Higher-order functions

Sure, let's dive deep into **Higher-Order Functions** in JavaScript.

🔁 What Are Higher-Order Functions?

A **higher-order function (HOF)** is a function that **does at least one** of the following:

1. Takes one or more functions as arguments (callbacks).
2. Returns a function as its result.

In JavaScript, functions are first-class citizens, meaning:

- You can assign them to variables,
- Pass them as arguments,
- Return them from other functions.

This makes it possible to create **higher-order functions**.

🔧 Syntax Example

```
function greet(name) {
  return `Hello, ${name}!`;
}

function higherOrderFunction(callback) {
  const result = callback("Abhi");
  console.log(result);
}

higherOrderFunction(greet);
// Output: Hello, Abhi!
```

In the above code:

- `greet` is a regular function.
- `higherOrderFunction` is a **higher-order function** because it takes another function (`greet`) as an argument.

📥 Passing Functions as Arguments (Callbacks)

```
function calculate(a, b, operation) {
  return operation(a, b);
}

function add(x, y) {
  return x + y;
}

function subtract(x, y) {
  return x - y;
}

console.log(calculate(10, 5, add));      // 15
console.log(calculate(10, 5, subtract)); // 5
```

Here, `calculate` is a higher-order function because it accepts `operation` as a function.

📤 Returning Functions

```
function multiplier(factor) {
  return function (number) {
    return number * factor;
  };
}

const double = multiplier(2);
console.log(double(5)); // 10
```

`multiplier` is a higher-order function because it returns another function.

📚 Real-World Examples in JavaScript

1. Array Methods – `.map()`, `.filter()`, `.reduce()` are all higher-order functions.

`map()`

```
const numbers = [1, 2, 3];
const squared = numbers.map(num => num * num);
console.log(squared); // [1, 4, 9]
```

`filter()`

```
const even = numbers.filter(num => num % 2 === 0);
console.log(even); // [2]
```

`reduce()`

```
const sum = numbers.reduce((acc, curr) => acc + curr, 0);
console.log(sum); // 6
```

Each of these methods:

- Takes a function as an argument
- Is therefore a higher-order function

🎯 Why Use Higher-Order Functions?

- **Modularity**: They help you separate concerns.
- **Reusability**: Common patterns can be abstracted and reused.
- **Composability**: You can combine small functions into bigger ones.

- **Functional Programming**: Enables a more declarative and concise coding style.

⚠️ Common Pitfall

Be careful not to confuse **invoking** a function and **passing** it.

```
// Incorrect
calculate(10, 5, add()); // This calls `add` immediately and passes the result

// Correct
calculate(10, 5, add);   // This passes the function `add` itself
```

✅ Summary

Feature	Explanation
Accepts function as argument	✅
Returns function	✅
Examples	`map`, `filter`, `reduce`, custom functions
Use cases	Callbacks, middleware, functional utilities

Let me know if you'd like a visual diagram or to move on to the next topic, like **callback hell** or **closures**.