Higher-Order Functions

Examples: map, filter, reduce, some and every

What are higher-order functions?

In javascript, a higher-order function is a function that accepts another function as an argument and/or returns a function as a value.

What are higher-order functions?

Let’s take a step back to think about what this means. We know already that functions operate on data. What are some of the types of data in Javascript?

What are higher-order functions?

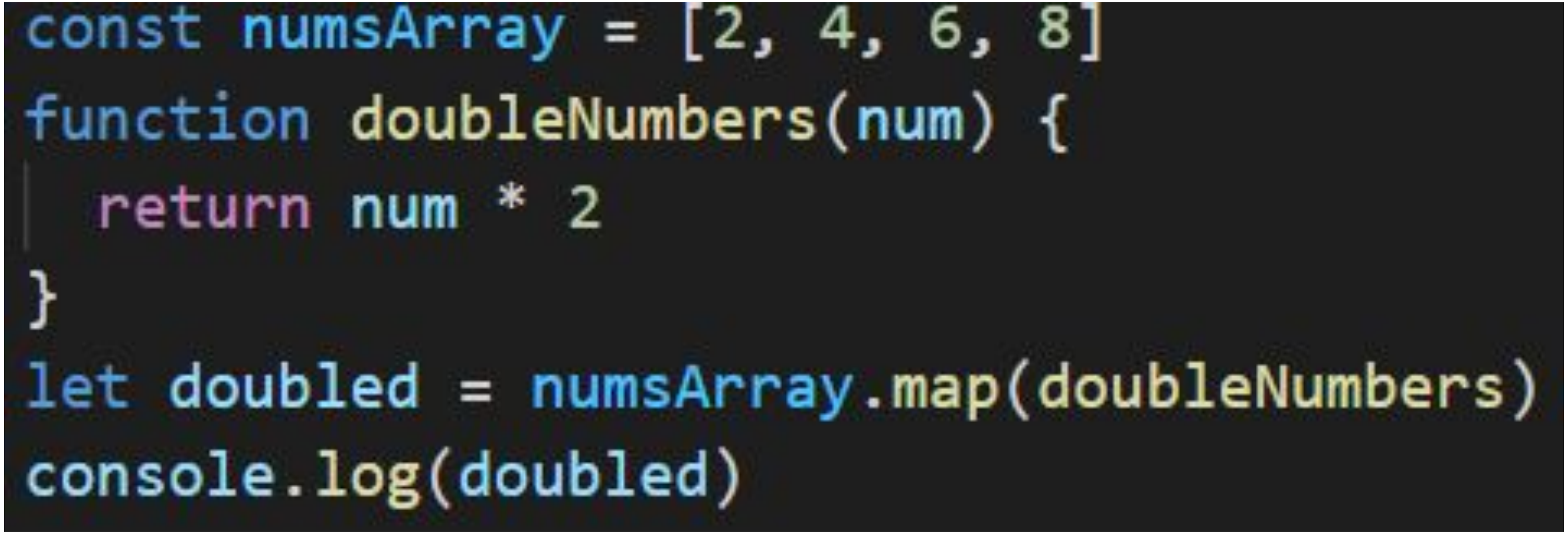
Strings, numbers, booleans and objects all are data types. Functions are data too.

What are higher-order functions?

Just like other types of data, we can store functions in arrays, set them as properties on objects (methods) assign them to variables (expressions) and pass them to other functions as arguments or return them as values (higher-order functions).

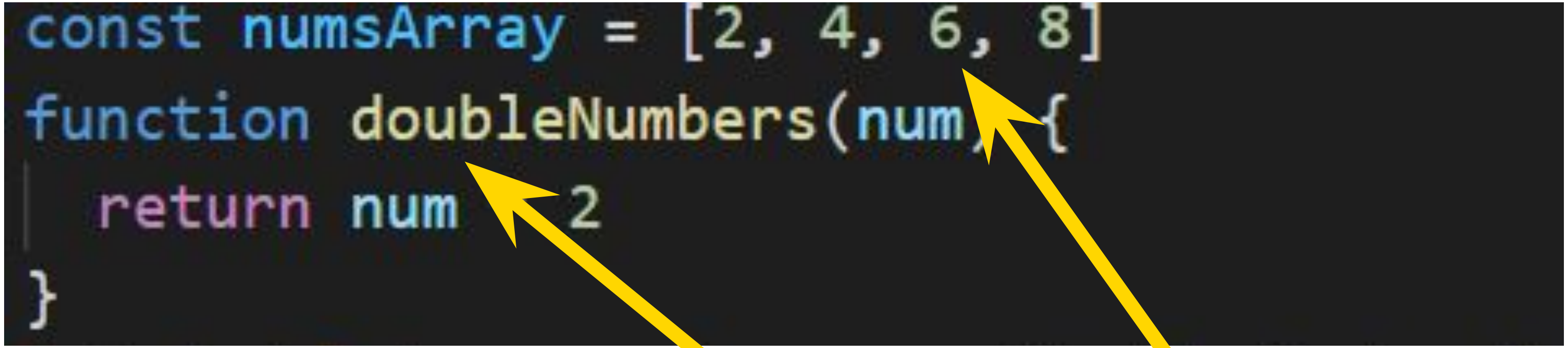
What are higher-order functions?

What are we doing with this code?



```
const numsArray = [2, 4, 6, 8]
function doubleNumbers(num) {
  return num * 2
}
let doubled = numsArray.map(doubleNumbers)
console.log(doubled)
```

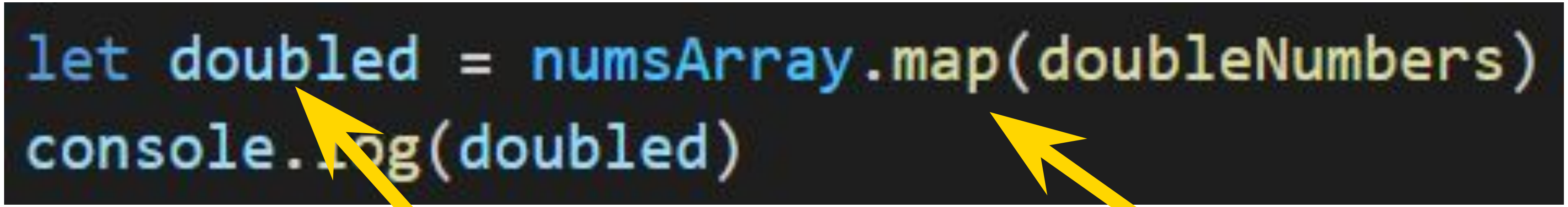
What are higher-order functions?



```
const numsArray = [2, 4, 6, 8]
function doubleNumbers(num) {
  return num * 2
}
```

Here, we’ve created a function and an array.

What are higher-order functions?

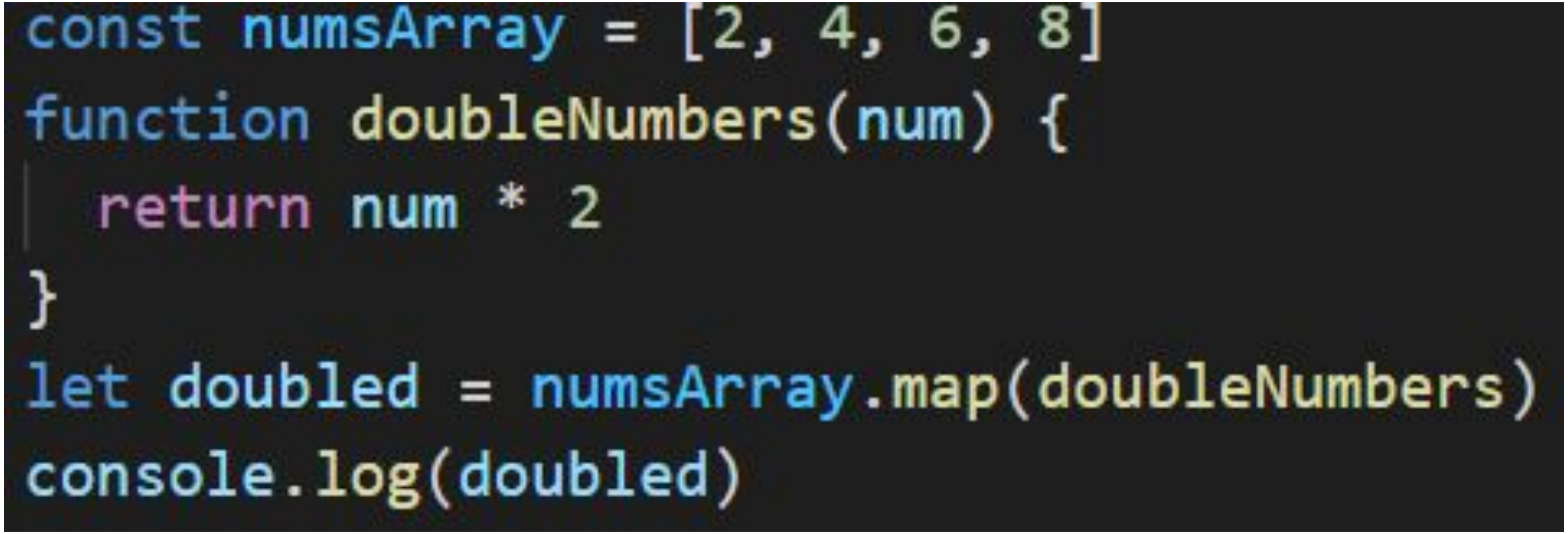


```
let doubled = numsArray.map(doubleNumbers)
console.log(doubled)
```

We assigned a variable the value of a function, and passed that function another function as an argument.

What are higher-order functions?

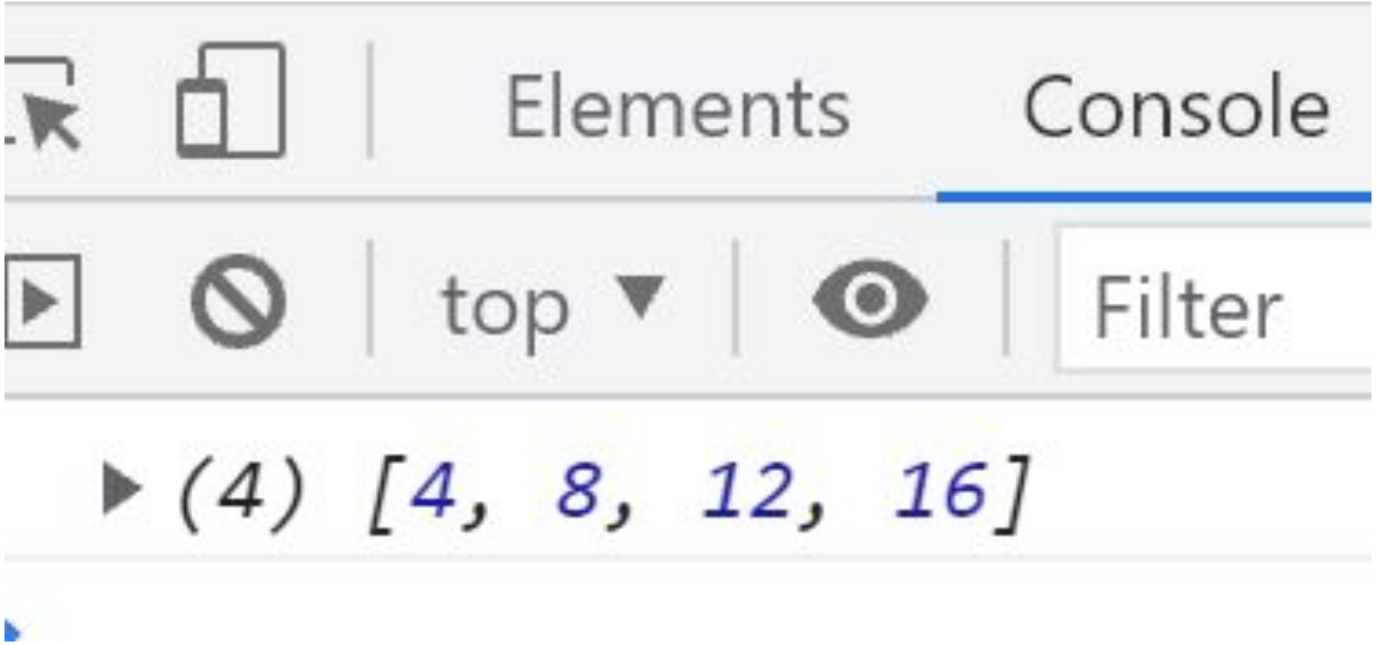
In this example, map is a higher-order function.



```
const numsArray = [2, 4, 6, 8]
function doubleNumbers(num) {
  return num * 2
}
let doubled = numsArray.map(doubleNumbers)
console.log(doubled)
```

What are higher-order functions?

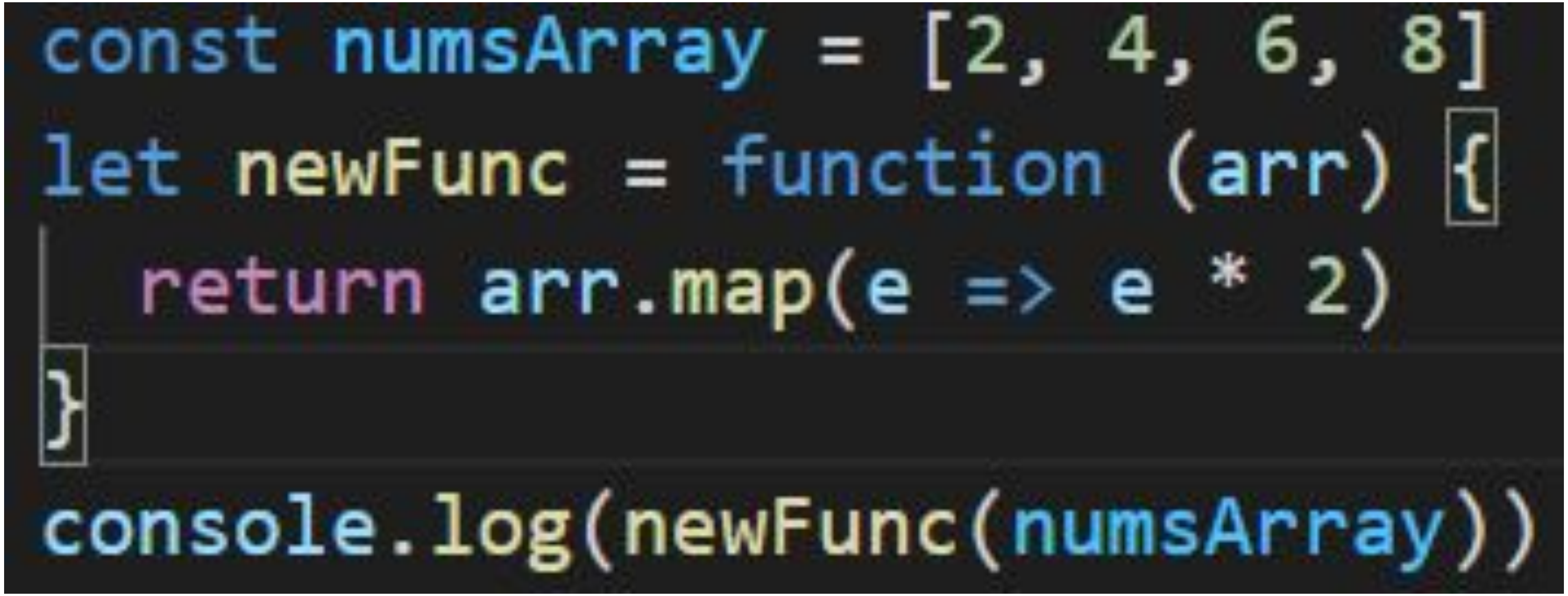
Ultimately, we’ll see this in our console.



```
▶ (4) [4, 8, 12, 16]
```

What are higher-order functions?

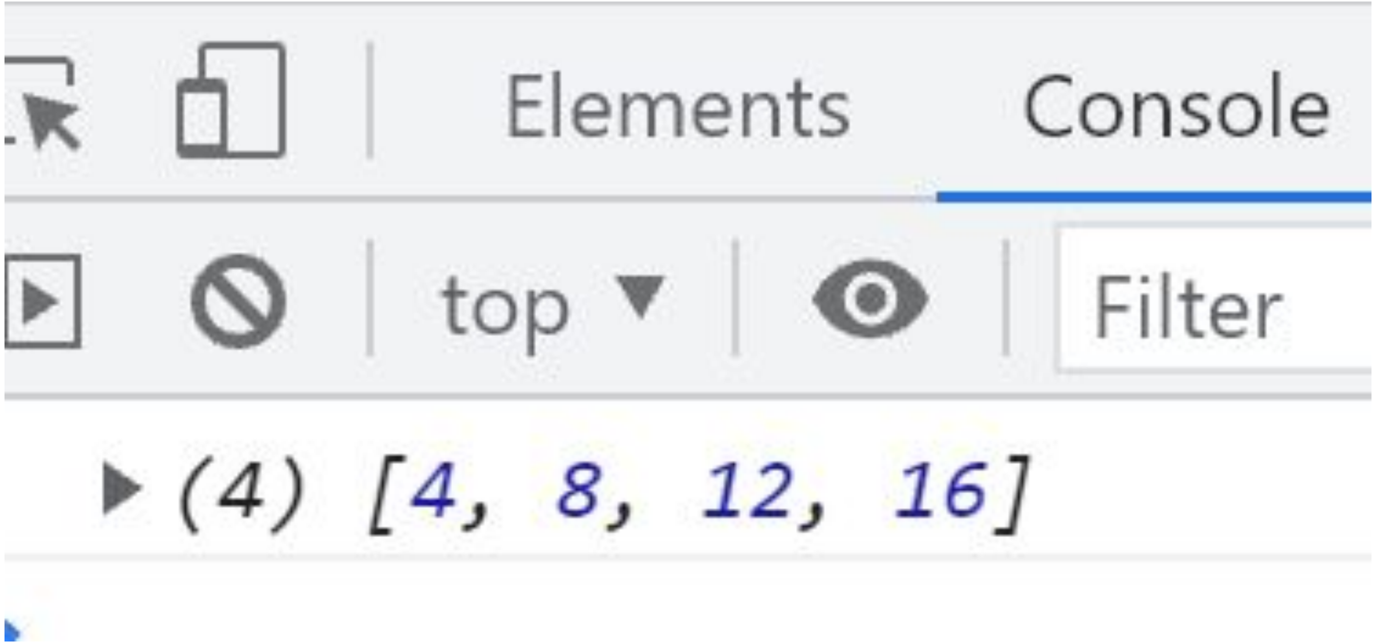
We can write the same thing this way:



```
const numsArray = [2, 4, 6, 8]
let newFunc = function (arr) {
    return arr.map(e => e * 2)
}
console.log(newFunc(numsArray))
```

What are higher-order functions?

We’ll see the same thing in our console.



```
▸ (4) [4, 8, 12, 16]
```

What are higher-order functions?

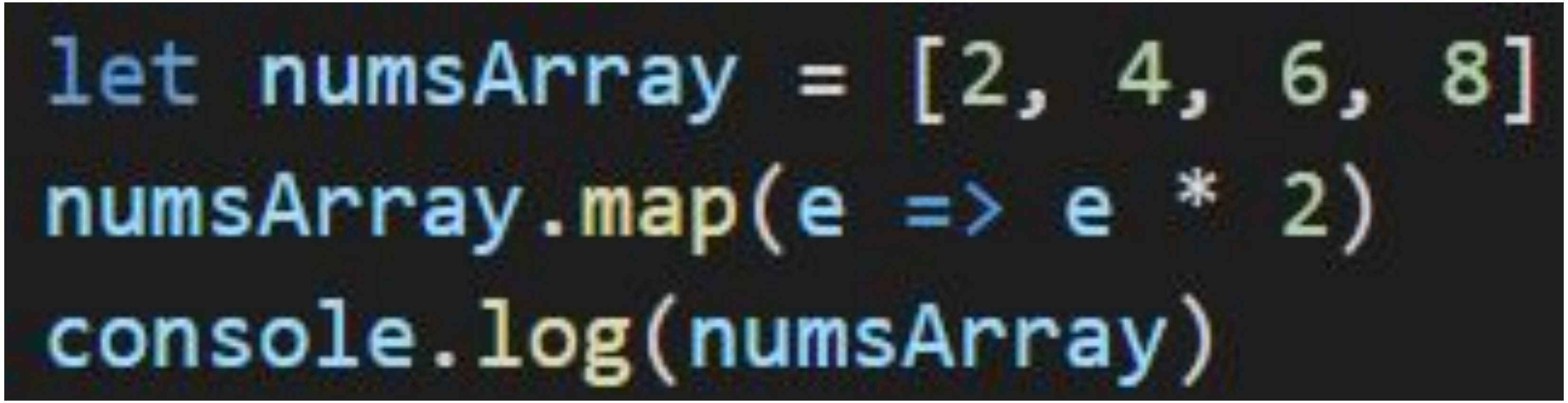
Common array methods such as .map(), .filter(), .reduce(), .some() and .every() all are higher-order functions because they take a function as an argument. There are many such functions in JS, and we can create our own higher-order functions.

How does .map() work?

The .map() method creates a new array, which is a modified copy of the original array, from the results of calling a function on each element in the calling array. We provide the function.

How does .map() work?

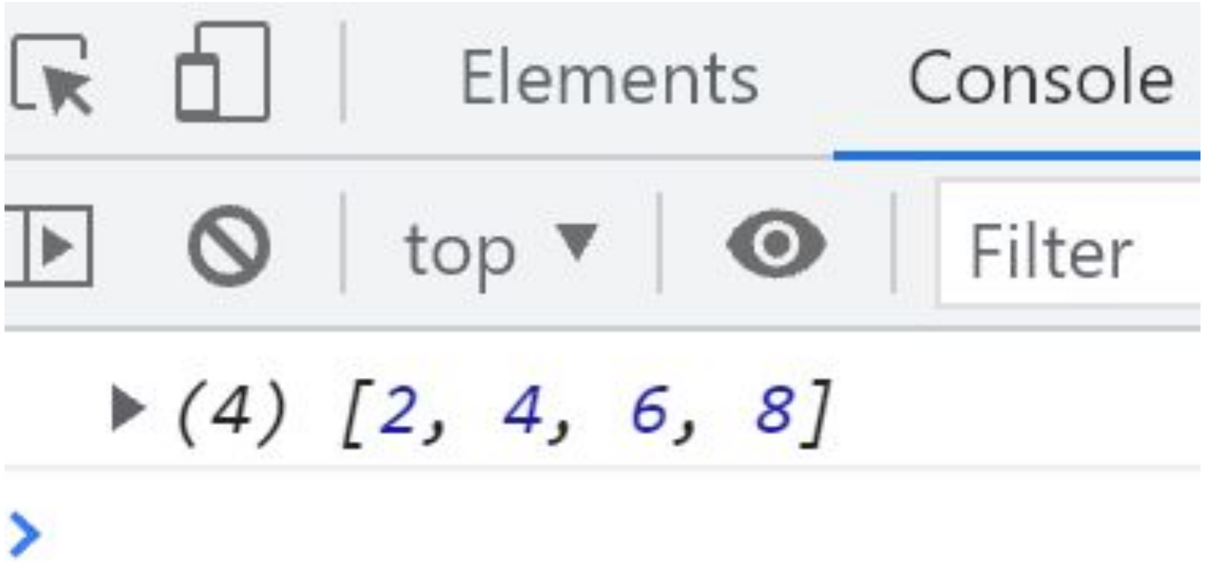
What will we see in our console?



```
let numsArray = [2, 4, 6, 8]
numsArray.map(e => e * 2)
console.log(numsArray)
```

How does .map() work?

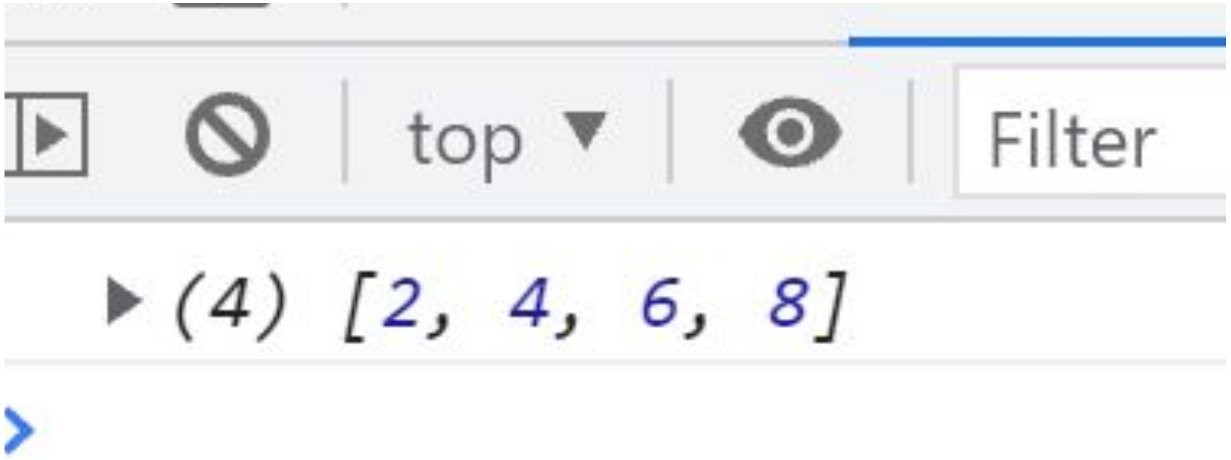
We’ll get this… But why?



```
▸ (4) [2, 4, 6, 8]
>
```

How does .map() work?

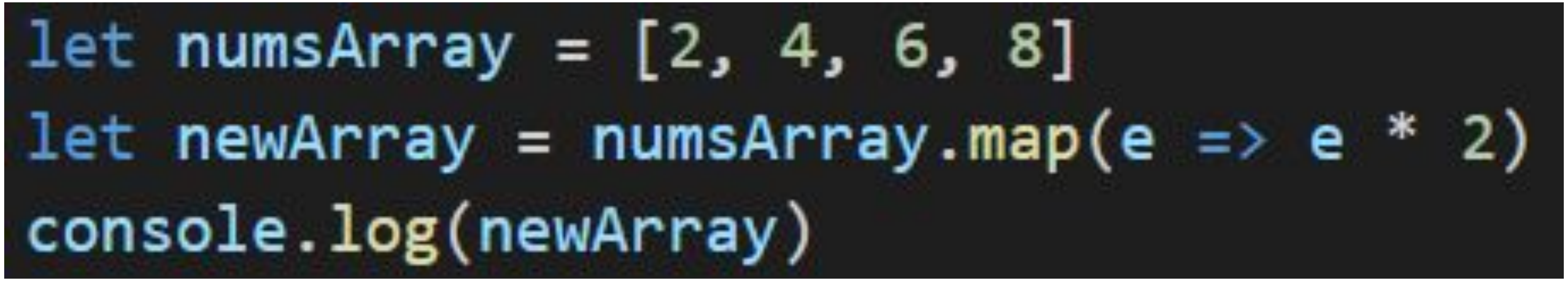
We get this because .map() creates a new array. We need a place to store that array.



```
▶ (4) [2, 4, 6, 8]
>
```

How does .map() work?

This method returns a new array. It doesn't change the original array.

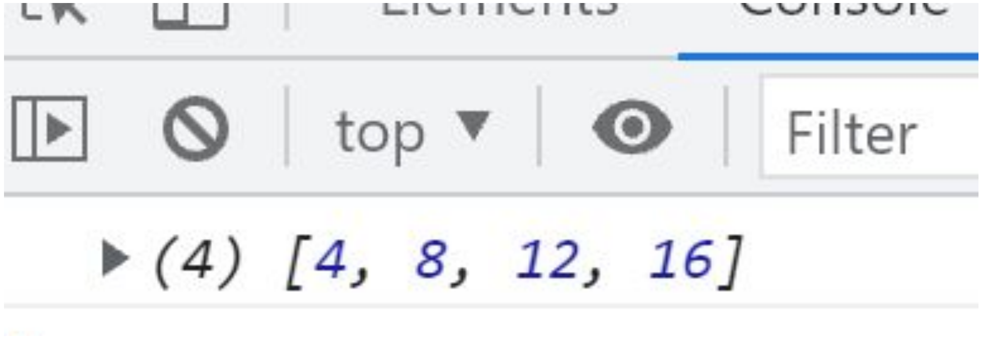


```
let numsArray = [2, 4, 6, 8]
let newArray = numsArray.map(e => e * 2)
console.log(newArray)
```

Our new array is the result of the map function call. Here we assign it to newArray.

How does .map() work?

Here are the results of the new array created by the function that we passed as an argument to the .map() method.



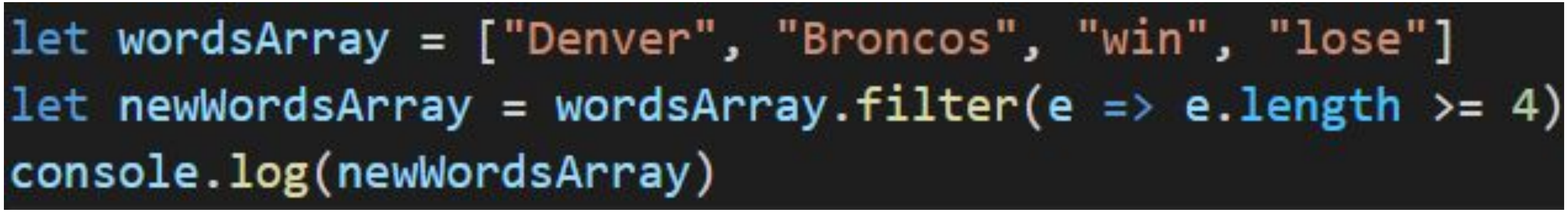
```
▶ (4) [4, 8, 12, 16]
```

How does .filter() work?

It works essentially the same way as .map(), except the new array it creates won’t necessarily be the same size as the original array. The new array will only include elements that pass the test we create.

How does .filter() work?

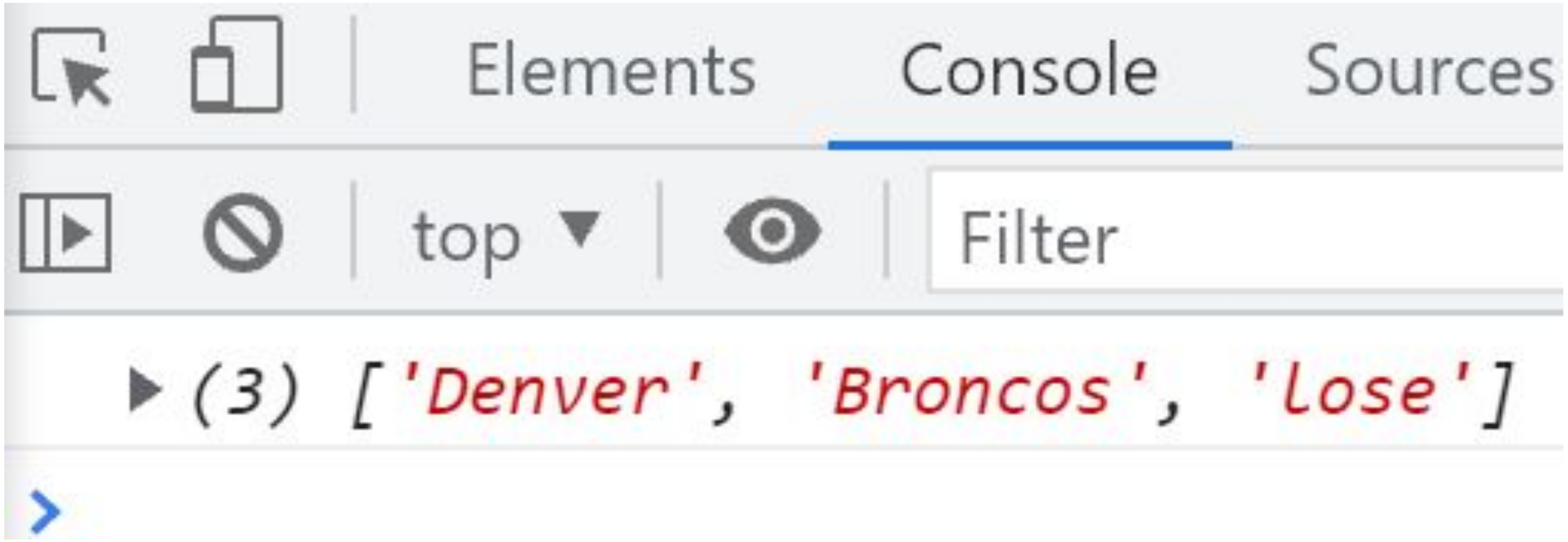
What will we see in our console?



```
let wordsArray = ["Denver", "Broncos", "win", "lose"]
let newWordsArray = wordsArray.filter(e => e.length >= 4)
console.log(newWordsArray)
```

How does .filter() work?

We will get this:



```
▸ (3) ['Denver', 'Broncos', 'Lose']
>
```

How does .reduce() work?

Just like the other array methods we’ve talked about, .reduce() will apply a function call to each element in an array. It will return a single value. In this case, we provide a “reducer” function.

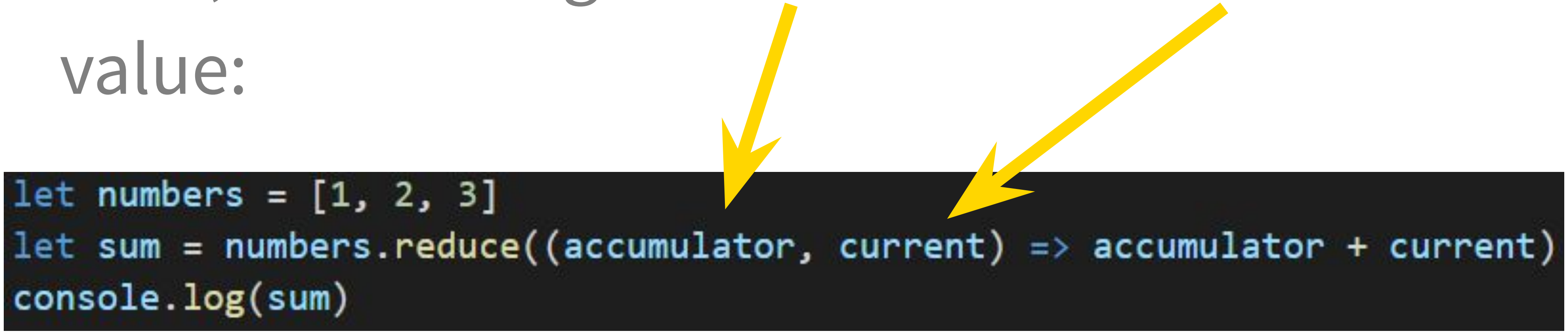
How does .reduce() work?

The reducer function can take four arguments:

- Accumulator
- Current value
- currentIndex - optional
- Array - optional

How does .reduce() work?

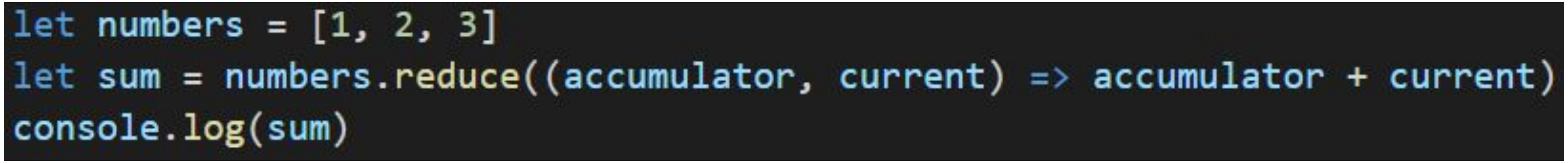
Here, we're using accumulator and current value:



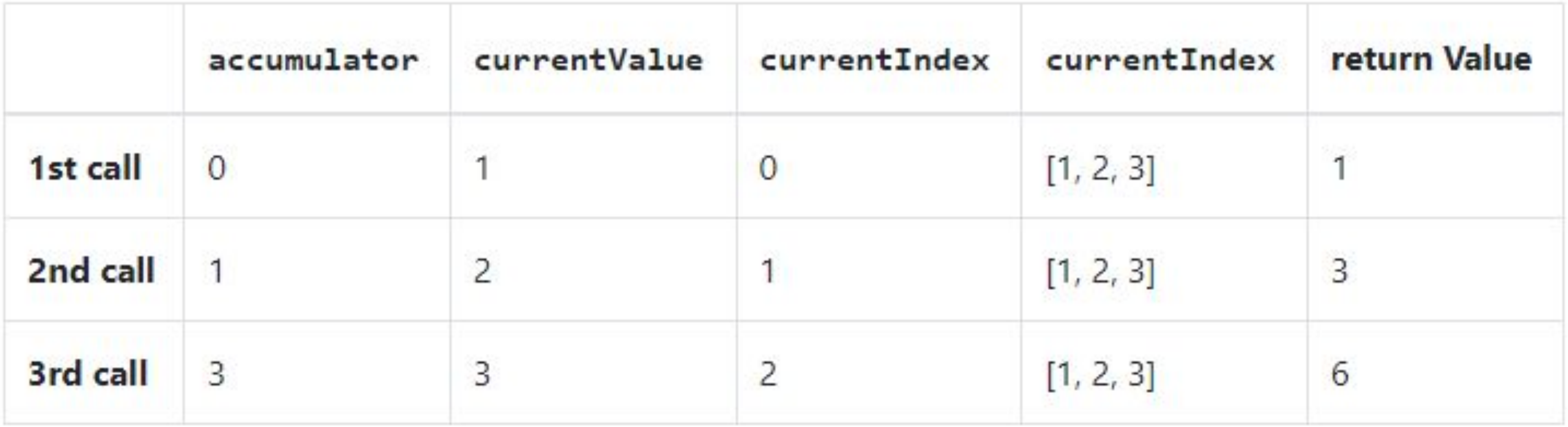
```
let numbers = [1, 2, 3]
let sum = numbers.reduce((accumulator, current) => accumulator + current)
console.log(sum)
```

How does this work?

How does .reduce() work?



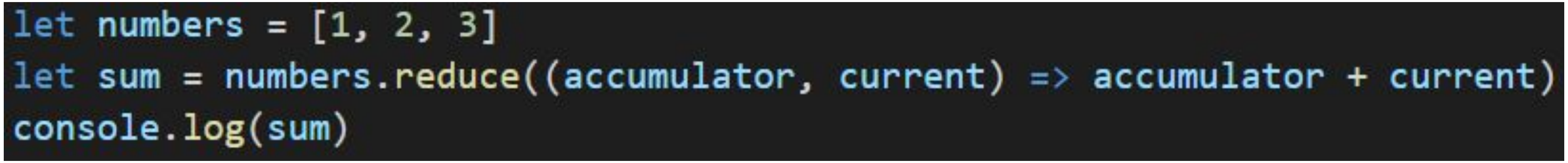
```
let numbers = [1, 2, 3]
let sum = numbers.reduce((accumulator, current) => accumulator + current)
console.log(sum)
```



	accumulator	currentValue	currentIndex	currentIndex	return Value
1st call	0	1	0	[1, 2, 3]	1
2nd call	1	2	1	[1, 2, 3]	3
3rd call	3	3	2	[1, 2, 3]	6

How does .reduce() work?

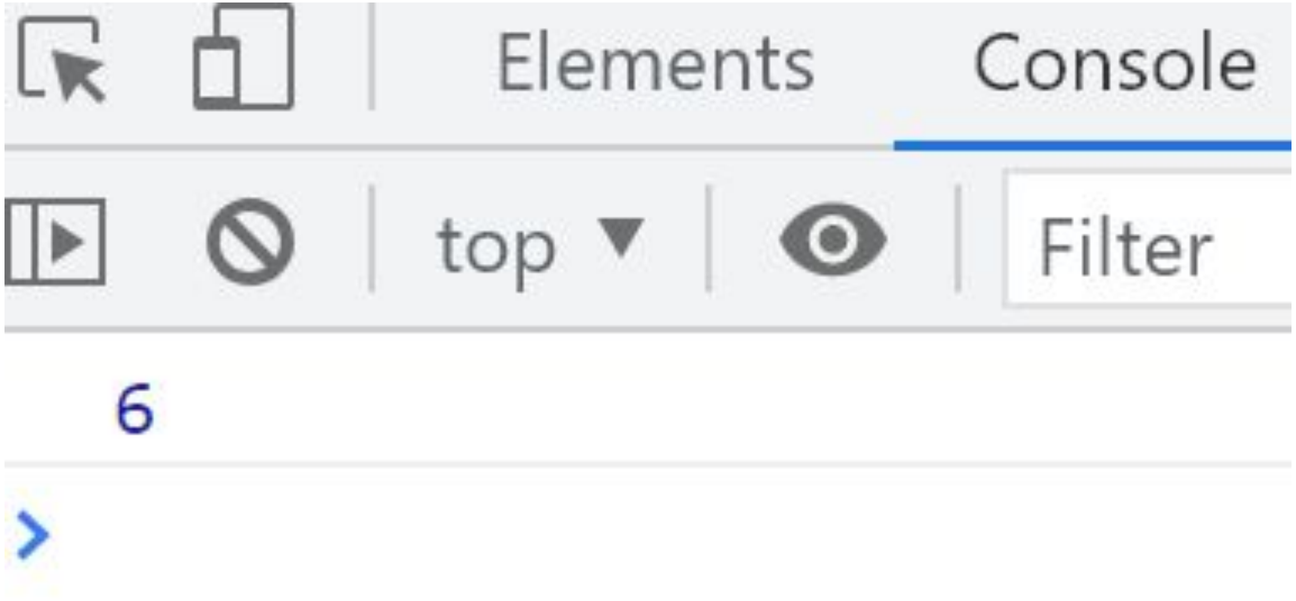
What will this code display in our console?



```
let numbers = [1, 2, 3]
let sum = numbers.reduce((accumulator, current) => accumulator + current)
console.log(sum)
```

How does .reduce() work?

What will this code display in our console?



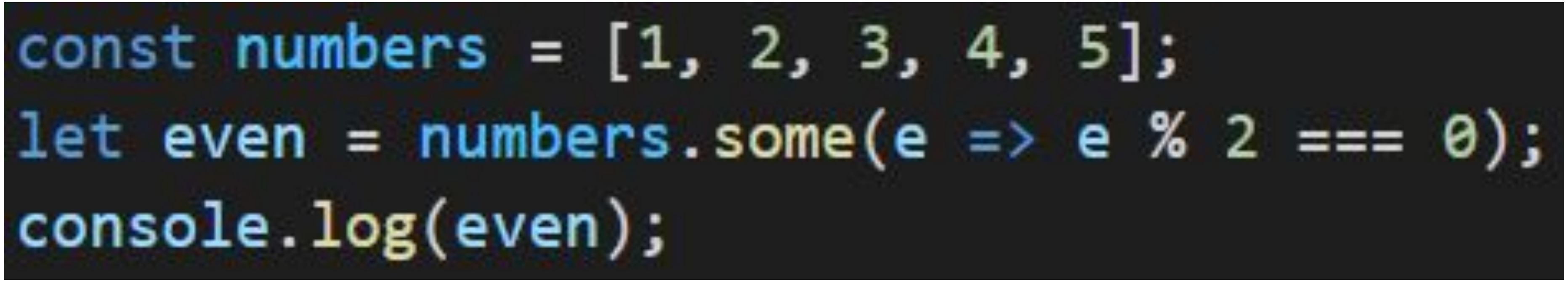
```
6
>
```

How does .some() work?

The .some() method tests whether at least one element in an array passes a test provided by a function. It returns true if, in the array, it finds an element for which the provided function returns true. It doesn’t alter the array.

How does .some() work?

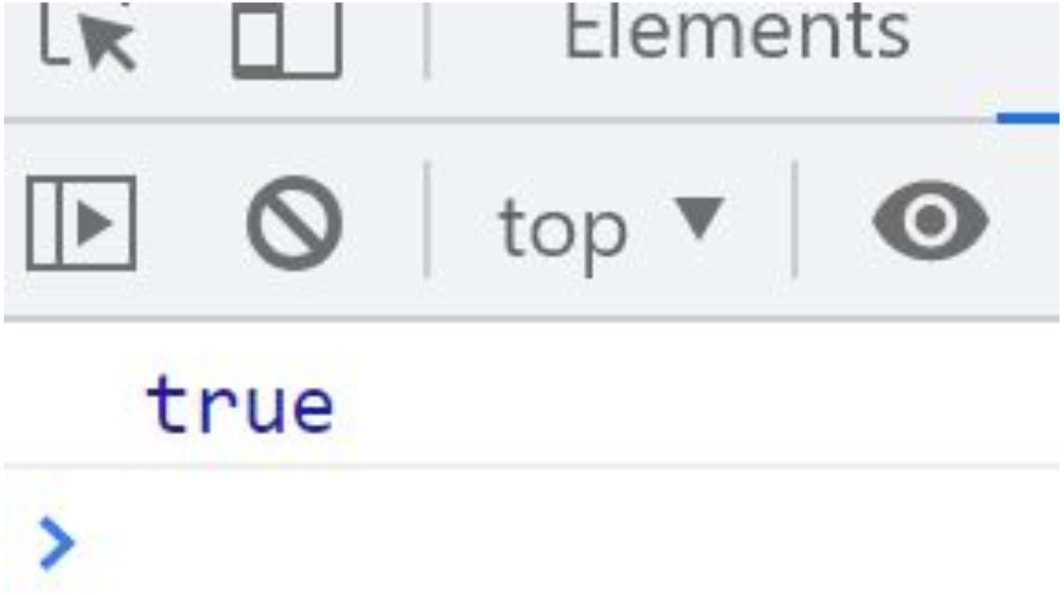
What will we see in our console?



```
const numbers = [1, 2, 3, 4, 5];
let even = numbers.some(e => e % 2 === 0);
console.log(even);
```

How does .some() work?

We’ll get this:



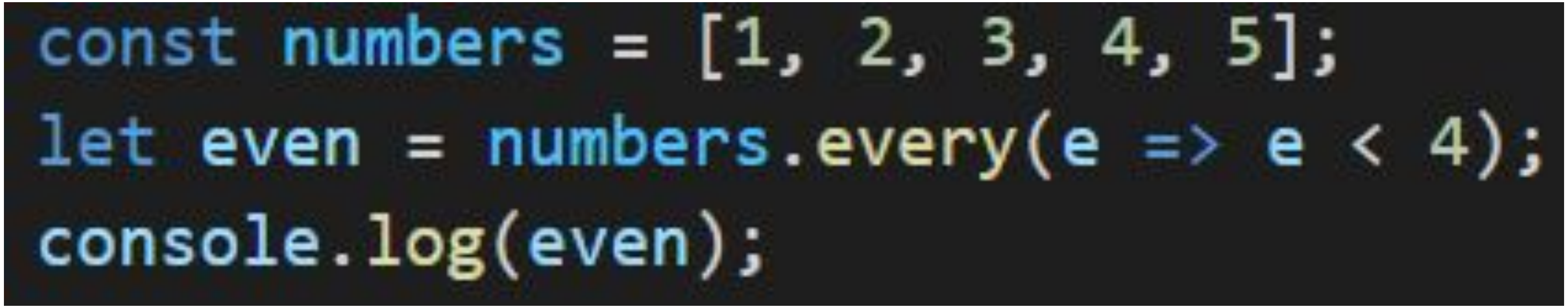
```
true
```

How does .every() work?

Just like the .some() method, .every() tests whether all elements in the array pass the test implemented by the provided function. It returns a boolean value.

How does .every() work?

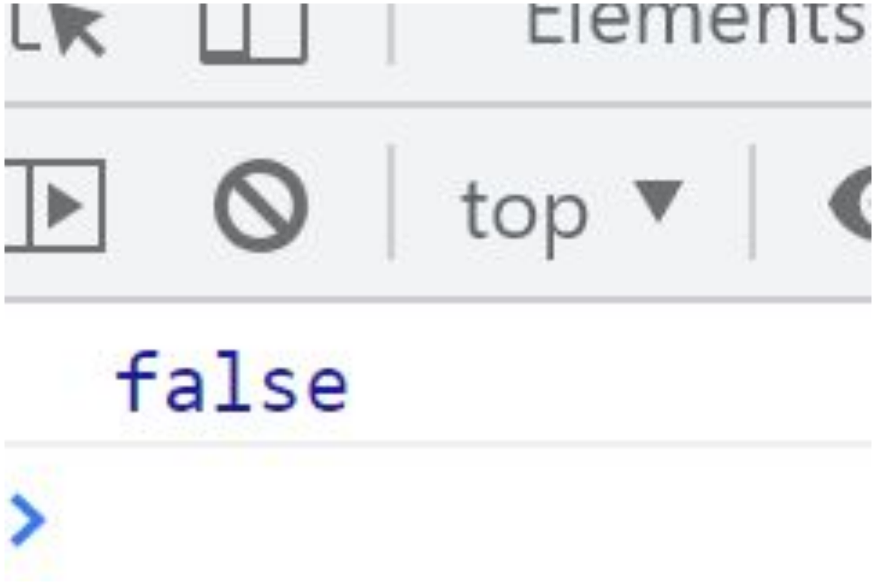
What will we see in our console?



```
const numbers = [1, 2, 3, 4, 5];
let even = numbers.every(e => e < 4);
console.log(even);
```

How does .every() work?

We'll get this:



```
false
>
```