Hip Hip Array, it's Friyay!

Array Methods

Mutating methods

Adding and removing items

- Push
- Pop
- Shift
- Unshift
- Splice

Push/Pop

PUSH

-Adds one (or more) elements to the **end** of the array

-Returns **new length of the array**

```
array.push('item')
```

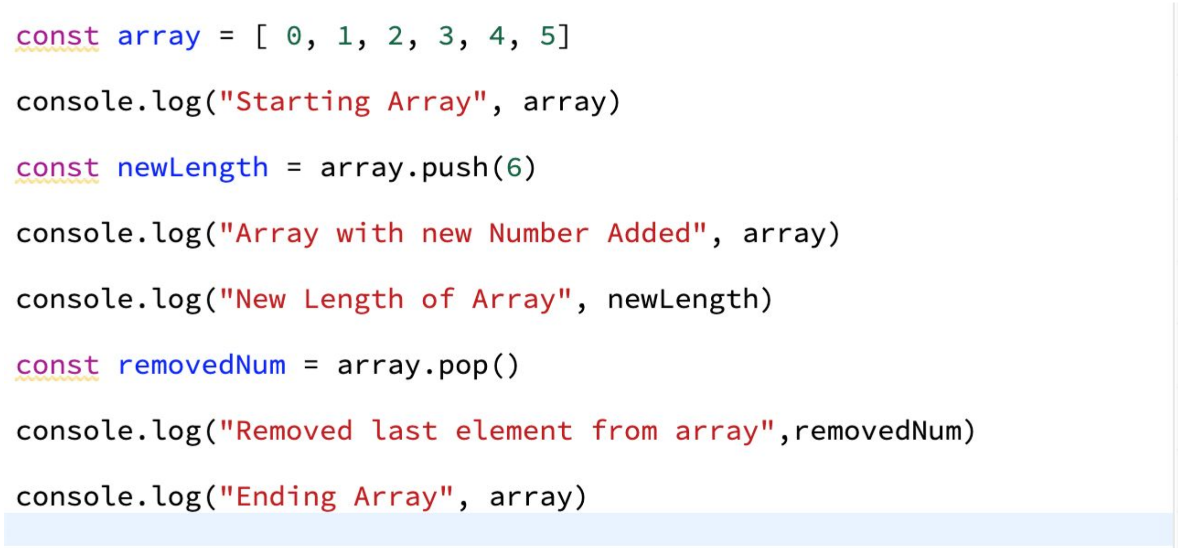
POP

-Removes the **last** element from the array

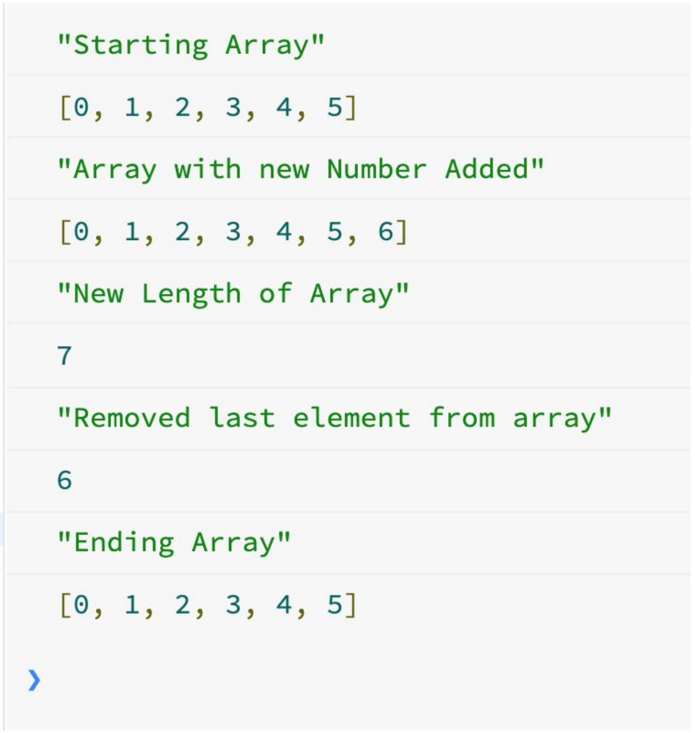
-Returns that element

```
array.pop()
```

See 'em in action



```
const array = [ 0, 1, 2, 3, 4, 5]
console.log("Starting Array", array)
const newLength = array.push(6)
console.log("Array with new Number Added", array)
console.log("New Length of Array", newLength)
const removedNum = array.pop()
console.log("Removed last element from array",removedNum)
console.log("Ending Array", array)
```



```
"Starting Array"
[0, 1, 2, 3, 4, 5]
"Array with new Number Added"
[0, 1, 2, 3, 4, 5, 6]
"New Length of Array"
7
"Removed last element from array"
6
"Ending Array"
[0, 1, 2, 3, 4, 5]
❯
```

Shift/Unshift

SHIFT

-Removes the **first** element from the array

-Returns that element

`array.shift()`

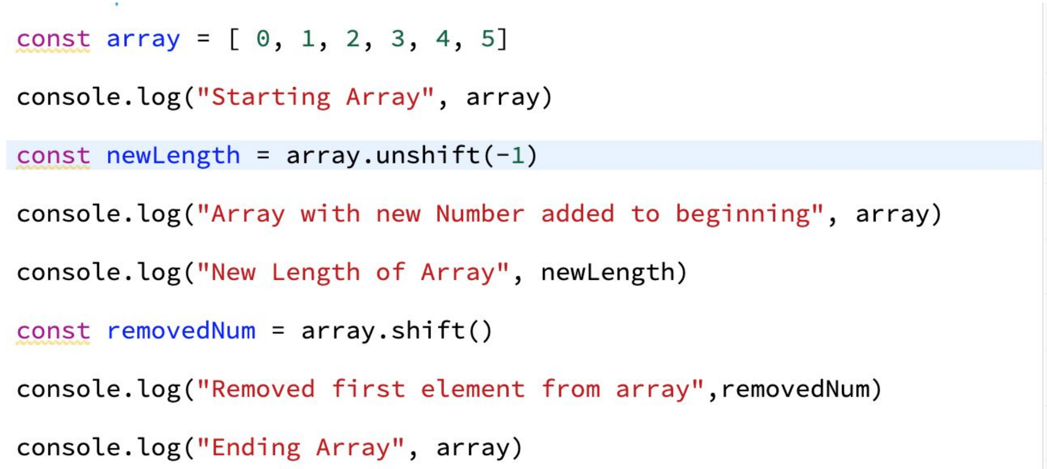
UNSHIFT

-Adds one or more elements to the **beginning** of the array

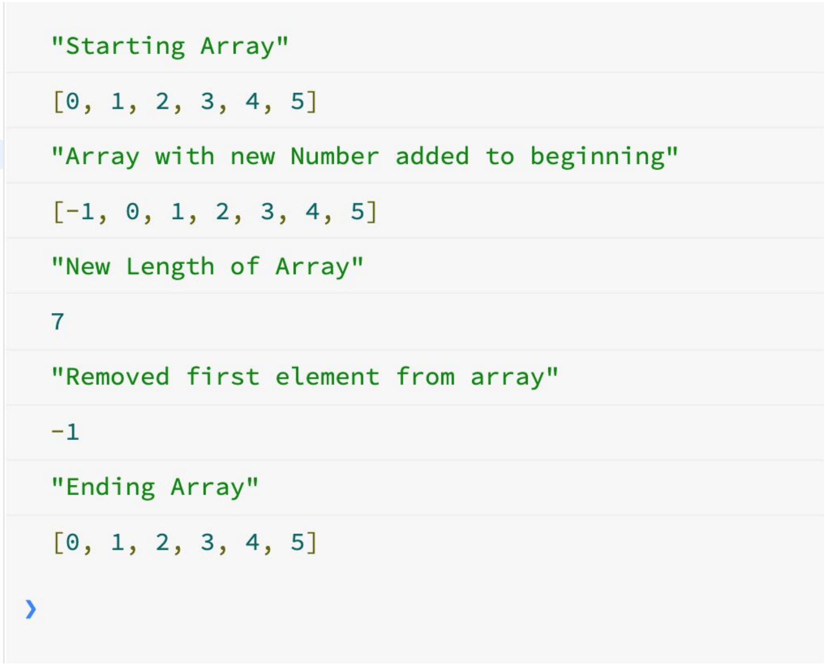
-Returns the new length of the array

`array.unshift(‘item’)`

See 'em in Action



```
const array = [ 0, 1, 2, 3, 4, 5]
console.log("Starting Array", array)
const newLength = array.unshift(-1)
console.log("Array with new Number added to beginning", array)
console.log("New Length of Array", newLength)
const removedNum = array.shift()
console.log("Removed first element from array",removedNum)
console.log("Ending Array", array)
```



```
"Starting Array"
[0, 1, 2, 3, 4, 5]
"Array with new Number added to beginning"
[-1, 0, 1, 2, 3, 4, 5]
"New Length of Array"
7
"Removed first element from array"
-1
"Ending Array"
[0, 1, 2, 3, 4, 5]
❯
```

Splice

Can do a lot!

- remove elements
- replace existing elements
- add new element(s) in place

Takes up to 3 optional arguments

Returns the removed element

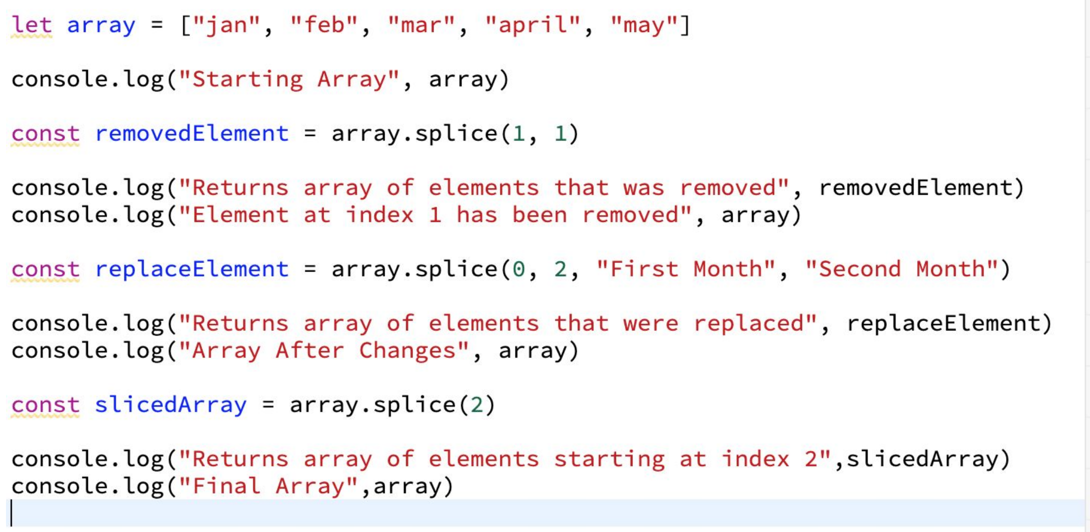
```
array.splice(x, y, z)
```

x: index at which to start changing the array

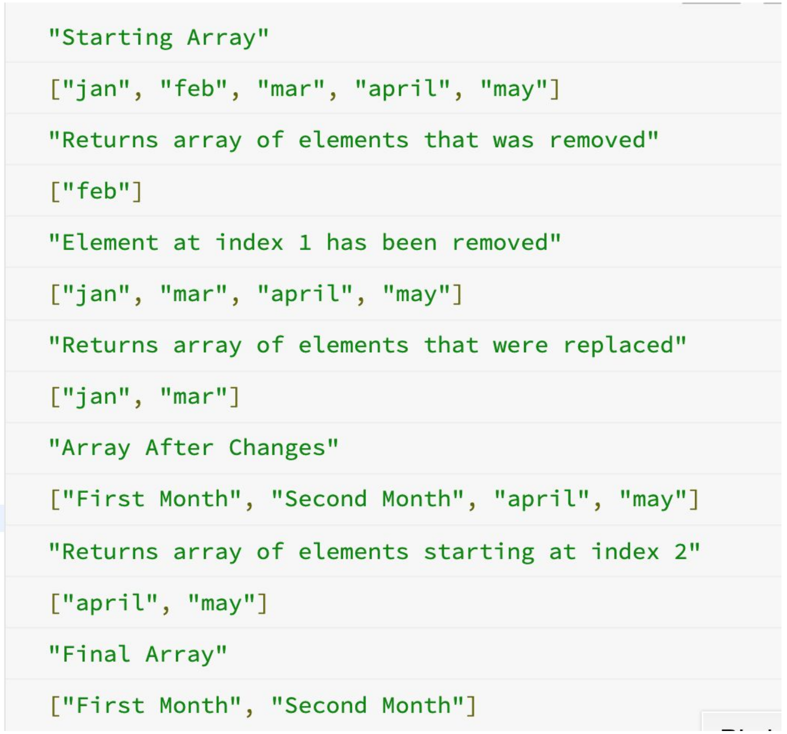
y: number of items to remove

z: item(s) to add to the array

See it in action



```
let array = ["jan", "feb", "mar", "april", "may"]

console.log("Starting Array", array)

const removedElement = array.splice(1, 1)

console.log("Returns array of elements that was removed", removedElement)
console.log("Element at index 1 has been removed", array)

const replaceElement = array.splice(0, 2, "First Month", "Second Month")

console.log("Returns array of elements that were replaced", replaceElement)
console.log("Array After Changes", array)

const slicedArray = array.splice(2)

console.log("Returns array of elements starting at index 2",slicedArray)
console.log("Final Array",array)
```



```
"Starting Array"
["jan", "feb", "mar", "april", "may"]
"Returns array of elements that was removed"
["feb"]
"Element at index 1 has been removed"
["jan", "mar", "april", "may"]
"Returns array of elements that were replaced"
["jan", "mar"]
"Array After Changes"
["First Month", "Second Month", "april", "may"]
"Returns array of elements starting at index 2"
["april", "may"]
"Final Array"
["First Month", "Second Month"]
```

Changing the order of arrays

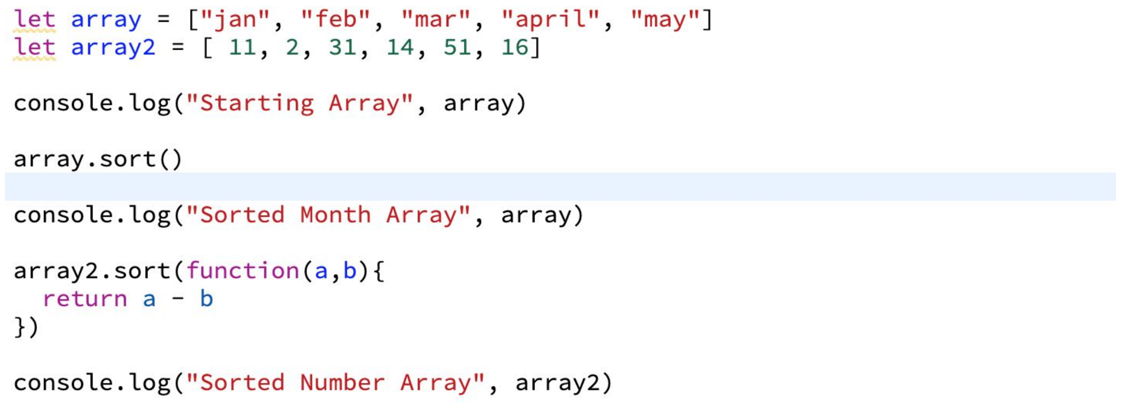
SORT

Sorts the elements in place

Returns sorted array

`array.sort()`

Note: this way sorts by UTF character, not numerically



```
let array = ["jan", "feb", "mar", "april", "may"]
let array2 = [ 11, 2, 31, 14, 51, 16]

console.log("Starting Array", array)

array.sort()

console.log("Sorted Month Array", array)

array2.sort(function(a,b){
  return a - b
})

console.log("Sorted Number Array", array2)
```

P.S. sort() can take additional arguments to sort by other means. We'll dive into that next week

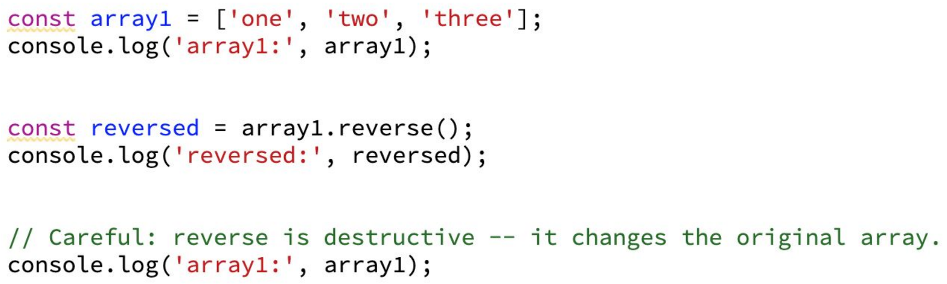
Changing the order of arrays

REVERSE

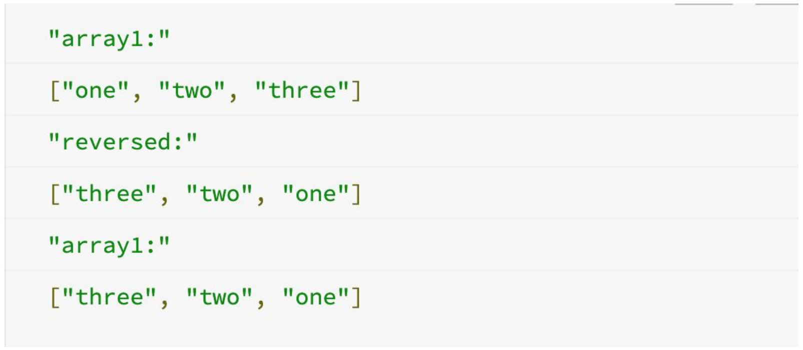
Reverses the elements of the array in place

`array.reverse()`

Returns reversed array



```
const array1 = ['one', 'two', 'three'];
console.log('array1:', array1);


const reversed = array1.reverse();
console.log('reversed:', reversed);


// Careful: reverse is destructive -- it changes the original array.
console.log('array1:', array1);
```



```
"array1:"
["one", "two", "three"]
"reversed:"
["three", "two", "one"]
"array1:"
["three", "two", "one"]
```

Join

Joins all elements of an array into a string

Returns new string

Syntax: array.join(separator)

```
    test_array = [‘I’, ‘love’, ‘cheese’]

    test_array.join(‘ ‘) ->  ‘I love cheese’

    test_array.join(‘+’) -> ‘I+love+cheese’
```

doesn’t actually mutate the array

Non-Mutating Methods

Generally...

Most array methods just perform a function on each element of the array (a callback method)

Checking Arrays

EVERY

Tests if **all** elements in array meet condition by provided function

Caution: empty arrays will always return true

SOME

Tests if **at least one** element in the array meets the provided condition

Caution: empty arrays will always return false

Caution 2: callback must return something

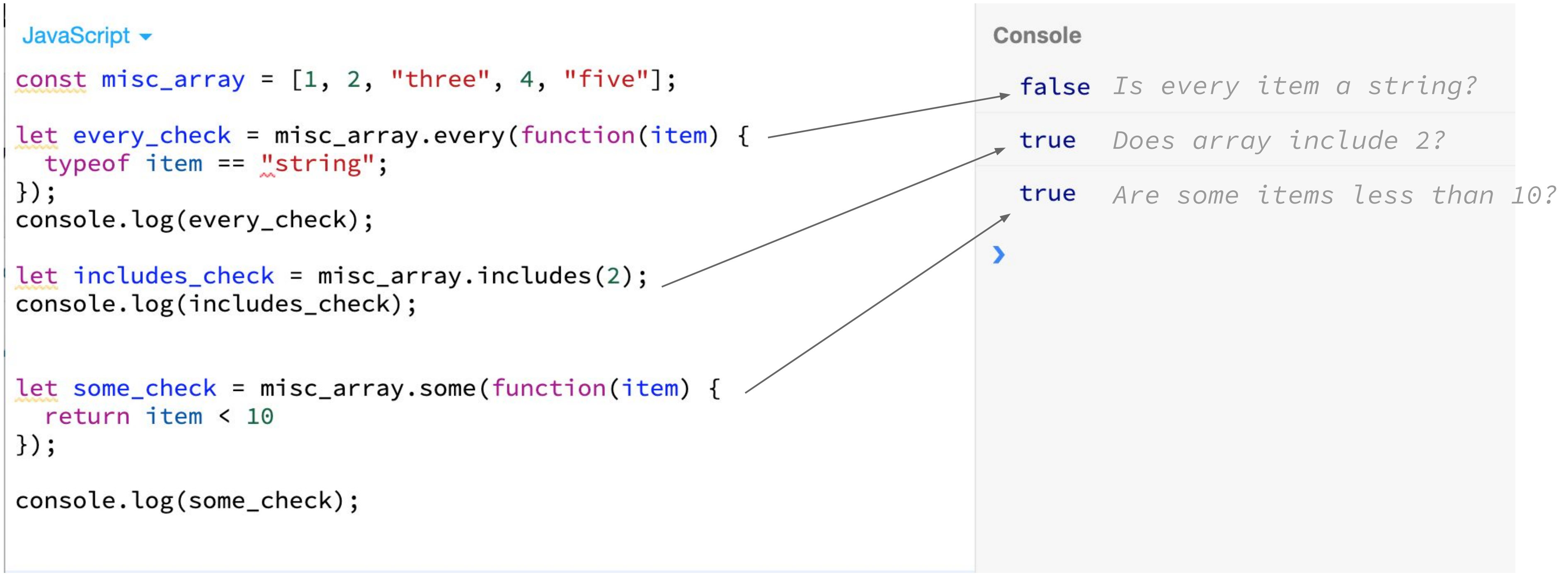
INCLUDES

Determines whether the array contains the value provided

Note: checks for each item to equal the passed argument (no callback method)

All of these return a boolean (true or false)

See em in action



```
const misc_array = [1, 2, "three", 4, "five"];

let every_check = misc_array.every(function(item) {
  typeof item == "string";
});
console.log(every_check);

let includes_check = misc_array.includes(2);
console.log(includes_check);


let some_check = misc_array.some(function(item) {
  return item < 10
});

console.log(some_check);
```

```
false   Is every item a string?
true    Does array include 2?
true    Are some items less than 10?
```

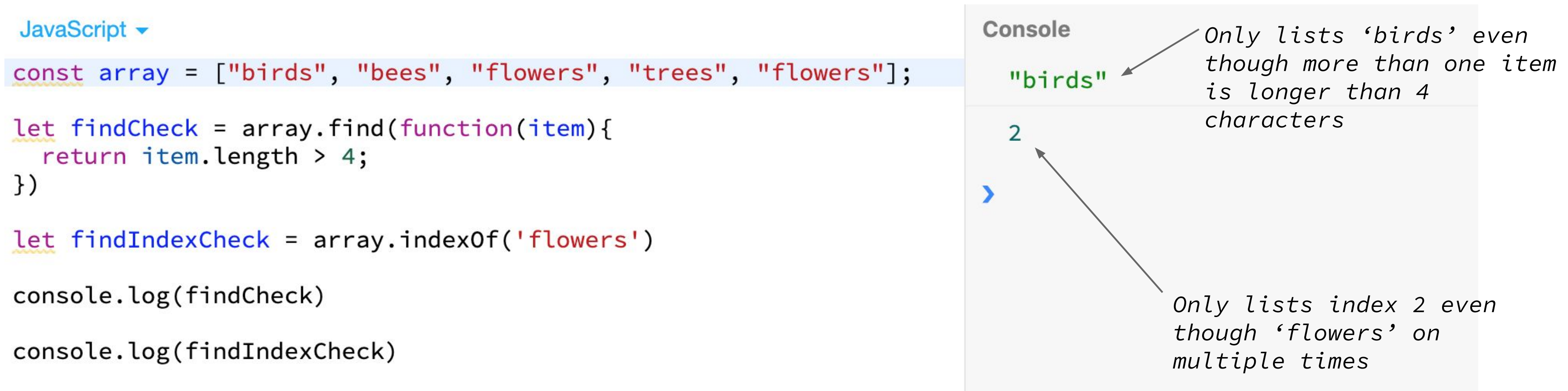
Finding stuff in arrays

FIND

Returns the **value** of the **first element** that meets the testing callback function

FINDINDEX

Returns the **index** of the **first element** in the array that satisfies the testing function



```
const array = ["birds", "bees", "flowers", "trees", "flowers"];

let findCheck = array.find(function(item){
  return item.length > 4;
})

let findIndexCheck = array.indexOf('flowers')

console.log(findCheck)

console.log(findIndexCheck)
```

```
"birds"
2
```

Only lists ‘birds’ even though more than one item is longer than 4 characters

Only lists index 2 even though ‘flowers’ on multiple times

More Methods To master

forEach

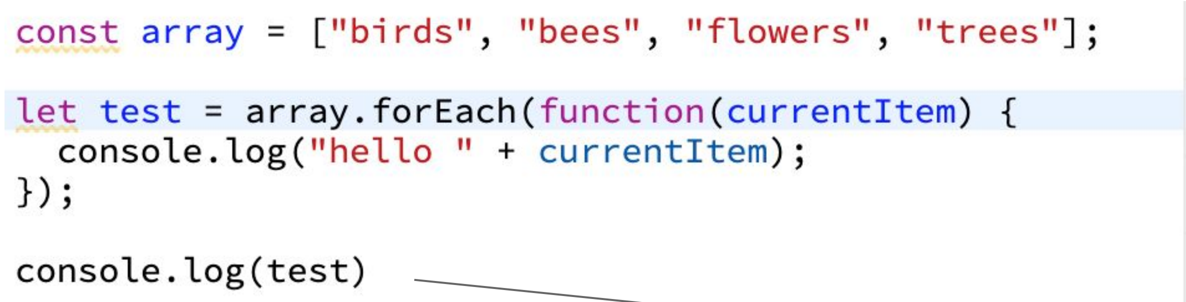
-Executes a provided function once for element in the array

-Returns undefined

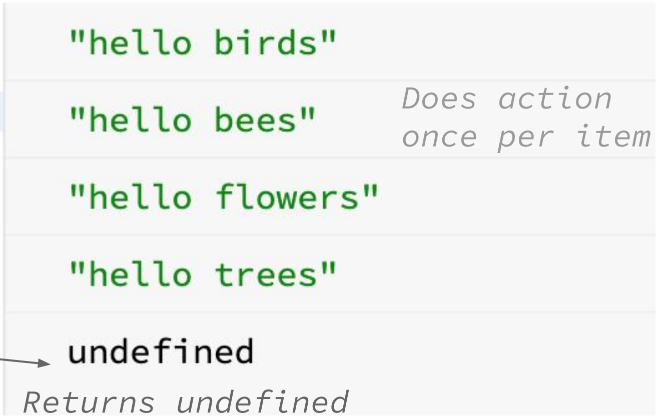
SYNTAX

```
array.forEach(function(currentItem) {
    action
    });
```

Note: this is simplest way. Can also take second argument, index, which is the currentItem's index



```
const array = ["birds", "bees", "flowers", "trees"];

let test = array.forEach(function(currentItem) {
  console.log("hello " + currentItem);
});

console.log(test)
```



```
"hello birds"
"hello bees"
"hello flowers"
"hello trees"
undefined
```

Does action once per item

Returns undefined

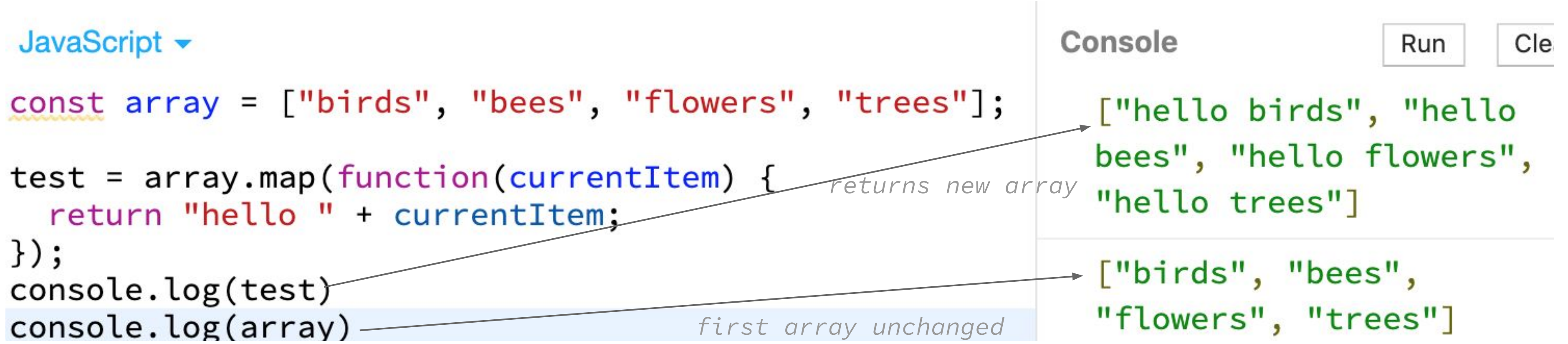
Map

Creates a new array of populated with results of calling provided function on every element of the previous array

Returns a new array

SYNTAX

```
array.map(function(currentItem){
     action
     });
```



```javascript
const array = ["birds", "bees", "flowers", "trees"];

test = array.map(function(currentItem) {
  return "hello " + currentItem;
});
console.log(test)
console.log(array)
```

```
["hello birds", "hello bees", "hello flowers", "hello trees"]

["birds", "bees", "flowers", "trees"]
```

Notes about map

DON’T USE MAP

- If you’re not using the returned array
- If you’re not returning a value from the callback

OTHER ARGUMENTS

- Map’s callback function can also take index as an argument if you need to access an individual item’s index

Filter

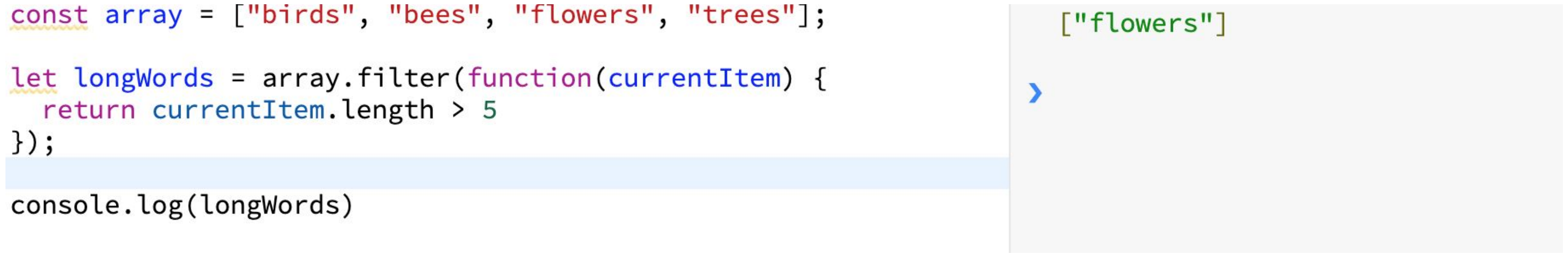
Creates a new array with all elements of the previous array that meet the condition

Callback function must return a Boolean

Returns the new array

SYNTAX

```
array.filter(function(currentItem){
    if(currentItem meets condition){
        return currentItem
        )}
    });
```



```
const array = ["birds", "bees", "flowers", "trees"];

let longWords = array.filter(function(currentItem) {
  return currentItem.length > 5
});

console.log(longWords)
```

```
["flowers"]

›
```

Reduce

Executes a function (called **a reducer**) on each element of the array, resulting in a single output value

Kind of like a ‘for loop’ using the array values to make something new

Reduce

Anatomy of reduce

First, define our callback function (called a reducer)

```
const reducer = (accumulator, currentValue) => accumulator +
currentValue;
```

accumulator: accumulated value previously returned in the
last invocation of the callback (or
initialValue if you give it one)

currentValue: current element of array

Reduce

Anatomy of reduce, part II

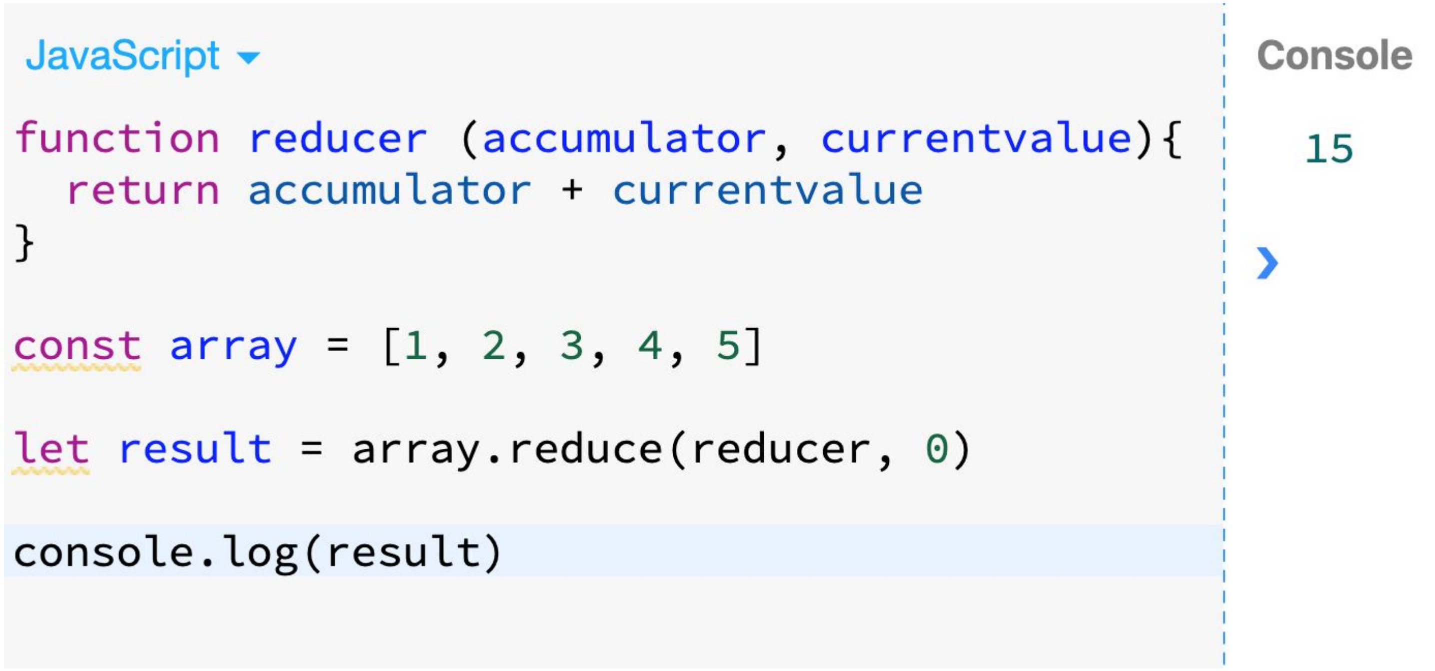
Next, use the reducer function when we ‘reduce’ the array

```
let result = array.reduce(reducer, initialValue);
```

reducer: callback function described in previous slide

initialValue: where ‘accumulator’ starts (optional - default
value is 0)

Reduce - a very simple example



```
function reducer (accumulator, currentvalue){
  return accumulator + currentvalue
}

const array = [1, 2, 3, 4, 5]

let result = array.reduce(reducer, 0)

console.log(result)
```

```
15
```

Reduce

Reduce can add up numbers, but it can also be used to make new arrays, new objects, or new arrays of objects

Basically, it can take an array and transform it into whatever you tell it to become!

Seems simple(?) but pretty powerful

“***reduce*** *is like one of those games where you can grasp the rules in an hour or two but still discover new ways of having fun for years to come.”* - Kristian Poslek

src: https://levelup.gitconnected.com/one-reduce-to-rule-them-all-504e1b790a83

Even More Methods..

```
copyWithin();

fill();

concat();

lastIndexOf();

slice();

toSource();

toString();

toLocaleString();

entries();

keys();

reduceRight();

values();
```

Finally...

It’s okay if you don’t really understand all of these!

The important thing is to know they exist - that way, you can use (and learn more about them) when needed.