Assignment-1.5

Roll.No:2303A51062
Batch-29

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:

```
# Accept user input
text = input("Enter a string: ")

# Reverse the string without using functions
reversed_text = ""

for char in text:
    reversed_text = char + reversed_text

# Display output
print("Reversed string:", reversed_text)
```

```
Enter a string: Hello World
Reversed string: dlroW olleH
```

Justification:

This task demonstrates how AI tools like GitHub Copilot can generate correct program logic without relying on modularization. Implementing string reversal directly in the main code helps beginners understand fundamental control structures such as loops and string manipulation. Avoiding user-defined functions keeps the logic simple and transparent, making it easier to follow the execution flow. Using Copilot in this scenario highlights its effectiveness in producing straightforward, readable code for basic text-processing tasks.

ask 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”

❖ **Expected Output**
➢ Original and optimized code versions
➢ Explanation of how the improvements reduce time complexity

code:

```python
# Original Code (Copilot Generated)
text = input("Enter a string: ")

reversed_text = ""

for char in text:
    reversed_text = char + reversed_text

print("Reversed string:", reversed_text)


# Optimized Code (Improved Readability & Efficiency)
text = input("Enter a string: ")

reversed_text = text[::-1]

print("Reversed string:", reversed_text)
```

```
Enter a string: Orange
Reversed string: egnarO
Enter a string: Pineapple
Reversed string: elppaeniP
```

Justification:

Using fruit names as input helps demonstrate string reversal in a clear and familiar way. Since fruit names contain only alphabetic characters, they make it easy to verify that the reversal logic works correctly without confusion from numbers or symbols. This approach improves understanding of string manipulation and helps users visually confirm the correctness of the output.

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

Code:

```
# Function to reverse a string
# This function takes a string as input and returns the reversed string
def reverse_string(text):
    # Reverse the string using slicing for efficiency
    return text[::-1]


# Sample test cases
words = ["Benny", "rose", "hani", "blessy"]

# Calling the function for each test case
for word in words:
    print("Original:", word, "| Reversed:", reverse_string(word))
```

```
Original: Benny | Reversed: ynneB
Original: rose | Reversed: esor
Original: hani | Reversed: inah
Original: blessy | Reversed: ysselb
```

Justification:

Using a function-based approach for string reversal improves modularity and reusability, especially when the same logic is required in multiple parts of an application. GitHub Copilot helps generate well-structured functions with meaningful comments, which enhances code readability and reduces development time. This modular design also simplifies debugging, testing, and future maintenance, making it suitable for scalable 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

Code:

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

reversed_text = ""

for char in text:
    reversed_text = char + reversed_text

print("Reversed string:", reversed_text)


def reverse_string(text):
    # Reverse the string using slicing
    return text[::-1]

text = input("Enter a string: ")
print("Reversed string:", reverse_string(text))
```

```
Enter a string: benny
Reversed string: ynneb
Enter a string: paul
Reversed string: luap
```

Justification:

During a code review, comparing procedural and modular approaches helps justify better design decisions. The procedural approach without functions is simple and suitable for small, one-time tasks, but it lacks reusability and becomes difficult to maintain as the codebase grows. In contrast, the modular approach using functions improves code clarity, allows reuse of logic, and makes debugging easier by isolating functionality. This structure is more suitable for large-scale applications where maintainability, scalability, and collaboration are important.

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:

```
# Loop-based string reversal
text = input("Enter a string: ")

reversed_text = ""
for char in text:
    reversed_text = char + reversed_text

print("Reversed string (Loop-based):", reversed_text)
```

```
Enter a string: hani
Reversed string (Loop-based): inah
```

Justification:

This task highlights how AI can generate multiple valid algorithmic solutions for the same problem, allowing developers to compare different logic paths. The loop-based

approach clearly demonstrates iterative execution and is useful for understanding basic control flow, while the slicing-based approach leverages Python’s built-in features for better efficiency and readability. Comparing both approaches helps justify choosing the most appropriate solution based on performance requirements, code clarity, and scalability, especially when handling large inputs.