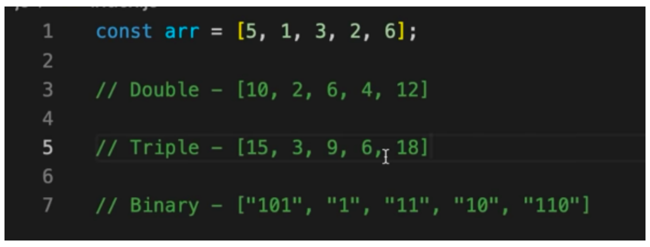
Map, Filter and Reduce

Monday, 21 August 2023 11:14 PM

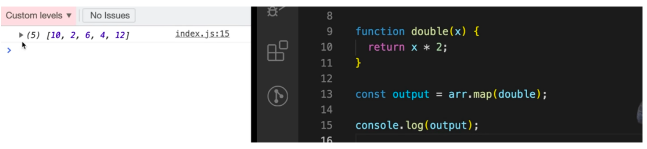
→ map() function: We use map() when we need to transform an array.



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

// Double - [10, 2, 6, 4, 12]

// Triple - [15, 3, 9, 6, 18]

// Binary - ["101", "1", "11", "10", "110"]
```

eg of transformation (use cases of map())

eg:



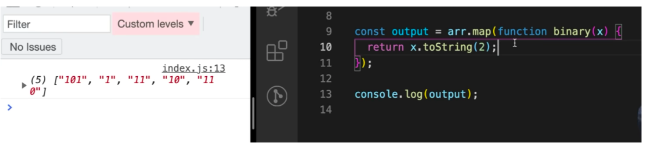
```
(5) [10, 2, 6, 4, 12]                index.js:15
```

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

const output = arr.map(double);

console.log(output);
```

eg of usage of map to double the values of an array.

map() takes a callback f^n inside it and runs that f^n for all values of the array it is pointing to. Internally, map() creates a new array and pushes the transformed values to this new array & returns it.

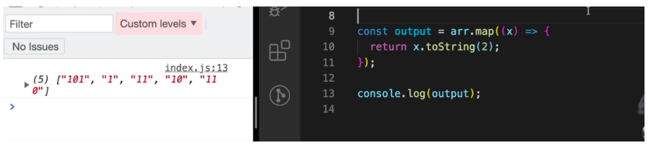
Note: Another way of writing map();



```
(5) ["101", "1", "11", "10", "110"]    index.js:13
```

```
const output = arr.map(function binary(x) {
  return x.toString(2);
});

console.log(output);
```

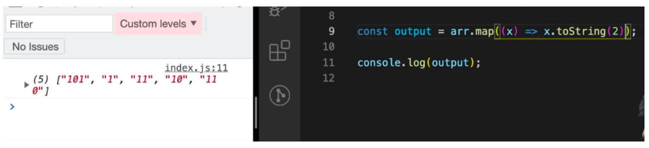
eg to convert numbers to binary.



```
(5) ["101", "1", "11", "10", "110"]    index.js:13
```

```
const output = arr.map((x) => {
  return x.toString(2);
});

console.log(output);
```

eg using arrow f^n



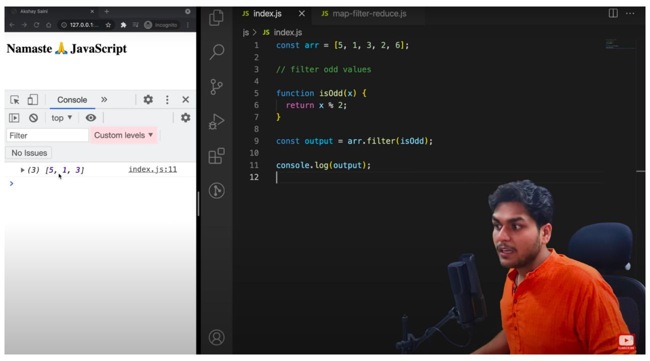
```
(5) ["101", "1", "11", "10", "110"]    index.js:11
```

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

console.log(output);
```

eg: we can write like this also if there is only 1 line of code

→ filter() function: We use filter() f^n to filter out values from an array.

eg: To filter out odd nos. inside an array.



Namaste 🙏 JavaScript

```
(3) [5, 1, 3]                index.js:11
```

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

// filter odd values

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

const output = arr.filter(isOdd);

console.log(output);
```

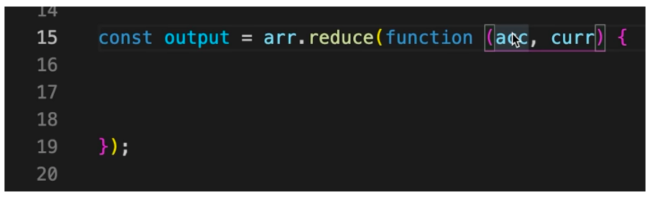
We can also use the other ways of writing map() in filter() also.

So, basically filter() f^n only returns the values stored in a new array which satisfy the condition present in the callback fn.

→ reduce() function : We use reduce() fn when we need to take all elements of an array and convert or reduce them to a single value.

use-cases: Find max. no. in an array, sum of all elements in an array.

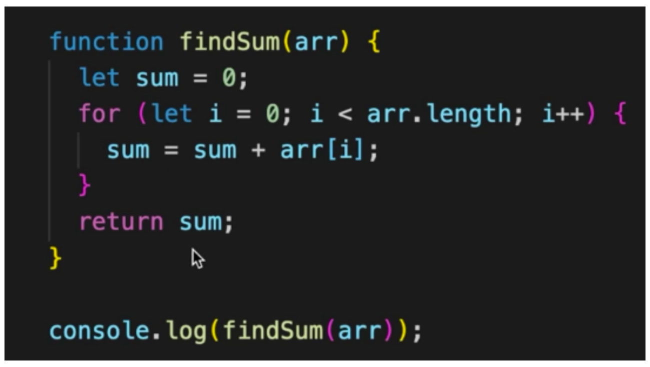
→ reduce() takes 2 values accumulator and current. Current means the current value reduce() is pointing to inside the array. Accumulator accumulates the value.



```
const output = arr.reduce(function (acc, curr) {



});
```

→ Syntax of reduce()

eg to understand what reduce() is doing internally:

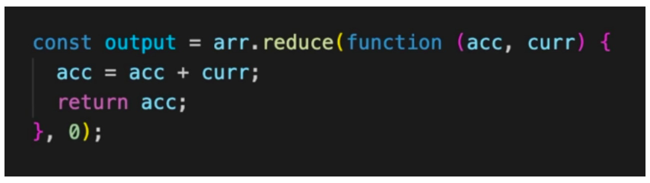


```
function findSum(arr) {
  let sum = 0;
  for (let i = 0; i < arr.length; i++) {
    sum = sum + arr[i];
  }
  return sum;
}

console.log(findSum(arr));
```

→ Here sum is the accumulator. arr[i] is the current.

→ This code basically takes in an array & returns its sum.

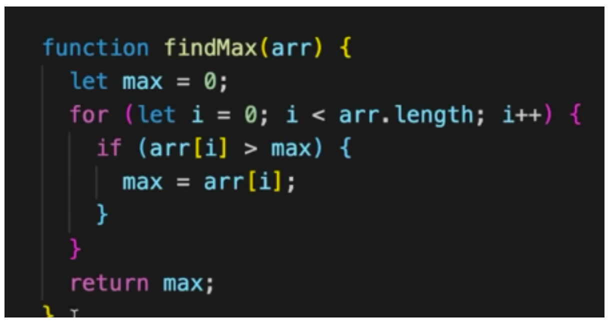
→



```
const output = arr.reduce(function (acc, curr) {
  acc = acc + curr;
  return acc;
}, 0);
```

So, the first argument of a reduce() fn is a callback fn & the second argument is the initial value of the accumulator. Because it must be given an initial value to start with like we've done in findSum() fn.

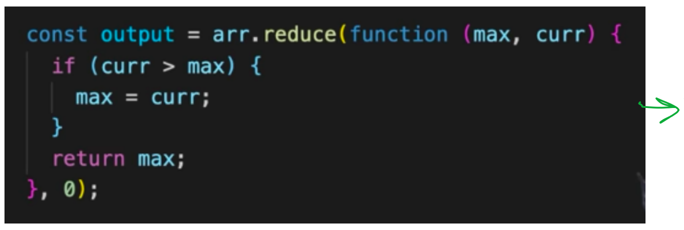
eg: Find max in an array (assuming array is non-empty & has +ve integers):



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

```
}
```

Now, using reduce() we can achieve the above logic.

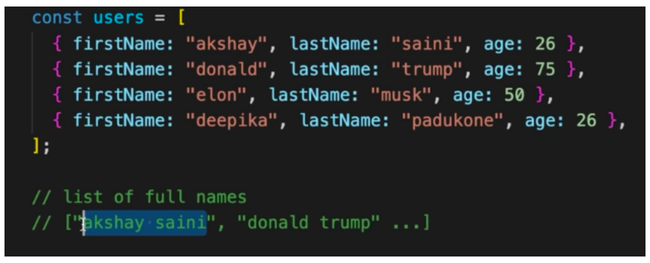


```
const output = arr.reduce(function (max, curr) {
  if (curr > max) {
    max = curr;
  }
  return max;
}, 0);
```

→ Here max is the accumulator & curr is current and we've initialised accumulator with 0.

We know that reduce() traverses through the whole array, thus we just need to write the condition inside reduce().

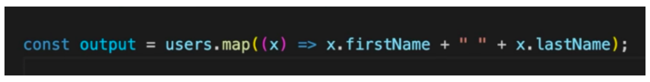
eg: Tricky example of map():



```
const users = [
  { firstName: "akshay", lastName: "saini", age: 26 },
  { firstName: "donald", lastName: "trump", age: 75 },
  { firstName: "elon", lastName: "musk", age: 50 },
  { firstName: "deepika", lastName: "padukone", age: 26 },
];

// list of full names
// ["akshay saini", "donald trump" ...]
```

We need to return full names from this array of objects.

Therefore we will use map().



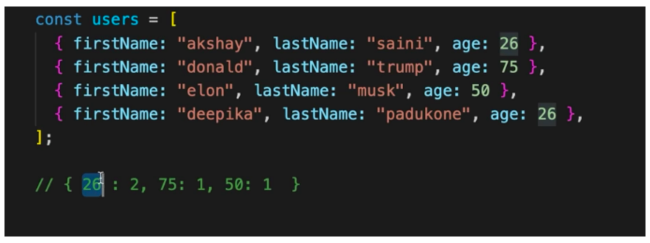
```
const output = users.map((x) => x.firstName + " " + x.lastName);
```

So, that's how we use map().

→ Tricky example using reduce():

Using above array only we need to find how many people have a particular age.

o/p should be something like this:



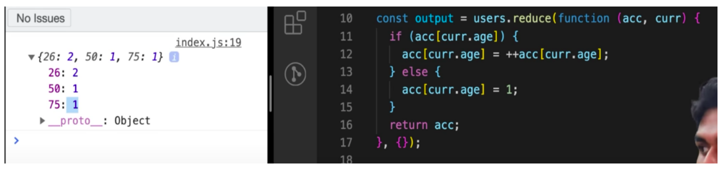
```
const users = [
  { firstName: "akshay", lastName: "saini", age: 26 },
  { firstName: "donald", lastName: "trump", age: 75 },
  { firstName: "elon", lastName: "musk", age: 50 },
  { firstName: "deepika", lastName: "padukone", age: 26 },
];

// { 26 : 2, 75: 1, 50: 1  }
```

We would be using reduce() as we want a single object as o/p and not an array.

We will reduce the array to a single value which in this case is an object.

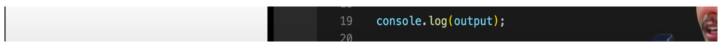
So, for this the accumulator would be: { }

and current would be: the individual object inside the array.



```
No Issues
                                   index.js:19
▼{26: 2, 50: 1, 75: 1}
    26: 2
    50: 1
    75: 1
  ▶ __proto__: Object
>
```

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



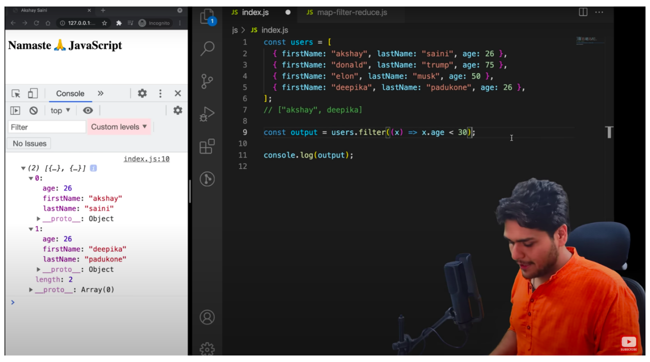
```
console.log(output);
```

So, what is happening in above is that, we write an if-else condition.

- else condition works when we are encountering an age for the first time, then we store it as '1' in the acc.
- if condition works when we've already stored an age in the acc and our code encounter that & increases its value by 1.

→ Example using filter():

Find firstName of all those people whose age < 30 in above array. Therefore we'll use filter()



```
const users = [
  { firstName: "akshay", lastName: "saini", age: 26 },
  { firstName: "donald", lastName: "trump", age: 75 },
  { firstName: "elon", lastName: "musk", age: 50 },
  { firstName: "deepika", lastName: "padukone", age: 26 },
];
// ["akshay", deepika]

const output = users.filter((x) => x.age < 30);

console.log(output);
```

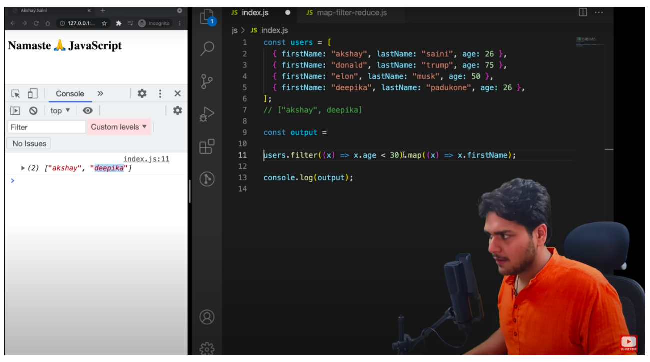
```
index.js:10
▼(2) [{…}, {…}]
  ▼0:
     age: 26
     firstName: "akshay"
     lastName: "saini"
    ▶__proto__: Object
  ▼1:
     age: 26
     firstName: "deepika"
     lastName: "padukone"
    ▶__proto__: Object
   length: 2
  ▶__proto__: Array(0)
>
```

→ we get the following o/p after using filter()

→ This is an array of objects returned to us by filter().

→ So we can basically use map() now to get the firstNames in an array. This is called chaining. map() will run on the o/p of filter().

→ We can chain map(), filter() & reduce().

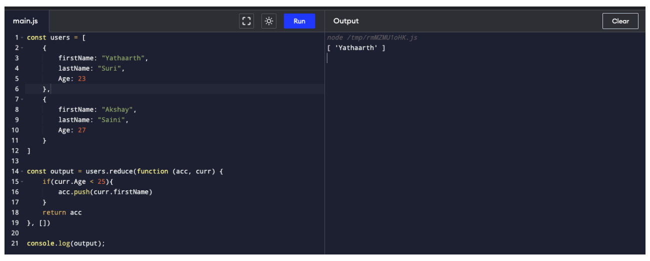


```
const users = [
  { firstName: "akshay", lastName: "saini", age: 26 },
  { firstName: "donald", lastName: "trump", age: 75 },
  { firstName: "elon", lastName: "musk", age: 50 },
  { firstName: "deepika", lastName: "padukone", age: 26 },
];
// ["akshay", deepika]

const output =

users.filter((x) => x.age < 30).map((x) => x.firstName);

console.log(output);
```

```
index.js:11
▶(2) ["akshay", "deepika"]
>
```

→ Code simulating the above logic.

Note: We can achive the same using just reduce() too.



```
const users = [
    {
        firstName: "Yathaarth",
        lastName: "Suri",
        Age: 23
    },
    {
        firstName: "Akshay",
        lastName: "Saini",
        Age: 27
    }
]

const output = users.reduce(function (acc, curr) {
    if(curr.Age < 25){
        acc.push(curr.firstName)
    }
    return acc
}, [])

console.log(output);
```

```
node /tmp/rmMZMU1oHK.js
[ 'Yathaarth' ]
```