Function composition

Sure, let's dive deep into **Function Composition** — a powerful technique in functional programming and JavaScript that allows for clean, readable, and reusable code.

🧩 What is Function Composition?

Function composition is the process of combining **two or more functions** to produce a new function, where the output of one function becomes the input of the next.

> In simple terms:
>
> 🎯 `compose(f, g)(x)` = `f(g(x))`

It's like chaining functions together in a logical **pipeline**.

🧠 Why Use It?

- Makes code **cleaner**, **modular**, and **declarative**
- Encourages **reusability** of small functions
- Helps to **abstract complex logic**

📦 Basic Example:

```
const double = x => x * 2;
const square = x => x * x;

const composed = x => square(double(x));

console.log(composed(3)); // double(3) = 6 → square(6) = 36
```

We manually composed `square` and `double` above.

⚙️ Manual `compose` Function

Let's create a generic `compose` function (right to left):

```
const compose = (f, g) => x => f(g(x));

const double = x => x * 2;
const square = x => x * x;

const composed = compose(square, double);

console.log(composed(5)); // 100 → square(double(5)) = square(10) = 100
```

🔁 Compose Multiple Functions

To handle **more than two** functions:

```
const compose = (...fns) => x => fns.reduceRight((acc, fn) => fn(acc), x);

const add = x => x + 1;
const multiply = x => x * 3;
const subtract = x => x - 2;

const pipeline = compose(subtract, multiply, add);

console.log(pipeline(4));
// add(4) = 5 → multiply(5) = 15 → subtract(15) = 13
```

> compose(subtract, multiply, add) is the same as subtract(multiply(add(x)))

🔄 vs. `pipe` (left to right)

Some libraries like **Ramda** or **RxJS** use `pipe`, which runs **left to right**:

```
const pipe = (...fns) => x => fns.reduce((acc, fn) => fn(acc), x);

const pipeline = pipe(add, multiply, subtract);
console.log(pipeline(4));
// add(4) = 5 → multiply(5) = 15 → subtract(15) = 13
```

`compose(f, g)(x)`	= `f(g(x))` (right to left)
`pipe(f, g)(x)`	= `g(f(x))` (left to right)

🧰 Real-World Use Case

✅ Clean input before submitting to backend:

```
const trim = str => str.trim();
const toLowerCase = str => str.toLowerCase();
const sanitize = str => str.replace(/[^\w\s]/gi, '');

const cleanInput = compose(sanitize, toLowerCase, trim);

console.log(cleanInput("  Hello@World! ")); // "helloworld"
```

🧠 Summary

Concept	Description
Function Composition	Combining multiple functions to form a new one
`compose`	Applies functions from **right to left**
`pipe`	Applies functions from **left to right**
Benefit	Makes code modular, readable, and reusable

🔧 Bonus: Use Lodash's `_.flow` or `_.flowRight`

Lodash offers built-in helpers for composition:

```
import { flowRight } from 'lodash';

const cleanInput = flowRight(sanitize, toLowerCase, trim);
```

Would you like a mini project that applies function composition (like a form cleaner or math pipeline), or should we do a **full review** of all advanced function topics next?