More JS!

Higher-Order Functions, Callbacks, and Array Methods

Higher-Order Functions

Higher-Order Functions

- A Higher-Order Function is a function that operates on other functions, either by taking them as arguments or returning them.

- In JavaScript, this is possible because functions are treated as **First-Class Functions**, meaning they can be assigned to variables, passed as arguments to other functions, and returned as values from other functions.

Higher Order Functions Continued

```
function outerFunc(cb) {
return cb();
}
```

In the above example, the outerFunc function is a Higher-Order Function. It takes in a callback function (cb) as an argument and then invokes that callback function by calling cb(). The outerFunc function essentially delegates some of its functionality to the callback function it receives.

Callback Functions

Callback Functions

- A callback function is a function that is passed as an argument to another function, and it is then invoked or executed inside the outer function to perform a specific routine or action.

- Callback functions can be declared functions, function expressions, or even anonymous functions, depending on the specific requirements and context in which they are used.

Callback Functions Continued

```
function outerFunc(cb) {
return cb();
}
```

In the previous example, the callback function is represented by the parameter cb in the outerFunc function. It is passed as an argument into the outerFunc function and then invoked by calling cb(). The purpose of the callback function is to be executed at a specific point inside the outerFunc function.

Array Methods

Array Methods

To enhance our understanding of higher-order functions and callbacks, let's explore some commonly used iterative array methods provided by JavaScript, including forEach, map, and reduce.

array.forEach()

- The forEach method is used to iterate over each element in an array. It takes a callback function as an argument and executes it once for each element in the array.

- You can also optionally pass an index argument to the callback function, allowing you to access the index of the current element within the iteration.

- It's important to note that the forEach function **does not modify the original array and does not return any value**. Its purpose is to perform some action or operation on each element of the array.

array.map()

- The map method is used to transform each element of an array by applying a callback function to it. It takes a callback function as an argument and creates a new array by performing the callback on each element of the original array.

- Similar to forEach, map can also optionally take an index argument, allowing you to access the index of the current element within the iteration.

- Importantly, the **map function does not modify the original array**. Instead, it returns a new array of the same length as the original array, containing the results of applying the callback function to each element.

array.reduce()

- The reduce method is used to reduce an array into a single value by applying a callback function to each element and accumulating the result. It takes a callback function and an iterator as arguments.

- The callback function used with reduce should have at least two parameters, commonly referred to as "previous" and "next." These parameters represent the accumulated value from the previous iteration and the current element being processed, respectively. The callback function should return a value that will be used as the accumulated value for the next iteration.

- It's important to note that **reduce does not modify the original array**. Instead, it returns a new value based on the result of the callback function applied to each element of the array.