JS:- Higher Order Functions.

07 28th Week • 188-177 JULY

or return another function as an argument.

Functions which take another function as an argument. These functions are known as Higher Order Functions.

Syntax:-

```
function f(x, fn) {
    console.log(x);   → 10
    fn();             → "I am a HOF"
}

f(10, function exec() {
    console.log("I am a HOF");
});
```

x:- number
fn:- function.

If we do console.log(fn) → [Function: exec]

Rough Sketch

```
function gun() {
    ...
}

function fun(gun) {
    ...
    gun();
}
```

HOF

Arrays:-

Arrays are custom objects in JS.

index of the element is the Key.

Element itself is the Value.

Ⓐ MAP Function:- MAP is a HOF, available with the arrays.

① It takes a function as an argument → (f)

② It returns an array in which every value is actually popwated by calling function (f) with original array element as argument.

```
function square (el) {
    return el * el;
}

const arr = [1, 2, 3, 4, 5];

const result = arr.map(square);

console.log(result)
```

o/p [1, 4, 9, 16, 25]

Explanation :-

① **MAP is a HOF**

arr.map(square)

So, from the object (arr); we are accessing a function (map). And inside that function we are sending square() - which happens to be a function.

∴ MAP is HOF ✓

② **It takes a function (f) as an argument**

arr.map(square) ✓

③ **It returns an array :-**

const (result) = (arr.map(square))

This is an array. which is the return type of

SUNDAY 09

④ Every value is actually populated by calling func. (f) with original array element as argument.

The content of the (result) array is populated by passing each element from the original array (const arr = [1,2,3,4,5]) one by one in the argument function (f). → square.

. Map internally iterates | loops over every element of the given original array pass that element in the argument function f and then store the returned value inside an array.

Use Case of MAP

In any situation when we have to do an operation on every element of the array map can be a good opn.

Example:

We have array of product objects. For each product object, I want to get product name.

We can we do, in the array we can do map and fetch object of each product and in that object fetch the Key name.

```
function isEvenOdd (ele) {
    if (ele%2 == 0) {
        return "Even";
    } else {
        return "Odd";
    }
}

const arr = [1, 2, 3, 4, 5];

const result = arr.map (isEvenOdd);

console.log (result);
```

*** Note ***

- We are just naming the funct We are not calling it. by saying isEvenOdd( )
- Same as saying we are just passing the func.

(8) **SORT:-** Sort the unsorted array.

```
let arr = [1, 10, 9, 100, 1000, 11, 12, 2, 3];

arr.sort();

console.log(arr);
```

O/P No change

- .sort() implementation does lexicographical sorting.

∴ Default implementation of arr.sort() is going to sort the array is lexicographical order.

lexicographical order :- Dictionary order.

2nd Use Case for MAP:-

```
const newArr = [9, 8, 7, 13, 15]

function print(element, index) {
    return `Element present at ${index} is ${element}`;
}

const result = newArr.map(print);
```

∴ In a nutshell, if we are passing func inside MAP takes 2 arguments.
The 1st argument will be the element
" 2nd " " " " element's index.

* Print takes 2 arguments. So when we do newArr.map(print).

∴ 1st argument will be the actual elem. in newArr
2nd argument will be the index of that element.

JULY 2017

WK	M	T	W	T	F	S	S
27	31					1	2
28	3	4	5	6	7	8	9
29	10	11	12	13	14	15	16
30	17	18	19	20	21	22	23
31	24	25	26	27	28	29	30

Here, MAP is looping over every element of then passing (element, index) in the function print.

Custom Mapper Function:-

To add an element inside an array → arr.push(elem)

```
function customMapper(array, func){
    let result = [];
    for(let i=0; i<array.length; i++){
        result.push(func(array[i], i));
    }
    return result;
}

let result = customMapper(newArr, print);
console.log(result);
```

"customMapper" is mimicking map function.

We are looping over the array

In the result array we are storing the return value of function (print).

Sorting (Continued) : Sort is a HOF

Sort takes a Comparator function as argument.

```
arr.sort(function(a,b){ return a-b });
```

If a<b ; then a-b is -ive → (a) is placed before (b).

a>b ; " a-b is +ive → (b) " " " (a).

FILTER :— ① Filter is a HOF

② Filter loops over the array elements

③ The special thing about filter is that the argument func. (f) which we have to pass inside filter should always return a Boolean. Otherwise o/p will be converted to a Boolean.

passes that element in the argument function and then if the o/p of this func. call is true, then it stores the original element in the new array otherwise doesn't add this elem. in the array.

```
function oddOrEven(x){
    return (x % 2 == 0);   // Boolean return
}

let arr = [1, 2, 3, 4, 5];

const result = arr.filter(oddOrEven);

console.log(result)  ⟶ [2, 4].
```

REDUCE :— ① Reduce is a HOF

② Reduce one by one goes to every element of the array. And pass that element to the function (f) and accumulate the result of further function calls with this particular result.

2017 FRIDAY

14 29th Week • 195-170 JULY

JULY 2017

Wk	M	T	W	T	F	S	S
27	31					1	2
28	3	4	5	6	7	8	9
29	10	11	12	13	14	15	16
30	17	18	19	20	21	22	23
31	24	25	26	27	28	29	30

REDUCE:- Reduce function is basically used at a place where you take all the elements of the array and come up with a single value out of it.

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

We have to iterate over all the elements in the array and come up with sum of all these elements. | or find out the max no. in the array.

Non-Functional way:-

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

(17)

Reduce takes 2 arguments.

1. A function
2. Initial value of accumulator.

Lets transform this logic and write a reduce function instead.

```
const output = arr.reduce( function(accumulator, current)
```

This function will be iterated over each & every elem. of the array.

The argument "current" represents the current value the pointer points during the loop.

"Accumulator" represents the result we need to get out of all these values.

AUGUST 2017

WK	M	T	W	T	F	S	S
32		1	2	3	4	5	6
33	7	8	9	10	11	12	13
34	14	15	16	17	18	19	20
35	21	22	23	24	25	26	27
36	28	29	30	31			

SATURDAY 2017

29th Week • 196-169

JULY 15

In the non-functional way,

We were accumulating our result inside variable (sum).

∴ Here (sum) becomes our (accumulator).

And arr[i] is the current.

```
∴ const output = arr.reduce(function (accumulator, current){
        accumulator = accumulator + current;
        return accumulator;
    }, 0)
```

0 → Initial value of accumulator.

```
console.log(output);
```
→ (17)

Finding Max

SUNDAY 16

```
const output = arr.reduce(function (accumulator, current){
    if (current > accumulator){
        Accumulator = current;
    }
    return accumulator
}, 0)
```

```
console.log(output)
```
→ (6)

JULY 2017
Wk M T W T F S S
27 31 1 2
28 3 4 5 6 7 8 9
29 10 11 12 13 14 15 16
30 17 18 19 20 21 22 23
31 24 25 26 27 28 29 30

⊙ Scenario:-

```
const users = [
{firstName: "Mr", lastname: "X", age: 23},
{firstName: "Donald", lastname: "Duck", age: 45},
{firstName: "Sharma", lastName: "Beta", age: 23}
];
```

Task 1

= We want to create an array with FULL Names.

ie ["Mr X", "Donald Duck", "Sharma Beta"].

```
function firstLastMerge(ele){
    let firstName = ele.firstName;
    let lastName = ele.lastName;
    let returnValue = firstName + " " + lastName;
    return returnValue;
}

const output = users.map(firstLastName);
console.log(output);
```

Task 2 We want firstName of all the users with age less than 30.

```
function ageMapper(ele){
    let age = ele.age;
    if(age<30){
        return true;
    }
    return false;
}
```

```
function getName(ele){
    let firstName = ele.firstName;
    return firstName;
}
```

```
const output = users.filter(ageMapper).map(getName);
```

Upon writing this we get

```
[{firstName:"Mr", lastName:"X", age:23},
{firstName:"Sharma", lastName:"Beta", age:23}]
```

→ We can use Map on this to get firstName

Same task via. Reduce:-

```
const output = users.reduce(function(acc, curr){
    let age = curr.age;
    if(age<30){
        acc.push(curr.firstName);
    }
    return acc;
}, []);
```

Initial val. of acc is [].

2017 FRIDAY

21 JULY

30th Week • 202 - 163

JULY 2017

WK	M	T	W	T	F	S	S
27	31						
28	3	4	5	6	7	8	9
29	10	11	12	13	14	15	16
30	17	18	19	20	21	22	23
31	24	25	26	27	28	29	30

AUGUST

WK	M
32	
33	7
34	14
35	21
36	28

09.00

10.00

11.00

12.00

01.00

02.00

03.00

04.00

05.00

07.00

⊙ Getting the Name initials:-

const input = " George Raymond Richard Martin ";

Expected output :- "GRRM".

```
const output = input.split(" ").map(f).reduce(function (acc, curr){
                                                    acc += curr;
                                                    return acc;
                                                }, "");

function f(elc){
    return elc[0];
}
```

⊙ Age diff. b/w the youngest and oldest.

```
const input = [ {name: "John", age: 13}, {name: "Mark", age: 56},
{name: "Rachel", age: 45}, {name: "Nate", age: 67},
{name: "Jennifer", age: 65} ].
```

o/p = [13, 67, 54]

```
✉ @ ☎ let o/p youngest = input.reduce(function (acc, curr){
                            if (curr.age < acc){
                                acc = curr.age;
                            }
                            return acc;
                        }, 100)
```

```
let opOldest = input.reduce(function (acc, curr){
    if (curr.age > acc){
        acc = curr.age;
    }
    return acc; }, 0);

console.log([opYoungest, opOldest, opOldest - opYoungest]);
```

⊙ Count elements in array of arrays

```
const input = [['a', 'b', 'c'], ['c', 'd', 'f'], ['d', 'f', 'g']];
```

Expected op { a:1, b:1, c:2, d:2, ... }.

Note arr.flat() is used to flatten a Nested 2D array.

```
const output = input.flat().reduce(function (acc, curr){
    if (acc[curr]){
        acc[curr]+=1;
    }else{
        acc[curr]=1;
    }
    return acc;
}, {});
```

2017 MONDAY

24 31st Week • 205 - 160
JULY

```
const products = [
  { name: 'product1 ', price: 20 , category : " Electronics" } ,
  { name: 'product2' , price:100 , category: " Clothes" } ,
  { name: 'product 3', price: 40 , category: " Electronics" } ] ;
```

Expected O/P

{ Electronics: 60 , Clothes: 100 }

For every dept. Find total price.

Step1 Can we make something like

{ Electronics : [20, 40] , Clothes: [100] } .

```
const tarOne = products.reduce( function (acc, curr) {
    let currCat = curr.category ;
    if (!acc[currCat]) {
        acc[currCat] = [curr.price];
    }
    else {
        acc[currCat].push(curr.price)
    }
return acc;}
} , { } );
```

AUGUST 2017

WK	M	T	W	T	F	S	S
32		1	2	3	4	5	6
33	7	8	9	10	11	12	13
34	14	15	16	17	18	19	20
35	21	22	23	24	25	26	27
36	28	29	30	31			

TUESDAY 2017

31st Week • 206-159

JULY 25

Step2 { Electronics : 60, clothes : 100 }.

09.00

10.00 Idea is, we will pick each key, and for every array corresponding to that

11.00 Key we will sum them up.

12.00 We Know,

Object.keys(ObjName) return us a list of keys.

01.00

02.00 const tarTwo = Object.keys(tarOne).reduce(funcLim(acc, curr){

03.00 return

["Electronics", 'Clothes"].

04.00

let currArr = tarOne[curr].reduce(

return

[20, 40]

[100]

05.00

06.00

func(sum, price){

sum = sum + price;

return sum;

}, 0)

07.00

will return 60 from [20, 40]

100 from [100].

acc[curr] = currArr;

return acc;

}, {});

console.log(tarTwo);