Memoization in JavaScript

Memoization is an optimization technique used to store the results of expensive function calls and return the cached result when the same inputs occur again. It is a fundamental concept in **Dynamic Programming**.

What is Caching?

Caching is a way to store values so that they can be reused later instead of recomputing them.

Memoization Explained:

Memoization involves storing function results in a cache and returning the cached value when the function is called again with the same arguments.

Example: Function Without Memoization

```
function add80(n) {
  return n + 80;
}

console.log(add80(89)); // 169
console.log(add80(89)); // 169 (Recomputes the value every time)
```

Example: Memoized Function

```
let cache = {};
const memoize = (n) => {
  if (n in cache) {
    return cache[n]; // Return cached result
  } else {
    console.log('Takes Time');
    cache[n] = n + 80; // Store result in cache
    return cache[n];
  }
};

console.log('1', memoize(5)); // Takes Time, Output: 85
console.log('2', memoize(5)); // Returns from cache, Output: 85
```

Benefits of Memoization:

- Reduces redundant calculations.
- Improves performance in recursive and computationally heavy functions.
- Optimizes dynamic programming problems like Fibonacci, Factorial, etc.

Memoization is commonly used in problems involving **recursion, API calls, and complex computations** to enhance efficiency.