



Intern Name: Fatima Wajid

Submitted to: Ali Hyder

Date: 23rd July 2025

Internship Role: Web Development

Objective of the Day

To understand and implement **high-order array methods** in JavaScript — namely `map()`, `filter()`, and `reduce()` — to manipulate and transform array data effectively in modern web development.

Introduction to Array Methods

JavaScript provides many powerful methods to work with arrays. Among the most commonly used are:

- `map()` – transforms each element in an array
- `filter()` – selects elements that match a condition
- `reduce()` – combines all elements into a single value

These methods help write **clean, concise, and functional code** for manipulating lists of data.

1. map() Method

The map() method creates a **new array** by applying a function to each element of the original array. It does **not modify** the original array.

Syntax:

```
const newArray = originalArray.map(function(currentValue, index, array) {  
  // return element to be added to newArray  
});
```

Example:

```
const numbers = [1, 2, 3, 4];  
const doubled = numbers.map(num => num * 2);  
console.log(doubled); // [2, 4, 6, 8]
```

Use Cases:

- Modifying data (e.g., converting prices, transforming names)
- Generating HTML from data arrays
- Changing object properties

2. filter() Method

The filter() method returns a **new array** with elements that **pass a test** (i.e., return true in the callback function).

Syntax:

```
const filteredArray = originalArray.filter(function(currentValue, index, array) {  
  return condition;  
});
```

Example:

```
const ages = [12, 19, 21, 17, 30];  
const adults = ages.filter(age => age >= 18);  
console.log(adults); // [19, 21, 30]
```

Use Cases:

- Filtering users based on age, status, role, etc.
- Removing unwanted items from a list
- Getting search results from a data array

3. reduce() Method

The reduce() method reduces an array to a **single value** (number, string, object, etc.) by applying a function to each element and accumulating the result.

Syntax:

```
const result = array.reduce(function(accumulator, currentValue, index, array) {  
    return updatedAccumulator;  
}, initialValue);
```

Example:

```
const numbers = [1, 2, 3, 4];  
const sum = numbers.reduce((total, num) => total + num, 0);  
console.log(sum); // 10
```

Use Cases:

- Calculating totals, averages, or scores
- Grouping or counting items
- Flattening arrays or combining objects

Comparison Table

Method	Purpose	Returns	Original Array Changed?	Use Case Examples
map()	Transform each element	New array	✗ No	Change values, update formats
filter()	Select elements by condition	New array	✗ No	Remove unwanted items, search results
reduce()	Reduce to a single value	Any data type	✗ No	Sum, average, total, merge data

What I Learned Today

- I learned how to use map() to transform arrays without mutating them.
- I practiced filter() to extract elements based on specific conditions.
- I understood how reduce() works by looping through array elements and accumulating a result.
- These methods are powerful tools for writing clean, readable, and functional code when working with data.

Code Snippet Example – Chaining Methods

```
const products = [  
  { name: "Shirt", price: 1000 },  
  { name: "Pants", price: 1500 },  
  { name: "Shoes", price: 3000 }  
];
```

```
// Apply 10% discount to items above 1000  
const discounted = products
```

```
.filter(product => product.price > 1000)
.map(product => ({
  ...product,
  discountedPrice: product.price * 0.9
}));
```

```
console.log(discounted);
```

Conclusion

Mastering array methods like `map()`, `filter()`, and `reduce()` is critical for any modern JavaScript developer. These functions allow us to process and transform data efficiently and are frequently used in real-world applications — from dynamic websites to APIs and beyond. Today's session significantly improved my ability to handle complex data manipulation tasks in a clean and optimized way.