Higher-order functions are a powerful concept in JavaScript, as well as in many other programming languages. A higher-order function is a function that either:

1. Takes one or more functions as arguments, or
2. Returns a function as its result.

These functions are essential for functional programming and enable more flexible, modular, and reusable code.

**Common Examples of Higher-Order Functions**

Let's go through some common higher-order functions in JavaScript.

**1. .map()**

The .map() function takes a function as an argument and applies it to each item in an array, returning a new array with the transformed values.

const numbers = [1, 2, 3, 4, 5];

const doubled = numbers.map((num) => num \* 2);

console.log(doubled); // [2, 4, 6, 8, 10]

**2. .filter()**

The .filter() function takes a predicate function as an argument and returns a new array with only the elements that satisfy the condition.

const numbers = [1, 2, 3, 4, 5];

const evens = numbers.filter((num) => num % 2 === 0);

console.log(evens); // [2, 4]

**3. .reduce()**

The .reduce() function reduces an array to a single value by applying a reducer function to each element in the array, carrying an accumulator with each iteration.

const numbers = [1, 2, 3, 4, 5];

const sum = numbers.reduce((accumulator, currentValue) => accumulator + currentValue, 0);

console.log(sum); // 15

**4. .forEach()**

The .forEach() function executes a provided function once for each array element.

const numbers = [1, 2, 3, 4, 5];

numbers.forEach((num) => console.log(num));

// Output: 1 2 3 4 5

**5. setTimeout and setInterval**

These are higher-order functions as they take other functions (callbacks) as arguments.

setTimeout(() => {

console.log("This runs after 1 second");

}, 1000);

**6. Custom Higher-Order Functions**

You can create your own higher-order functions, like a function that returns another function.

function createMultiplier(multiplier) {

return function (num) {

return num \* multiplier;

};

}

const double = createMultiplier(2);

console.log(double(5)); // 10

**Benefits of Higher-Order Functions**

* **Code Reusability**: You can pass different functions to achieve different behaviors.
* **Cleaner Code**: Helps reduce redundancy and increase readability.
* **Function Composition**: Allows chaining and combining functions in powerful ways.

**Summary**

Higher-order functions are central to JavaScript's functional programming capabilities, enabling more concise and expressive code.