

Assignment - 1

Name: B.Sai Tejaswi

Roll Number: 2303A51184

Batch - 03

AI Assisted Coding

07-01-2026

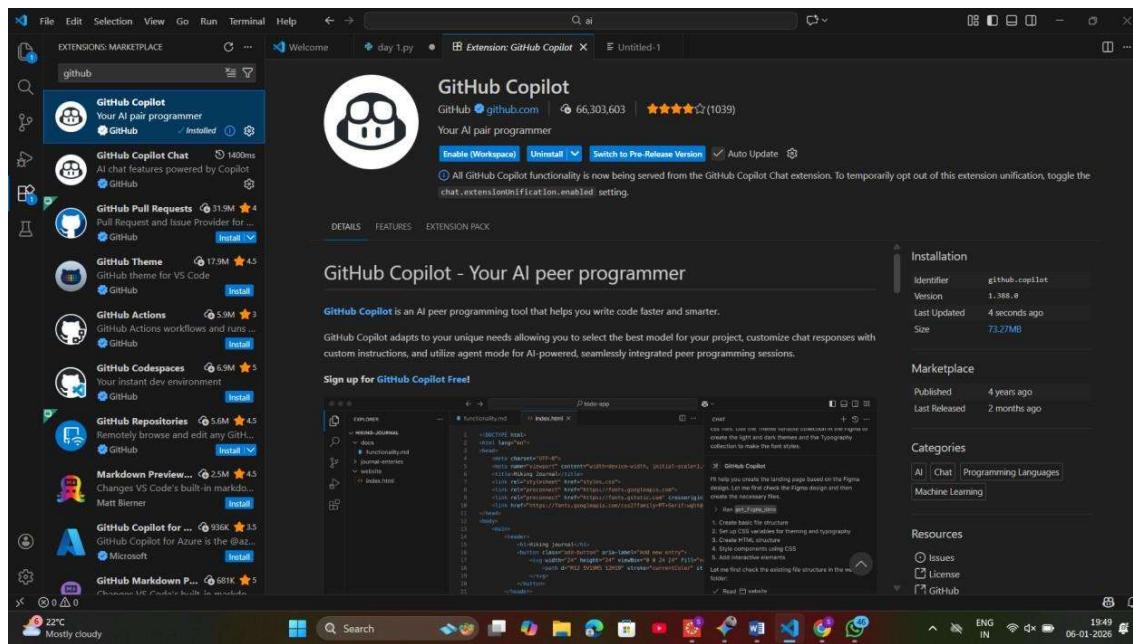
Task 0: Environment Setup:-

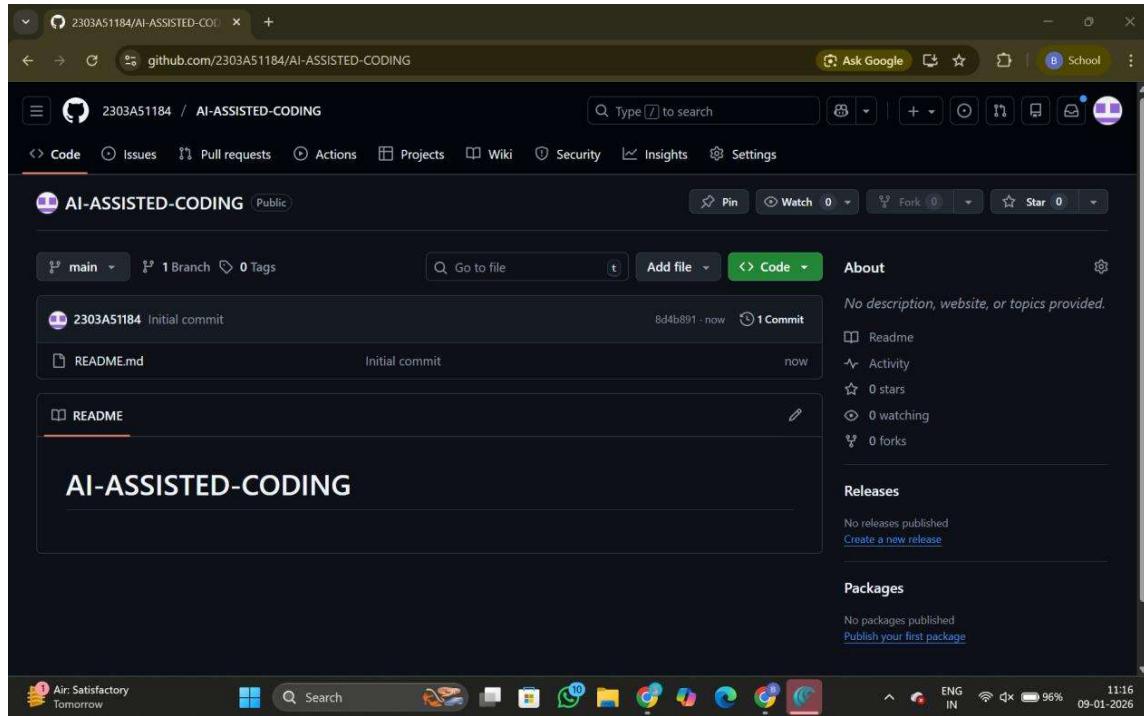
Task 0

- 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 Microsoft Visual Studio Code interface. In the center, there is a code editor with the following Python code:

```

1 # Task 1: Procedural Factorial Implementation
2 num = int(input("Enter a number: "))
3 factorial = 1
4
5 if num < 0:
6     print("Factorial does not exist for negative numbers")
7 elif num == 0:
8     print("The factorial of 0 is 1")
9 else:
10    temp = num
11    while temp > 0:
12        factorial *= temp
13        temp -= 1
14    print(f"The factorial of {num} is {factorial}")

```

Below the code editor is a terminal window showing the output of running the code:

```

Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>
PS C:\Users\hp\OneDrive\Desktop\HPC>
PS C:\Users\hp\OneDrive\Desktop\HPC> cd 'C:\Users\hp\OneDrive\Desktop\HPC' & 'C:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'C:\Users\hp\vscode\extensions\ms-python.python\debugpy-2023.10.0-win32-x64\foundled\l11bs\Debug\Python\Launcher' '62650' & ''C:\Users\hp\OneDrive\Desktop\ai\task1.py'
PS C:\Users\hp\OneDrive\Desktop\HPC>

```

To the right of the code editor is a "CHAT" panel with an AI-generated message:

python code

Asked for clarification on Python code request

I'd be happy to help! However, your request is unclear. Could you clarify what you'd like me to do?

For example:

- View existing code: Should I look at lab1.py or task1.py?
- Write new code: What should the code do?
- Fix/debug code: What problem needs fixing?
- Run code: Which file should I execute?
- Explain code: Which code would you like me to explain?

Please let me know what you're trying to accomplish!

This screenshot shows the Microsoft Visual Studio Code interface with a terminal window open. The terminal output is as follows:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... | X
PS C:\Users\hp\OneDrive\Desktop\HPC> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/hp/OneDrive/Desktop/ai/task1.py
Enter a number: 5
Factorial is: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>

```

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).

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. In the center, there are two tabs: 'lab1.py' and 'task1.py'. The 'task1.py' tab contains the following Python code:

```

C:\> Users > hp > OneDrive > Desktop > ai > task1.py > ...
1 # Task 2: optimized Factorial
2 num = int(input("Enter a number: "))
3 factorial = 1
4
5 for i in range(1, num + 1):
6     factorial *= i
7
8 print(f"Factorial: {factorial}")

```

Below the code editor is a terminal window showing the output of running the script:

```

PS C:\Users\hp\OneDrive\Desktop\ai> python task1.py
Enter a number: 5
Factorial: 120
PS C:\Users\hp\OneDrive\Desktop\ai>

```

To the right of the code editor is a 'CHAT' sidebar titled 'CLARIFICATION ON PYTHON CODE REUSE'. It contains a message from the AI asking for clarification on Python code requests, followed by a list of options for the user to choose from:

- View existing code: Should I look at lab1.py or task1.py?
- Write new code: What should the code do?
- Fix/debug code: What problem needs fixing?
- Run code: Which file should I execute?
- Explain code: Which code would you like me to explain?

At the bottom of the sidebar, there is a text input field with the placeholder 'Describe what to build next.' and a button labeled 'Ask'.

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.

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The Explorer sidebar on the left shows files: 'lab1.py' and 'task1.py'. The 'task1.py' editor tab contains the following Python code:

```
C:\> Users > hp > OneDrive > Desktop > ai > task1.py
1 def calculate_factorial(n):
2     """Calculates the factorial of a given number iteratively."""
3     result = 1
4     for i in range(1, n + 1):
5         result *= i
6     return result
7
8 if __name__ == "__main__":
9     user_input = int(input("Enter number: "))
10    print(f"Result: {calculate_factorial(user_input)}")
```

The Terminal tab at the bottom shows the command-line output:

```
PS C:\Users\hp\OneDrive\Desktop\ai>
PS C:\Users\hp\OneDrive\Desktop\ai>
PS C:\Users\hp\OneDrive\Desktop\ai> cd "c:\Users\hp\OneDrive\Desktop\ai" & "c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.11.exe" "c:\Users\hp\.vscode\extensions\ms-python.python.debug-2025.18.0-win32-x64\bundled\libs\debug\launcher" "65d97" -- "c:\Users\hp\OneDrive\Desktop\ai\task1.py"
PS C:\Users\hp\OneDrive\Desktop\ai> Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop\ai>
```

A GitHub Copilot chat window is open on the right, titled 'CLARIFICATION ON PYTHON CODE REQUI...'. It shows a message from the AI asking for clarification on the Python code request. Below the message, there's a list of options for interacting with the code, such as 'View existing code', 'Write new code', and 'Fix/debug code'. At the bottom of the chat window, there's a text input field with the placeholder 'Describe what to build next'.

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. |

| Criteria | Procedural (Task 1 & 2) | Modular (Task 3) |
|----------------------------|---|--|
| 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 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 the Microsoft Visual Studio Code interface. In the Explorer sidebar, there is a folder named 'HPC' containing a file 'labt.py'. The 'CODE' tab displays two Python functions:

```
C:\> Users > hp > OneDrive > Desktop > ai > labt.py > factorial_iterative
1 def factorial_iterative(n):
2     res = 1
3     for i in range(2, n + 1):
4         res *= i
5     return res
6
7 def factorial_recursive(n):
8     if n == 0 or n == 1:
9         return 1
10    return n * factorial_recursive(n - 1)
```

The 'TERMINAL' tab shows a terminal session where the user has run a script. The output is:

```
Enter number: 5
Result: 120
```

The status bar at the bottom right indicates the system is running at 15:40, the date is 07-01-2026, and the language is set to English (IN).

Assignment - 1.5

Name: B.Sai Tejaswi

Roll Number: 2303A51184

Batch - 03

AI Assisted Coding

09-01-2026

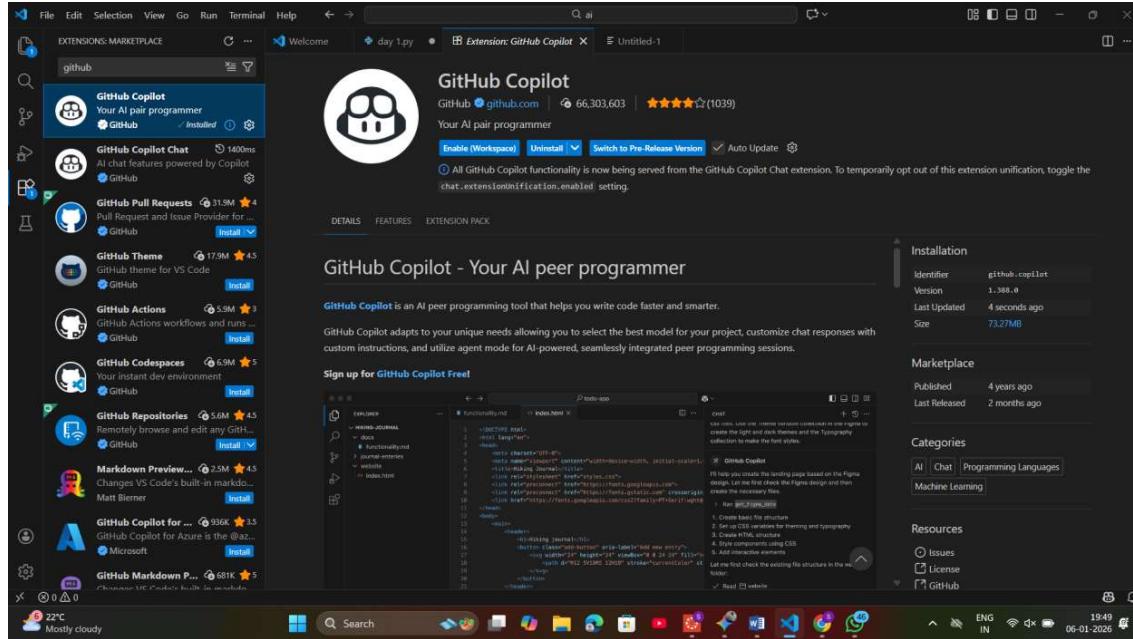
Task 0: Environment Setup:-

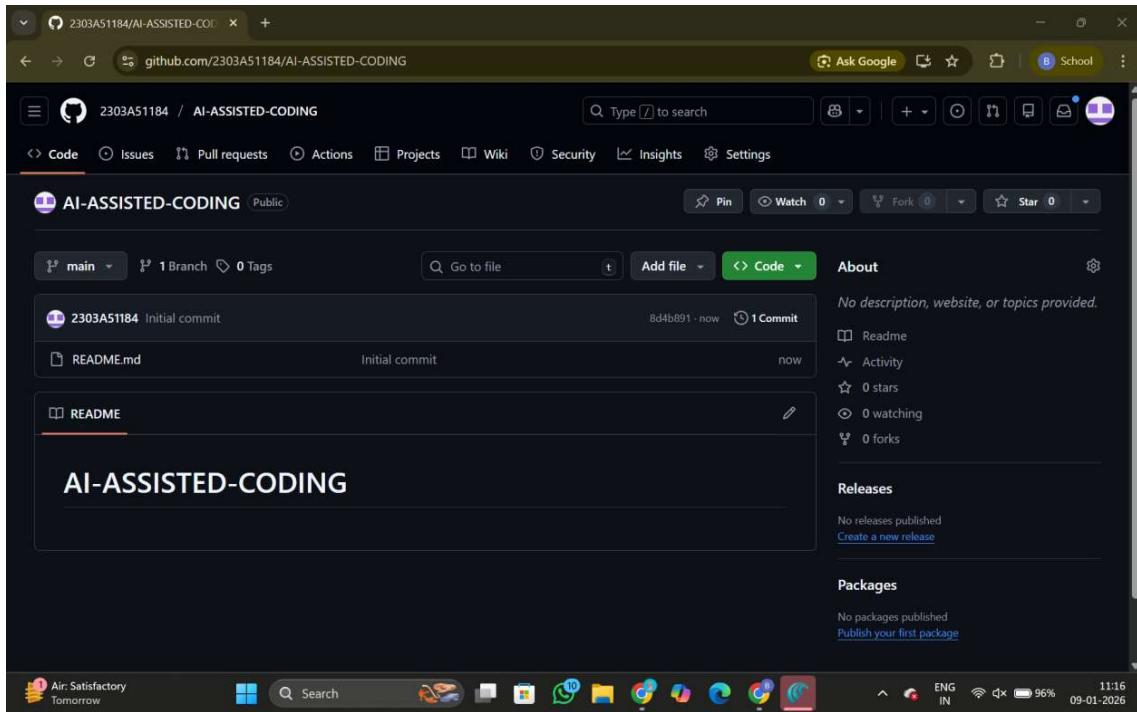
Task 0

- 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 Visual Studio Code interface. In the top editor pane, there is a file named 'task1.py' containing the following Python code:

```
C:\> Users > hp > OneDrive > Desktop > ai > task1.py > ...
1 # Accepting user input
2 user_input = input("Enter a string to reverse: ")
3
4 # Initializing an empty string to store the result
5 reversed_string = ""
6
7 # Logic to reverse the string using a loop
8 for i in range(len(user_input) - 1, -1, -1):
9     reversed_string += user_input[i]
10
11 # Printing the result
12 print("Original String:", user_input)
13 print("Reversed String:", reversed_string)
```

Below the editor is a terminal window showing the execution of the script and its output:

```
PS C:\Users\hp> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/hp/OneDrive/Desktop/ai/task1.py
Enter a string to reverse: 2 3 4 5 6
Original String: 2 3 4 5 6
Reversed String: 6 5 4 3 2
PS C:\Users\hp>
```

The bottom part of the screenshot shows the Windows taskbar with various pinned 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**

```
task1.py
C:\Users\hp\OneDrive\Desktop\ai> task1.py
1 user_input = input("Enter a string: ")
2
3 # Using Python's slicing for maximum efficiency
4 reversed_string = user_input[::-1]
5
6 print(f"Reversed: {reversed_string}")

PS C:\Users\hp\OneDrive\Desktop\ai> & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\lib
s\debugpy\launcher' '80075' '--' 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
Enter a string: 48 50 60 70
Reversed: 07 06 05 04
PS C:\Users\hp\OneDrive\Desktop\ai>
```

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 code editor with a Python file named `task1.py` and a terminal window below it.

`task1.py` content:

```
C:\> Users > hp > OneDrive > Desktop > ai > task1.py > ...
1 def reverse_string_functional(text):
2     """
3         Reverses the input string and returns it.
4     """
5     reversed_text = ""
6     for char in text:
7         reversed_text = char + reversed_text
8     return reversed_text
9
10 # Testing the function
11 input_str = input("Enter text: ")
12 result = reverse_string_functional(input_str)
13 print(f"Result: {result}")
```

Terminal output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PUBLISH
5 4 3 2 1
5 4 3 2 1
PS C:\Users\hp\OneDrive\Desktop\ai> ^C
PS C:\Users\hp\OneDrive\Desktop\ai>
PS C:\Users\hp\OneDrive\Desktop\ai> cd 'c:\Users\hp\OneDrive\Desktop\ai'; & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundles\libs\debugpy\launcher' '54371' '...'\c:\Users\hp\OneDrive\Desktop\ai\task1.py
Enter text: Teju
Result: ujet
PS C:\Users\hp\OneDrive\Desktop\ai>
```

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 the VS Code interface with the file `task1.py` open. The code defines two functions: `reverse_iterative` and `reverse_slicing`, both of which return the reverse of a given string. A test input is provided to demonstrate the functionality.

```
C:\> Users > hp > OneDrive > Desktop > ai > task1.py > ...
1  v def reverse_iterative(input_string):
2      reversed_str = ""
3  v     for char in input_string:
4      |         reversed_str = char + reversed_str
5  v     return reversed_str
6
7  v def reverse_slicing(input_string):
8      |     return input_string[::-1]
9
10 test_input = input("Enter a string: ")
11
12 print(reverse_iterative(test_input))
13 print(reverse_slicing(test_input))
```

The screenshot shows the VS Code interface with the terminal tab active. The terminal displays the execution of `task1.py`. The user enters the string "54321" and the program outputs the reversed strings from both functions.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ v ... | x
s\debugpy\launcher' '5836' '... 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
PS C:\Users\hp\OneDrive\Desktop\ai>
PS C:\Users\hp\OneDrive\Desktop\ai> c:: cd 'c:\Users\hp\OneDrive\Desktop\ai'; & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\.vscode\extensions\ms-python.on.debugger-2025.18.0-win32-364\bundled\libs\debugpy\launcher' '57517' '... 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
Enter a string: 54321
5 4 3 2 1
PS C:\Users\hp\OneDrive\Desktop\ai>
```