

Lab Assignment 1.2 – AI Assisted Coding

V.Amruth Sagar

2403A51L44

B:52

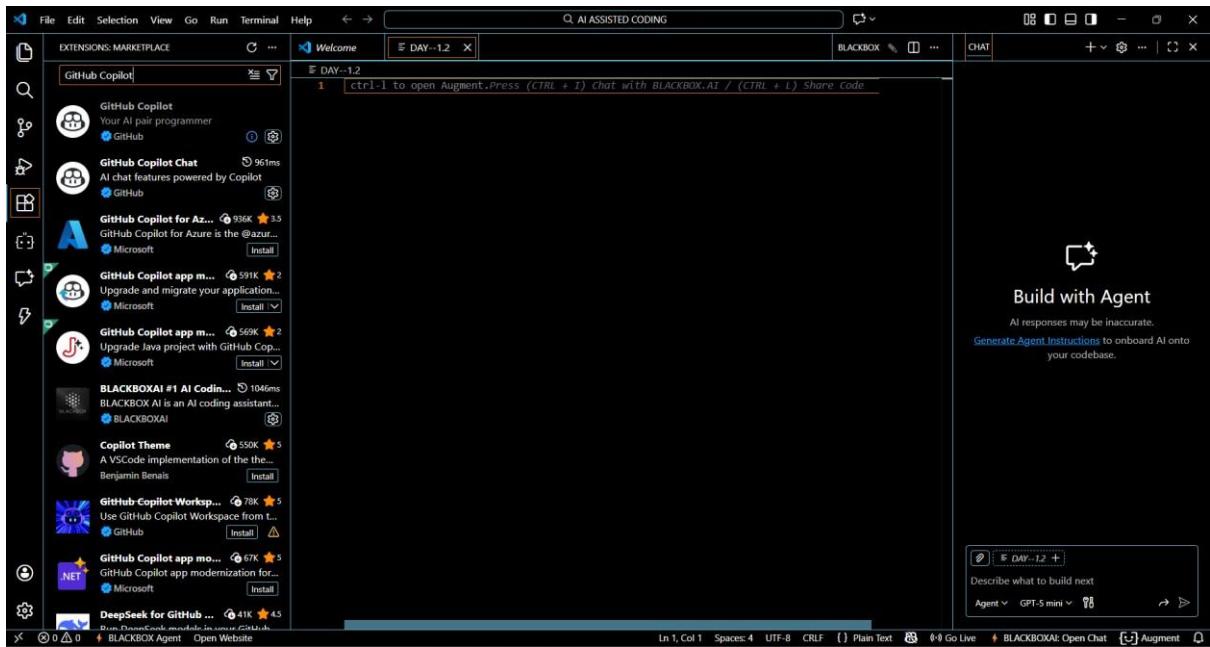
Task 0: GitHub Copilot Installation & Configuration

Steps Followed:

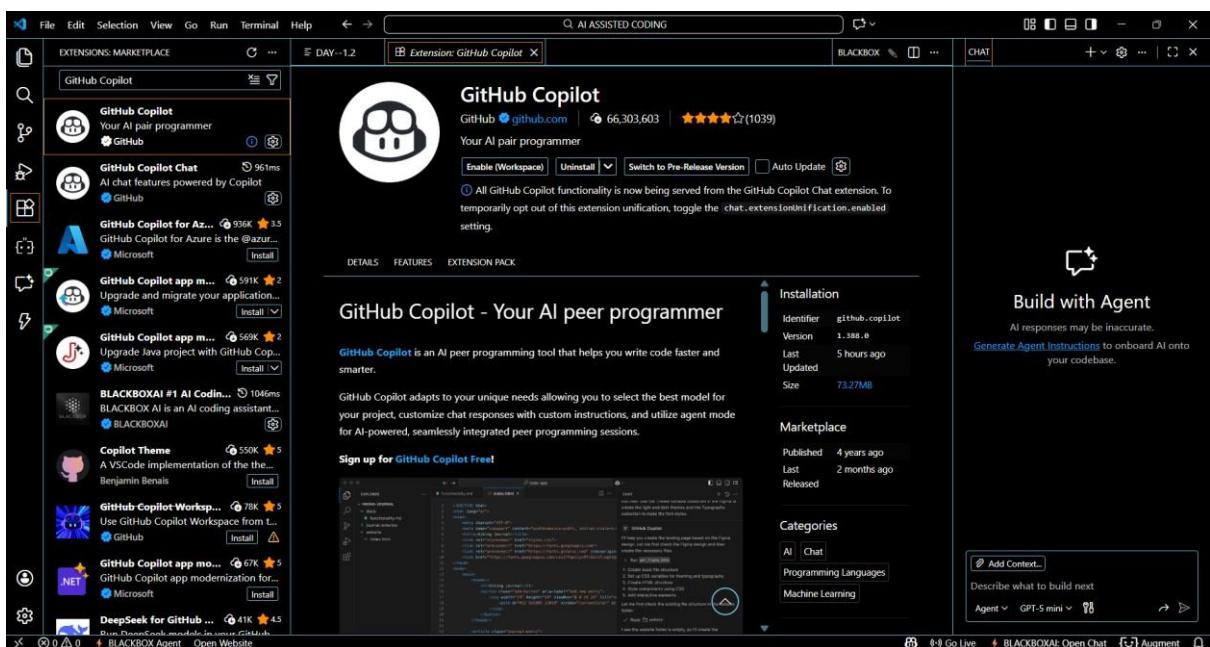
1. Installed Visual Studio Code
2. Opened Extensions Marketplace



3. Searched for GitHub Copilot



4. Clicked Install



5. Signed in with GitHub Account

6. Enabled Copilot suggestions

7. Verified Copilot inline suggestions in Python file

The screenshot shows the Microsoft Visual Studio Code interface with the "AI ASSISTED CODING" extension active. The top bar has tabs for "File", "Edit", "Selection", "View", "Go", "Run", "Terminal", and "Help". The title bar says "AI ASSISTED CODING". The left sidebar includes icons for Explorer, Search, Find, Open, Save, Undo, Redo, and Settings. The main editor window displays a Python script named "DAY-1.2.py" with code to calculate factorial. A tooltip from the "Accept Word" feature is shown over the word "factorial". The bottom navigation bar includes "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "TERMINAL" (which is selected), "PORTS", and "AUGMENT NEXT EDIT". The terminal pane shows command-line output for running the script. On the right, there's a "Build with Agent" panel with a "Generate Agent Instructions" button and a status message about AI responses being inaccurate.

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

Prompt Used: “Write a Python program to calculate factorial of a number using loops only, without defining any function.”

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

- File Explorer:** Shows a folder named "AI ASSISTED CODING" containing files "DAY-1.2.py" and "DAY-1.2.py > ...".
- Editor:** Displays the content of "DAY-1.2.py". The code calculates the factorial of a number using loops only, without defining any functions.
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, AUGMENT NEXT EDIT, and icons for Python, Run, Stop, and Save.
- Terminal:** Shows the command "PS C:\Users\sarik\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/sarik/OneDrive/Desktop/ai_assisted_coding/day-1.2.py"" followed by user input "Enter a number: 4" and the output "Factorial is: 24".
- Right Sidebar:** Features a "Build with Agent" section with a "Build" button, a note about AI responses being inaccurate, and a "Generate Agent Instructions" link.
- Bottom Status Bar:** Shows file path "ln 7, Col 31", character count "Spaces: 4", encoding "UTF-8", line endings "CRLF", Python version "3.13.9 (Microsoft Store)", and status icons for Go Live, BLACKBOX, Open Chat, and Augment.

GitHub Copilot was very helpful for a beginner as it generated correct logic instantly.

It followed basic Python syntax and loop structure accurately.

The code was readable and easy to understand.
However, it did not include input validation automatically.
Best practices like modular design were not applied unless explicitly prompted.

Task 2: AI Code Optimization & Cleanup Original

Code:

The screenshot shows the AI ASSISTED CODING interface. In the Explorer panel, there is a file named 'DAY-1.2.py'. The code in the editor is:

```
1 """Write a Python program to calculate factorial of a number using loops only, without defining any function."""
2
3 n = int(input("Enter a number: "))
4 result = 1
5 for i in range(1, n + 1):
6     result = result * i
7 print("Factorial is:", result)
```

In the Terminal tab, the command 'python DAY-1.2.py' is run, and the output is:

```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> python DAY-1.2.py
Enter a number: 2
Factorial is: 2
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING>
```

The interface includes a sidebar for 'Build with Agent' and a bottom right corner for 'Describe what to build'.

Prompt Used: “Optimize this code and make it more readable”

```

1  """Write a Python program to calculate factorial of a number using loops only, without defining any function."""
2
3  n = int(input("Enter a number: "))
4  result = 1
5  for i in range(1, n + 1):
6      result *= i
7  print("Factorial is:", result)
8
9
10 """Optimize this code and make it more readable"""
11
12 n = int(input("Enter a number: "))
13 factorial = 1
14 for i in range(1, n + 1):
15     factorial *= i
16 print(f"Factorial of {n} is: {factorial}")

```

The optimized version improves clarity, maintainability, and readability without affecting performance.

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

Prompt Used: “Create a Python function to calculate factorial and call it from main block”

```

1  """Create a Python function to calculate factorial and call it from main block"""
2
3  def calculate_factorial(num):
4      """Returns factorial of a number"""
5      result = 1
6      for i in range(1, num + 1):
7          result *= i
8      return result
9
10 number = int(input("Enter a number: "))
11 print("Factorial is:", calculate_factorial(number))

```

Modularity improves reusability by allowing the same function to be used across multiple programs. It also simplifies testing and debugging.

Task 4: Comparative Analysis

Procedural vs Modular AI Code

Criteria	Without Function		With Function
	Moderate	High	
Logic Clarity	No	Yes	
Debugging Ease	Difficult	Easy	
Large Project Suitability	Poor	Excellent	
AI Dependency Risk	Higher	Lower	

Conclusion:

Function-based design is more scalable and suitable for real-world applications.

Task 5: Iterative vs Recursive AI Code

Prompt Used: “Generate iterative and recursive factorial programs in Python”

The screenshot shows a software interface titled "AI ASSISTED CODING". On the left is an "EXPLORER" sidebar with a tree view showing "AI ASSISTED CODING" and "DAY-1.2.py". The main area displays Python code for calculating factorials:

```

30     """Generate iterative and recursive factorial programs in Python"""
31
32     """
33     Iterative Version"""
34     def factorial_iterative(n):
35         result = 1
36         for i in range(1, n + 1):
37             result *= i
38         return result
39
40     """
41     Recursive Version"""
42     def factorial_recursive(n):
43         if n == 0 or n == 1:
44             return 1
45         return n * factorial_recursive(n - 1)
46 number = int(input("Enter a number: "))
47 print("Iterative Factorial is:", factorial_iterative(number))
48 print("Recursive Factorial is:", factorial_recursive(number))
49

```

Below the code editor is a "TERMINAL" tab showing command-line interaction:

```

PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/users/sarik/OneDrive/Desktop/AI ASSISTED CODING/DAY-1.2.py"
● Enter a number: 4
Iterative Factorial is: 24
Recursive Factorial is: 24
○ PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING>

```

At the bottom right, there's a "Build with Agent" panel with a "Generate Agent Instructions to onboard AI onto your codebase" button.

Execution Flow Explanation:

- Iterative version uses a loop and constant memory.
- Recursive version uses function calls and stack memory.

Comparison:

Aspect	Iterative	Recursive
Readability	Simple	Elegant
Stack Usage	No	Yes
Performance	Faster	Slower
Risk	Low	Stack Overflow
Recommendation	Preferred	Avoid for large inputs