("\nFunction-based approach:")
34 print("  Logic clarity: High - isolated and documented")
35 print("  Reusability: High - can be imported and used anywhere")
36 print("  Debugging ease: Easy - logic is testable and modular")
37 print("  Large projects: Excellent - follows best practices")
38 print("  AI dependency: Lower - cleaner code requires less AI assistance")
39 print("\nRecommendation: Use function-based approach for production code")

```

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

- ☐ Logic clarity
- ☐ Reusability

Terminal output:

```

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

```


The screenshot shows a VS Code editor with a file named `factorial.py`. The code implements a factorial function and compares two approaches: a non-function approach and a function-based approach. The terminal output shows the results of running the script.

```

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 Output:

```

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

--- Comparison of Approaches ---
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

```

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 VS Code editor with a file named `factorial.py`. A prompt is entered in the editor: "write the python program using recursion for factorial of number". The terminal output shows the results of running the script.

```

1

```

Terminal Output:

```

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

--- Comparison of Approaches ---
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

```

```

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}")
  
```

```

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>
  
```

Explanation:

How the Function Works

- Negative number check**
Factorials are not defined for negative numbers. If the input is negative, the program raises an error message.
 - Base cases**
For 0 and 1, the factorial is defined as 1. This acts as the stopping condition for recursion.
 - 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).

