SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 1

Name of Student : **Thota Varshith**
Enrollment No. : **2303A510E9**
Batch No. : **21**

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

Prompt:-
write a python code to calculate the factorial of a number without using functions**.**

Code:-

```python
# write a python code to calculate the factorial of a number without using functions.
number = int(input("Enter a number to calculate its factorial: "))
factorial = 1
for i in range(1, number + 1):
    factorial *= i
print(f"The factorial of {number} is {factorial}")

```

Output:-

```
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
PS C:\AIAC LAB>
```

Justification:-

Think of this code as a straight-line solution it solves the problem quickly and directly without any extra packaging. It’s perfect for a one-off task where you just need the answer right away, though it isn't built to be shared or reused easily in bigger projects.

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

Prompt:-
Optimize the above code by removing unnecessary variables, simplifying the loop, and improving readability.

Code:-

```python
# write a code Optimize the above code by removing unnecessary variables, simplifying the loop, and improving readability.
number = int(input("Enter a number to calculate its factorial: "))
factorial = 1
for i in range(1, number + 1):
    factorial *= i
print(f"The factorial of {number} is {factorial}")

```

Output:-

```
Enter a number to calculate its factorial: 9
The factorial of 9 is 362880
PS C:\AIAC LAB>
```

Justification:-

This version streamlines the logic by stripping away clutter and focusing on the most direct way to get the result. It's essentially a cleaner, more grown-up version of the previous code that's easier for a teammate to read and maintain.

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

Prompt:-
write a python code on factorial with function

Code:-

```python
# write a python code on factorial with function
def factorial(n):
    """Calculate the factorial of a number n."""
    if n < 0:
        return "Factorial is not defined for negative numbers."
    elif n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result
# Example usage
number = 5
print(f"The factorial of {number} is {factorial(number)}")
```

Output:-

```
The factorial of 5 is 120
PS C:\AIAC LAB>
```

Justification:-

This version wraps the logic into a tidy, reusable "tool" called a function, making the code much more professional and organized. By separating the calculation from the display, you can now use this same logic anywhere else in your project without having to rewrite it from scratch.

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

Prompt:-
Write a program that compares an AI-written script that's all in one block and a version that uses clear, well- named functions, explaining which is better for understanding, reuse, debugging, big projects, and avoiding over- reliance on AI, and show it in a small table.

Code:-

task4.py > ...

```python
#Write a program that compares an AI-written script that's all in one blo
def calculate_factorial(number):
    if number == 0:
        return 1
    else:
        return number * calculate_factorial(number - 1)

number = int(input("Enter a number to calculate its factorial: "))
result = calculate_factorial(number)
print(f"The factorial of {number} is {result}")
```

Output:-

```
Enter a number to calculate its factorial: 6
The factorial of 6 is 720
PS C:\AIAC LAB>
```

Justification:-

This version uses a recursive function to elegantly handle the logic, making the code much easier to read and test compared to a single long script. It's built for modern teamwork, where clear, reusable "blocks" of code are essential for building and debugging large, professional projects.

Task 5: AI-Generated Iterative vs Recursive Thinking

Prompt:-
write a program on create iterative version logic vs recursive version logic of factorial and explain their flow and compare readability, stack usage, performance, and when recursion should be avoided**.**

Code:-

task5.py > ...

```python
# write a program on create iterative version logic

def factorial_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

def factorial_recursive(n):
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)

print("Iterative version:")
print(factorial_iterative(5))

print("Recursive version:")
print(factorial_recursive(5))
```

Output:-

```
Iterative version:
120
Recursive version:
120
PS C:\AIAC LAB>
```

Justification:-

This comparison highlights the two main ways computers "think": the iterative version uses a simple, efficient loop for direct calculation, while the recursive version breaks the problem down into smaller, repeated steps. While recursion looks more elegant and mathematical, the iterative approach is often safer for very large numbers because it doesn't risk overloading the computer's memory stack.