

## Assignment – 1.1

Name: A Akshith Reddy

Roll Number: 2303A51177

Batch - 03

AI Assisted Coding

07-01-2026

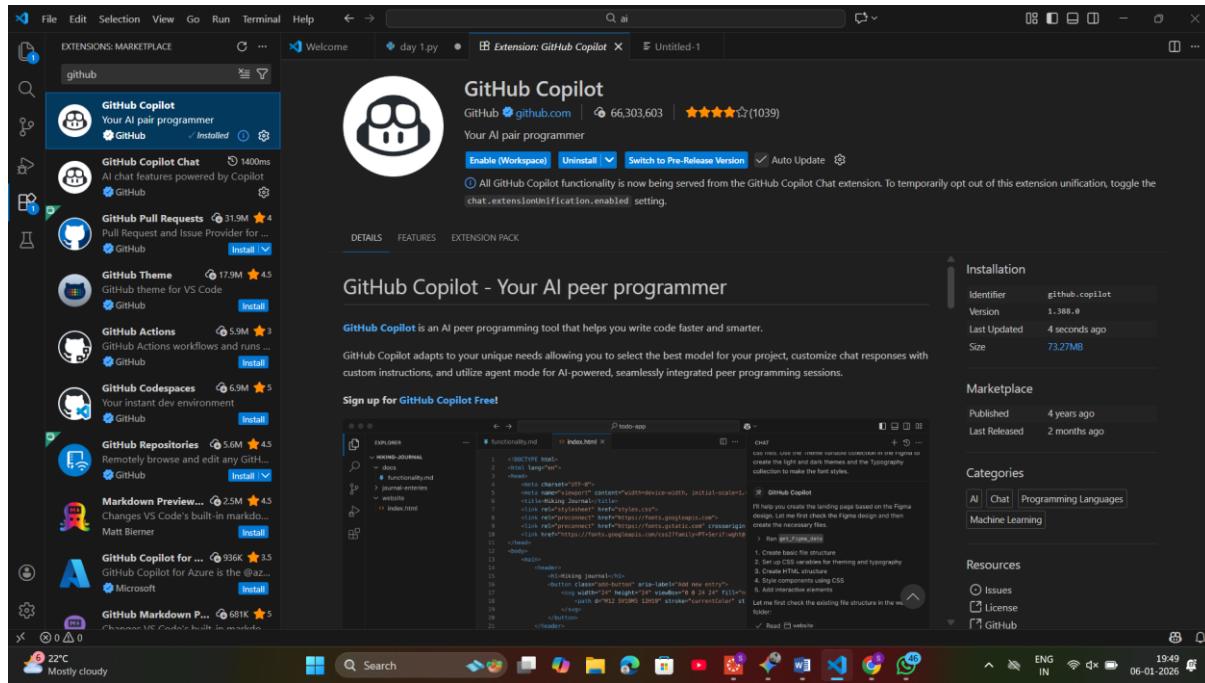
### Task 0: Environment Setup:-

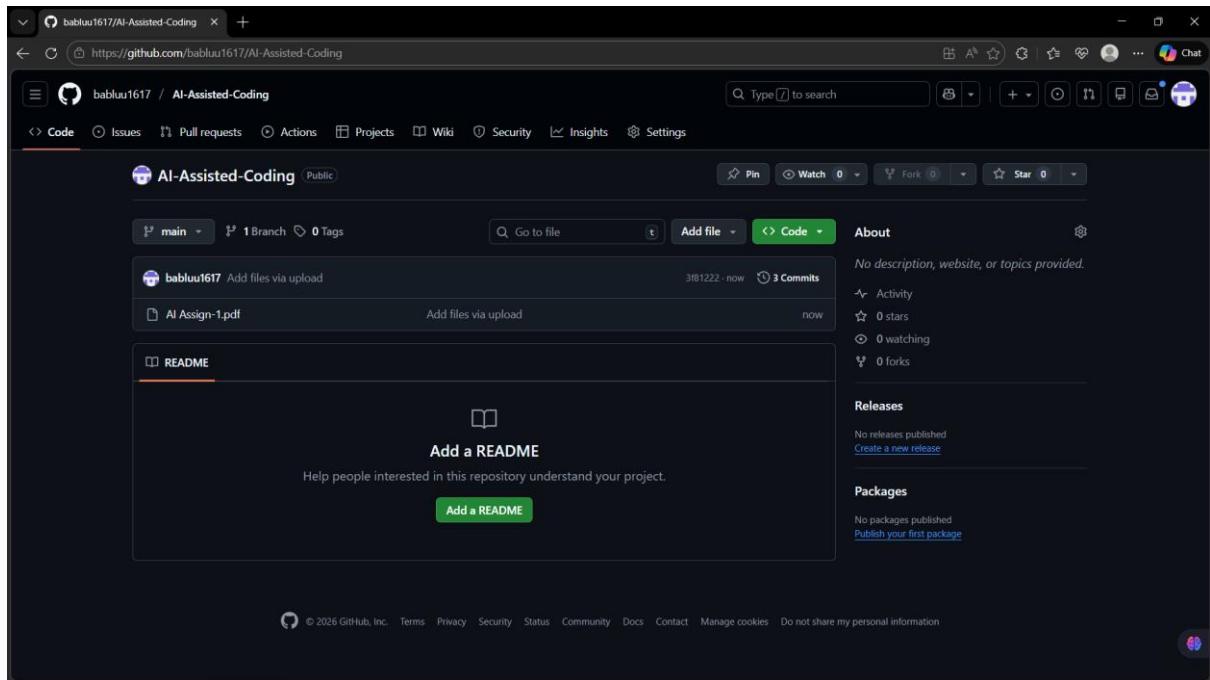
#### Task 0

- 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:

```
C:\> Users > hp > OneDrive > 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}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

To the right of the code editor, there is a sidebar titled "CHAT" which contains 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!

At the bottom of the interface, there is a terminal window showing the command-line output of the Python code execution.

The screenshot shows the Microsoft Visual Studio Code interface with the terminal tab selected. The terminal output is as follows:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + v ... 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. The Explorer sidebar on the left shows a folder named 'HPC' containing a file 'lab1.py'. The main editor area displays the following Python code:

```

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("Factorial: ", factorial)

```

The terminal at the bottom shows the output of running the code:

```

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\bundled\libs\debug\Launcher" "56820" "-c" "c:\Users\hp\OneDrive\Desktop\HPC\lab1.py"
Enter a number: 5
Factorial: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>

```

A Chat window on the right is open with the message "python code". Below it, a message from the AI says: "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:" followed by a list of options: View existing code, Write new code, Fix/debug code, Run code, Explain code.

### 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 > ai > lab1.py > ... > 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 right sidebar (CHAT) has a message from 'GitHub Copilot' asking for clarification on Python code requests. Below it, there's a list of options like 'View existing code', 'Write new code', and 'Fix/debug code'. At the bottom, there's a terminal window showing the command to run the script and its output:

```
PS C:\Users\hp\OneDrive\Desktop\HPC> ^C
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\python3.11.exe' 'C:\Users\hp\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\build\libs\debugpy\launcher' '65497' -- 'C:\Users\hp\OneDrive\Desktop\ai\task1.py'
Enter number: 5
Result: 120
PS C:\Users\hp\OneDrive\Desktop\HPC>
```

## 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).**

<b>Criteria</b>	<b>Procedural (Task 1 &amp; 2)</b>	<b>Modular (Task 3)</b>
<b>Logic Clarity</b>	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.
<b>Reusability</b>	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.
<b>Debugging Ease</b>	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)
<b>Project Suitability</b>	Suitable only for small, one-off scripts or prototypes.	Essential for enterprise-level, large-scale software development.
<b>AI Dependency Risk</b>	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)**

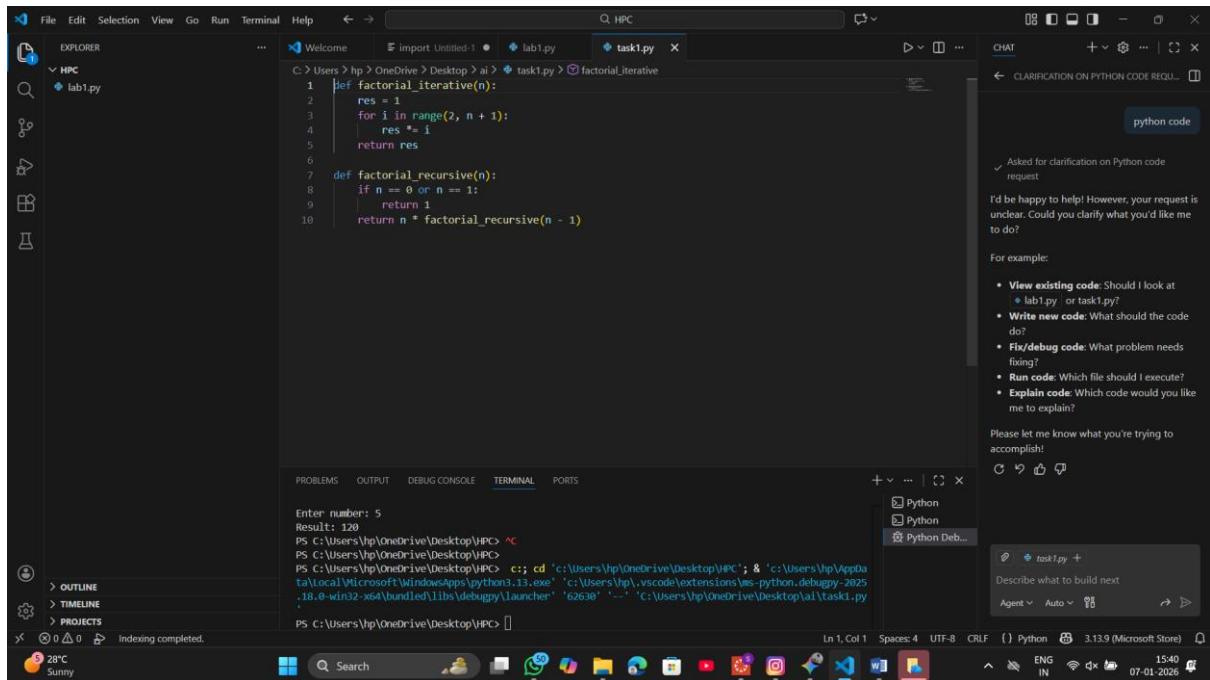
**Comparison covering:**

**➤ Readability**

**➤ Stack usage**

**➤ Performance implications**

**➤ When recursion is not recommended.**



## Assignment – 1.5

Name: A. Akshith Reddy

Roll Number: 2303A51177

Batch - 03

AI Assisted Coding

09-01-2026

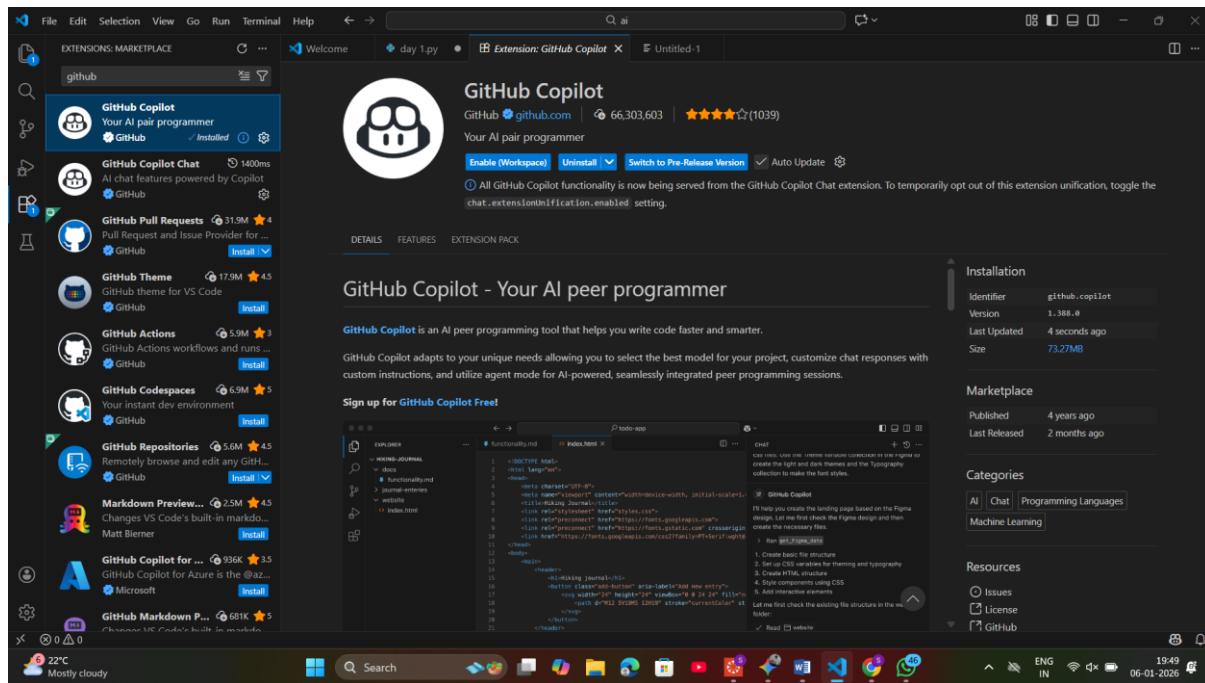
### Task 0: Environment Setup:-

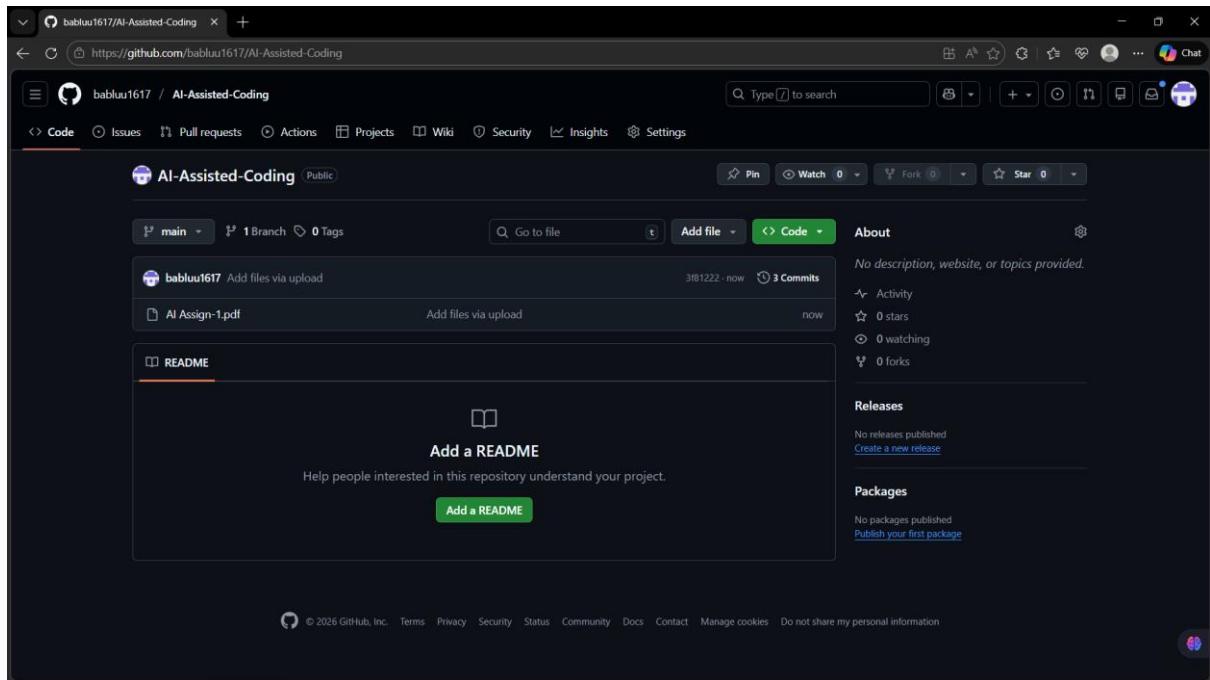
#### Task 0

- 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

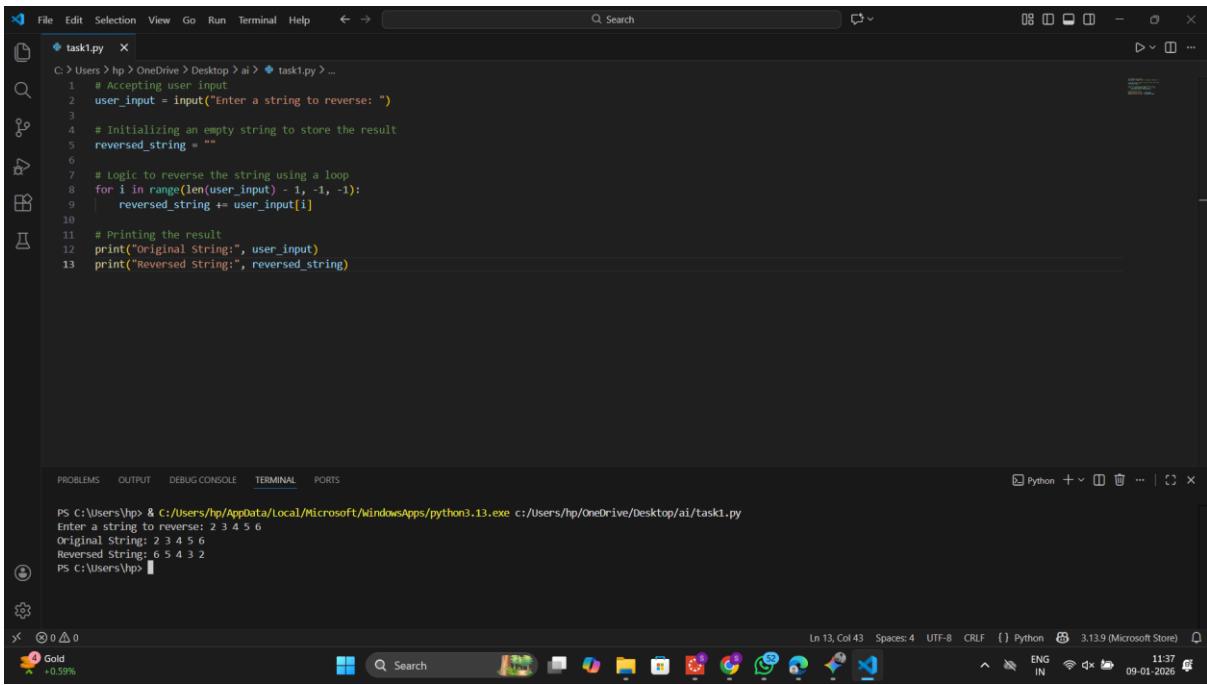
Use GitHub Copilot to generate a Python program that:

- 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. The top part displays the code file `task1.py` which contains a Python script to reverse a string. The bottom part shows the terminal window with the command `python task1.py` run, producing the expected output of reversing the input string "2 3 4 5 6".

```
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
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>
```



This screenshot shows a dark-themed terminal window with the same command and output as the one above, demonstrating the execution of `task1.py`.

```
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>
```

## 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:**

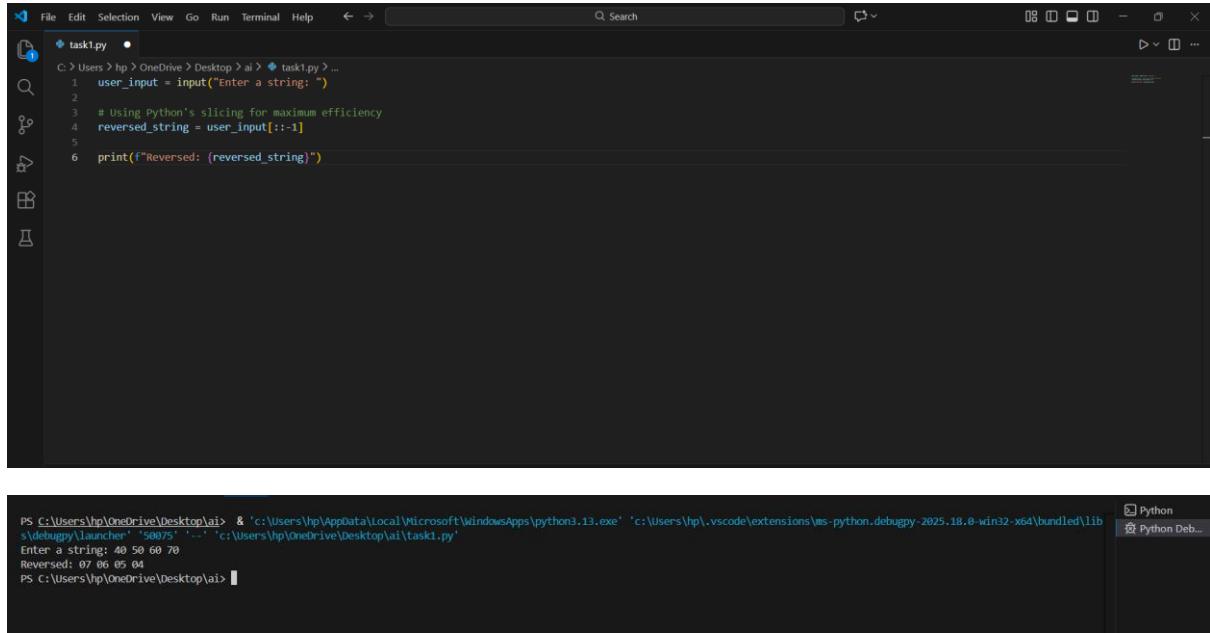
**Prompt Copilot with phrases like**

**“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 interface. On the left, the code editor displays a file named 'task1.py' with the following content:

```
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}")
```

On the right, a terminal window shows the execution of the script. The command run was:

```
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\site-packages\debugpy\launcher" "50075" ... "c:\Users\hp\OneDrive\Desktop\ai\Task1.py"
```

The output of the script is:

```
Enter a string: 40 50 60 70
Reversed: 70 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

```
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}")
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Enter text: Teju
Result: ujet
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\python3.13.exe' 'c:\Users\hp\.vscode\extensions\ms-python.on.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' "59916" ...; c:\Users\hp\OneDrive\Desktop\ai\task1.py
Enter text: Akshith
Result: htsikA
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)
<b>Code Clarity</b>	Easy for tiny scripts; messy for large ones.	Very high; logic is isolated and named.
<b>Reusability</b>	Must copy-paste code to use it again.	Can be called anywhere in the app.
<b>Debugging</b>	Harder to isolate where an error occurs.	Easy to unit test the specific function.
<b>Scalability</b>	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.**

```
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
12 print(reverse_iterative(test_input))
13 print(reverse_slicing(test_input))
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + v ... | ⌂ X
Python
Python Deb...
Python Deb...
PS 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-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> [Delta 0] Indexing completed.
```