***Benefits of Modularity***

*Modularity refers to the practice of dividing a program into smaller, self-contained units called functions. Each function has a specific task and interacts with other functions through well-defined interfaces (parameters and return values). Here's how it helps:*

***Reusability:****Modular functions can be reused in different parts of the program or even in other programs, reducing code duplication and development time.*

***Organization****: Code becomes more organized and easier to understand by breaking down complex tasks into smaller, manageable functions.*

***Maintainability:****Changes or bug fixes can be isolated within a specific function, making maintenance easier and reducing the risk of unintended side effects.*

***Function Design***

*1. factorial(n)*

*This function calculates the factorial of a non-negative integer n. The factorial of a number is the product of all positive integers less than or equal to that number.*

***Pseudocode:***

***function factorial(n)***

***if n < 0 then***

***return "Error: Factorial is not defined for negative numbers"***

***else if n == 0 then***

***return 1***

***else fact := 1***

***for i := 1 to n do***

***fact := fact \* i***

***end for return fact***

***end if***

***end function***

***Explanation:***

*The function checks for invalid input (negative numbers) and returns an error message.*

*For valid input (0 or positive number), it initializes a variable fact to 1 and iterates through a loop from 1 to n, multiplying fact by the current loop variable i in each iteration.*

*Finally, the function returns the calculated factorial.*

1. ***fibonacci(n)***

*This function calculates the nth Fibonacci number. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding numbers, starting from 0 and 1.*

**Pseudocode:**

***function fibonacci(n)***

***if n < 0 then***

***return "Error: Fibonacci sequence is not defined for negative numbers"***

***else if n <= 1 then***

***return n***

***else***

***return fibonacci(n-1) + fibonacci(n-2)***

***end if***

***end function***

*Explanation:*

*The function checks for invalid input (negative numbers) and returns an error message.*

*For valid non-negative input, it uses recursion. If n is 0 or 1, it directly returns n as these are the base cases in the Fibonacci sequence.*

*For other values of n, the function recursively calls itself twice: once with n-1 and again with n-2. This breaks down the calculation of the nth term into the sum of the (n-1)th and (n-2)th terms.*

***Explanation:***

*The function checks for invalid input (negative numbers) and returns an error message.*

*For valid non-negative input, it uses recursion. If n is 0 or 1, it directly returns n as these are the base cases in the Fibonacci sequence.*

*For other values of n, the function recursively calls itself twice: once with n-1 and again with n-2. This breaks down the calculation of the nth term into the sum of the (n-1)th and (n-2)th terms.*

***Conclusion :***

*By designing these functions as modular units, we achieve reusability and improve code organization. The factorial function can be used in various contexts where factorial calculations are needed, and the fibonacci function can be used in programs that require Fibonacci number generation. Modularity promotes well-structured and maintainable code.*