Episode 19 : map, filter & reduce

> map, filter & reducer are Higher Order Functions.

Map function

It is basically used to transform a array. The map() method creates a new array with the results of calling a function for every array element.

const output = arr.map(*function*) // this *function* tells map that what transformation I want on each element of array

```
const arr = [5, 1, 3, 2, 6];
// Task 1: Double the array element: [10, 2, 6, 4, 12]
function double(x) {
    return x * 2;
}
const doubleArr = arr.map(double); // Internally map will run double
function for each element of array and create a new array and returns it.
console.log(doubleArr); // [10, 2, 6, 4, 12]
```

```
// Task 2: Triple the array element
const arr = [5, 1, 3, 2, 6];
// Transformation logic
function triple(x) {
    return x * 3;
}
const tripleArr = arr.map(triple);
console.log(tripleArr); // [15, 3, 9, 6, 18]
```

```
// Task 3: Convert array elements to binary
const arr = [5, 1, 3, 2, 6];
// Transformation logic:
function binary(x) {
    return x.toString(2);
}
const binaryArr = arr.map(binary);

// The above code can be rewritten as :
const binaryArr = arr.map(function binary(x) {
    return x.toString(2);
}

// OR -> Arrow function
const binaryArr = arr.map((x) => x.toString(2));
```

So basically map function is mapping each and every value and transforming it based on given condition.

Filter function

Filter function is basically used to filter the value inside an array. The arr.filter() method is used to create a new array from a given array consisting of only those elements from the given array which satisfy a condition set by the argument method.

```
const array = [5, 1, 3, 2, 6];
// filter odd values
function isOdd (x) {
    return x % 2;
}
const oddArr = array.filter(isOdd); // [5,1,3]

// Other way of writing the above:
const oddArr = arr.filter((x) => x % 2);
```

Filter function creates an array and store only those values which evaluates to true.

Reduce function

It is a function which take all the values of array and gives a single output of it. It reduces the array to give a single output.

```
const array = [5, 1, 3, 2, 6];
// Calculate sum of elements of array - Non functional programming way
function findSum(arr) {
    let sum = 0;
    for (let i = 0; i < arr.length; i++) {
        sum = sum + arr[i];
    }
    return sum;
}
console.log(findSum(array)); // 17

// reduce function way
const sumOfElem = arr.reduce(function (accumulator, current) {
    // current represent the value of array
    // accumulator is used the result from element of array.
    // In comparison to previous code snippet, *sum* variable is
*accumulator* and *arr[i]* is *current*
    accumulator = accumulator + current;
    return accumulator;
}, 0); //In above example sum was initialized with 0, so over here
accumulator also needs to be initialized, so the second argument to reduce
function represent the initialization value.
console.log(sumOfElem); // 17
```

```
// find max inside array: Non functional programming way:
const array = [5, 1, 3, 2, 6];
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(array)); // 6

// using reduce
const output = arr.reduce((acc, current) => {
    if (current > acc ) {
        acc = current;
    }
    return acc;
}, 0);
console.log(output); // 6

// acc is just a label which represent the accumulated value till now,
// so we can also label it as max in this case
const output = arr.reduce((max, current) => {
    if (current > max) {
        max= current;
    }
    return max;
}, 0);
console.log(output); // 6
```

Tricky MAP

```
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 array of full name : ["Alok Raj", "Ashish Kumar", ...]
const fullNameArr = users.map((user) => user.firstName + " " +
user.lastName);
console.log(fullNameArr); // ["Alok Raj", "Ashish Kumar", ...]

------------------------------------------------------------

// Get the count/report of how many unique people with unique age are
there
```

```
// like: {29 : 2, 75 : 1, 50 : 1}
// We should use reduce, why? we want to deduce some information from the
array. Basically we want to get a single object as output
const report = users.reduce((acc, curr) => {
    if(acc[curr.age]) {
        acc[curr.age] = ++ acc[curr.age] ;
    } else {
        acc[curr.age] = 1;
    }
}, {})
console.log(report) // {29 : 2, 75 : 1, 50 : 1}
```

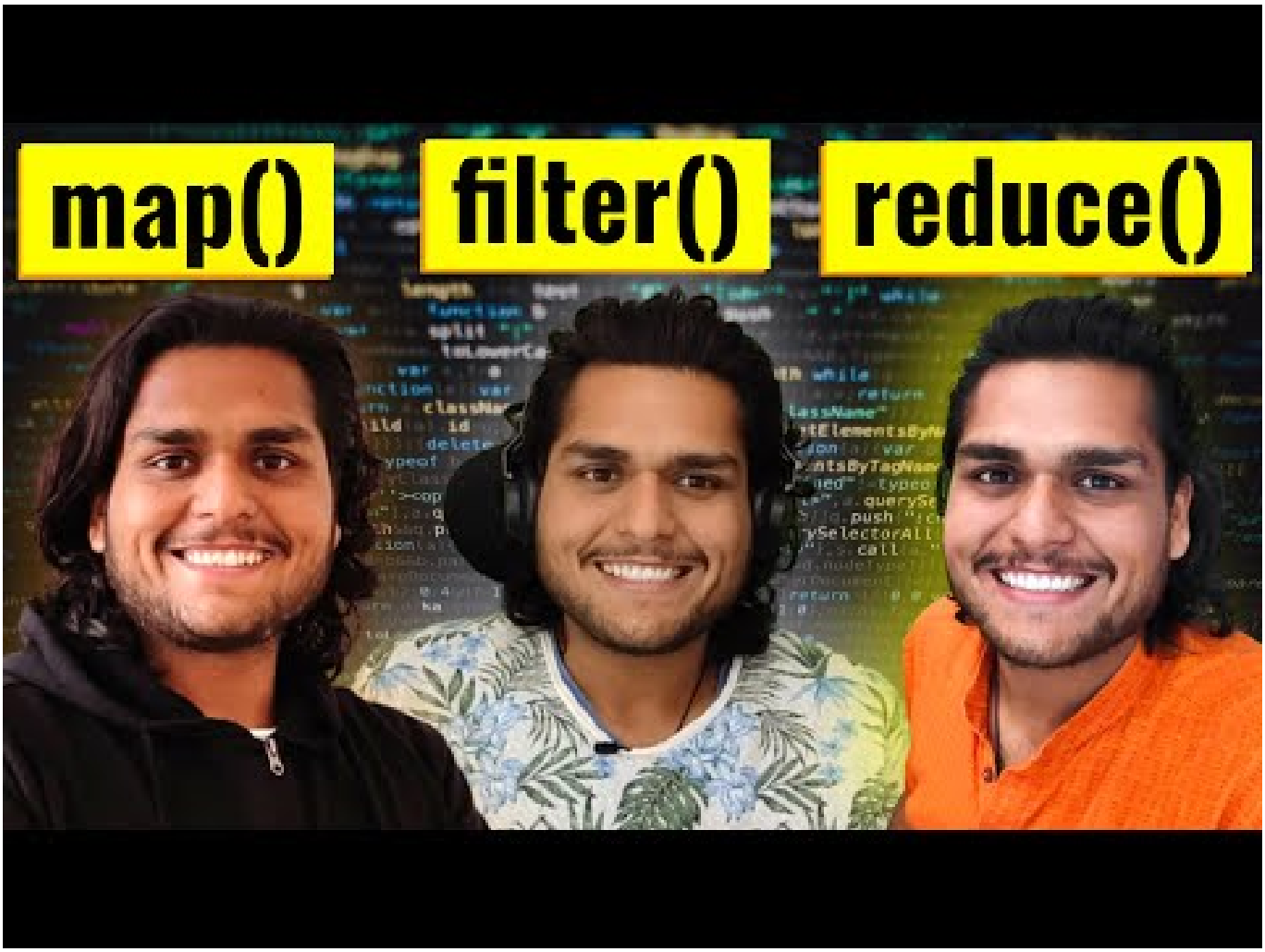
Function Chaining

```
// First name of all people whose age is less than 30
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 },
];

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

// Homework challenge: Implement the same logic using reduce
const output = users.reduce((acc, curr) => {
    if (curr.age < 30) {
        acc.push(curr.firstName)
    }
    return acc;
}, []);
console.log(output); // ["Alok", "Ashish", "Ankit"]
```

Watch Live On Youtube below:


map()
filter()
reduce()