AI-ASSISTED CODING

Assignment-1.5

Name: CH.Akhila

Hall Ticket: 2303A51417

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

```
# Generate string reversal without using functions
input_string = "Hello, World!"
reversed_string = ""
for char in input_string:
    reversed_string = char + reversed_string
print("Original string:", input_string)
print("Reversed string:", reversed_string)
```

Output:

```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
ktop/Ai-Assisted Coding/Assignment-2.py"
Original string: Hello, World!
Reversed string: !dlroW ,olleH
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
```

Justification:

The given program reverses a string without using functions by iterating through each character and adding it to the beginning of a new string. This logic correctly produces the reversed output as shown. The approach is simple and easy to understand for small programs. However, the code is not reusable since the logic is written directly in the main block. Debugging and maintenance become difficult if the program grows. Using functions would improve readability, reusability, and suitability for larger applications.

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

```
# Improve readability by using descriptive variable names
original_string = "Hello, World!"
reversed_string_manual = ""
for character in original_string:
    reversed_string_manual = character + reversed_string_manual
print("Original string:", original_string)
print("Reversed string (manual):", reversed_string_manual)

# Generate string reversal using slicing
reversed_string_slicing = original_string[::-1]
print("Reversed string (slicing):", reversed_string_slicing)
```

Output:

```
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
ktop/Ai-Assisted Coding/Assignment-2.py"
Original string: Hello, World!
Reversed string (manual): !dlroW ,olleH
Reversed string (slicing): !dlroW ,olleH
PS C:\Users\chara\OneDrive\Desktop\Ai-Assisted Coding>
```

Justification:

The manual approach reverses the string using a loop by adding each character to the beginning of a new string. The slicing approach reverses the string using Python's built-in slicing [::-1], which is shorter and more readable. Both methods produce the same correct output. The manual method helps in understanding the logic behind string reversal. The slicing method is more efficient and preferred for real-world applications. Hence, slicing is best for performance, while the manual approach is useful for learning purposes.

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

```python
# Generate string reversal using a function
def reverse_string(s):
    return s[::-1]

reversed_string_function = reverse_string(original_string)
print("Reversed string (function):", reversed_string_function)

# Generate string reversal using reversed() built-in function
reversed_string_builtin = ''.join(reversed(original_string))
print("Reversed string (built-in reversed):", reversed_string_builtin)

# Comparative Analysis – Procedural vs Modular Approach

# Procedural approach (without functions)
print("\nProcedural Approach:")
print("Original string:", original_string)
print("Reversed string (manual):", reversed_string_manual)
print("Reversed string (slicing):", reversed_string_slicing)
print("Reversed string (built-in reversed):", reversed_string_builtin)
```

Output:

```
ktop/Ai-Assisted Coding/Assignment-2.py"
Reversed string (function): !dlroW ,olleH
Reversed string (built-in reversed): !dlroW ,olleH

Procedural Approach:
Original string: Hello, World!
Reversed string (manual): !dlroW ,olleH
Reversed string (slicing): !dlroW ,olleH
Reversed string (built-in reversed): !dlroW ,olleH
```

Explanation:

The function-based approach reverses the string using a reusable function, making the code clean and well-structured. The procedural approach performs string reversal directly in the main program without functions. Both approaches produce the same correct output. The function-based method is easier to reuse and maintain in larger programs. Debugging is simpler in the modular approach compared to procedural code. Hence, the function-based approach is more suitable for real-world applications, while the procedural approach is good for small 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
# 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\chara\OneDrive\Desktop\Ai-Assisted Coding> &
ktop/Ai-Assisted Coding/Assignment-2.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\chara\OneDrive\Desktop\Ai-Assisted Coding>
```

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.