LAB – 1.5

2403A51L19
Batch - 51

Problem 1 :- Factorial

Task 1 :

Prompt : AI-Generated Logic Without Modularization (Factorial without Functions)

Fact.py

```python
# Get input once
num = int(input("Enter a number: "))

print("\n" + "="*60)
print("FACTORIAL CALCULATOR - ALL 5 APPROACHES")
print("="*60 + "\n")

# Approach 1: AI-Generated Logic Without Modularization
print("#1 - AI-Generated Logic Without Modularization")
print("    (Factorial without Functions)")
if num < 0:
    print("    Factorial is not defined for negative numbers.")
elif num == 0 or num == 1:
    print(f"    The factorial of {num} is 1.")
else:
    factorial = 1
    for i in range(2, num + 1):
        factorial *= i
    print(f"    The factorial of {num} is {factorial}.")
print()
```

Output :

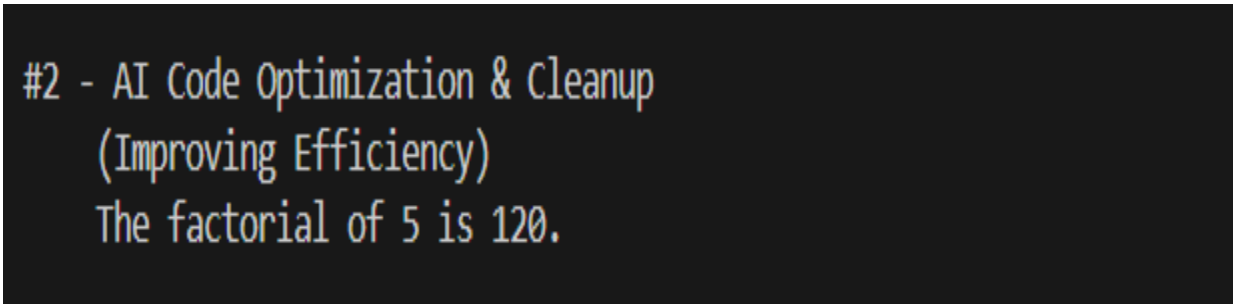
```
PS C:\Users\manik\OneDrive\Documents\AIC> python .\Fact.py
Enter a number: 5

============================================================
FACTORIAL CALCULATOR - ALL 5 APPROACHES
============================================================

#1 - AI-Generated Logic Without Modularization
    (Factorial without Functions)
    The factorial of 5 is 120.
```

Explanation : Uses a single code block without any function encapsulation
Calculates factorial by iterating from 2 to n and multiplying all values
Simple and direct approach, but lacks reusability and code organization

Task  2 : AI Code Optimization & Cleanup (Improving Efficiency)

```python
# Approach 2: AI Code Optimization & Cleanup
print("#2 - AI Code Optimization & Cleanup")
print("    (Improving Efficiency)")
if num < 0:
    print("    Factorial is not defined for negative numbers.")
elif num == 0 or num == 1:
    print(f"    The factorial of {num} is 1.")
else:
    factorial = 1
    for i in range(2, num + 1):
        factorial *= i
    print(f"    The factorial of {num} is {factorial}.")
print()
```

Output :




Explanation :
Improves the previous approach by initializing variables more efficiently
Uses a for loop instead of while loop for cleaner and more Pythonic iteration
Better organized with early condition handling for negative and edge case inputs

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

```python
# Approach 3: Modular Design Using AI Assistance
print("#3 - Modular Design Using AI Assistance")
print("    (Factorial with Functions)")
def calculate_factorial(n):
    if n < 0:
        return None
    elif n == 0 or n == 1:
        return 1
    else:
        factorial = 1
        for i in range(2, n + 1):
            factorial *= i
        return factorial

result = calculate_factorial(num)
if result is None:
    print("    Factorial is not defined for negative numbers.")
else:
    print(f"    The factorial of {num} is {result}.")
print()
```

Output :

```
#3 - Modular Design Using AI Assistance
    (Factorial with Functions)
    The factorial of 5 is 120.
```

Explanation : Encapsulates factorial logic in a reusable function for better code organization
Function returns None for invalid input, enabling proper error handling and validation
Promotes code reusability, maintainability, and easier testing of individual

Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

```python
# Approach 4: Comparative Analysis – Procedural vs Modular AI Code
print("#4 - Comparative Analysis – Procedural vs Modular AI Code")
print("    (Procedural Approach)")
if num < 0:
    print("    Factorial is not defined for negative numbers.")
elif num == 0 or num == 1:
    print(f"    The factorial of {num} is 1.")
else:
    factorial = 1
    for i in range(2, num + 1):
        factorial *= i
    print(f"    The factorial of {num} is {factorial}.")
print()
```

Output :

```
PS C:\Users\manik\OneDrive\Documents\AIC> python .\Fact.py

#4 - Comparative Analysis – Procedural vs Modular AI Code
    (Procedural Approach)
    The factorial of 5 is 120.
```

Explanation :

Demonstrates the traditional procedural programming approach without functions
Uses iterative multiplication similar to Approach 2, focusing on sequential execution
Useful for comparing procedural vs functional paradigms and understanding code structure differences

Task 5: AI-Generated Iterative vs Recursive Thinking

```python
# Approach 5: Recursive Approach Using AI
print("#5 - Recursive Approach Using AI")
print("    (Factorial with Recursion)")
def factorial_recursive(n):
    if n < 0:
        return None
    elif n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)

result = factorial_recursive(num)
if result is None:
    print("    Factorial is not defined for negative numbers.")
else:
    print(f"    The factorial of {num} is {result}.")
```

Output :

```
#5 - Recursive Approach Using AI
    (Factorial with Recursion)
    The factorial of 5 is 120.
```

Explanation :

Uses recursion where the function calls itself with decremented input until base case
Each recursive call reduces n by 1 until reaching 0 or 1, then multiplies results back
Demonstrates functional programming paradigm but may be less efficient for large numbers due to call stack overhead