🎬 Episode 19: Map, Filter & Reduce

🔁 Higher-Order Functions (HOFs)

👉 `map()`, `filter()`, and `reduce()` are classic examples of **Higher-Order Functions** in JavaScript.
They either **take a function as an argument** or **return a function**.

🗺️ Map Function

> **Purpose**: Transforms each element of an array and returns a **new array** of the same length.

☑️ Syntax:

```
const output = arr.map(callback);
```

- The `callback` function is executed on **each element**.

🔧 Example Tasks:

◇ **Task 1: Double Each Element**

```
const arr = [5, 1, 3, 2, 6];

function double(x) {
  return x * 2;
}

const doubleArr = arr.map(double);
console.log(doubleArr); // [10, 2, 6, 4, 12]
```

◇ **Task 2: Triple Each Element**

```
function triple(x) {
  return x * 3;
}

const tripleArr = arr.map(triple);
console.log(tripleArr); // [15, 3, 9, 6, 18]
```

◇ **Task 3: Convert to Binary**

```
function binary(x) {
  return x.toString(2);
}

const binaryArr = arr.map(binary);
console.log(binaryArr); // ["101", "1", "11", "10", "110"]
```

🔄 Or using arrow function:

```
const binaryArr = arr.map((x) => x.toString(2));
```

🧹 Filter Function

> **Purpose**: Filters elements based on a condition and returns a **new array**.

✅ Syntax:

```
const filteredArr = arr.filter(callback);
```

🔧 Example: Filter Odd Numbers

```
const arr = [5, 1, 3, 2, 6];

function isOdd(x) {
  return x % 2;
}

const oddArr = arr.filter(isOdd);
console.log(oddArr); // [5, 1, 3]
```

🔄 Or using arrow function:

```
const oddArr = arr.filter((x) => x % 2);
```

🧮 Reduce Function

> **Purpose**: Reduces the array to a **single output value** by accumulating results.

✅ Syntax:

```
const result = arr.reduce((acc, curr) => {
  // logic
  return acc;
}, initialValue);
```

🔧 Examples:

- **Sum of All Elements (Non-Functional Way)**

```
function findSum(arr) {
  let sum = 0;
  for (let i = 0; i < arr.length; i++) {
    sum += arr[i];
  }
  return sum;
}
console.log(findSum(arr)); // 17
```

- **Using `reduce`**

```
const sumOfElem = arr.reduce((acc, curr) => acc + curr, 0);
console.log(sumOfElem); // 17
```

🔧 Find Maximum Element

Non-functional:

```
function findMax(arr) {
  let max = 0;
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] > max) max = arr[i];
  }
  return max;
}
console.log(findMax(arr)); // 6
```

With `reduce`:

```
const output = arr.reduce((max, curr) => (curr > max ? curr : max), 0);
console.log(output); // 6
```

🧑‍🏭 Tricky `map()` with Objects

```
const users = [
  { firstName: "Alok", lastName: "Raj", age: 23 },
  { firstName: "Ashish", lastName: "Kumar", age: 29 },
  { firstName: "Ankit", lastName: "Roy", age: 29 },
  { firstName: "Pranav", lastName: "Mukherjee", age: 50 },
];
```

◇ Get Full Names

```
const fullNameArr = users.map((user) => `${user.firstName} ${user.lastName}`);
console.log(fullNameArr); // ["Alok Raj", "Ashish Kumar", "Ankit Roy", "Pranav
Mukherjee"]
```

📊 Reduce to Group Data by Age

> 🎯 Goal: Create an object like `{29: 2, 23: 1, 50: 1}`

```
const report = users.reduce((acc, curr) => {
  acc[curr.age] = (acc[curr.age] || 0) + 1;
  return acc;
}, {});
console.log(report); // {23: 1, 29: 2, 50: 1}
```

🔗 **Function Chaining**

> 🎯 Get first names of users whose age is < 30

```
const output = users
  .filter((user) => user.age < 30)
  .map((user) => user.firstName);
console.log(output); // ["Alok", "Ashish", "Ankit"]
```

🏠 Homework Challenge: Do It With `reduce`

```
const output = users.reduce((acc, curr) => {
  if (curr.age < 30) acc.push(curr.firstName);
```

```
    return acc;
  }, []);
  console.log(output); // ["Alok", "Ashish", "Ankit"]
```

🌟 Summary:

Function	Purpose	Returns
`map()`	Transform each item	New transformed array
`filter()`	Filter items based on condition	New filtered array
`reduce()`	Accumulate data to one value	Single output value

🧠 Learn → Practice → Master