

Assignment -1.5

Name: N. SAI PAVAN

Roll Number: 2303A51117

Batch - 03

AI Assisted Coding

09-01-2026

Task 0: Environment Setup:-

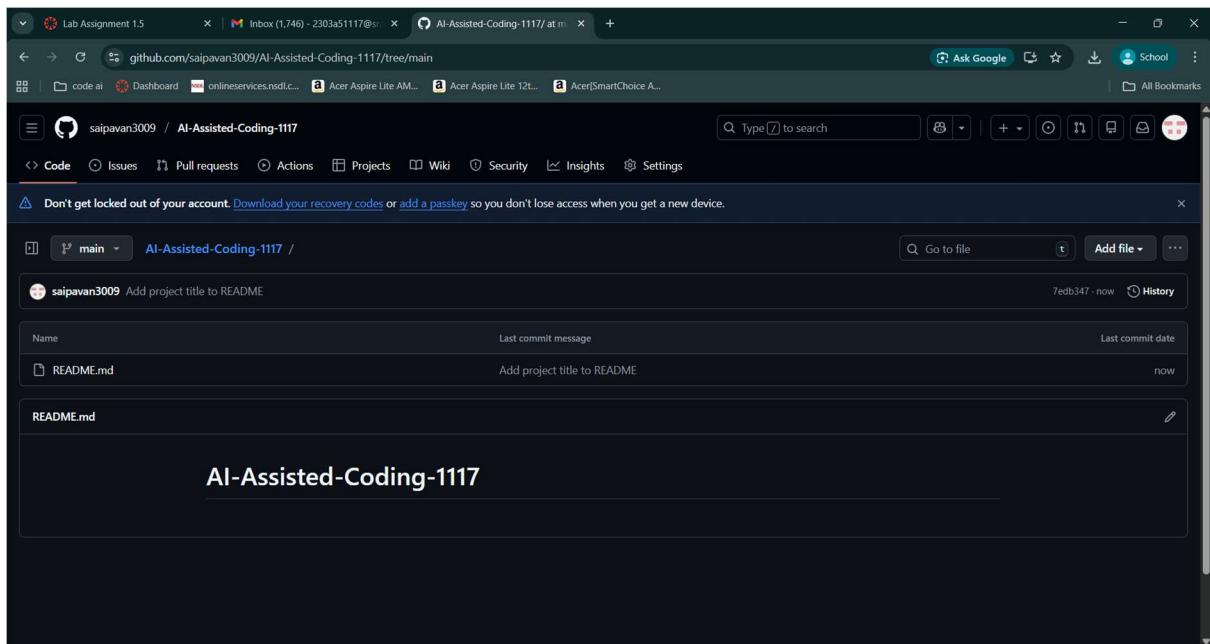
Task 0

- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

Expected Output

- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.





Task 1: Non-Modular Logic (Factorial):-

: AI-Generated Logic Without Modularization (String Reversal Without Functions)

❖ Scenario

You are developing a basic text-processing utility for a messaging application.

❖ Task Description

Use GitHub Copilot to generate a Python program that:

- Reverses a given string
- Accepts user input
- Implements the logic directly in the main code
- Does not use any user-defined functions

❖ Expected Output

- Correct reversed string
- Screenshots showing Copilot-generated code suggestions

➤ Sample inputs and outputs

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

- File Explorer:** Shows files: factorial using recursion.py, lab_assignment_1_5.py (selected), and lab_assignment_1_5.py > ...
- Code Editor:** Displays Python code to reverse a string without using a function.

```
1 #write a program to reverse a string without unsing function
2 input_string = input("Enter a string: ")
3 reversed_string = ""
4 for char in input_string:
5     reversed_string = char + reversed_string
6 print("Reversed string:", reversed_string)
```
- Terminal:** Shows the execution of the code and its output.

```
AI Assisted Coding/lab_assignment_1_5.py"
Enter a string: APPLE
Reversed string: ELPPA
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_1_5.py"
Enter a string: madam
Reversed string: madam
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_1_5.py"
Enter a string: sir
Reversed string: ris
PS D:\AI Assisted Coding>
```
- Output Panel:** Shows Python logs with some warning icons.

Task 2: AI Code Optimization:-

Efficiency & Logic Optimization (Readability Improvement)

❖ Scenario

The code will be reviewed by other developers.

❖ Task Description

Examine the Copilot-generated code from Task 1 and improve it by:

- **Removing unnecessary variables**
- **Simplifying loop or indexing logic**
- **Improving readability**
- **Use Copilot prompts like:**
 - **“Simplify this string reversal code”**
 - **“Improve readability and efficiency”**

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

❖ Expected Output

➤ Original and optimized code versions

➤ Explanation of how the improvements reduce time complexity

The screenshot shows the Visual Studio Code (VS Code) interface with the "AI Assisted Coding" extension active. The terminal window displays the following Python code and its execution:

```
#write a program to reverse a string without unsing function
input_string = input("Enter a string: ")
reversed_string = input_string[::-1]
print("Reversed string:", reversed_string)
```

When run, the terminal shows:

```
Enter a string: 457
Reversed string: 754
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_1_5.py"
Enter a string: 4567
Reversed string: 7654
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/lab_assignment_1_5.py"
Enter a string: 457
Reversed string: 754
Enter a string: 
```

Task 3: Modular Design Using AI Assistance (String Reversal Using Functions)

❖ Scenario

The string reversal logic is needed in multiple parts of an application.

❖ Task Description

Use GitHub Copilot to generate a function-based Python program that:

➤ Uses a user-defined function to reverse a string

➤ Returns the reversed string

➤ Includes meaningful comments (AI-assisted)

❖ Expected Output

➤ Correct function-based implementation

➤ Screenshots documenting Copilot's function generation

➤ Sample test cases and outputs

The screenshot shows a dark-themed instance of Visual Studio Code. In the center, there is a code editor window displaying a Python script named `lab_assignment_1_5.py`. The script contains the following code:

```
#write a program to print reverse of a string using function
def reverse_string(s):
    reversed_s = ""
    for char in s:
        reversed_s = char + reversed_s
    return reversed_s
input_string = input("Enter a string: ")
print("Reversed string using function:", reverse_string(input_string))
```

Below the code editor, the terminal window shows the execution of the script. It prompts the user to enter a string, receives "457", and prints "Reversed string: 754". This indicates that the function is working correctly.

At the bottom right of the interface, there are several status indicators and icons, including a Python icon with a warning sign, which typically means there are errors or warnings in the Python code.

Task 4: Comparative Analysis – Procedural vs Modular Approach (With vs Without Functions)

❖ Scenario

You are asked to justify design choices during a code review.

❖ Task Description

Compare the Copilot-generated programs:

➤ Without functions (Task 1)

➤ With functions (Task 3)

Analyze them based on:

➤ Code clarity

➤ Reusability

➤ Debugging ease

➤ Suitability for large-scale applications

❖ Expected Output

Comparison table or short analytical report

Feature	Procedural (Without Functions)	Modular (With Functions)
Code Clarity	Easy for tiny scripts; messy for large ones.	Very high; logic is isolated and named.
Reusability	Must copy-paste code to use it again.	Can be called anywhere in the app.
Debugging	Harder to isolate where an error occurs.	Easy to unit test the specific function.
Scalability	Not suitable for large applications.	Essential for professional development.

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different

Algorithmic Approaches to String Reversal)

❖ **Scenario**

Your mentor wants to evaluate how AI handles alternative logic paths.

❖ **Task Description**

Prompt GitHub Copilot to generate:

- **A loop-based string reversal approach**
- **A built-in / slicing-based string reversal approach**

❖ **Expected Output**

- **Two correct implementations**

➤ **Comparison discussing:**

- **Execution flow**
- **Time complexity**
- **Performance for large inputs**
- **When each approach is appropriate.**

The screenshot shows a code editor interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Title Bar:** AI Assisted Coding
- Left Sidebar:** Includes icons for AI ASSISTED CODING, factorial using recursion.py, lab_assignment_1_5.py, and a search icon.
- Code Editor:** Displays Python code for reversing strings.

```
31 #write a program for Iterative vs Recursive string reverse (Different Algorithmic Approaches to String Reversal)
32 def reverse_string_iterative(s):
33     reversed_s = ""
34     for char in s:
35         reversed_s = char + reversed_s
36     return reversed_s
37 input_string = input("Enter a string: ")
38 print("Reversed string using iterative approach:", reverse_string_iterative(input_string))
39 def reverse_string_recursive(s):
40     if len(s) == 0:
41         return s
42     else:
43         return s[-1] + reverse_string_recursive(s[:-1])
44 input_string = input("Enter a string: ")
45 print("Reversed string using recursive approach:", reverse_string_recursive(input_string))
46
```
- Terminal:** Shows the output of the code execution:

```
Reversed string using iterative approach: dlrow olleh
Enter a string: hello world
Reversed string using recursive approach: dlrow olleh
PS D:\AI Assisted Coding>
```
- Bottom Status Bar:** Line 46, Col 1, Spaces:4, UTF-8, CRLF, Python 3.11.4, Go Live, Prettier

Assignment – 1.1

Name: N. SAI PAVAN

Roll Number: 2303A51117

Batch - 03

AI Assisted Coding

07-01-2026

Task 0: Environment Setup:-

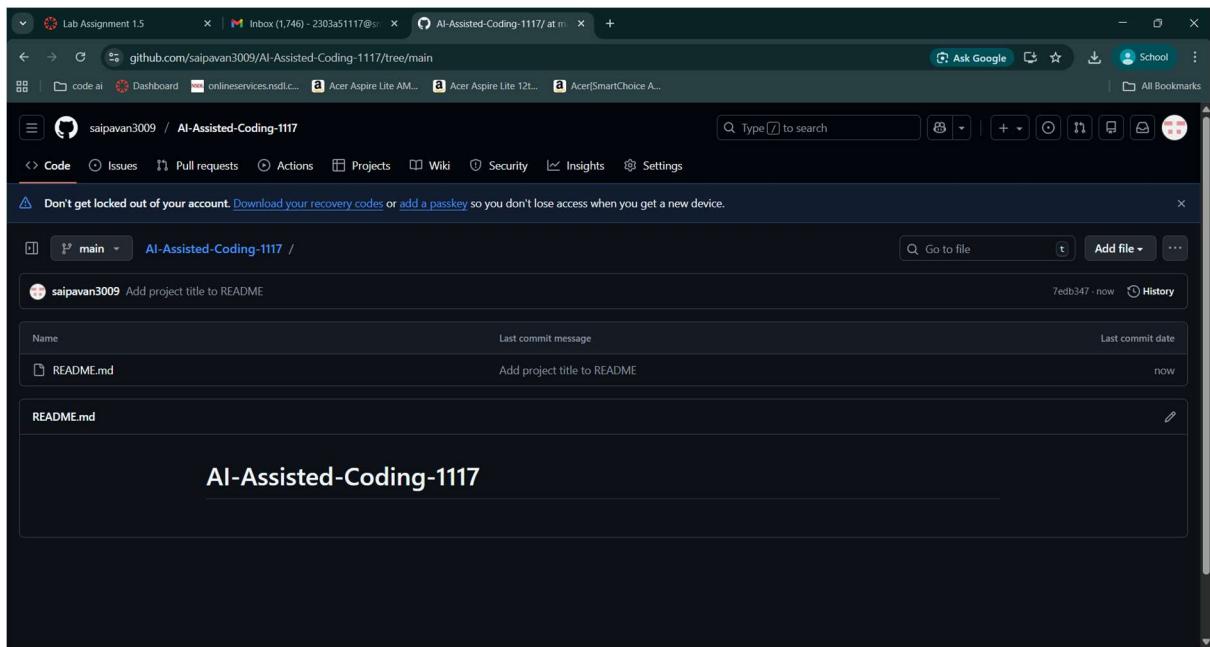
Task 0

- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

Expected Output

- Install and configure GitHub Copilot in VS Code. Take screenshots of each step.





Task 1: Non-Modular Logic (Factorial):-

AI-Generated Logic Without Modularization (Factorial without Functions)

- Scenario

You are building a small command-line utility for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

- 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.

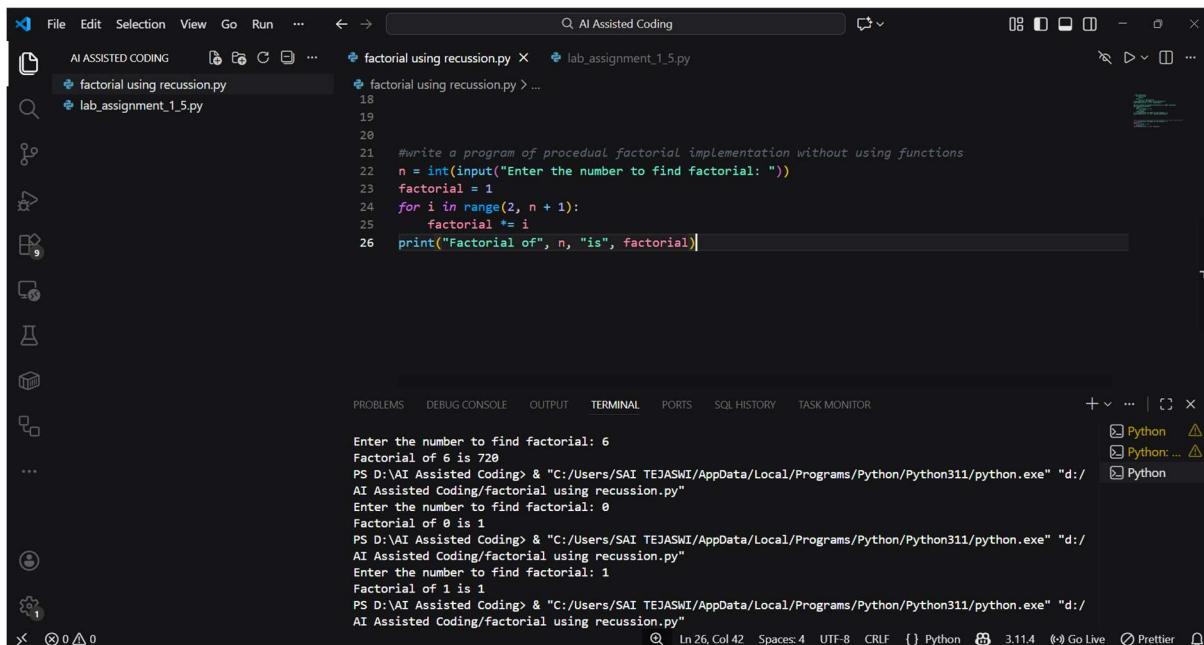
- Constraint:

- Do not define any custom function
- Logic must be implemented using loops and variables only

- Expected Deliverables

- A working Python program generated with Copilot assistance
- Screenshot(s) showing:

- The prompt you typed
- Copilot's suggestions
- Sample input/output screenshots
- Brief reflection (5–6 lines):
- How helpful was Copilot for a beginner?
- Did it follow best practices automatically?



The screenshot shows the VS Code interface with the title bar "AI Assisted Coding". In the left sidebar, there are icons for file operations like Open, Save, and Close, along with a search icon. The main workspace shows two files: "factorial using recursion.py" and "lab_assignment_1_5.py". The "factorial using recursion.py" file contains the following code:

```

18
19
20
21 #write a program of procedural factorial implementation without using functions
22 n = int(input("Enter the number to find factorial: "))
23 factorial = 1
24 for i in range(2, n + 1):
25     factorial *= i
26 print("Factorial of", n, "is", factorial)

```

The terminal at the bottom shows the execution of the program:

```

Enter the number to find factorial: 6
Factorial of 6 is 720
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"
AI Assisted Coding/factorial using recursion.py"
Enter the number to find factorial: 0
Factorial of 0 is 1
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"
Enter the number to find factorial: 1
Factorial of 1 is 1
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"

```

At the bottom right of the terminal, there are status indicators: Line 26, Column 42, Spaces: 4, UTF-8, CRLF, Python 3.11.4, Go Live, and Prettier.

Task 2: AI Code Optimization:-

AI Code Optimization & Cleanup (Improving Efficiency)

❖ Scenario

Your team lead asks you to review AI-generated code before committing it to a shared repository.

❖ Task Description

Analyze the code generated in Task 1 and use Copilot again to:

- Reduce unnecessary variables
- Improve loop clarity
- Enhance readability and efficiency

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

❖ Expected Deliverables

➤ Original AI-generated code

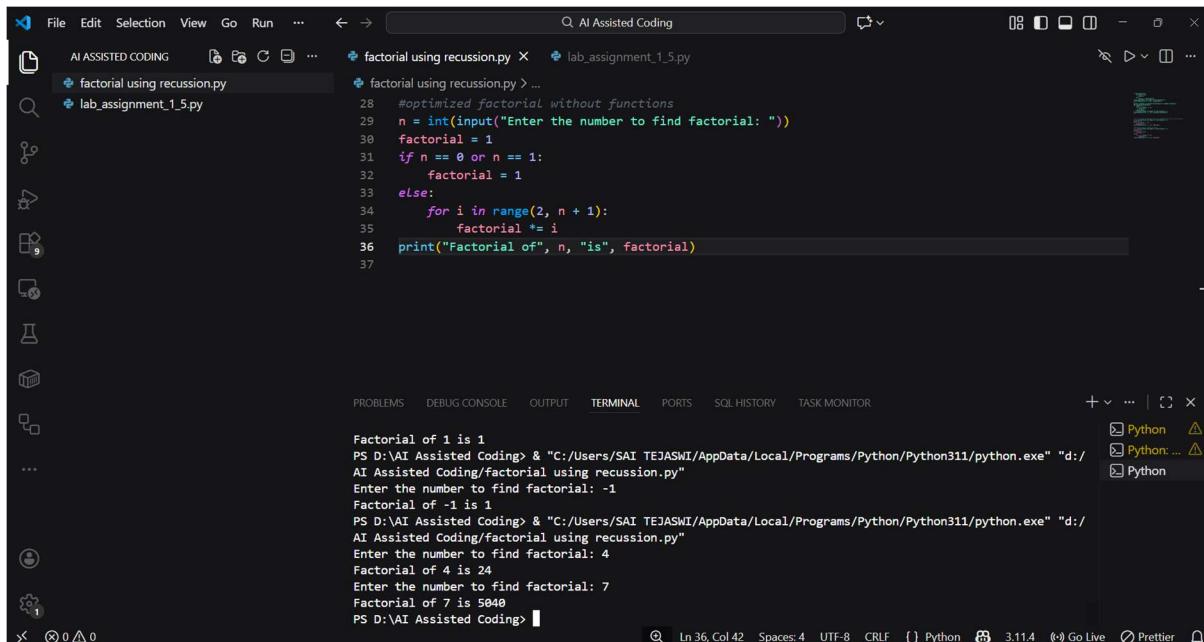
➤ Optimized version of the same code

➤ Side-by-side comparison

➤ Written explanation:

■ What was improved?

■ Why the new version is better (readability, performance, maintainability.)



```
#optimized factorial without functions
n = int(input("Enter the number to find factorial: "))
factorial = 1
if n == 0 or n == 1:
    factorial = 1
else:
    for i in range(2, n + 1):
        factorial *= i
print("Factorial of", n, "is", factorial)
```

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

❖ Scenario

The same logic now needs to be reused in multiple scripts.

❖ 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

❖ Constraints

➤ Use meaningful function and variable names

➤ Include inline comments (preferably suggested by Copilot)

❖ Expected Deliverables

➤ AI-assisted function-based program

➤ Screenshots showing:

o Prompt evolution

o Copilot-generated function logic

➤ Sample inputs/outputs

➤ Short note:

o How modularity improves reusability.

```
factorial using recursion.py X lab_assignment_1_5.py
factorial using recursion.py > ...
9  #write a program to calculate the factorial of a number iteratively
10 def factorial_iterative(n):
11     result = 1
12     for i in range(2, n + 1):
13         result *= i
14     return result
15 n = int(input("Enter the number to find factorial: "))
16 print("Factorial of", n, "is", factorial_iterative(n))

PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"
Enter the number to find factorial:5
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"
Enter the number to find factorial:5
Factorial of 5 is 120
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"
Enter the number to find factorial: 5
Factorial of 5 is 120
PS D:\AI Assisted Coding>
```

Task 4: Comparative Analysis:-

Comparative Analysis – Procedural vs Modular AI Code (With vs

Without Functions)

❖ Scenario

As part of a code review meeting, you are asked to justify design choices.

❖ **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**

❖ **Expected Deliverables**

Choose one:

- **A comparison table**

OR

- **A short technical report (300–400 words).**

Criteria	Procedural (Task 1 & 2)	Modular (Task 3)
Logic Clarity	Linear and straightforward for very small tasks but becomes "spaghetti code" as complexity grows.	High clarity; the mathematical logic is isolated from the input/output logic.
Reusability	None. To use the logic elsewhere, the code must be manually copied and pasted.	High. The function can be imported into other Python files or called multiple times in one script.
Debugging Ease	Difficult. Errors in logic are mixed with errors in user input handling.	Simple. You can test the function with specific values (Unit Testing) to ensure the math is correct.
Project Suitability	Suitable only for small, one-off scripts or prototypes.	Essential for enterprise-level, large-scale software development.
AI Dependency Risk	High. AI might generate redundant variables or inefficient loops in long scripts.	Low. AI is highly specialized and accurate when asked to

Criteria	Procedural (Task 1 & 2)	Modular (Task 3)
		write specific, single-purpose functions.

Task 5: Iterative vs Recursive Thinking:-

: AI-Generated Iterative vs Recursive Thinking

❖ **Scenario**

Your mentor wants to test how well AI understands different computational paradigms.

❖ **Task Description**

Prompt Copilot to generate:

An iterative version of the logic

A recursive version of the same logic

❖ **Constraints**

Both implementations must produce identical outputs

Students must not manually write the code first

❖ **Expected Deliverables**

Two AI-generated implementations

Execution flow explanation (in your own words)

Comparison covering:

➤ **Readability**

➤ **Stack usage**

➤ **Performance implications**

➤ **When recursion is not recommended.**

The screenshot shows a code editor interface with a dark theme. In the top left, there's a navigation bar with File, Edit, Selection, View, Go, Run, etc. Below it is a search bar. The main area has two tabs: "factorial using recursion.py" and "lab_assignment_1_5.py". The "factorial using recursion.py" tab contains the following Python code:

```
1 def factorial(n):
2     if n==0 or n==1:
3         return 1
4     else:
5         return n * factorial(n-1)
6 n=int(input("Enter the number to find factorial:"))
7 print("Factorial of", n, "is", factorial(n))
```

Below the code editor is a toolbar with various icons. Underneath the toolbar are tabs for PROBLEMS, DEBUG CONSOLE, OUTPUT, TERMINAL, PORTS, SQL HISTORY, and TASK MONITOR. The TERMINAL tab is active, showing command-line history:

```
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"
Enter the number to find factorial:5
Factorial of 5 is 120
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"
Enter the number to find factorial:5
PS D:\AI Assisted Coding> & "C:/Users/SAI TEJASWI/AppData/Local/Programs/Python/Python311/python.exe" "d:/AI Assisted Coding/factorial using recursion.py"
Enter the number to find factorial:5
Factorial of 5 is 120
PS D:\AI Assisted Coding>
```

At the bottom right, there are status indicators: Line 7, Col 45, Spaces:4, UTF-8, CRLF, Python 3.11.4, Go Live, Prettier, and a lock icon.