Lab assignment(2403A52399):

# Prompt1: generate a Python program that calculates the factorial of a number without defining any functions

Code: def factorial\_iterative(n):

    result = 1

    for i in range(1, n + 1):

        result \*= i

    return result

# output:

Enter a number: 5

Sum of factorials up to 5 is: 153

Prompt2: generate a Python program that demonstrate how previous code’s efficiency can be Improved

Code: from functools import lru\_cache

@lru\_cache(maxsize=None)

def factorial\_memoized(n):

if n == 0 or n == 1:

return 1

return n \* factorial\_memoized(n - 1)

print(factorial\_memoized(5))

print(factorial\_memoized(10))

print(factorial\_memoized(20))

# output:

120

3628800

2432902008176640000

Prompt3: generate a Python program that calculates the factorial of a number using a user-defined function.

# Code:

def factorial(n):

result = 1

for i in range(1, n + 1):

result \*= i

return result

# Main program

num = int(input("Enter a number: "))

if num < 0:

print("Factorial is not defined for negative numbers.")

else:

print(f"The factorial of {num} is {factorial(num)}")

# output:

Enter a number: 5

The factorial of 5 is 120

# Prompt4:Differentiate between the Copilot-generated factorial program with functions and without functions in terms of logic, reusability, and execution.

# Output:

| **Aspect** |  |  |  |  | **With Functions** | **Without Functions** |
| --- | --- | --- | --- | --- | --- | --- |
| **Logic** |  |  |  |  | Factorial calculation is separated into a function factorial(n) → clean and modular. | Entire factorial logic is inside main program → less structured. |
| **Reusability** |  |  |  |  | Function can be reused anywhere (e.g., factorial in multiple parts of the program or imported into other files). | Code cannot be reused directly — would need to rewrite factorial logic each time. |
| **Execution** |  |  |  |  | Slightly more overhead because of function calls, but negligible. Program is clean, easy to debug. | Direct execution is a bit faster (no function call overhead), but difference is tiny for small programs. |
| **Readability** |  |  |  |  | More readable and maintainable — factorial logic is separate. | Less readable if program grows larger — everything is in one place. |
| **Best Use Case** |  |  |  |  | Larger programs, when factorial is needed multiple times. | Very small/one-time scripts. |

# Prompt5: generate both iterative and recursive versions of the factorial function in python

Code: def factorial\_iterative(n):

    result = 1

    for i in range(1, n + 1):

        result \*= i

    return result

def factorial\_recursive(n):

    if n == 0 or n == 1:

        return 1

    else:

        return n \* factorial\_recursive(n - 1)

# output:

Enter a number: 5

Iterative factorial of 5 is: 120

Recursive factorial of 5 is: 120