Lab Assignment 1.5 – AI ASSISTED CODING

Name: Syed Murtaza

H.T No: 2303A51259

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

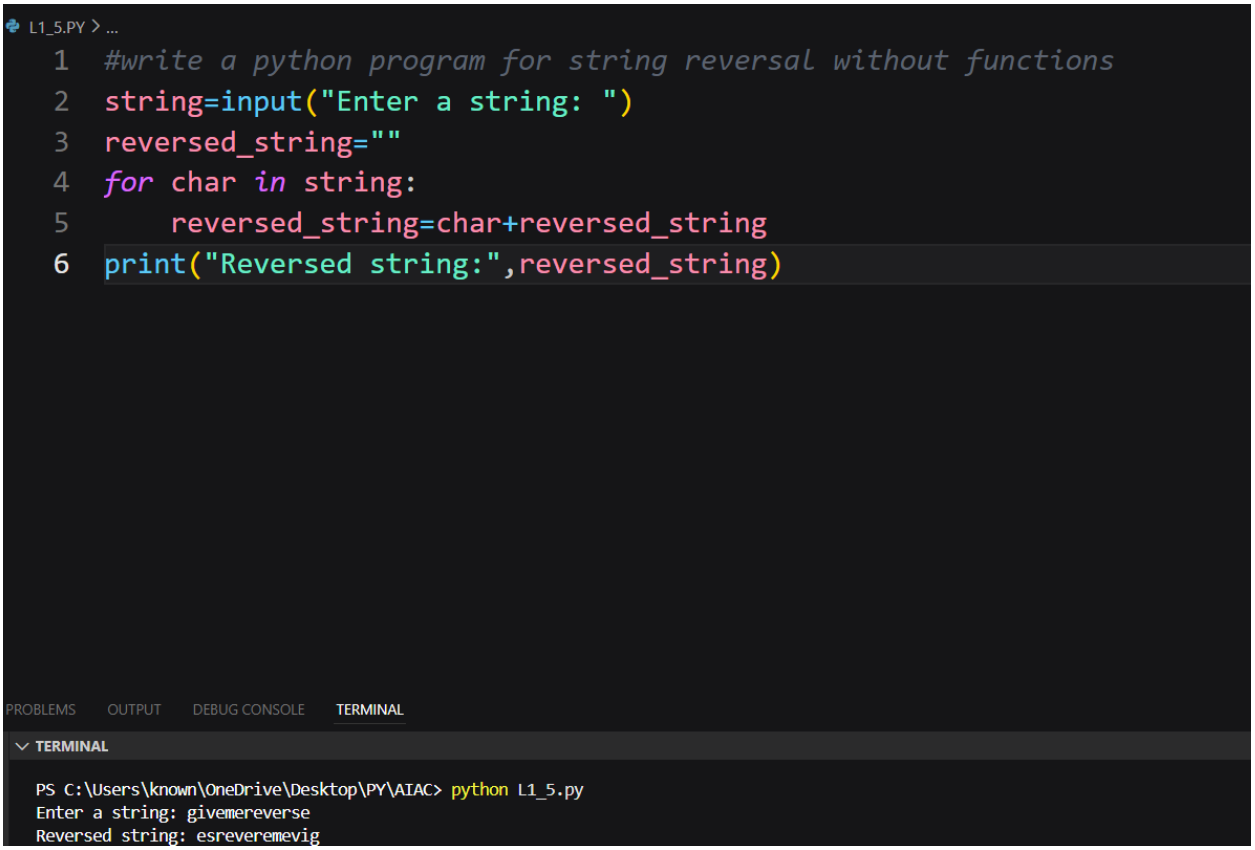
➢ Implements the logic directly in the main code

➢ Does not use any user-defined functions

Prompt:

#Write a python program for string reversal without functions

Code:



```python
#write a python program for string reversal without functions
string=input("Enter a string: ")
reversed_string=""
for char in string:
    reversed_string=char+reversed_string
print("Reversed string:",reversed_string)
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L1_5.py
Enter a string: givemereverse
Reversed string: esreveremevig
```

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

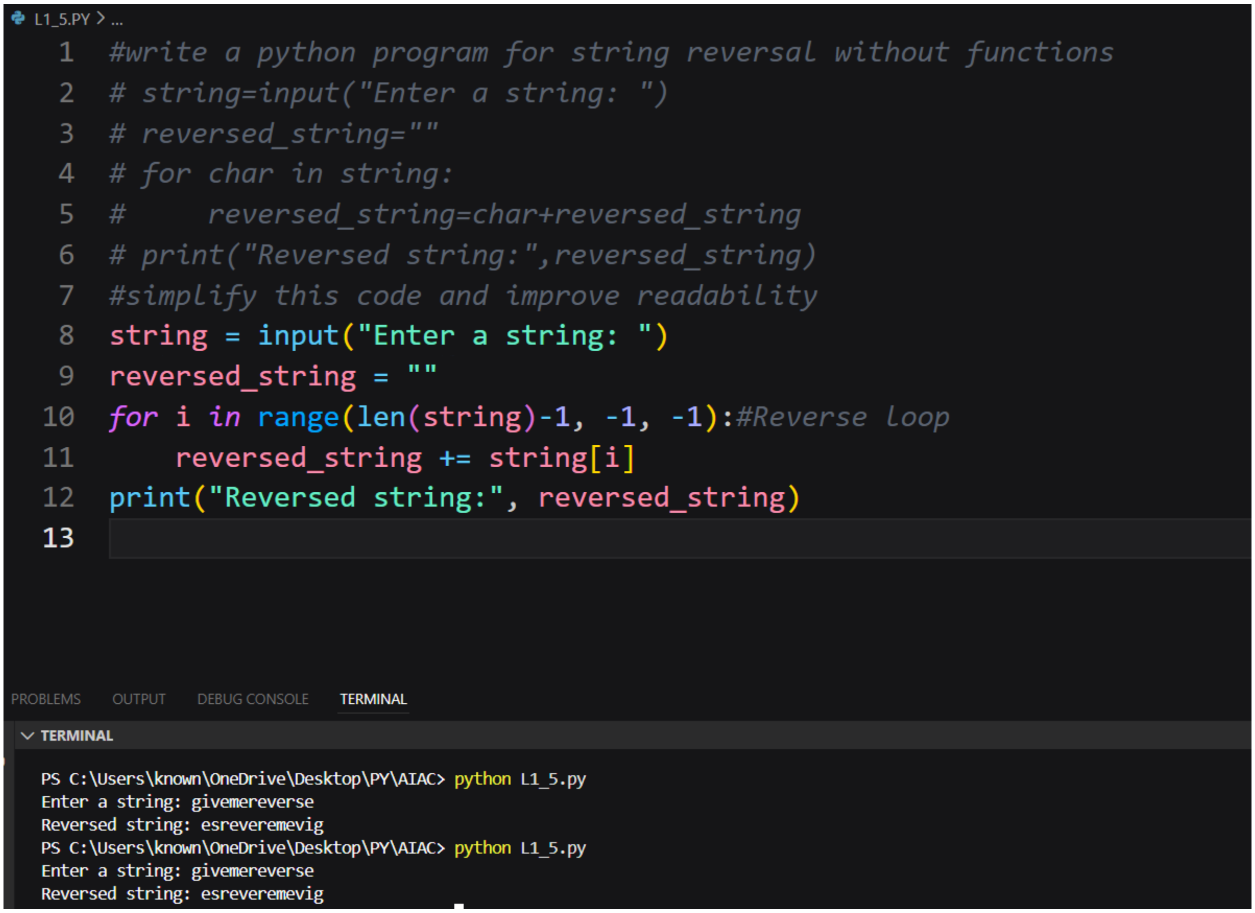
➢ Simplifying loop or indexing logic

➢ Improving readability

Prompt:

#Simplify the above code and improve readability

Code:



```python
#write a python program for string reversal without functions
# string=input("Enter a string: ")
# reversed_string=""
# for char in string:
#     reversed_string=char+reversed_string
# print("Reversed string:",reversed_string)
#simplify this code and improve readability
string = input("Enter a string: ")
reversed_string = ""
for i in range(len(string)-1, -1, -1):#Reverse loop
    reversed_string += string[i]
print("Reversed string:", reversed_string)
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L1_5.py
Enter a string: givemereverse
Reversed string: esreveremevig
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L1_5.py
Enter a string: givemereverse
Reversed string: esreveremevig
```

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:

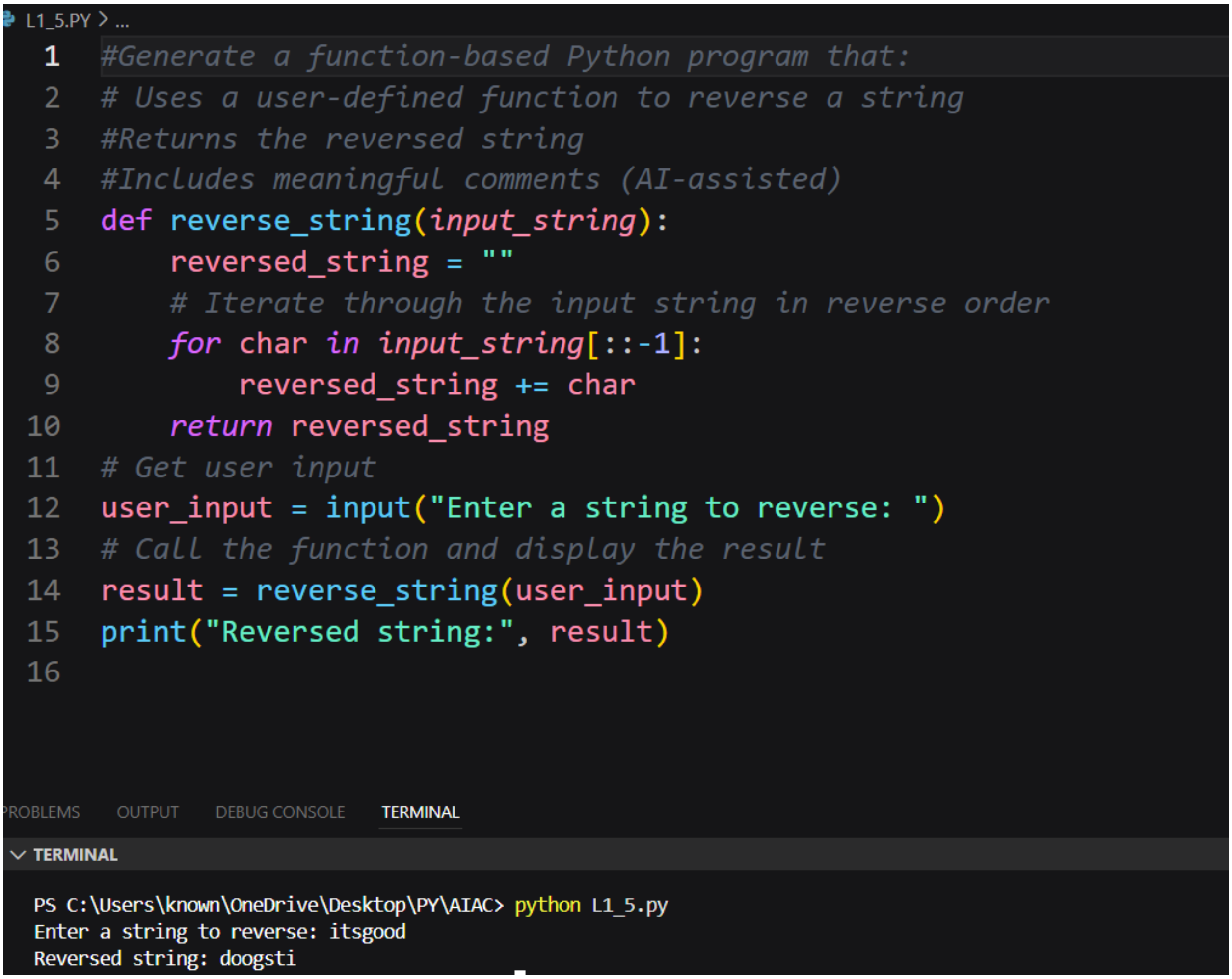
➢ Uses a user-defined function to reverse a string

➢ Returns the reversed string

➢ Includes meaningful comments (AI-assisted)

Prompt:

#Generate a function based program that: Uses a user-defined function to reverse a string, returns the reversed string, includes meaningful comments(AI-assisted)



```python
#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)
def reverse_string(input_string):
    reversed_string = ""
    # Iterate through the input string in reverse order
    for char in input_string[::-1]:
        reversed_string += char
    return reversed_string
# Get user input
user_input = input("Enter a string to reverse: ")
# Call the function and display the result
result = reverse_string(user_input)
print("Reversed string:", result)

```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python L1_5.py
Enter a string to reverse: itsgood
Reversed string: doogsti
```

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

Comparison Table:

Task 4: Comparative Analysis – Procedural vs Modular Approach

Criteria	**Without Functions (Task 1 – Procedural)**	**With Functions (Task 3 – Modular)**
Code Clarity	Code is simple and easy to understand for small programs, but logic and execution are mixed together.	Code is well-structured. Logic is separated into a function, making it cleaner and more readable.
Reusability	Low reusability. The code must be rewritten if string reversal is needed again.	High reusability. The function can be called multiple times with different inputs.
Debugging Ease	Debugging becomes harder as the program grows since all logic is in one block.	Easier to debug because issues can be isolated within the function.
Suitability for Large-Scale Applications	Not suitable for large programs. Code becomes lengthy and difficult to manage.	Highly suitable. Modular design supports scalability and teamwork.

Conclusion

The **procedural approach (Task 1)** is suitable for small, simple programs, while the **modular approach (Task 3)** is better for larger, maintainable, and reusable applications. Using functions improves code organization, debugging, and scalability.

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

Comparison Table:

Task 5: Comparison – Loop-Based vs Slicing-Based String Reversal

Aspect	**Loop-Based Approach**	**Slicing-Based Approach**
Execution Flow	Iterates through the string character by character and builds the reversed string step by step.	Uses Python’s built-in slicing mechanism to reverse the string in a single operation.
Time Complexity	**O(n)** — each character is processed once.	**O(n)** — slicing traverses the entire string once internally.
Performance for Large Inputs	Slightly slower due to explicit looping and repeated string concatenation.	Faster and more optimized because slicing is implemented in C internally.
Memory Usage	Uses extra memory for constructing the reversed string incrementally.	Creates a new reversed string in one step; generally more memory-efficient.
Readability	More verbose but clearly shows the reversal logic.	Very concise and easy to read.
When Appropriate	Useful for learning, interviews, or when custom logic per character is required.	Best choice for real-world applications where simplicity and performance matter.

Conclusion

Both approaches have the same time complexity, but the **slicing-based method** is more efficient and concise for large inputs. The **loop-based method** is valuable for understanding algorithmic flow and applying custom processing during reversal.