forEach

Does it return Something?

go to every element of the array, & apply the cb on that element

map

vvi

return an new array

is capable to go to every element of the array. & apply a cb.

cb of map method, we can actually return something & map picks the value of each return statement & puts it in a new array.

[2, 3, 1, 4, 5]

[4 6 1 .....]

response = arr.map ((el) => {
    return el*2;

3)

Filter

↙

it filters out elements based on some condition

[1, 2, 3, 4, 5] ← we want to fetch all the odd elements.

In simple terms, we want to filter out odd elements

① → true

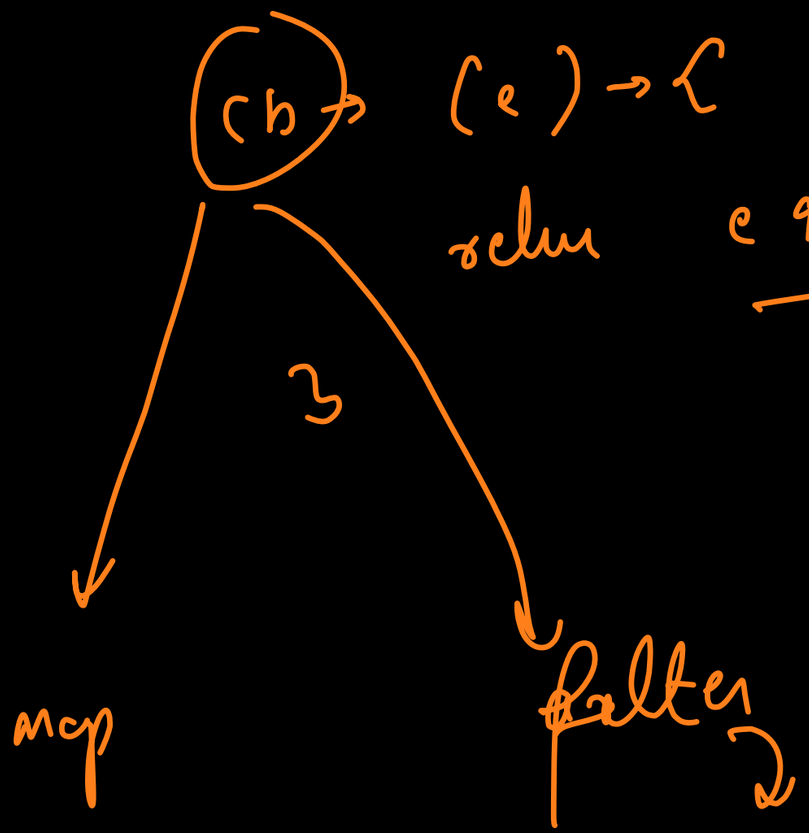
```
1  const arr = [1,2,3,4,5,6,7];
2
3  const oddelements = arr.filter((element) ⇒ {
4      // logic you write here
5      return element % 2 ≠ 0;
6  })
7
8  console.log(oddelements); // new array
```

cb

→ for all those elements of the array for whom the cb returns true, those original elent will be part of a new filtered array:

[1, 3, 5 ....]

↳ new array



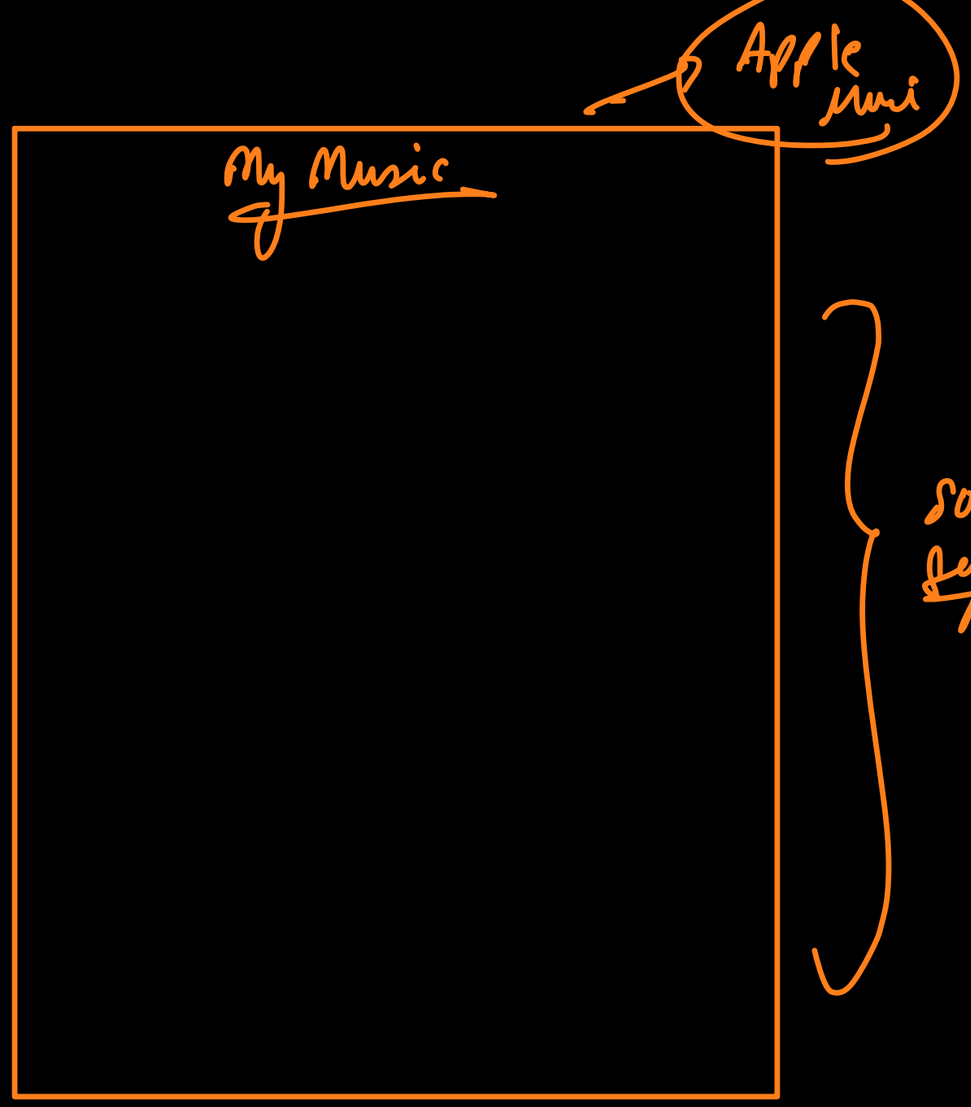
(e) -> {

return e % 2 == 0;

[1, 2, 3, 7, 5]

[f, f, f, f, f]

[2, 4]



eg

includes all songs are part of the My music playlist

[1, 2, 3, 4,]

Reduce Method

this method is used to reduce an array to a single value.

Cart → products added to cart

we can represent this cart as array of products

[{ product: 'Nike air force', price: 25000, quantity: 1 },
{ product: 'airpods', price: 20000, quantity: 2 },
{ product: 'power adapter', price: 2500, quantity: 4 }
]

find the total cost of the cart:

// approach 1

```
let value = 0
cart.forEach (product => {
    value += product.quantity * product.price;
});
return value;
```

value → similar to accumulator

0 → similar to initialValue

product → similar to currentValue

array.reduce(reducerfunc, initialValue);

returns the final single value

reducerfunc

how this looks like ??

→ accumulator.

→ currentValue

```js
const shoppingCart = [
    { product: 'Shoes', price: 80, quantity: 1 },
    { product: 'Bag', price: 120, quantity: 2 },
    { product: 'Watch', price: 250, quantity: 4 }
];


// I want to calculate total price of the cart

/**
 * let value = 0;
 * shoppingCart.forEach(product => {
 *  value += product.price * product.quantity;
 * })
 */

const result = shoppingCart.reduce(function reducer(accumulator, currentValue) {
    return accumulator + currentValue.price * currentValue.quantity;
}, 0)

console.log(result);
```

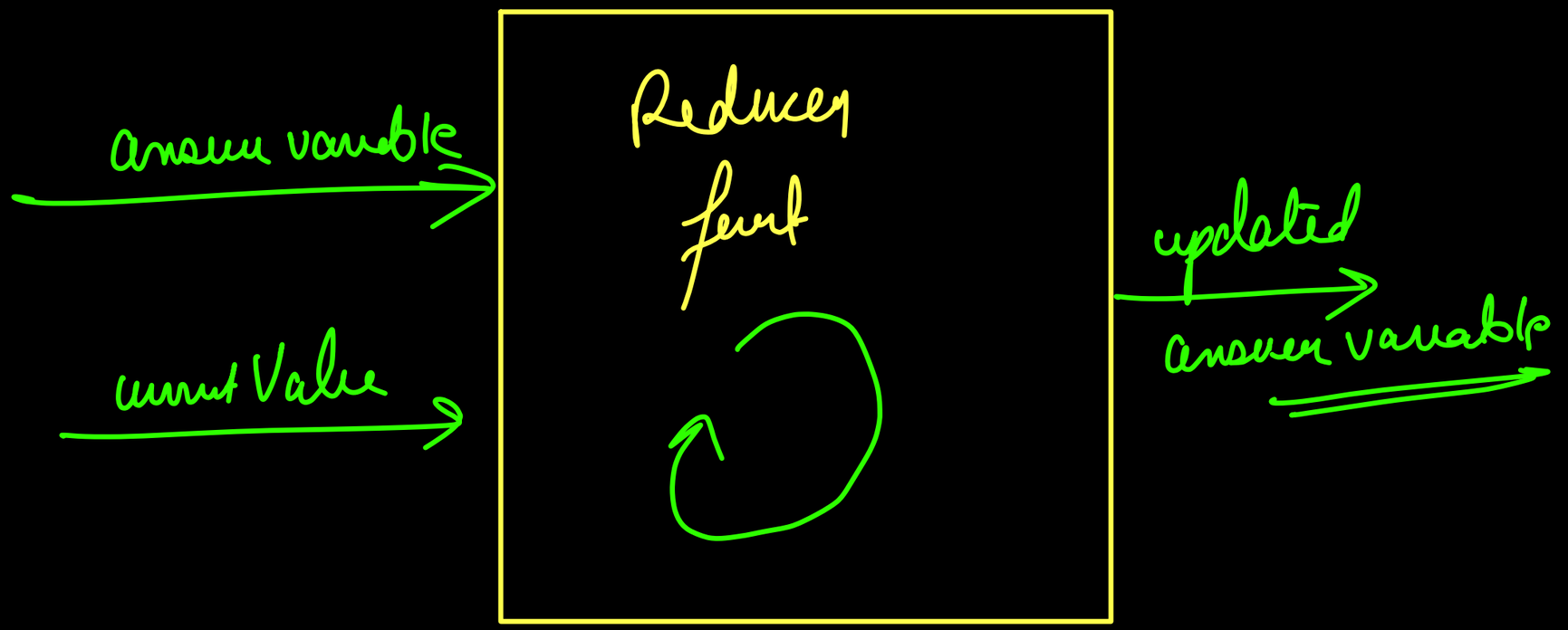
variable for final output

accumulator → 0
currentValue

accumulator → 80
currentValue

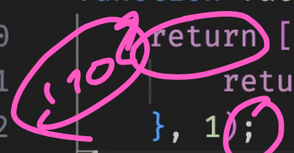
accumulator → 320
currentValue

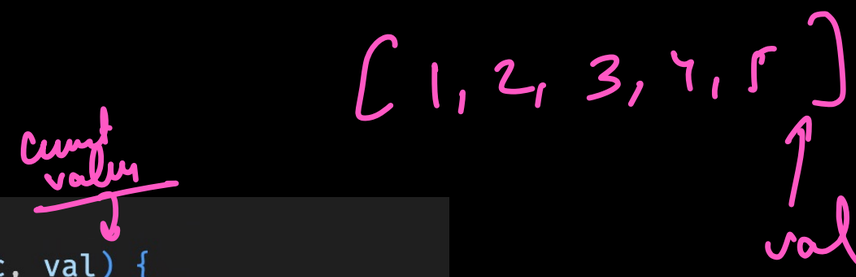
accumulator
→ 320

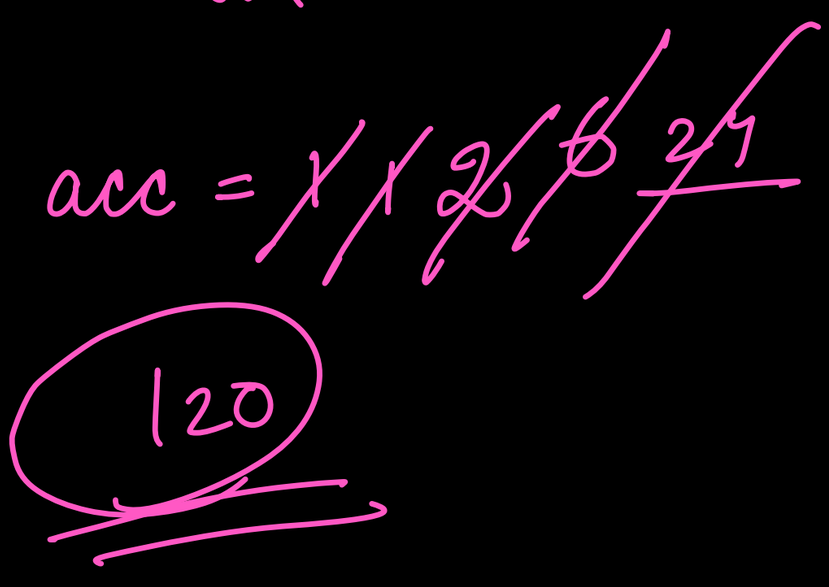


Whatever our return becomes
updated answer variable

```
9   function factorial() {
10      return [1,2,3,4,5].reduce(function reducer(acc, val) {
11          return acc * val;
12      }, 1);
13  }
14
15  console.log(factorial());
```







Array.from (——— ——— cb

Array like objects → [ ] [1, 2, 3]

{ length: 3 }

d

{ 0: 1, 1: 2, length: 3 }

c

b

key : —

[50, 3, 1, 9, 8]
0 1 2 3 4

[, , , ,]
len : 5

{
0 : 50
1 : 3
2 : 1
3 : 9
4 : 8
}

y

JOIN THE DARKSIDE

why was passing an array like object imp <u>??</u>

Array.from uses an array like obj instead of any object because

Array like objects has a mandatory length property.

the cb fnc^{n} of Array.from is called length no. of times

Array.from ({ length: num } , (_ , i) => i + 1)

current element — index we are iterating on — the length loop

obj

Array(num)

obj

cb runs length no. of times

for (i = 0 ; i < len ; i++) {

(undefined, 0)

(undefined, 1)

(undefined, 2)

⋮

}

cb(obj[i], i)