AI-ASSISTED CODING

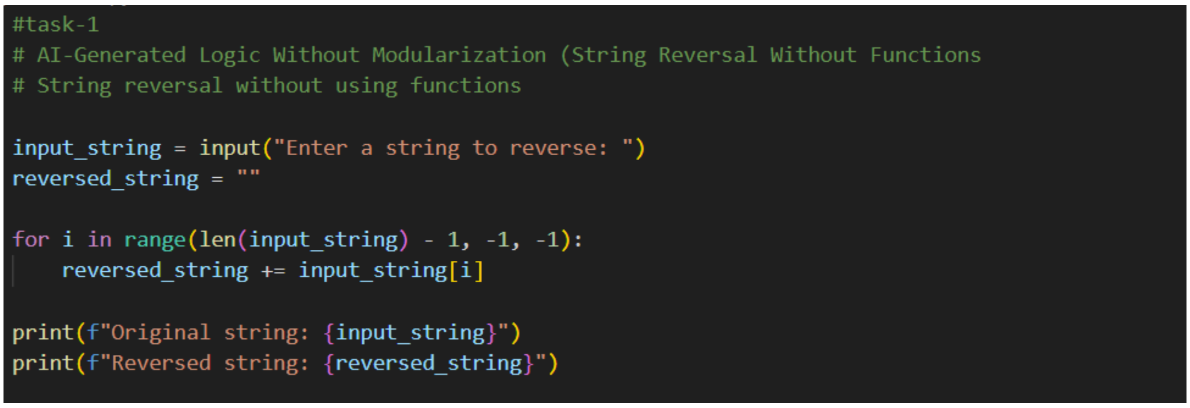
Assignment-1.5

Name: Praneeth banda

Hall Ticket: 2303A51711

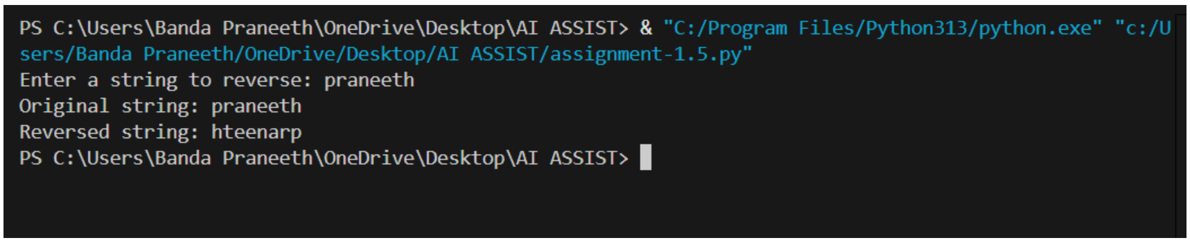
Batch-20

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



```python
#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}")
```

Output:

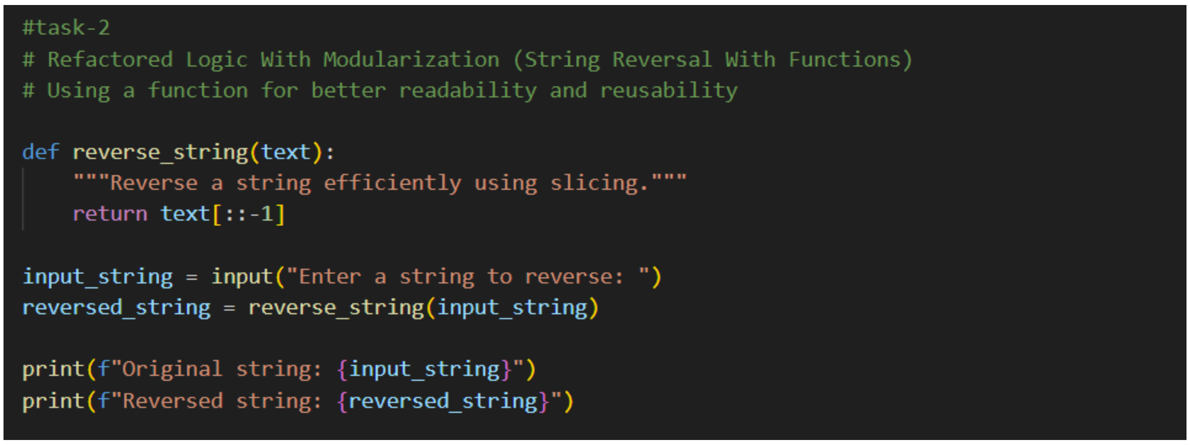


```
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/U
sers/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.5.py"
Enter a string to reverse: praneeth
Original string: praneeth
Reversed string: hteenarp
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST>
```

Justification:

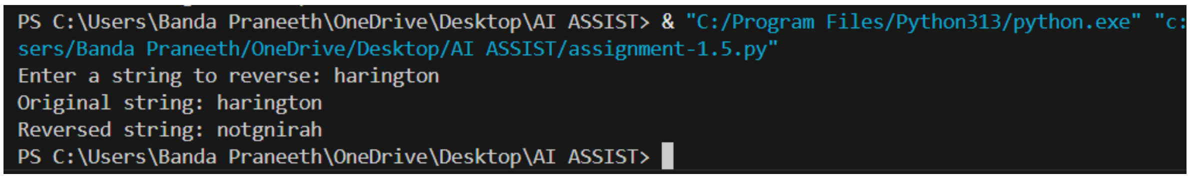
The program provided reverses a string without using any functions by looping through each character and inserting it at the start of a new string. This method successfully generates the reversed string as demonstrated. The logic is straightforward and easy to follow for small-scale programs. However, the code lacks reusability because the entire logic is placed inside the main section. As the program becomes larger, debugging and maintaining the code can become challenging. Implementing functions would enhance clarity, reusability, and make the program more appropriate for larger projects.

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



```python
#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}")
```

Output:

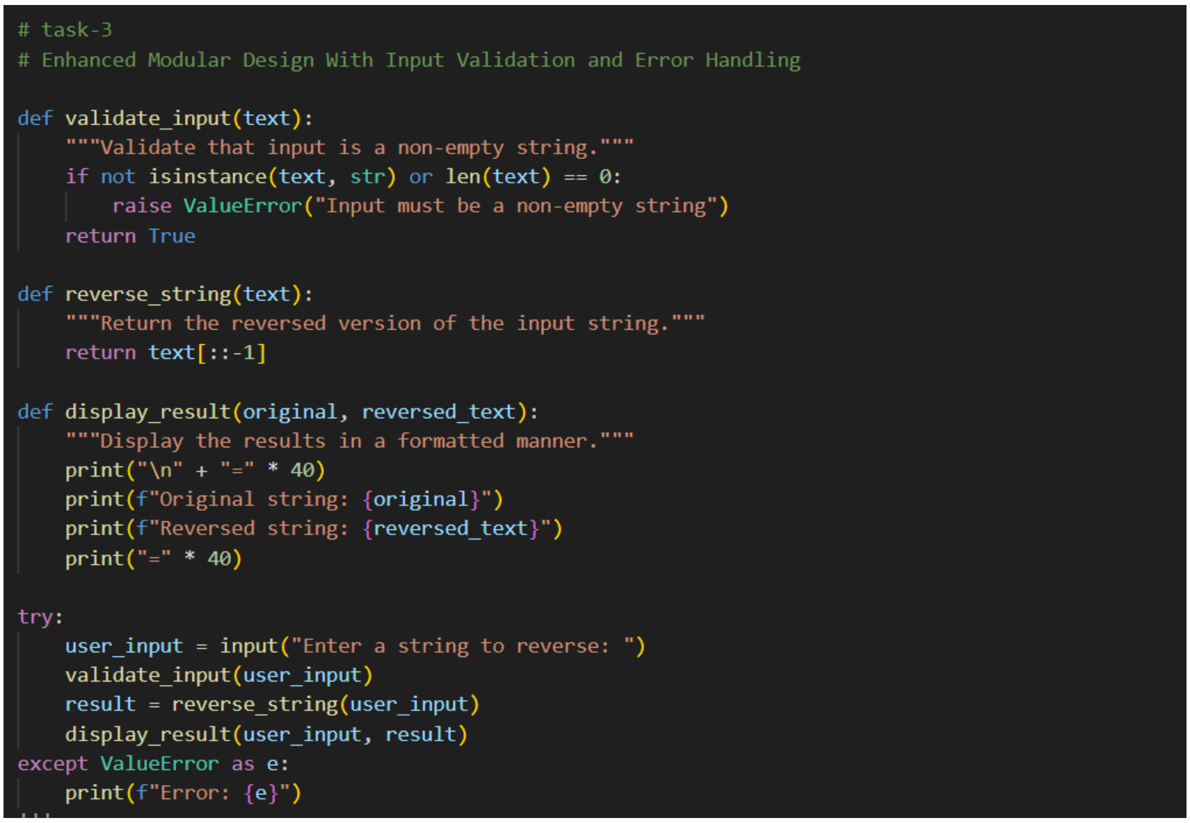


```
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:
sers/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.5.py"
Enter a string to reverse: harington
Original string: harington
Reversed string: notgnirah
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST>
```

Justification:

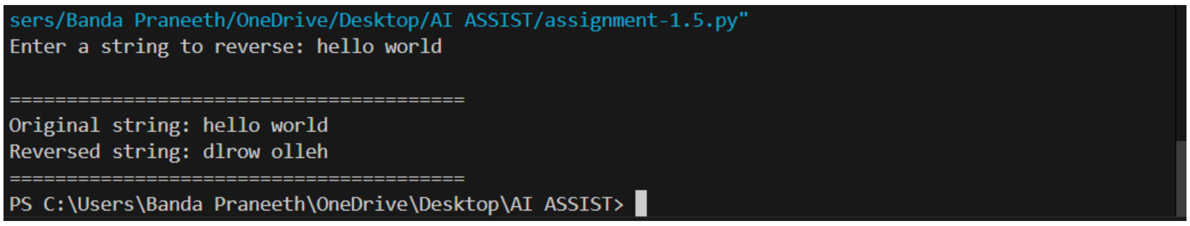
The manual technique reverses a string by looping through it and placing each character at the front of a new string. The slicing technique, on the other hand, uses Python’s built-in [::-1] slicing syntax, making it shorter and easier to read. Both approaches result in the same correct output. The manual method is helpful for understanding the underlying logic of string reversal. The slicing method is more efficient and commonly used in practical applications. Therefore, slicing is better for performance, while the manual method is valuable for learning concepts.

Task-3: Modular Design Using AI Assistance (String Reversal Using Functions)



```python
# task-3
# Enhanced Modular Design With Input Validation and Error Handling

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 = input("Enter a string to reverse: ")
    validate_input(user_input)
    result = reverse_string(user_input)
    display_result(user_input, result)
except ValueError as e:
    print(f"Error: {e}")
```

Output:



```
sers/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-1.5.py"
Enter a string to reverse: hello world

========================================
Original string: hello world
Reversed string: dlrow olleh
========================================
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST>
```

Explanation:

The function-oriented method reverses a string by using a reusable function, which keeps the code organized and structured. In contrast, the procedural method carries out the string reversal directly within the main program without using functions. Both methods generate the same correct result. The function-based approach is more convenient to reuse and maintain in larger applications. Debugging is also easier with a modular structure than with purely procedural code. Therefore, the function-based method is better suited for real-world projects, while the procedural method works well for small and simple tasks

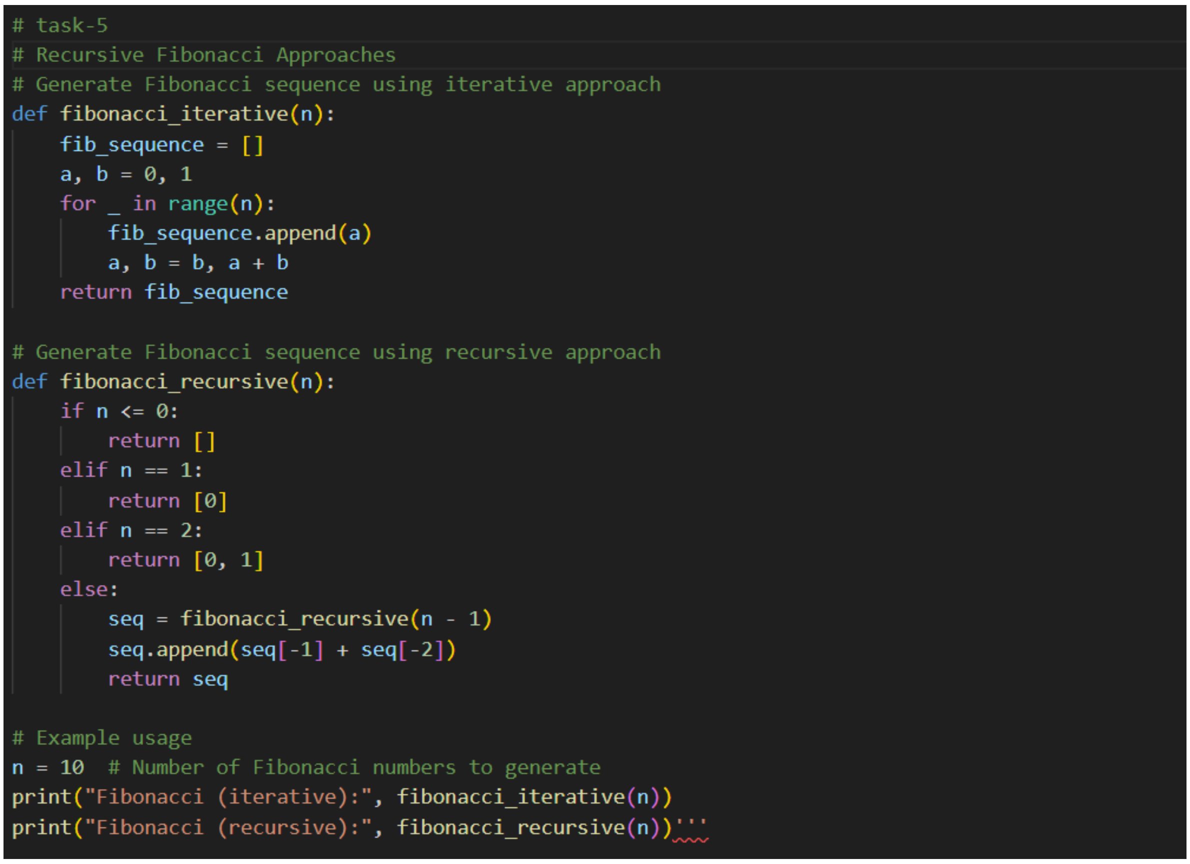
.Task-4: Comparative Analysis – Procedural vs Modular Approach (With vs Without Functions)

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

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

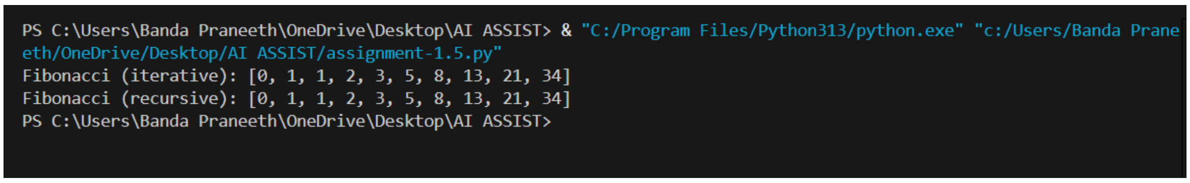
Aspect	Without Functions	With Functions
Code clarity	Hard to understand when code increases	Easy to read and understand
Reusability	Cannot reuse code	Function can be reused
Debugging	Difficult	Easy
Maintainability	Hard to modify	Easy to modify
Large applications	Not suitable	Suitable

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to String Reversal)



```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))'''
```

Output:



```
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/Users/Banda Prane
eth/OneDrive/Desktop/AI ASSIST/assignment-1.5.py"
Fibonacci (iterative): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Fibonacci (recursive): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST>
```

Explanation:

Both iterative and recursive approaches generate the same Fibonacci sequence correctly. The iterative approach uses a loop and runs faster with less memory usage. The recursive approach follows a mathematical definition and is easier to understand conceptually. However, recursion involves repeated function calls, which increases time and space usage. For large input values, the iterative method is more efficient and reliable. Therefore, iteration is preferred for performance, while recursion is useful for learning and clarity.