AI ASSITANT CODING

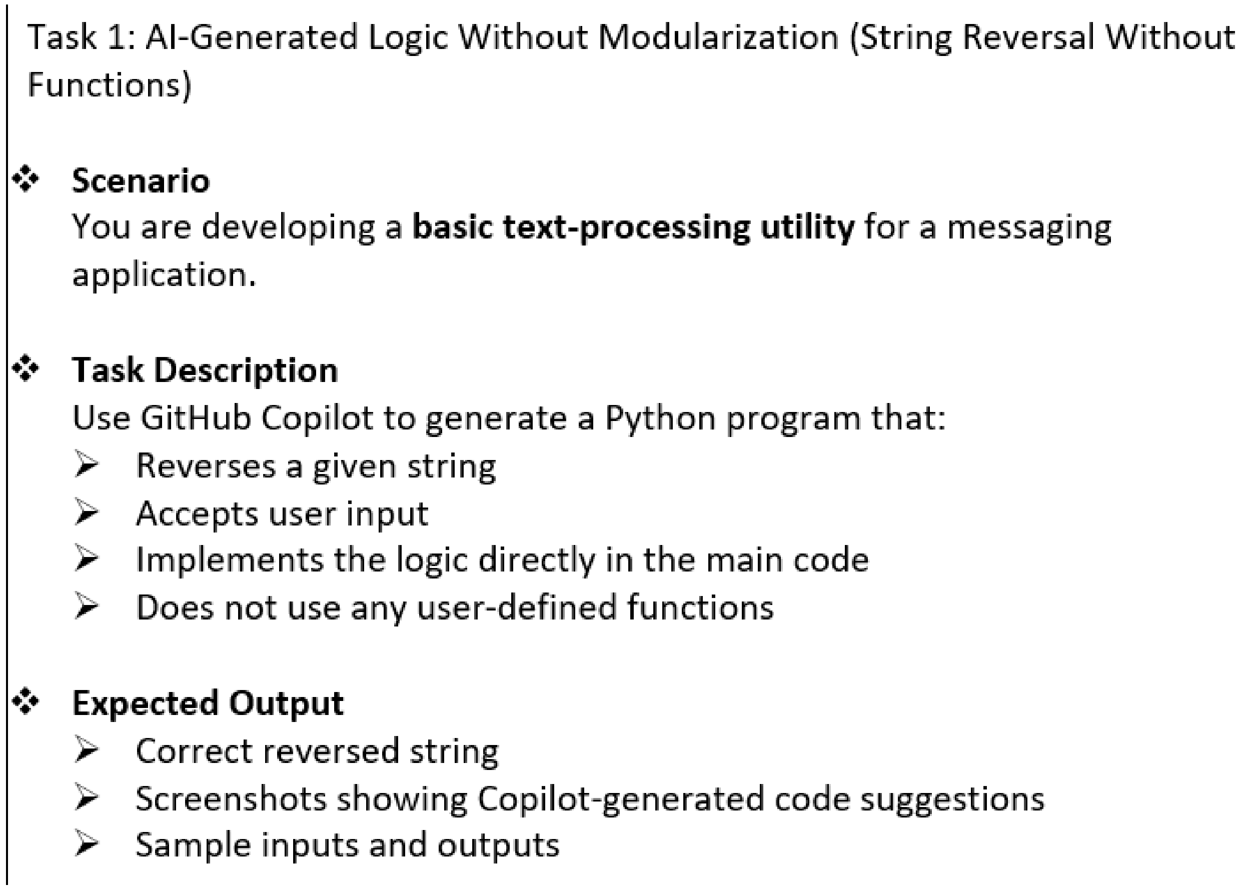
ASSIGNMENT - 1.5

NAME-S.SAI

H:NO-2303A52492

BATCH-50

TASK-1



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:

write a python code to reverse a given string without user defined functions input should be taken from the user.

CODE LOGIC:

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

OUTPUT:

```
Enter a string: sai

Reversed string: ias
```

JUSTIFICATION:

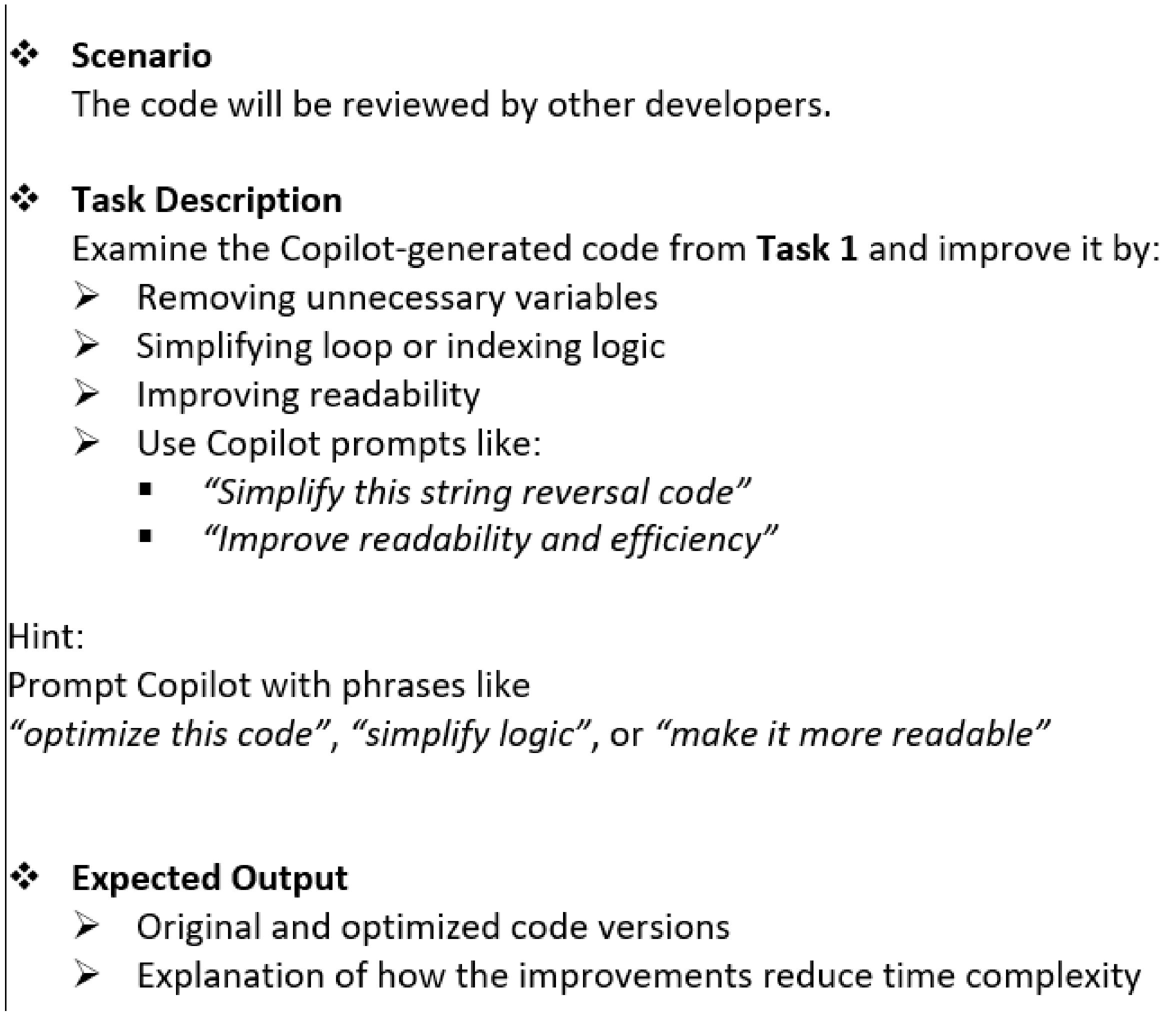
Memory Efficiency: It creates a new string in a single, highly optimized pass.

Readability: For experienced Python developers, this is the "standard" way. However, for beginners, the `[::-1]` syntax can look a bit like "magic" code.

TASK-2

Task 2: Efficiency & Logic Optimization (Readability Improvement)

QUESTION:



❖ **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:

```
Make The Code more readable and decrease the time complexityMake The Code
more readable and decrease the time complexity
```

CODE:

```
string = input("Enter a string: ")
reversed_string = ''.join(reversed(string))
print("Reversed string:", reversed_string)
```

OUTPUT:

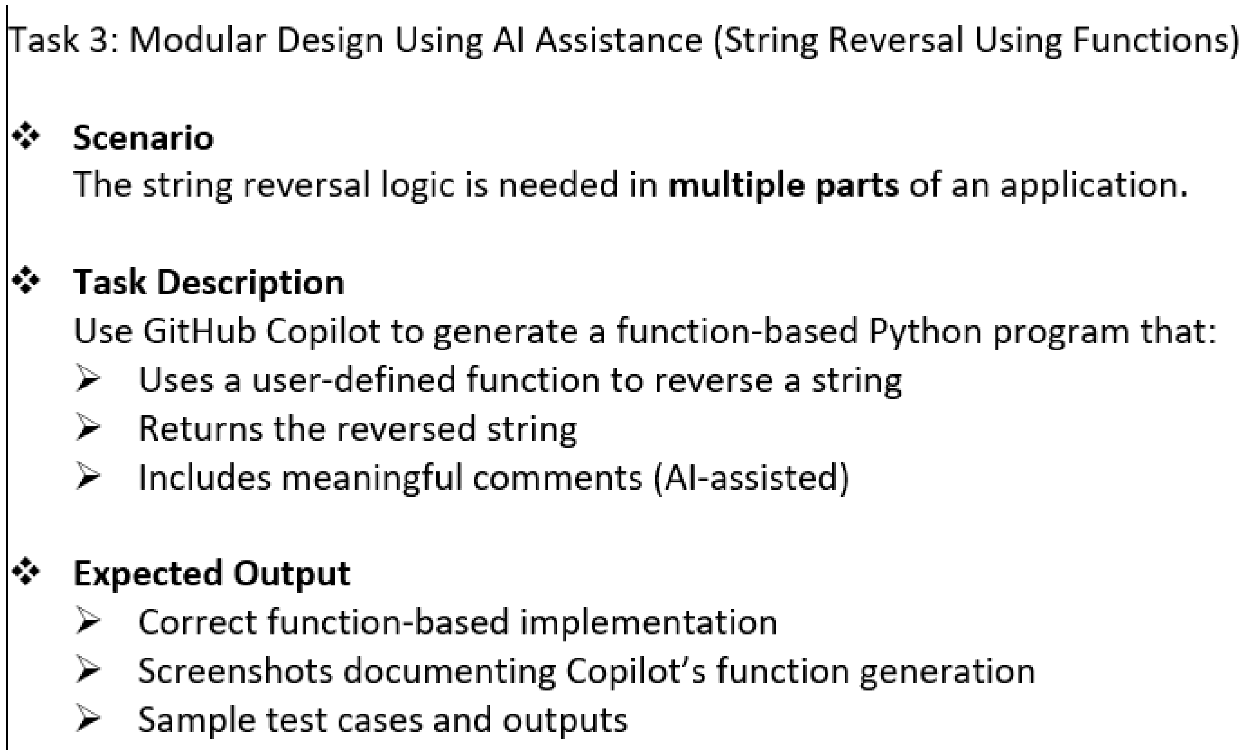
```
Enter a string: HELLO WORLD
Reversed string: DLROW OLLEH
```

JUSTIFICATION:

Readability: It is very explicit. A human reading the code knows exactly what is happening: you are *reversing* the string and then *joining* the characters back together.
Overhead: Technically, this is **slower** than slicing. Here is why:

1. `reversed(string)` creates an **iterator** object.
2. `.join()` then has to loop through that iterator and build the string. These extra steps (creating an object and calling a method) add "overhead" that makes it slower than the direct slicing method.

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:

write a python code that reverse a string with user defined functions that return a reversed string and comments for understand the code.

CODE:

```
def reverse_string(s):
    # This function takes a string as input and returns the reversed string
    return s[::-1]
```

```
# Take input from the user
user_input = input("Enter a string: ")
# Call the function and store the result
result = reverse_string(user_input)
# Print the reversed string
print("Reversed string:", result)
```

OUTPUT:

```
Enter a string: Hello and welcome sai
Reversed string: ias emoclew dna olleH
```

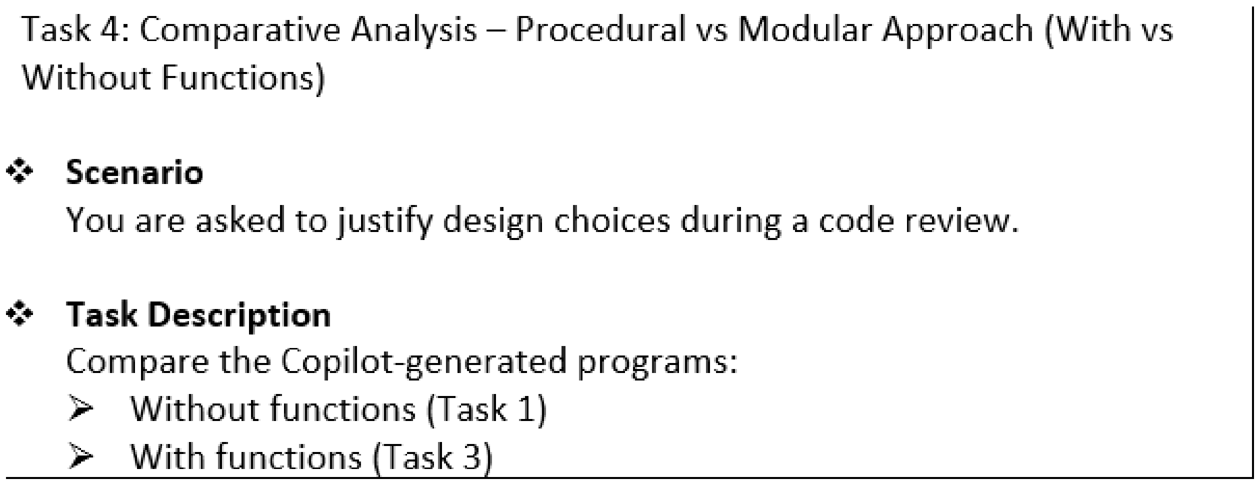
JUSTIFICATION:

Reusability:
By defining a function, you can reverse strings multiple times throughout a larger program without rewriting the slicing logic.

Modularity:
It separates the core logic (reversing) from the user interface (input/output), making the code cleaner and easier to maintain.

TASK 4

QUESTION:

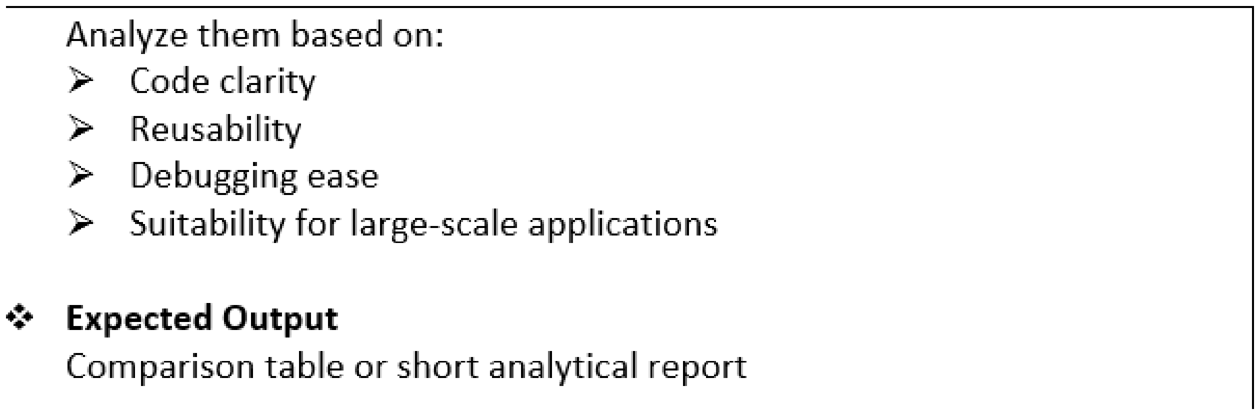


Task 4: Comparative Analysis – Procedural vs Modular Approach (With vs Without Functions)

❖ **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