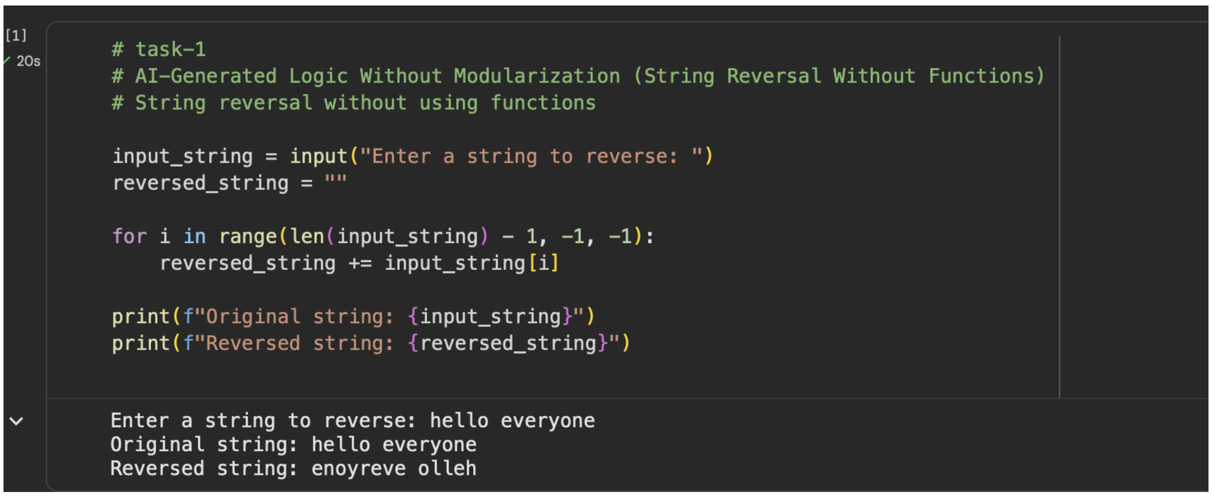
ASSIGNMENT 1.5

J.Sujith
2303A51327
BATCH 20

Task-1: AI-Generated Logic Without Modularization (String Reversal Without Functions)

Explanation

In Task-1, a string reversal program is written without using any functions.
The program takes a string as input from the user and reverses it using a for loop.
The loop starts from the last character of the string and moves backwards, appending each character to a new string.
This is a procedural approach, where all logic is written in a single block.



```
[1]
20s
# task-1
# AI-Generated Logic Without Modularization (String Reversal Without Functions)
# String reversal without using functions

input_string = input("Enter a string to reverse: ")
reversed_string = ""

for i in range(len(input_string) - 1, -1, -1):
    reversed_string += input_string[i]

print(f"Original string: {input_string}")
print(f"Reversed string: {reversed_string}")
```

```
Enter a string to reverse: hello everyone
Original string: hello everyone
Reversed string: enoyreve olleh
```

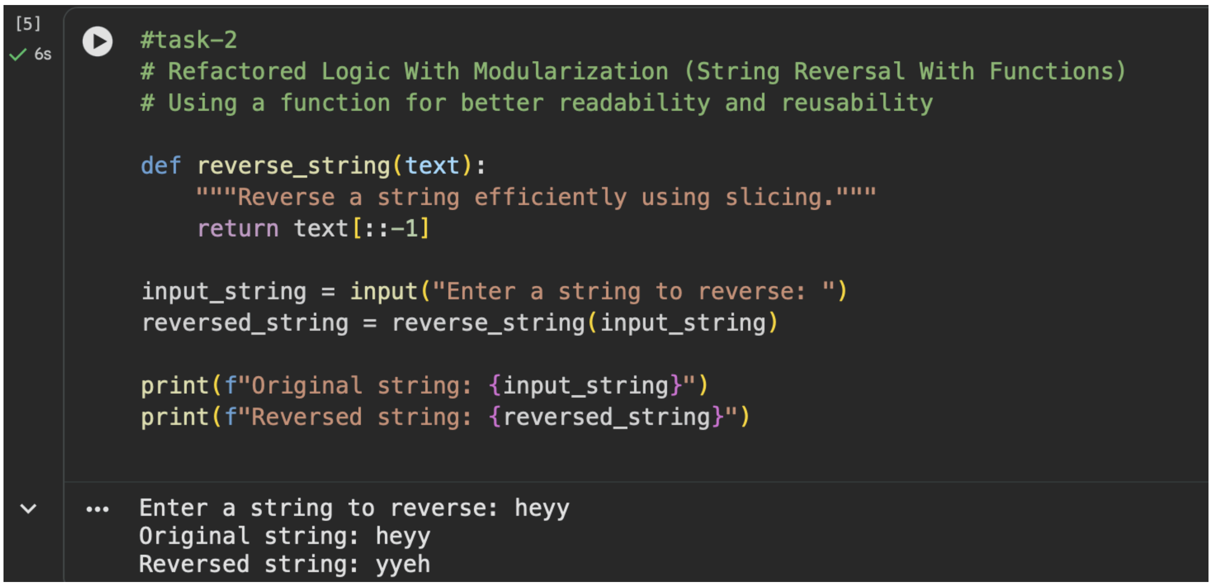
Justification

This task demonstrates how string reversal works at a basic level without modularization.
It helps beginners understand looping, indexing, and string manipulation clearly.
However, as the program grows, such code becomes harder to read, reuse, and maintain.

Task-2: Efficiency & Logic Optimization (Readability Improvement)

Explanation

In Task-2, the same string reversal logic is improved by using a function.
The reversal is done using Python's slicing technique (`[::-1]`), which reverses the string efficiently.
The logic is placed inside a function to improve readability and reusability.



```
[5]
✓ 6s
#task-2
# Refactored Logic With Modularization (String Reversal With Functions)
# Using a function for better readability and reusability

def reverse_string(text):
    """Reverse a string efficiently using slicing."""
    return text[::-1]

input_string = input("Enter a string to reverse: ")
reversed_string = reverse_string(input_string)

print(f"Original string: {input_string}")
print(f"Reversed string: {reversed_string}")
```

```
Enter a string to reverse: heyy
Original string: heyy
Reversed string: yyeh
```

Justification

Using functions makes the code cleaner and easier to understand.
The slicing method reduces code length and improves performance.
This approach is more suitable for real-world applications compared to the procedural method used in Task-1.

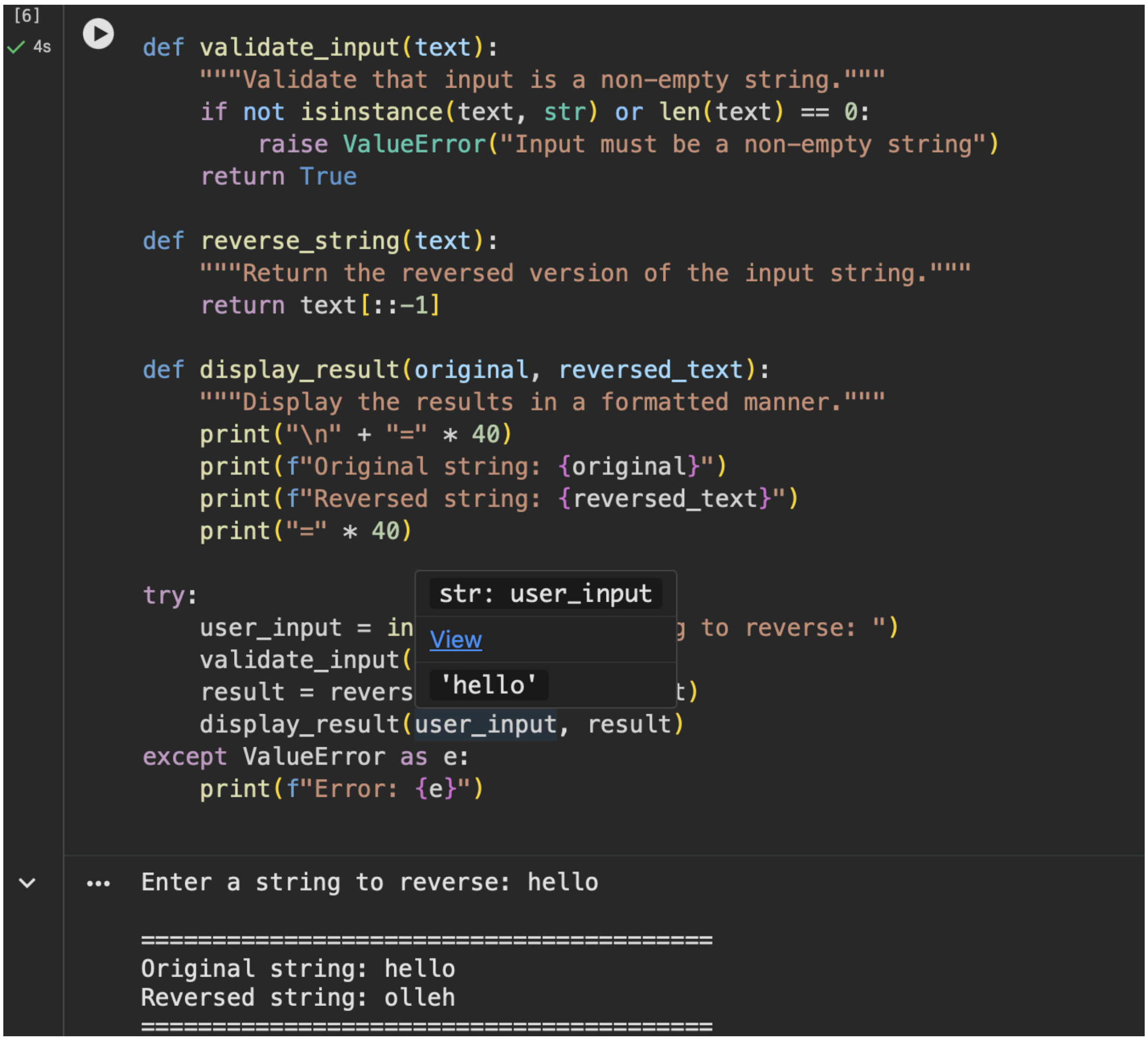
Task-3: Enhanced Modular Design with Input Validation and Error Handling

Explanation

Task-3 further improves the program by introducing:

- **Input validation**
- **Error handling**
- **Multiple well-defined functions**

The program checks whether the input is a **non-empty string**.
If invalid input is given, a `ValueError` is raised and handled using a `try-except` block.
Results are displayed using a separate display function.



```python
def validate_input(text):
    """Validate that input is a non-empty string."""
    if not isinstance(text, str) or len(text) == 0:
        raise ValueError("Input must be a non-empty string")
    return True

def reverse_string(text):
    """Return the reversed version of the input string."""
    return text[::-1]

def display_result(original, reversed_text):
    """Display the results in a formatted manner."""
    print("\n" + "=" * 40)
    print(f"Original string: {original}")
    print(f"Reversed string: {reversed_text}")
    print("=" * 40)

try:
    user_input = in                  g to reverse: ")
    validate_input(
    result = revers                  t)
    display_result(user_input, result)
except ValueError as e:
    print(f"Error: {e}")
```

```
Enter a string to reverse: hello

========================================
Original string: hello
Reversed string: olleh
========================================
```

Justification

This task shows how to write **robust and user-friendly code**.
Input validation prevents runtime errors and unexpected behavior.
Modular design improves debugging, testing, and maintenance, making the program suitable for large applications.

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

In this task, two string reversal programs generated using GitHub Copilot are compared:

- **Without functions (Procedural approach)**
- **With functions (Modular approach)**

Comparison Table

Aspect	Without Functions	With Functions
Code clarity	• Hard to understand when code increases	• Easy to read and understand
Reusability	• Cannot reuse code	• Function can be reused
Debugging	• Difficult	• Easy
Maintainability	• Hard to modify	• Easy to modify
Large applications	• Not suitable	• Suitable

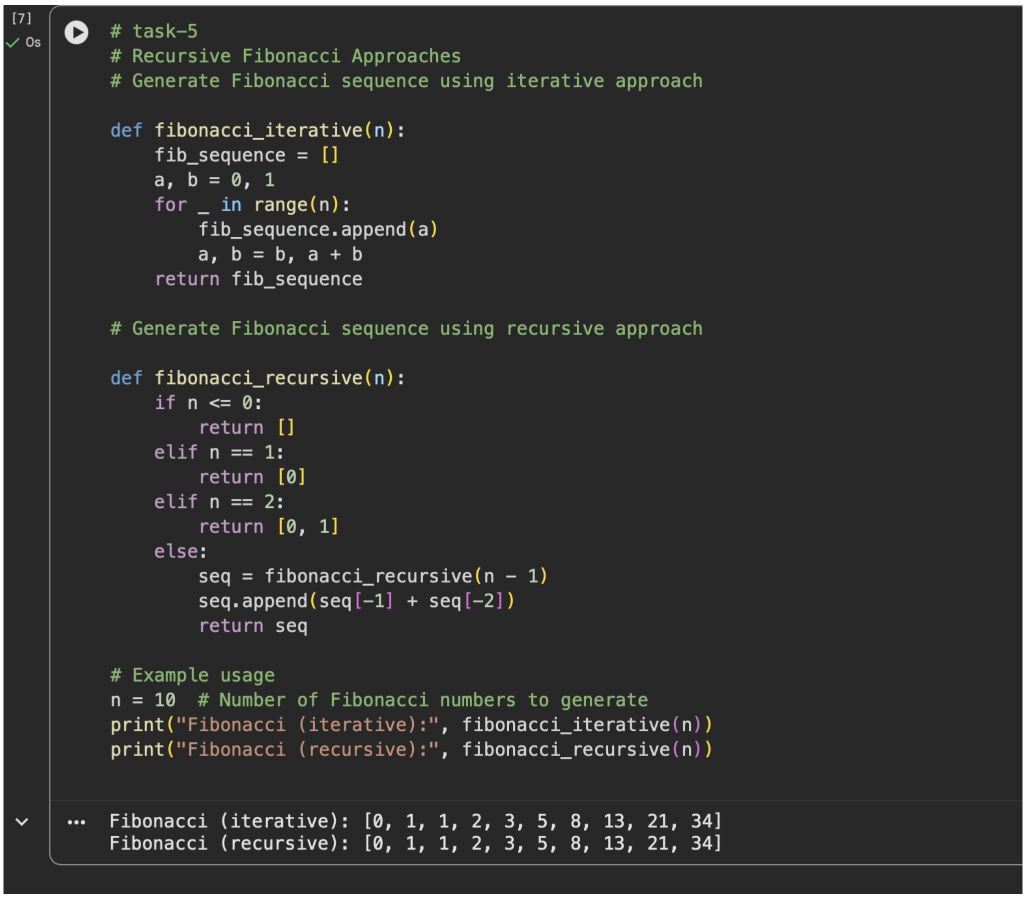
Task-5: AI-Generated Iterative vs Recursive Fibonacci Approaches

Explanation

Task-5 demonstrates two different algorithmic approaches for generating the Fibonacci sequence:

- **Iterative approach** using loops
- **Recursive approach** using function calls

Both methods generate the same Fibonacci sequence, but they differ in implementation style and efficiency.



```
[7]
0s
```

```python
# task-5
# Recursive Fibonacci Approaches
# Generate Fibonacci sequence using iterative approach

def fibonacci_iterative(n):
    fib_sequence = []
    a, b = 0, 1
    for _ in range(n):
        fib_sequence.append(a)
        a, b = b, a + b
    return fib_sequence

# Generate Fibonacci sequence using recursive approach

def fibonacci_recursive(n):
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        seq = fibonacci_recursive(n - 1)
        seq.append(seq[-1] + seq[-2])
        return seq

# Example usage
n = 10  # Number of Fibonacci numbers to generate
print("Fibonacci (iterative):", fibonacci_iterative(n))
print("Fibonacci (recursive):", fibonacci_recursive(n))
```

```
Fibonacci (iterative): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Fibonacci (recursive): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Justification

This task helps understand different problem-solving techniques.
The iterative approach is faster and uses less memory.
The recursive approach is easier to understand conceptually but less efficient for large inputs.
It shows how algorithm choice affects performance.