Memoization

Memoization is a programming technique used to optimize the performance of functions by caching the results of expensive function calls and returning the cached result when the same inputs occur again.

Here's how memoization typically works:

1. **Function Invocation**: When a function is called with certain inputs, the memoized function checks whether it has already computed the result for those inputs.
2. **Cache Lookup**: If the result for the given inputs is already stored in a cache, the memoized function returns the cached result instead of recomputing it.
3. **Computing and Storing Results**: If the result is not found in the cache, the function computes the result for the given inputs, stores it in the cache, and then returns the result.
4. **Reuse of Cached Results**: Subsequent calls to the function with the same inputs can now directly retrieve the result from the cache.

Ex:

// Function to calculate the nth Fibonacci number

function fibonacci(n) {

if (n <= 1) {

return n;

}

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

}

// Memoized version of the Fibonacci function

function memoizedFibonacci() {

let cache = {}; // Cache to store results

return function fib(n) {

if (n in cache) {

return cache[n]; // Return cached result if available

}

if (n <= 1) {

return n;

}

// Calculate Fibonacci number recursively

const result = fib(n - 1) + fib(n - 2);

// Cache the result for future use

cache[n] = result;

return result;

};

}

// Usage of memoized Fibonacci function

const memoizedFib = memoizedFibonacci();

console.log(memoizedFib(5)); // Output: 5

console.log(memoizedFib(10)); // Output: 55

console.log(memoizedFib(15)); // Output: 610