Armen Melkumyan • 1st
Technical / Solutions Architect
4mo •

From JavaScript Technical Interviews:

Using Closures to Optimize Fibonacci Calculations with Caching

When you’re asked to write a function that calculates Fibonacci numbers, the real challenge isn’t just getting the sequence right it’s about how you optimize your solution. Let’s take it up a caching with closures.

```
function createFibonacciCache() {
 const cache = {};

 return function fibonacci(n) {
 if (n in cache) {
 console.log(`Fetching from cache: ${n}`);
 return cache[n];
 }

 console.log(`Calculating: ${n}`);
 if (n <= 1) return n;

 cache[n] = fibonacci(n - 1) + fibonacci(n - 2);
 return cache[n];
 };
}

const fib = createFibonacciCache();
console.log(fib(6)); // Calculates and caches
console.log(fib(6)); // Fetches from cache
console.log(fib(8)); // Fetches from cache for 1-6, calculates 7 and 8, caches them
```

console.log(fib(8)); // Fetches from cache

As an interviewer, I'm not just asking you to calculate Fibonacci numbers—I'm looking for a solution that:

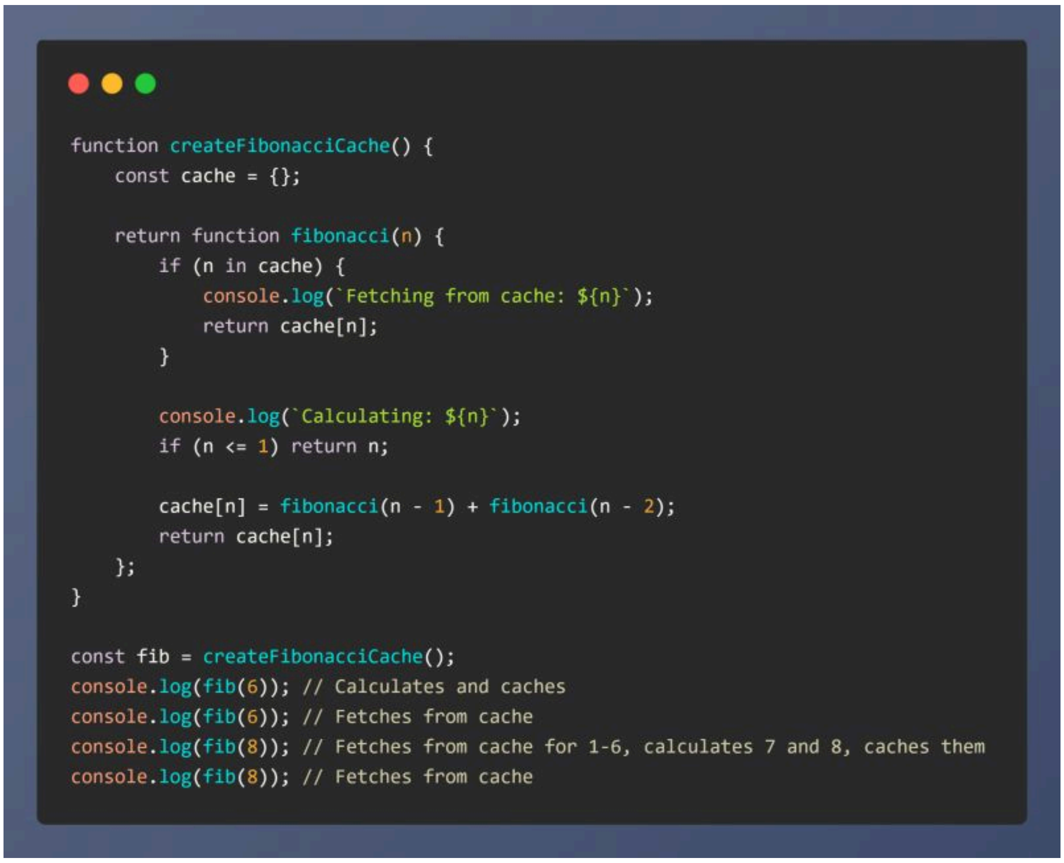
Optimizes Performance: Using caching avoids redundant calculations. For example, when you call fib(8), it reuses cached results for fib(1) through fib(6) from the previous call, calculating only fib(7) and fib(8).

Demonstrates Mastery of Closures: The cache is neatly encapsulated within the closure, making the function self-contained and preventing global variable clutter.

Balances Trade-offs: Efficient caching improves performance but also requires memory. Knowing when and where to apply caching is just as important as implementing it.

Yeah you are right when n < 0 make sense to throw an error, but it's a question to interviewer))))

#JavaScript #TechInterview



```javascript
function createFibonacciCache() {
    const cache = {};

    return function fibonacci(n) {
        if (n in cache) {
            console.log(`Fetching from cache: ${n}`);
            return cache[n];
        }

        console.log(`Calculating: ${n}`);
        if (n <= 1) return n;

        cache[n] = fibonacci(n - 1) + fibonacci(n - 2);
        return cache[n];
    };
}

const fib = createFibonacciCache();
console.log(fib(6)); // Calculates and caches
console.log(fib(6)); // Fetches from cache
console.log(fib(8)); // Fetches from cache for 1-6, calculates 7 and 8, caches them
console.log(fib(8)); // Fetches from cache
```

119 7 comments 5 reposts