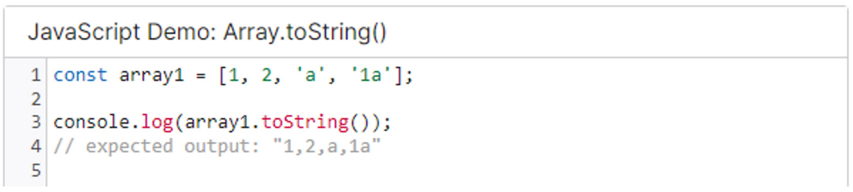
Array Methods

25 November 2022 22:41

1. Array.prototype.toString()

The **toString()** method returns a string representing the specified array and its elements.

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/toString>



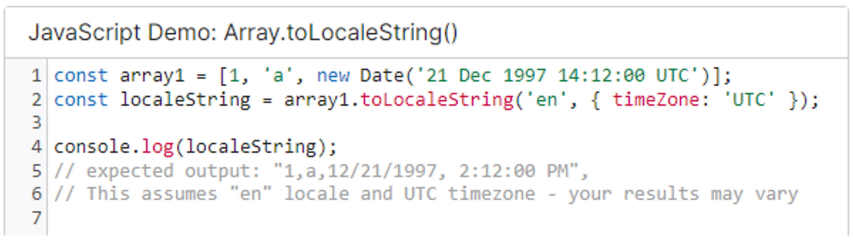
JavaScript Demo: Array.toString()

```javascript
const array1 = [1, 2, 'a', '1a'];

console.log(array1.toString());
// expected output: "1,2,a,1a"

```

2. Array.prototype.toLocaleString()

The **toLocaleString()** method returns a string representing the elements of the array. The elements are converted to Strings using their toLocaleString methods and these Strings are separated by a locale-specific String (such as a comma ",").

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/toLocaleString>



JavaScript Demo: Array.toLocaleString()

```javascript
const array1 = [1, 'a', new Date('21 Dec 1997 14:12:00 UTC')];
const localeString = array1.toLocaleString('en', { timeZone: 'UTC' });

console.log(localeString);
// expected output: "1,a,12/21/1997, 2:12:00 PM",
// This assumes "en" locale and UTC timezone - your results may vary

```

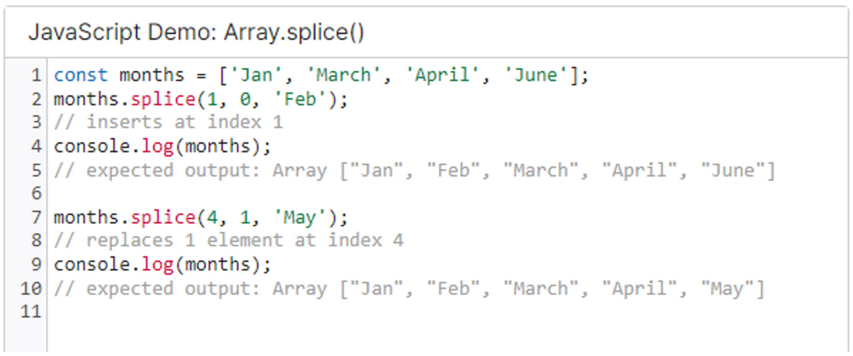
3. Array.prototype.splice()

The **splice()** method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place. To access part of an array without modifying it, see slice().

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice>

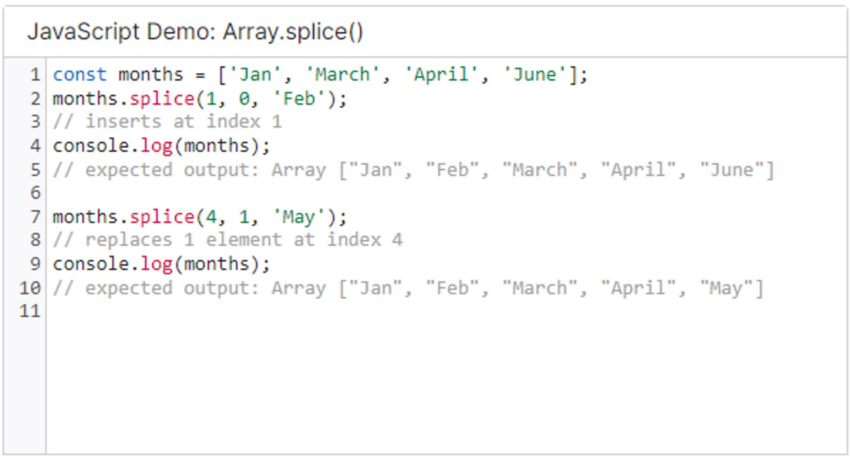
Syntax#

```javascript
splice(start)
splice(start, deleteCount)
splice(start, deleteCount, item1)
splice(start, deleteCount, item1, item2, itemN)
```



JavaScript Demo: Array.splice()

```javascript
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
// inserts at index 1
console.log(months);
// expected output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');
// replaces 1 element at index 4
console.log(months);
// expected output: Array ["Jan", "Feb", "March", "April", "May"]

```

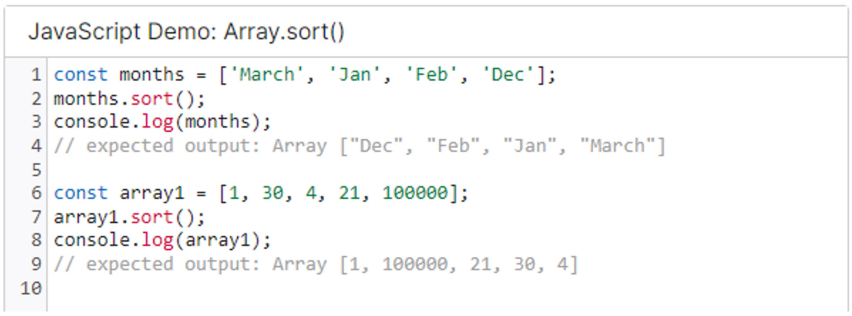

JavaScript Demo: Array.splice()

```javascript
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
// inserts at index 1
console.log(months);
// expected output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');
// replaces 1 element at index 4
console.log(months);
// expected output: Array ["Jan", "Feb", "March", "April", "May"]

```

4. Array.prototype.sort()

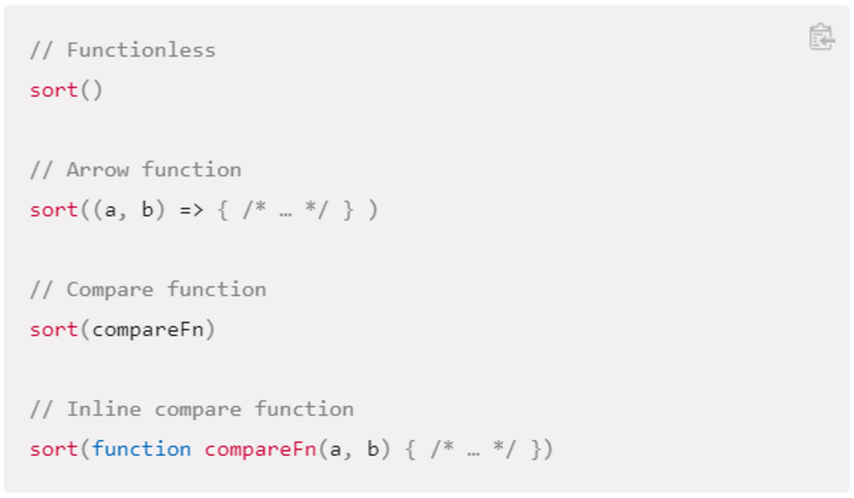
The **sort()** method sorts the elements of an array *in place* and returns the reference to the same array, now sorted. The default sort order is ascending, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values. The time and space complexity of the sort cannot be guaranteed as it depends on the implementation.

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort>


JavaScript Demo: Array.sort()

```javascript
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months);
// expected output: Array ["Dec", "Feb", "Jan", "March"]

const array1 = [1, 30, 4, 21, 100000];
array1.sort();
console.log(array1);
// expected output: Array [1, 100000, 21, 30, 4]

```

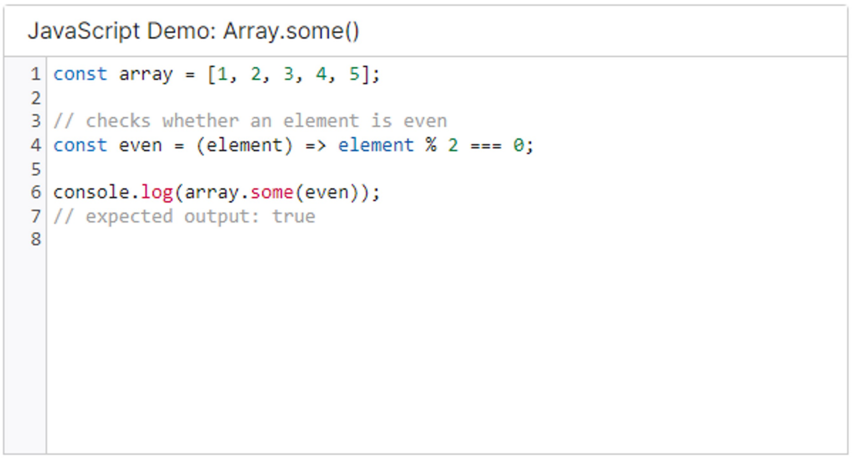
Syntax


```javascript
// Functionless
sort()

// Arrow function
sort((a, b) => { /* … */ } )

// Compare function
sort(compareFn)

// Inline compare function
sort(function compareFn(a, b) { /* … */ })
```

5. Array.prototype.some()

The **some()** method tests whether at least one element in the array passes the test implemented by the provided function. It returns true if, in the array, it finds an element for which the provided function returns true; otherwise it returns false. It doesn't modify the array.

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/some>

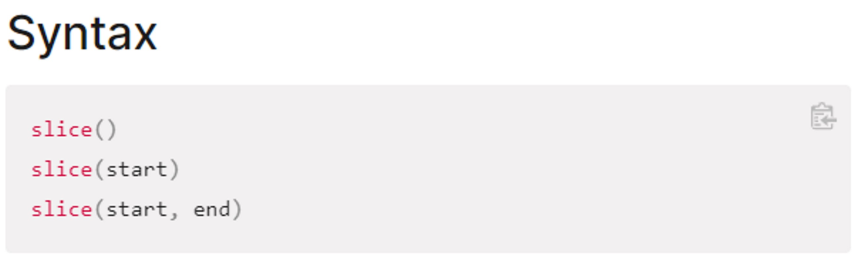

JavaScript Demo: Array.some()

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

// checks whether an element is even
const even = (element) => element % 2 === 0;

console.log(array.some(even));
// expected output: true

```

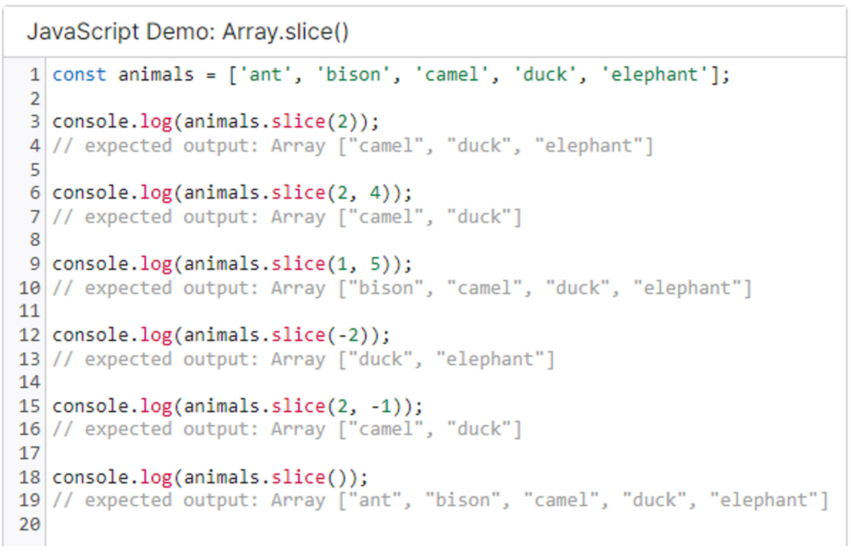
6. Array.prototype.slice()

The **slice()** method returns a shallow copy of a portion of an array into a new array object selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be modified.

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice>


Syntax

```
slice()
slice(start)
slice(start, end)
```

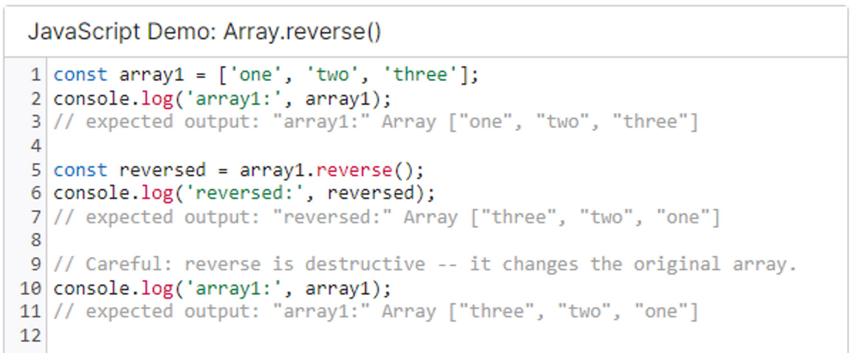

JavaScript Demo: Array.slice()

```
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));
// expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// expected output: Array ["camel", "duck"]

console.log(animals.slice(1, 5));
// expected output: Array ["bison", "camel", "duck", "elephant"]

console.log(animals.slice(-2));
// expected output: Array ["duck", "elephant"]

console.log(animals.slice(2, -1));
// expected output: Array ["camel", "duck"]

console.log(animals.slice());
// expected output: Array ["ant", "bison", "camel", "duck", "elephant"]

```

7. Array.prototype.reverse()

The **reverse()** method reverses an array *in place* and returns the reference to the same array, the first array element now becoming the last, and the last array element becoming the first. In other words, elements order in the array will be turned towards the direction opposite to that previously stated.

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reverse>

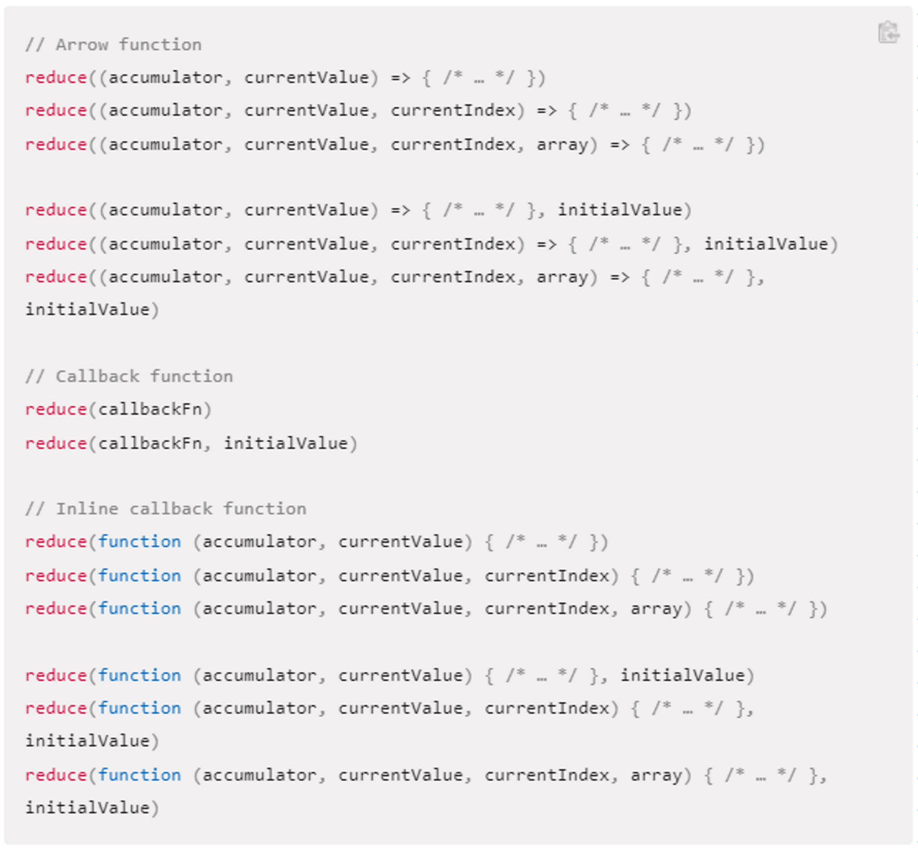

JavaScript Demo: Array.reverse()

```
const array1 = ['one', 'two', 'three'];
console.log('array1:', array1);
// expected output: "array1:" Array ["one", "two", "three"]

const reversed = array1.reverse();
console.log('reversed:', reversed);
// expected output: "reversed:" Array ["three", "two", "one"]

// Careful: reverse is destructive -- it changes the original array.
console.log('array1:', array1);
// expected output: "array1:" Array ["three", "two", "one"]

```

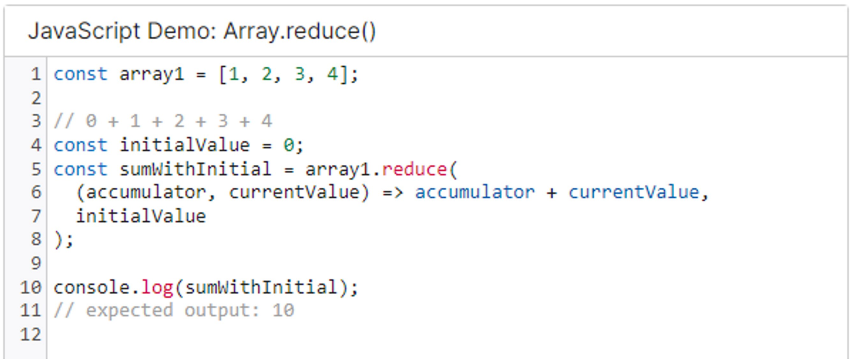
8. Array.prototype.reduce()

The **reduce()** method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.
The first time that the callback is run there is no "return value of the previous calculation". If supplied, an initial value may be used in its place. Otherwise the array element at index 0 is used as the initial value and iteration starts from the next element (index 1 instead of index 0).
Perhaps the easiest-to-understand case for reduce() is to return the sum of all the elements in an array:

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce>



```
// Arrow function
reduce((accumulator, currentValue) => { /* … */ })
reduce((accumulator, currentValue, currentIndex) => { /* … */ })
reduce((accumulator, currentValue, currentIndex, array) => { /* … */ })

reduce((accumulator, currentValue) => { /* … */ }, initialValue)
reduce((accumulator, currentValue, currentIndex) => { /* … */ }, initialValue)
reduce((accumulator, currentValue, currentIndex, array) => { /* … */ },
initialValue)

// Callback function
reduce(callbackFn)
reduce(callbackFn, initialValue)

// Inline callback function
reduce(function (accumulator, currentValue) { /* … */ })
reduce(function (accumulator, currentValue, currentIndex) { /* … */ })
reduce(function (accumulator, currentValue, currentIndex, array) { /* … */ })

reduce(function (accumulator, currentValue) { /* … */ }, initialValue)
reduce(function (accumulator, currentValue, currentIndex) { /* … */ },
initialValue)
reduce(function (accumulator, currentValue, currentIndex, array) { /* … */ },
initialValue)
```


JavaScript Demo: Array.reduce()

```
const array1 = [1, 2, 3, 4];

// 0 + 1 + 2 + 3 + 4
const initialValue = 0;
const sumWithInitial = array1.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  initialValue
);

console.log(sumWithInitial);
// expected output: 10

```

Examples

How reduce() works without an initial value

The code below shows what happens if we call reduce() with an array and no initial value.
constarray =[15,16,17,18,19];functionreducer(accumulator,currentValue,index){constreturns =accumulator +currentValue;console.log(`accumulator: ${accumulator}, currentValue: ${currentValue}, index: ${index}, returns: ${returns}`,);returnreturns;}array.reduce(reducer);
Copy to Clipboard
The callback would be invoked four times, with the arguments and return values in each call being as follows:

	accumulator	**currentValue**	**index**	**Return value**
First call	15	16	1	31
Second call	31	17	2	48
Third call	48	18	3	66
Fourth call	66	19	4	85

The array parameter never changes through the process — it's always [15, 16, 17, 18, 19]. The value returned by reduce() would be that of the last callback invocation (85).

How reduce() works with an initial value

Here we reduce the same array using the same algorithm, but with an initialValue of 10 passed as the second argument to reduce():
[15,16,17,18,19].reduce((accumulator,currentValue)=>accumulator +currentValue,10,);
Copy to Clipboard
The callback would be invoked five times, with the arguments and return values in each call being as follows:

	accumulator	**currentValue**	**index**	**Return value**
First call	10	15	0	25
Second call	25	16	1	41
Third call	41	17	2	58
Fourth call	58	18	3	76
Fifth call	76	19	4	95

The value returned by reduce() in this case would be 95.

Sum of values in an object array

To sum up the values contained in an array of objects, you **must** supply an initialValue, so that each item passes through your function.
constobjects =[{x:1},{x:2},{x:3}];constsum =objects.reduce((accumulator,currentValue)=> accumulator +currentValue.x,0,);console.log(sum);// 6
Copy to Clipboard

Flatten an array of arrays

constflattened =[[0,1],[2,3],[4,5],].reduce((accumulator,currentValue)=> accumulator.concat(currentValue),[]);// flattened is [0, 1, 2, 3, 4, 5]

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce>

9. Array.of()

The **Array.of()** method creates a new Array instance from a variable number of arguments, regardless of number or type of the arguments.

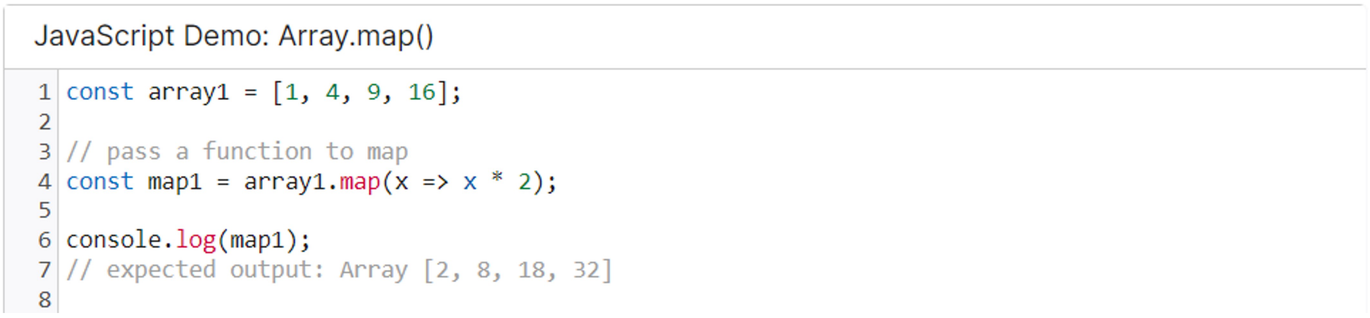
From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/of>

```
Array.of(7); // [7]
Array(7); // array of 7 empty slots

Array.of(1, 2, 3); // [1, 2, 3]
Array(1, 2, 3); // [1, 2, 3]
```

10. Array.prototype.map()

The **map()** method **creates a new array** populated with the results of calling a provided function on every element in the calling array.

From <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map>



JavaScript Demo: Array.map()

```
const array1 = [1, 4, 9, 16];

// pass a function to map
const map1 = array1.map(x => x * 2);

console.log(map1);
// expected output: Array [2, 8, 18, 32]

```