Sheryians Coding School

Live Cohort

Notes Day 24

JavaScript Advanced HOFs, Callbacks, and Closures

Understanding Higher-Order Functions (HOFs)

A Higher-Order Function (HOF) is a function that either takes another function as an argument or returns a function. These are widely used in JavaScript to enhance code reusability and maintainability.

1 Delayed Execution Using Callback (HOF + Callback)

Concept ; -

A function can accept another function as an argument (callback) and execute it after a delay using setTimeout.

Implementation:

```
function delayedExecution(callback) {
    setTimeout(callback, 3000);
    // Calls the callback function after 3 seconds
}

// Example usage
delayedExecution(() =>
console.log("Executed after 3 seconds"));
```

Explanation:

- delayedExecution accepts a function callback.
- setTimeout is used to delay the execution of callback for 3 seconds.
- When called, it logs a message to the console after 3 seconds.

JavaScript Advanced HOFs, Callbacks, and Closures

Implementing Custom Function (HOF)

Concept:

The .map() method is used to transform an array by applying a callback function to each element. We can create our own version of .map().

Implementation:

```
function customMap(array, callback) {
    let result = [];
    for (let i = 0; i < array.length; i++) {
        result.push(callback(array[i], i, array));
        // Apply callback to each element
    }
    return result;
}

// Example usage
console.log(customMap([1, 2, 3], num => num * 2));
// Output: [2, 4, 6]
```

Explanation:

- customMap takes an array and a callback function.
- Iterates over the array, applies the callback function to each element, and stores the result.
- Works similarly to Array.prototype.map but is implemented manually.

JavaScript Advanced HOFs, Callbacks, and Closures

Closures: Creating a Counter Function

Concept:

A closure is a function that retains access to variables from its outer scope even after the outer function has finished execution.

Implementation:

```
function createCounter() {
    let count = 0;
    return function() { // Closure retains access to `count`
        return ++count;
    };
}

// Example usage
const counter = createCounter();
console.log(counter()); // Output: 1
console.log(counter()); // Output: 2
console.log(counter()); // Output: 3
```

Explanation:

- createCounter defines a variable count and returns a function.
- The inner function forms a closure, keeping count in memory.
- Each time the inner function is called, count is incremented and returned.

JavaScript Advanced HOFs, Callbacks, and Closures

Limiting Function Calls (Closure + HOF)

Concept:

A function should only be executed a limited number of times. This can be achieved using closures.

Implementation:

```
function limit(fn, limit) {
    let calledTimes = 0;
    return function () {
        if (calledTimes < limit) {
            calledTimes++;
            fn();
        }
    };
}

// Example usage
let fn = limit(() => console.log("hello"), 3);
fn(); // "hello"
fn(); // "hello"
fn(); // "hello"
fn(); // (No output, limit reached)
```

Explanation:

- limit takes a function fn and a limit value.
- It tracks the number of times the function has been called.
- Once the limit is reached, further calls do nothing.

JavaScript Advanced HOFs, Callbacks, and Closures

Conclusion

- Higher-Order Functions (HOFs) enable cleaner and reusable code by accepting functions as arguments.
- Callbacks allow asynchronous behavior and function execution control.
- Closures help retain variable states and create private data.
- These concepts are crucial in modern JavaScript development, especially in functional programming and asynchronous operations.