Higher Order Functions (Functions That Control Functions)

Subtitle: Learn how functions become data, how they accept or return other functions, and how modern JavaScript is built on this concept.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

🧠 Introduction

So far, functions were treated as tools that:

- take input
- run logic
- return output

Higher Order Functions (HOF) change that model completely.

In real JavaScript systems, functions are not just logic blocks — they are **controllable units**.

A function can accept another function OR return another function.

This is the foundation of:

- React patterns
- async programming
- array methods (map, filter)
- middleware systems
- event handling systems

⚙️ Step 1: What is a Higher Order Function?

A function is called Higher Order if it does at least one of these:

- takes a function as an argument
- returns a function

Example 1: Function as Argument

```
function greet(name) {

  return "Hello " + name;

}



function processUser(fn) {
```

```
  console.log(fn("Ali"));
}

processUser(greet);
```

Key Idea:

We are not passing data — we are passing behavior.

🧠 Step 2: Why This is Powerful

Normal functions:

- handle data

Higher order functions:

- control logic flow

Mental Model:

Data → Functions → Behavior Control → Dynamic Logic

⚙️ Step 3: Function Returning Function

```
function multiplier(x) {
  return function (y) {
    return x * y;
  };
}

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

Output:

10

What happened:

- outer function fixed x
- inner function uses y
- closure + HOF combined

🔄 Step 4: Real JavaScript Uses (Critical Section)

1. Array Methods (Core HOF usage)

map()

```
const nums = [1, 2, 3];


const doubled = nums.map(function (n) {
  return n * 2;
});
```

filter()

```
const nums = [1, 2, 3, 4];


const evens = nums.filter(function (n) {
  return n % 2 === 0;
});
```

reduce()

```
const nums = [1, 2, 3];


const sum = nums.reduce(function (acc, n) {
  return acc + n;
}, 0);
```

Insight:

These are all higher order functions.

🌍 Step 5: Real-World System Design

1. Middleware Systems (Backend)

```
function logger(req, next) {
  console.log("Request received");
  next();
}
```

2. Event Systems (Frontend)

```
button.addEventListener("click", function () {
  console.log("Clicked");
});
```

3. React-like Logic Thinking

- components = functions
- props = data
- callbacks = behavior injection

🧠 Step 6: Why HOF Exists (Engineering Reason)

HOF solves 3 problems:

1. Code Reusability

Same function works with different behaviors

2. Abstraction

Logic is separated from execution

3. Flexibility

Behavior can be changed at runtime

⚠️ Step 7: Common Mistakes

❌ 1. Thinking functions are only for data

They also control behavior

❌ 2. Confusing callback with HOF

Callback = function passed
HOF = function that uses/returns function

❌ 3. Not understanding execution timing

HOF enables delayed execution patterns

❌ 4. Overcomplicating simple logic

Not every function needs HOF

🧪 Step 8: Mini Exercises

Exercise 1

Create a function that takes another function and executes it twice.

Exercise 2

Create a function that returns a function to subtract numbers.

Exercise 3

Implement a custom map function.

🧠 Step 9: Mini Quiz

1. What is a Higher Order Function?
2. Why are functions passed as arguments?
3. What is the role of closures in HOF?
4. Name 3 built-in HOFs in JavaScript

🚀 Step 10: Thinking Upgrade

If you understand this properly:

- functions are not static
- they are controllable logic units
- JavaScript becomes **behavior-driven system**

📌 Step 11: Summary

- Higher Order Functions accept or return functions

- They enable functional programming patterns
- Used heavily in arrays, events, APIs
- Combine with closures for powerful systems
- Foundation of modern JavaScript frameworks