Assignment 3:

**Function Design and Modularization** –

Create a document that describes the  
design of two modular functions: one that returns the factorial of a number,  
and another that calculates the nth Fibonacci number. Include pseudocode and  
a brief explanation of how modularity in programming helps with code reuse  
and organization.

**Factorial Function: -**

The factorial of a number ‘n’ is the product of all the positive integers less than or equal to number itself.

Design: -

* + Check if the input number n is less than 0. If so, raise an error or return None since factorial is not defined for negative numbers.
  + Initialize a variable result to 1.
  + Iterate from 1 to n (inclusive) and multiply each number with result.
  + Finally, return result which will contain the factorial of n.

**Pseudo Code**

function factorial(n):

if n is 0:

return 1

else:

return n \* factorial(n-1)

**Fibonacci Function: -**

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, starting from 0 and 1. The nth Fibonacci number, denoted as F(n), can be calculated recursively or iteratively.

**Design: -**

* + If n is 0 or 1, return n itself as the Fibonacci sequence starts with 0 and 1.
  + Initialize variables a and b to 0 and 1 respectively, as these are the first two numbers in the Fibonacci sequence.
  + Loop n-1 times and in each iteration, update a and b to b and a+b respectively.
  + After the loop, return the value of b which will be the nth Fibonacci number.

**Pseudo code: -**

function fibonacci(n):

if n <= 1:

return n

else:

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

A brief explanation of how modularity in programming helps with code reuse  
and organization.

Modularity in programming refers to the practice of breaking down a program into smaller, self-contained modules or functions. Each module or function performs a specific task, and these tasks are often related to each other within the context of the larger program.

**Code Reusability:**

Modular design allows functions or modules to be reused in different parts of a program or in different programs altogether. For instance, the factorial and Fibonacci functions designed here can be reused in various mathematical calculations without rewriting the code.

**Organized Codebase:**

By separating functionalities into modules, the codebase becomes more organized and easier to understand. Each module focuses on a specific task, making it simpler to maintain and debug.

**Scalability**: Modularity facilitates scalability as new features can be added by creating new modules or extending existing ones without affecting the entire codebase. This makes the code more adaptable to changes and future requirements.

**Encapsulation:**

Modular code promotes encapsulation, which is the practice of hiding internal implementation details and exposing only necessary interfaces.

Modules provide well-defined interfaces for interaction, allowing other parts of the program to use their functionality without needing to know how they are implemented internally.

**Testing and Debugging:**

Modular code facilitates unit testing, as individual modules can be tested independently of the rest of the program.

When bugs occur, modular code makes it easier to identify the specific module responsible for the issue, reducing the time and effort required for debugging.

Overall, modularity promotes code reuse, improves organization, enhances maintainability, and facilitates collaboration in software development projects. By breaking down complex systems into smaller, manageable components, developers can build more robust, scalable, and maintainable software solutions.

**Collaboration**: Modular code allows multiple developers to work on different modules simultaneously, promoting collaboration and speeding up the development process.