

Assignment - 1

Name: A.Sai Varshith

Roll Number: 2303A51831

Batch - 04

AI Assisted Coding

07-01-2026

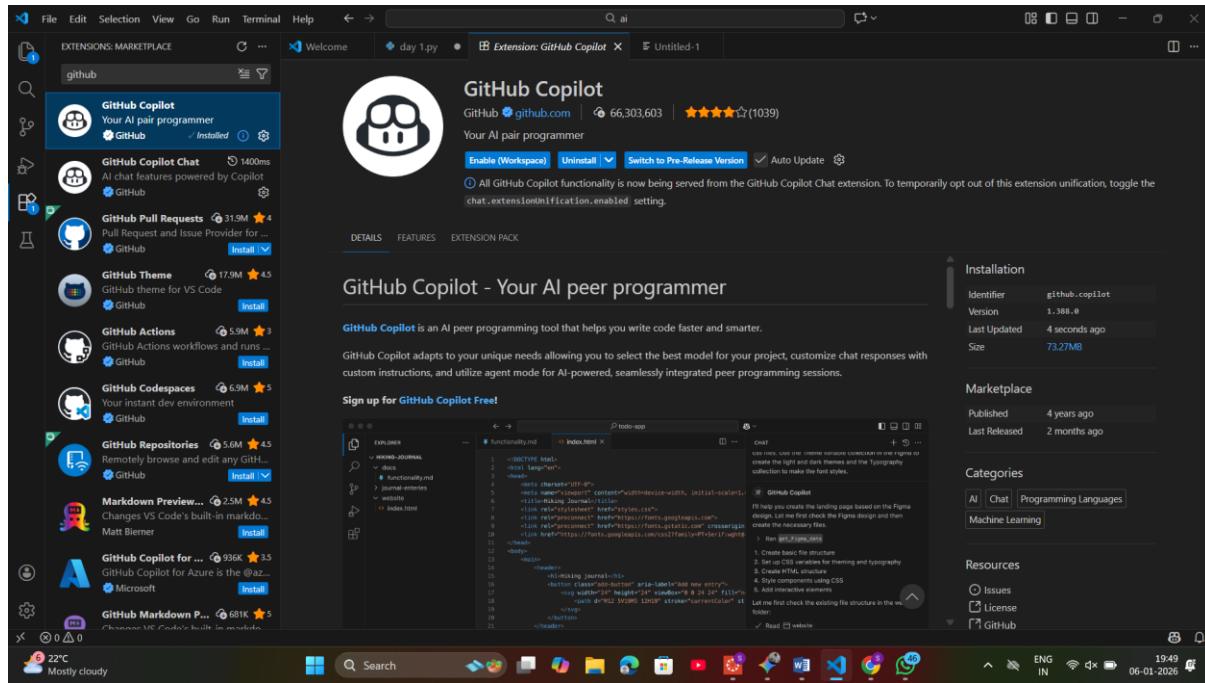
Task 0: Environment Setup:-

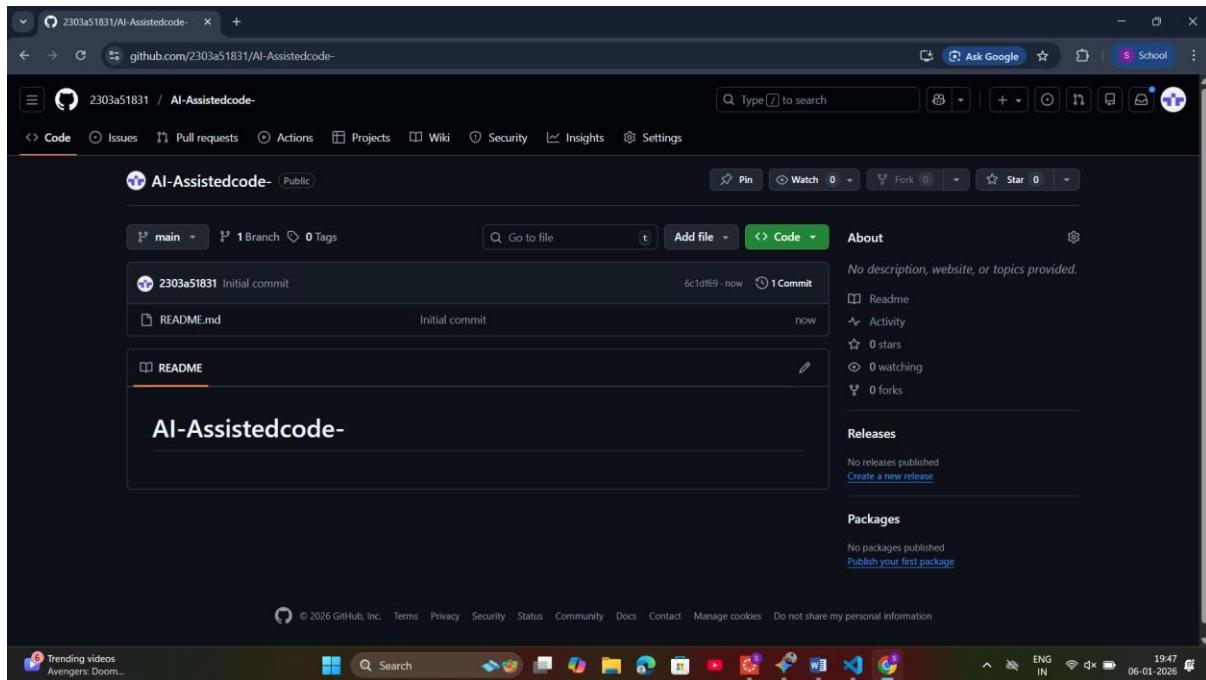
Task 0

- 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. On the left is the Explorer sidebar with a project named 'HPC' containing files 'lab1.py' and 'task1.py'. The main editor area displays the following Python code:

```

C:\> Users > hp > Desktop > ai > task1.py > ...
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}")

```

To the right of the editor is a 'CHAT' sidebar with a message from 'Copilot' asking for clarification on Python code requests. Below the editor are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab shows the command to run the script and its execution:

```

Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop\HPC> & c:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/hp/Desktop/ai/task1.py
PS C:\Users\hp\OneDrive\Desktop\HPC>

```

The status bar at the bottom indicates the file is indexed and shows system information like weather (28°C, sunny), battery level (1549), and date/time (07-01-2026).

This screenshot shows the terminal window in Visual Studio Code. It displays the command to run the script and the resulting output:

```

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

```

The status bar at the bottom shows the file is indexed and provides system details.

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. The left sidebar has 'EXPLORER' open, showing a folder named 'HPC' containing 'lab1.py'. The main editor area has a file named 'task1.py' with the following code:

```

1 # Task 2: Optimized Factorial
2 num = int(input("Enter a number: "))
3 factorial = 1
4 for i in range(1, num + 1):
5     factorial *= i
6 print("Factorial: ", factorial)

```

The bottom terminal window shows the output of running the script:

```

PS C:\Users\hp\OneDrive\Desktop\HPC> & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\vscode\extensions\ms-python.python.debug-2025.18.0-win32-x64\bundle\libs\debug\launcher' '56820' '<-' 'C:\Users\hp\OneDrive\Desktop\HPC\task1.py'
Enter a number: 5
Factorial: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>

```

The right side of the interface features the 'CHAT' panel, which is currently active. It shows a message from 'python code' asking for clarification on Python code requests. Below this, there's a list of options for interacting with the code, such as 'View existing code', 'Write new code', and 'Fix/debug code'. A message at the bottom asks for the user's goal: 'Please let me know what you're trying to accomplish!'. The status bar at the bottom right shows the date and time as '07-01-2026 15:35'.

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:

- Prompt evolution
- Copilot-generated function logic

➤ Sample inputs/outputs

➤ Short note:

○ How modularity improves reusability.

The screenshot shows the Microsoft Visual Studio Code (VS Code) interface. The left sidebar (EXPLORER) shows a folder named 'HPC' containing a file 'lab1.py'. The main editor area displays a Python script named 'task1.py' with the following code:

```
C:\Users\hp\OneDrive\Desktop>cd > 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 at the bottom shows the command-line output of running the script:

```
PS C:\Users\hp\OneDrive\Desktop> cd > task1.py > ...
PS C:\Users\hp\OneDrive\Desktop> PS C:\Users\hp\OneDrive\Desktop> c; 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-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65497' '--' 'c:\Users\hp\OneDrive\Desktop\HPC\task1.py'
Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop>
```

The status bar at the bottom right indicates the system is at 28°C, sunny, and the date is 07-01-2026.

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 (VS Code) interface. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search field. The Explorer sidebar on the left shows a project named "HPC" containing a file "lab1.py". The main editor area displays two Python files: "factorial_iterative.py" and "factorial_recursive.py".

```
C:\Users\hp> OneDrive > Desktop > ai > task1.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 below shows the execution of the code:

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

The status bar at the bottom indicates indexing completed, the current file is "task1.py", and the system status is 28°C, Sunny.

A Chat window on the right side of the interface is open, showing a conversation about Python code requests. It includes a list of common code-related questions and a message asking for clarification on the user's request.

Assignment – 1.5

Name: A.Sai Varshith

Roll Number: 2303A51831

Batch - 04

AI Assisted Coding

09-01-2026

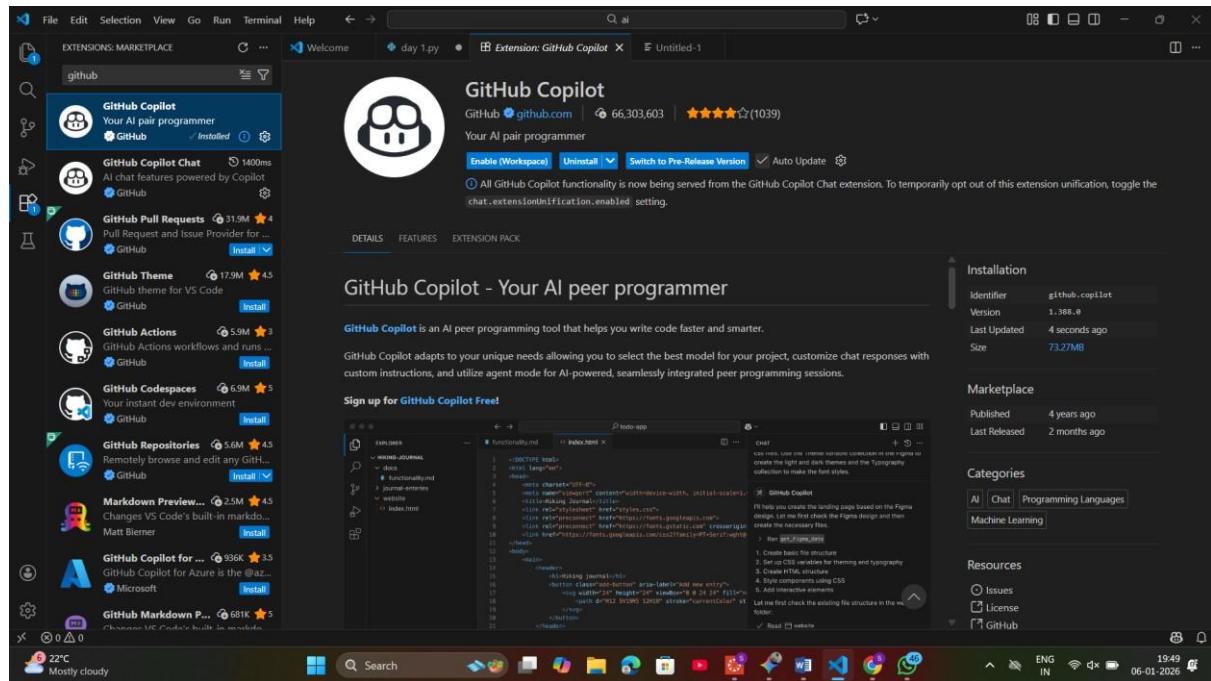
Task 0: Environment Setup:-

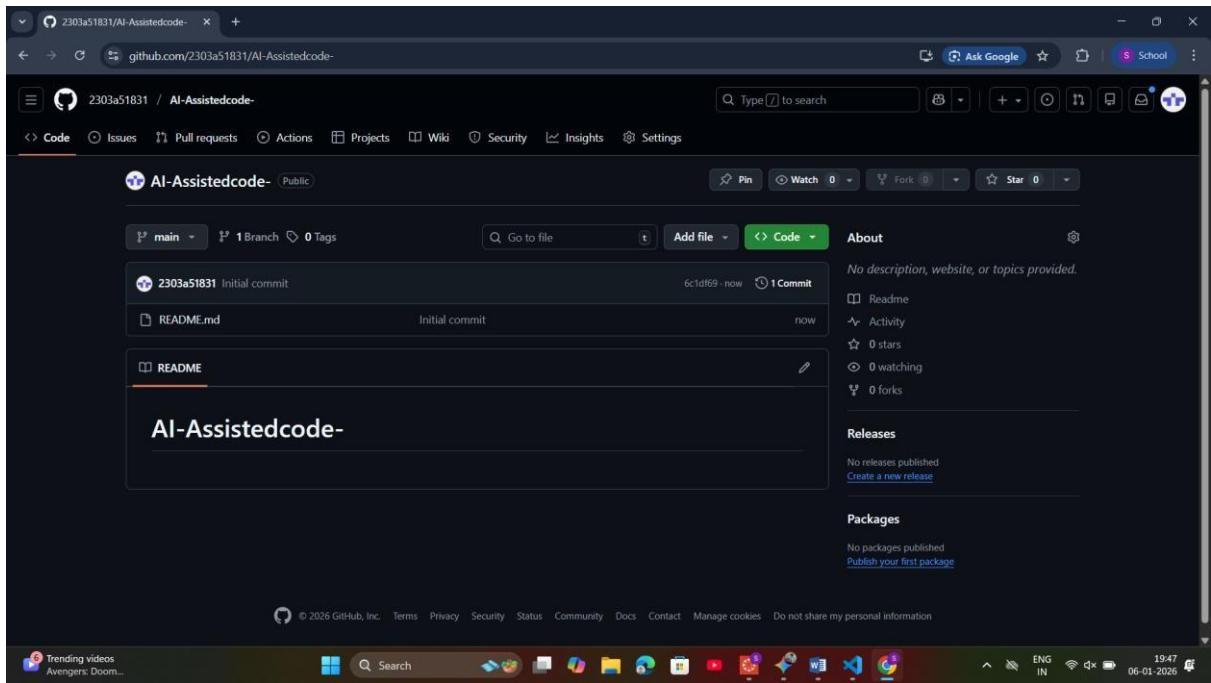
Task 0

- **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

```

task1.py
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)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + × × ... | ×
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> [REDACTED]
[REDACTED]
In 13, Col 43 Spaces: 4 UTF-8 CRLF {} Python 🐍 3.13.9 (Microsoft Store) 🌐
ENG IN 11:37 09-01-2026 📅
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
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> [REDACTED]
[REDACTED]

```

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 a dark-themed VS Code interface. The left sidebar has icons for file operations like Open, Save, and Close. The main editor area contains the following Python code:

```
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}")
```

To the right of the editor is a terminal window with the following output:

```
PS C:\Users\hp\OneDrive\Desktop\ai> & 'c:\Users\hp\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\hp\vscode\extensions\ms-python.python-2025.18.0-win32-x64\bundled\lib
s\debug\Launcher' '50075' '--' 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
Enter a string: 40 50 60 70
Reversed: 07 60 50 40
PS C:\Users\hp\OneDrive\Desktop\ai>
```

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

❖ Scenario

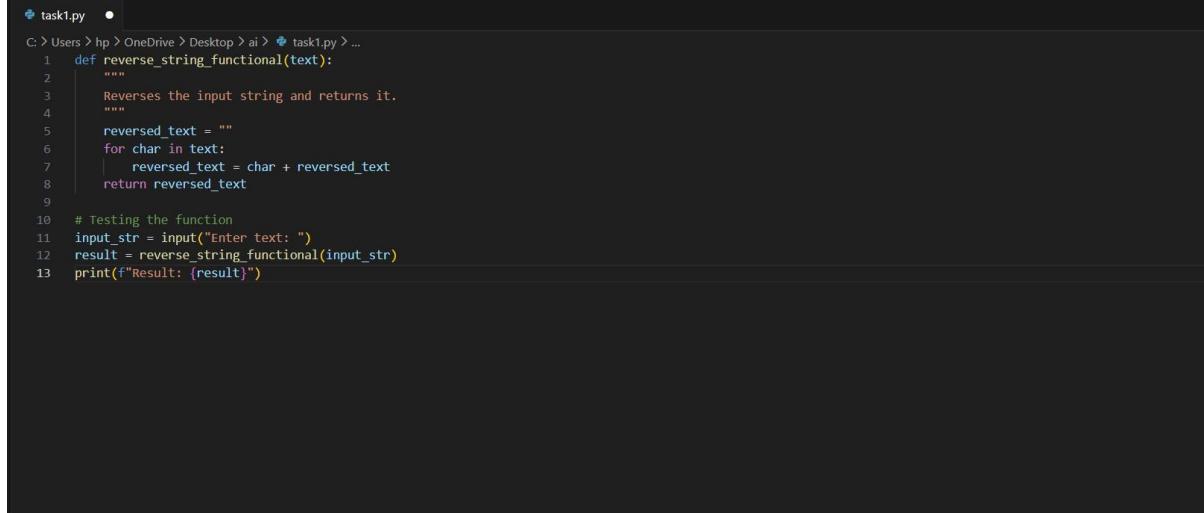
The string reversal logic is needed in multiple parts of an application.

❖ 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

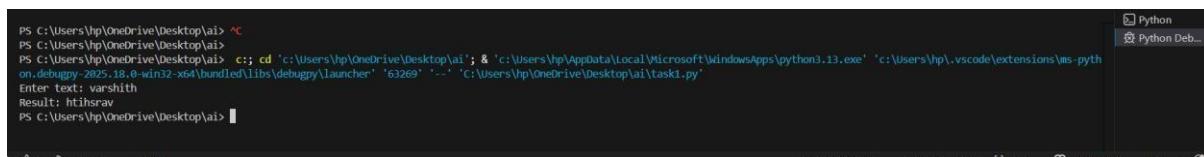


```

task1.py

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}")

```

```

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.python.debug-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63269' '--' 'C:\Users\hp\OneDrive\Desktop\ai\task1.py'
Enter text: varshith
Result: htihsrav
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; large ones.	Very high; logic is isolated and named.
Reusability	Must copy-paste code again.	Can be called anywhere in the app.
Debugging	Harder to isolate what 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

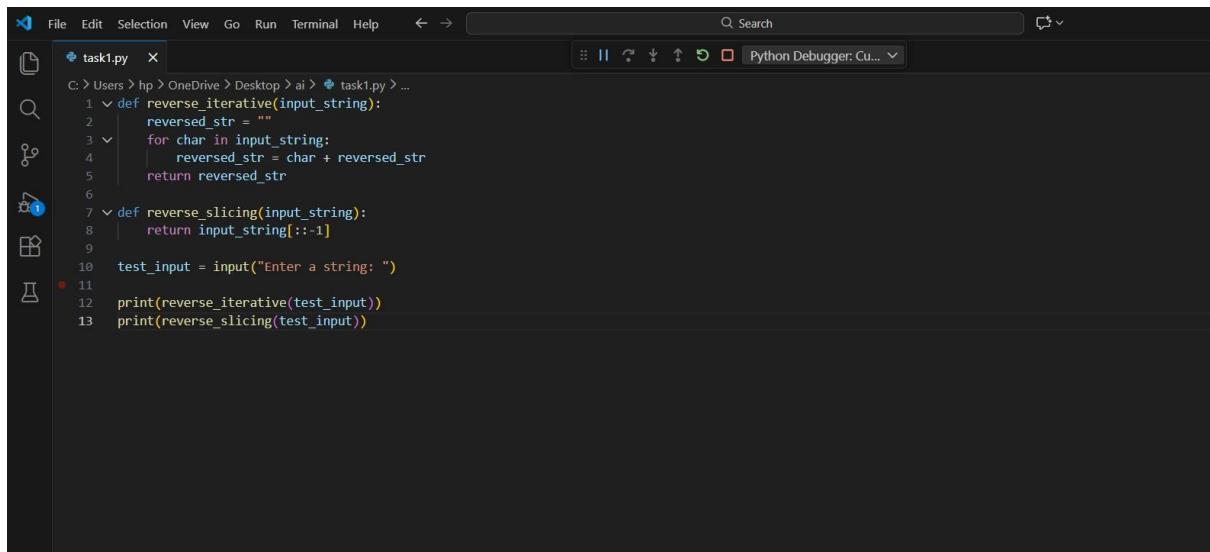
❖ **Expected Output**

- Two correct implementations

➤ Comparison discussing:

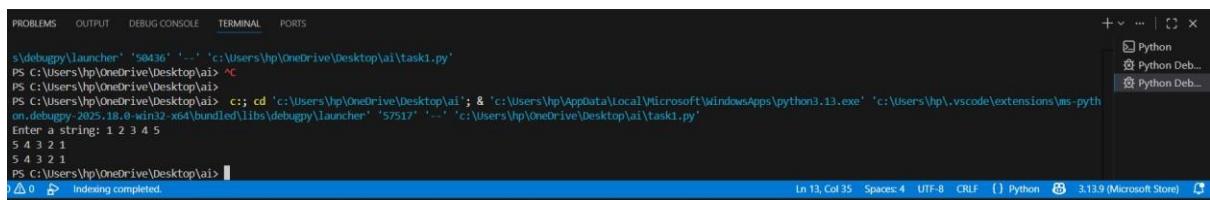
- Execution flow
- Time complexity
- Performance for large inputs

- When each approach is appropriate.



A screenshot of the Visual Studio Code interface. The left sidebar shows icons for file, search, and other workspace functions. The main editor area displays a Python script named 'task1.py'. The code contains two functions: 'reverse_iterative' which uses a for loop to build a reversed string character by character, and 'reverse_slicing' which uses Python's slicing feature to reverse the string. A red dot on the left margin indicates a breakpoint at line 11. The status bar at the bottom shows the file path 'C:\Users\hp\OneDrive\Desktop\ai> task1.py > ...'.

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



A screenshot of the Visual Studio Code interface focusing on the terminal tab. The terminal window shows the command 'task1.py' being run, followed by the output of the reversed strings. The status bar at the bottom indicates the terminal has completed indexing.

```
s\debugpy\launcher` '58436' `--> 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
PS C:\Users\hp\OneDrive\Desktop\ai> ^C
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_debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57517' `--> 'c:\Users\hp\OneDrive\Desktop\ai\task1.py'
Enter a string: 1 2 3 4 5
5 4 3 2 1
5 4 3 2 1
PS C:\Users\hp\OneDrive\Desktop\ai>
```