

# AI ASSISTED CODING

**Name: O.Sai Keerthana**

**Ht.No: 2303A52397**

**Batch: 36**

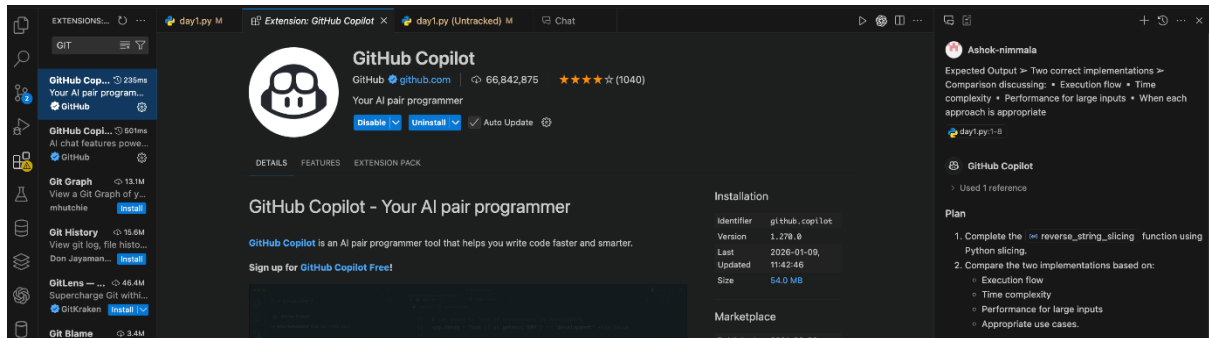
## Assignment-1.5

### 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:

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

### Scenario:

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

Application

## Task Description

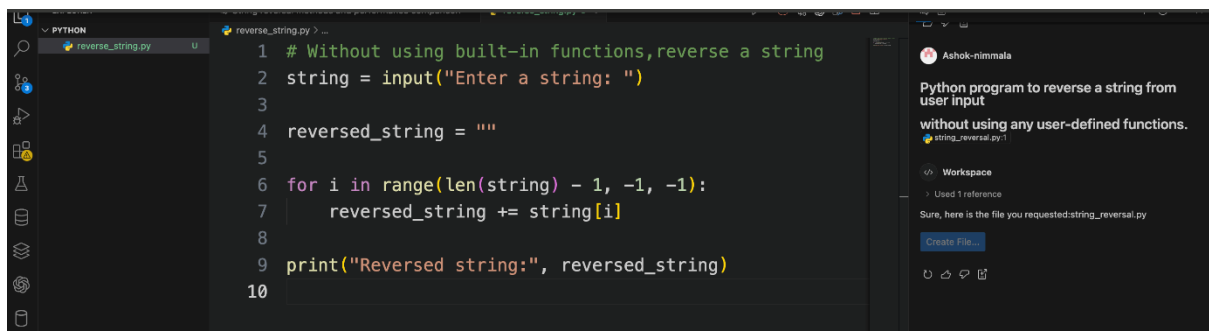
Use GitHub Copilot to generate a Python program that:

- 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

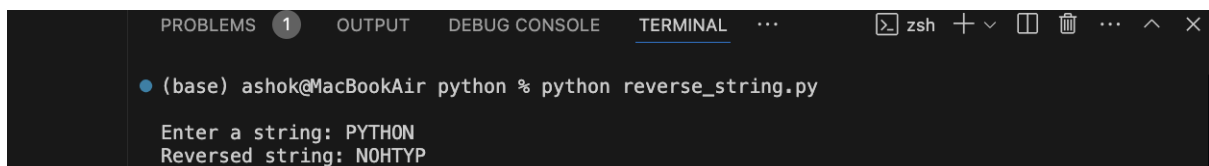
## Code

A screenshot of a code editor interface. The main editor window shows a Python file named 'reverse\_string.py' with the following code:

```
1 # Without using built-in functions,reverse a string
2 string = input("Enter a string: ")
3
4 reversed_string = ""
5
6 for i in range(len(string) - 1, -1, -1):
7     reversed_string += string[i]
8
9 print("Reversed string:", reversed_string)
10
```

The left sidebar shows the file explorer with 'reverse\_string.py' selected. The right sidebar shows a Copilot-generated description: 'Python program to reverse a string from user input without using any user-defined functions.' and a 'Create File...' button.

## Output:

A screenshot of a terminal window. The terminal shows the command '(base) ashok@MacBookAir python % python reverse\_string.py' being executed. The output is:

```
Enter a string: PYTHON
Reversed string: NOHTYP
```

## **Task-2:**

### **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:**

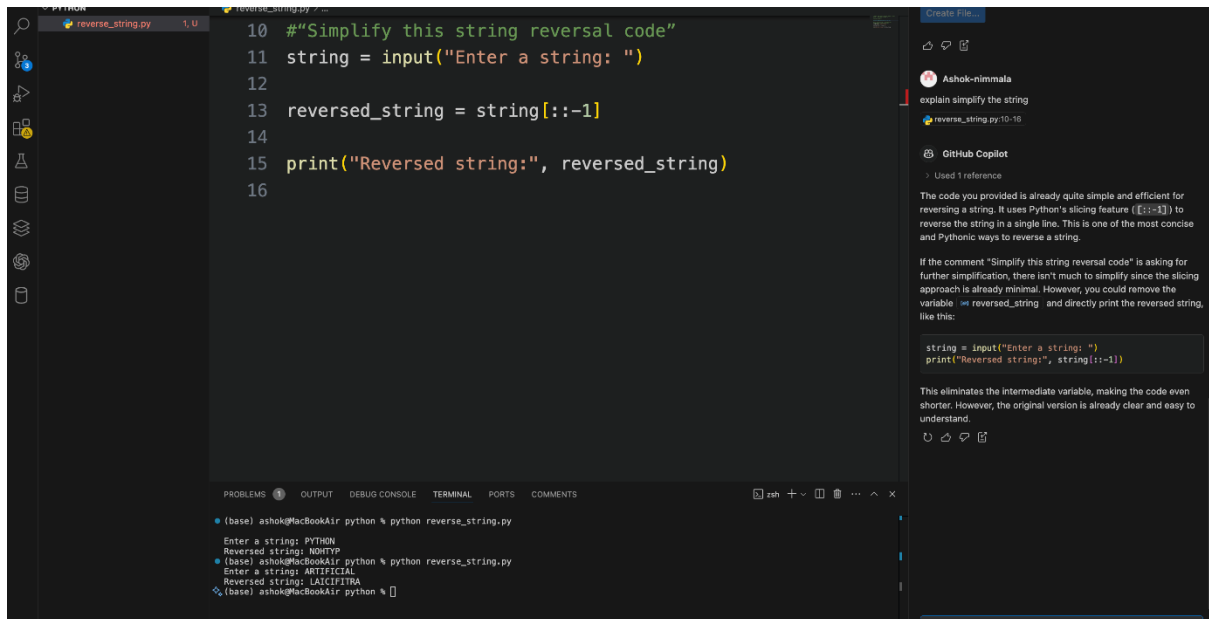
Prompt Copilot with phrases like

“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

## **Code & Output:**



## Justification:

By prompting Copilot with

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

the code was reduced from a loop-based  $O(n^2)$  approach to an efficient  $O(n)$  solution using slicing

## 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

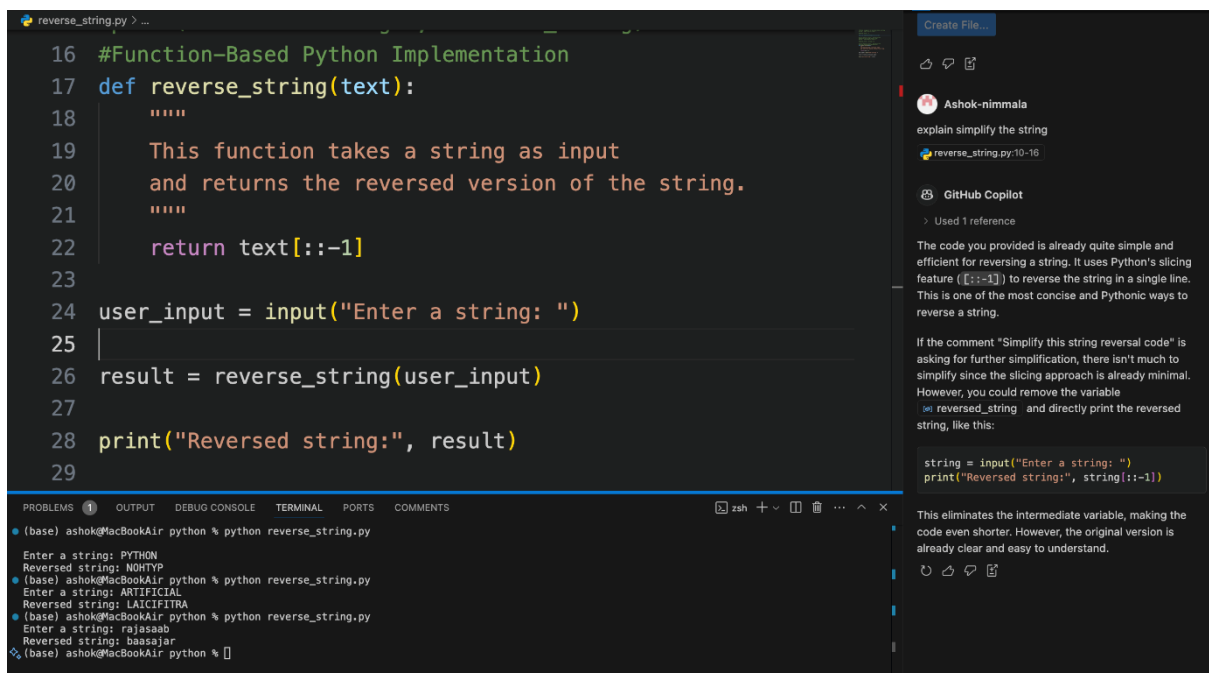
Use GitHub Copilot to generate a function-based Python program that:

- 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

## Output:



The screenshot displays a code editor with a Python script named `reverse_string.py`. The script defines a function `reverse_string(text)` that takes a string as input and returns its reversed version using slicing (`text[::-1]`). Below the function, the script prompts the user to enter a string, calls the `reverse_string` function, and prints the result.

The terminal output shows the script being executed with several test cases:

```
(base) ashok@MacBookAir python % python reverse_string.py
Enter a string: PYTHON
Reversed string: NOTYPO
(base) ashok@MacBookAir python % python reverse_string.py
Enter a string: ARTIFICIAL
Reversed string: LAICIFITRA
(base) ashok@MacBookAir python % python reverse_string.py
Enter a string: rajasaab
Reversed string: baasajar
(base) ashok@MacBookAir python %
```

The sidebar on the right shows the GitHub Copilot interface, which provides an explanation of the string reversal code and suggests a more concise implementation using slicing directly in the print statement:

```
string = input("Enter a string: ")
print("Reversed string:", string[::-1])
```

## 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

### **Report:**

In the procedural approach (Task 1), the string reversal logic is implemented directly in the main code. While this method is simple and suitable for very small programs, it leads to poor reusability and reduced readability when the same logic is required multiple times. Debugging also becomes challenging as the application grows.

In contrast, the modular approach (Task 3) uses a user-defined function to encapsulate the string reversal logic. This significantly improves code clarity, maintainability, and scalability. The function-based design allows the logic to be reused across multiple modules, making it ideal for large-scale and real-world applications. Additionally, modular code aligns with best software engineering practices and simplifies debugging and testing.

### **Conclusion:**

During a code review, the modular (function-based) approach should be preferred over the procedural approach because it:

Enhances readability

Promotes reusability

Simplifies debugging

Supports scalable application design

## **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

### **Code:**

```

reverse_string.py > reverse_string_slice
29 #Implementation 1: Loop-Based String Reversal (Iterative)
30 def reverse_string_loop(text):
31     reversed_text = ""
32
33     for char in text:
34         reversed_text = char + reversed_text
35
36     return reversed_text
37
38 #Implementation 2: Built-in / Slicing-Based String Reversal
39 user_input = input("Enter a string: ")
40 print("Reversed string (loop):", reverse_string_loop(user_input))
41
42 def reverse_string_slice(text):
43     return text[::-1]
44
45
46 user_input = input("Enter a string: ")
47 print("Reversed string (slicing):", reverse_string_slice(user_input))

```

## Comparison:

### Execution Flow

Loop-Based: Reverses the string character by character using a loop.

Slicing-Based: Uses Python's built-in slicing to reverse the string in one step.

### Time Complexity

Loop-Based:  $O(n^2)$  due to repeated string concatenation.

Slicing-Based:  $O(n)$  because it processes the string in a single pass.

### Performance for Large Inputs

Loop-Based: Slower and inefficient for large strings.

Slicing-Based: Fast and efficient even for large inputs.

When Each Approach is Appropriate



Loop-Based: Suitable for learning basic logic and algorithm understanding.

Slicing-Based: Best for real-world applications and production code.

## **Conclusion:**

Both approaches work correctly, but the slicing-based method is preferred due to better performance, simplicity, and scalability.