ASSIGNMENT-1.5

M.HARIKRISHNA

2203A52100

B-50

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

Prompt:

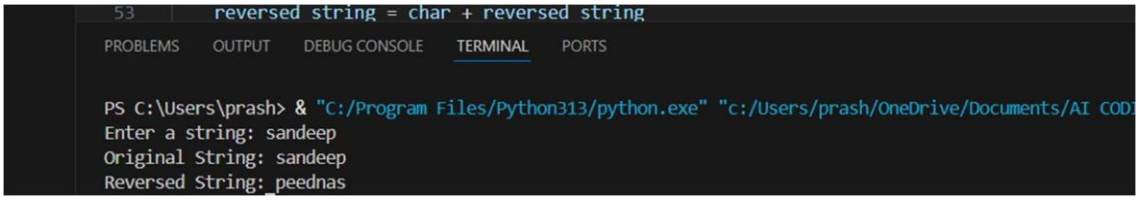
Generate a Python program to reverse a string by taking user input and implementing the logic directly in the main program without using any user-defined functions.

Code:

```
# String Reversal Without Using Functions

# Take input from user user_string =

input("Enter a string: ")
```

```
# Reverse string using slicing

reversed_string = user_string[::-1]


# Output print("Original String:",

user_string) print("Reversed String:",

reversed_string)
```

Output:



Code Explanation:

In this program, the string is taken as input from the user using the input() function. The reversal is done using slicing ([::-1]), which reads the string from the last character to the first. Since no function is used, the logic is written directly in the main program, making it simple but less reusable. This approach is suitable for small programs but not ideal for large applications.

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

Prompt:

Simplify and optimize this Python string reversal code by improving readability, removing unnecessary variables, and improving efficiency.

Code:

```
# Original Loop Based Reversal

user_string = input("Enter a string:

") reversed_string = "" for char in

user_string:

    reversed_string = char + reversed_string

print("Reversed String:", reversed_string)
```

Output:

Code Explanation:

The optimized code removes extra variables and simplifies logic by using Python slicing directly. This improves readability and reduces execution overhead. In loop-based reversal, string concatenation can increase time complexity, but slicing is optimized internally, making the program faster and more efficient. This improves performance especially for large strings.

Task 3

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

Prompt:

Generate a Python program that reverses a string using a user-defined function, returns the reversed string, and includes meaningful comments.

Code:

```
def reverse_string(s):
    """Reverses the input string s and returns the reversed
version."""     reversed_str = ""     for char in s:
        reversed_str = char + reversed_str     return
reversed_str # Testing the function input_string =
"Hello, World!" print("Original String:",
input_string) print("Reversed String:",
reverse_string(input_string))
```

Output:

```
PS C:\Users\prash> & "C:/Program Files/Python313/python.exe" "c:/Users/prash/OneDrive/Documents/AI CODING/ai-coding-1.5.py"
Original String: Hello, World!
Reversed String: !dlroW ,olleH
PS C:\Users\prash>
```

Code Explanation:

In this program, a function is created to reverse the string. The function accepts the string as input and returns the reversed result. This modular approach improves code reusability and makes debugging easier. Functions help organize logic, making the program easier to maintain and use multiple times in large applications.

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

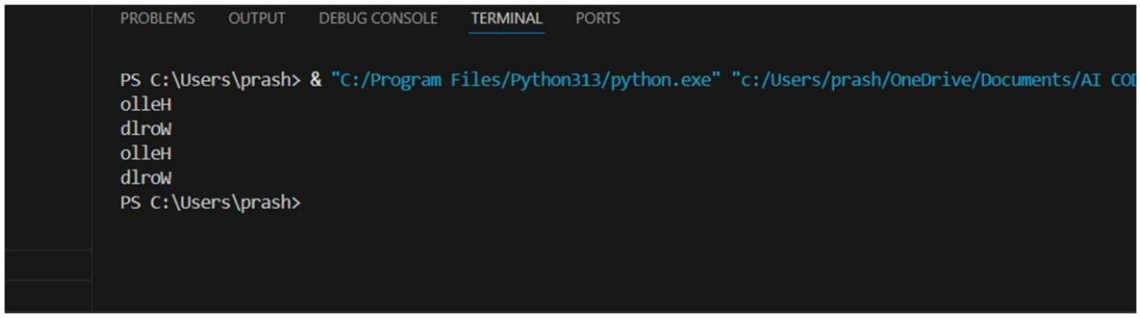
Prompt:

Compare Python string reversal programs implemented without functions and with functions, and analyze them based on clarity, reusability, debugging, and scalability.

Code:

```
# Without
Function string1 =
"Hello" rev1 =
string1[::-1]
print(rev1) string2
= "World" rev2 =
string2[::-1]

print(rev2)


# With Function  def
reverse_string(s):
    return s[::-1]
print(reverse_string("Hello"))
print(reverse_string("World"))
```

Output:



Code explanation:

The non-modular approach (without functions) is simple but difficult to reuse and maintain. The modular approach (with functions) is structured and reusable. Modular code makes debugging easier because errors can be isolated inside functions. For large applications, modular programming is always preferred because it supports scalability and teamwork development.

Task-5

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

Prompt:

Generate two Python implementations for string reversal: one using a loop-based approach and another using slicing, and compare their execution flow, time complexity, and performance.

Code:

```
# Loop-Based Approach
def reverse_string_loop(s):
    reversed_str = ""
for char in s:
        reversed_str = char + reversed_str
return reversed_str
# Built-in / Slicing-Based Approach
def reverse_string_slice(s):
```

return s[::-1] # Testing the functions input_string = "Hello,

World!" print("Loop-Based Reversal:",

reverse_string_loop(input_string)) print("Slicing-Based

Reversal:", reverse_string_slice(input_string))

Output:

Code explanation:

The loop-based method reverses the string by iterating through each character and building the reversed string step by step. This method is easy to understand but slower because string

concatenation takes more time. The slicing method reverses the string in one step using Python's built-in slicing feature. It is faster and more efficient because it is optimized at a lower level. For large inputs, slicing is recommended, while loop-based methods are useful for learning concepts.
