Functional array methods (map, filter, reduce, forEach, find, some, every)

Here's a detailed explanation of **functional array methods** in JavaScript:

🔧 Functional Array Methods

Functional methods treat arrays as **data to transform**, not mutate. These include:

Method	Purpose	Returns
`map()`	Transform each element	New array
`filter()`	Select elements by condition	New array
`reduce()`	Accumulate values to single result	Any type
`forEach()`	Loop through each element	`undefined`
`find()`	Find **first** matching element	Single value
`some()`	At least one element matches?	Boolean
`every()`	All elements match?	Boolean

1️⃣ `map()`

Returns a **new array** by applying a function to each element.

```
const numbers = [1, 2, 3];
const squared = numbers.map(n => n * n);

console.log(squared); // [1, 4, 9]
```

> 🧠 Use when transforming each item in the array.

2️⃣ filter()

Returns a **new array** with elements that pass the test.

```
const numbers = [1, 2, 3, 4];
const evens = numbers.filter(n ⇒ n % 2 === 0);

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

> ✅ Use when selecting items that meet a condition.

3️⃣ reduce()

Reduces the array to a **single value**.

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce((acc, curr) ⇒ acc + curr, 0);

console.log(sum); // 10
```

Parameters:

- `acc` → Accumulator (result so far)
- `curr` → Current item
- `0` → Initial value

> 🔥 Great for totals, object grouping, or flattening arrays.

4️⃣ forEach()

Executes a function for each array element.

⚠️ It does **not return anything**.

```
const numbers = [1, 2, 3];
numbers.forEach(n => console.log(n * 2));
```

> 🔁 Use for side effects like logging or updating UI.

5️⃣ find()

Returns the **first element** that matches a condition.

```
const users = [
  { name: "Abhi", age: 22 },
  { name: "John", age: 30 }
];

const user = users.find(u => u.age > 25);

console.log(user); // { name: "John", age: 30 }
```

> 🚀 Useful when you only need the first match.

6️⃣ some()

Checks if **at least one** element passes the condition.

```
const nums = [1, 3, 5, 8];
const hasEven = nums.some(n => n % 2 === 0);

console.log(hasEven); // true
```

> ✅ Returns a boolean.

7️⃣ every()

Checks if **all elements** pass the condition.

```
const nums = [2, 4, 6];
const allEven = nums.every(n => n % 2 === 0);

console.log(allEven); // true
```

> 🔒 Often used for validation.

🧠 Summary Table

Method	Mutates?	Returns	Use For
`map()`	❌	New array	Transform each item
`filter()`	❌	New array	Select items
`reduce()`	❌	Any value	Aggregate to one result
`forEach()`	❌	`undefined`	Perform side effects
`find()`	❌	First match	Find one matching element
`some()`	❌	Boolean	At least one matches?
`every()`	❌	Boolean	All match?

💡 Tip:

These methods do **not mutate** the original array — they return new results, making your code more **predictable and functional**.

Let me know if you'd like hands-on problems or visual illustrations of how they work behind the scenes.