Sheryians Coding School

Live Cohort

Notes Day 25

JavaScript Advanced HOFs, Callbacks, and Closures

1. Repeating a Function at Intervals (Using Callbacks)

Concept:

A callback function is a function passed as an argument to another function. Using setInterval, we can execute a callback function repeatedly at specified intervals.

Implementation:

```
function repeatFunction(callback, interval) {
    setInterval(callback, interval * 1000);
}

// Example usage
repeatFunction(() => console.log("Repeating..."), 2);
// Logs "Repeating..." every 2 seconds
```

Explanation:

- The function repeatFunction takes two parameters:
  - callback: The function to execute.
  - interval: The time in seconds between executions.
- setInterval is used to call callback repeatedly after every interval seconds.
- In the example, the function logs "Repeating..." every 2 seconds.

JavaScript Advanced HOFs, Callbacks, and Closures

2. Creating a Function with a Preset Greeting (Using Closures)

Concept:

A closure allows a function to "remember" variables from its outer scope even after the outer function has finished executing.

Implementation:

```
function greetUser(greeting) {
    return function (name) {
        return `${greeting}, ${name}!`;
    };
}

// Example usage
const greetHello = greetUser("Hello");
console.log(greetHello("Alice")); // "Hello, Alice!"
console.log(greetHello("Bob"));   // "Hello, Bob!"
```

Explanation:

- greetUser is a higher-order function that returns another function.
- The returned function remembers the greeting value from greetUser (closure property).
- When greetHello("Alice") is called, it uses the stored greeting ("Hello") and returns "Hello, Alice!".
- This technique is useful for creating pre-configured functions.

JavaScript Advanced HOFs, Callbacks, and Closures

3. Executing a Function Only Once (Using HOFs + Closures)

Concept:

A function should only be executed once, no matter how many times it is called.

Implementation:

```
function once(fn) {
    let executed = false;
    return function (...args) {
        if (!executed) {
            executed = true;
            return fn(...args);
        }
    };
}

// Example usage
const init = once(() => console.log("Initialized!"));
init(); // "Initialized!"
init(); // (No output)
```

Explanation:

- The once function wraps another function (fn) and ensures it only executes once.
- The variable executed keeps track of whether the function has already been called.
- The first call executes fn, but all subsequent calls do nothing.
- Useful for initialization functions.

JavaScript Advanced HOFs, Callbacks, and Closures

4. Throttling a Function (Using HOFs + Closures)

Concept:

Throttling ensures that a function is not executed more than once within a specified time period. This is useful in event listeners to improve performance.

Implementation:

```
function throttle(fn, delay) {
    let lastCall = 0;
    return function (...args) {
        let now = Date.now();
        if (now - lastCall >= delay) {
            lastCall = now;
            fn(...args);
        }
    };
}

// Example usage
const throttledFn = throttle(() =>
console.log("Throttled Execution"), 2000);
throttledFn();
throttledFn();
throttledFn();
// Only executes the first call,
others are ignored until 2 sec passes
```

JavaScript Advanced HOFs, Callbacks, and Closures

Explanation:

- throttle ensures that fn runs only once in every delay milliseconds.
- The lastCall variable stores the last execution time.
- If delay hasn't passed, additional calls are ignored.
- Useful in scenarios like scrolling events or resizing windows.