Javascript Resume

Inhoud

[ARRAY functions 3](#_Toc121338160)

[1. .toString 3](#_Toc121338161)

[2. .join 3](#_Toc121338162)

[3. .concat 3](#_Toc121338163)

[4. .splice 3](#_Toc121338164)

[5. .slice 4](#_Toc121338165)

[6. .indexOf 4](#_Toc121338166)

[7. .lastIndexOf 4](#_Toc121338167)

[8. .forEach 4](#_Toc121338168)

[9. .map 5](#_Toc121338169)

[10. .filter 5](#_Toc121338170)

[11. .reduce 6](#_Toc121338171)

[12. .some 7](#_Toc121338172)

[13. .every 8](#_Toc121338173)

[14. .flat 8](#_Toc121338174)

[15. .find 8](#_Toc121338175)

[16. .findIndex 9](#_Toc121338176)

[17. .sort 9](#_Toc121338177)

[Local Storage 10](#_Toc121338178)

[1. Definition 10](#_Toc121338179)

[2. Storing : setItem() 10](#_Toc121338180)

[3. Retrieving : getItem() 10](#_Toc121338181)

[4. Deleting 11](#_Toc121338182)

[Promises 12](#_Toc121338183)

# ARRAY functions

### .toString

Converts an array into a string of comma-separated array values:

let names = ["Zoe", "Adam", "Dan"]

let strNames = names.toString() // Zoe,Adam,Dan

### .join

Similar to toString, but you can specify the separator:

let names = ["Zoe", "Adam", "Dan"]

let strNames = names.join(" and ") // Zoe and Adam and Dan

### .concat

Create a new array by concatenating existing arrays:

let nums = [1, 2, 3]

let nums2 = [4, 5, 6]

let nums3 = [7, 8, 9]

let concatArr = nums.concat(nums2, nums3)

// [1, 2, 3, 4, 5, 6, 7, 8, 9]

### .splice

The splice() method can be used to add new items to an array:

let arr = ["Danny", "Joe"]

arr.splice(1, 0, "Alice", "Amy")

console.log(arr) // ['Danny', 'Alice', 'Amy', 'Joe']

The first parameter (1) defines the index from where the new elements should be added (spliced in).

The second parameter (0) defines how many elements should be removed.

The rest of the parameters ('Alice', 'Amy') define the new elements to be added.

splice() returns an array with the deleted items:

let arr = ["Danny", "Joe"]

let spliced = arr.splice(1, 1, "Alice", "Amy") // ['Joe']

console.log(arr) // ['Danny', 'Alice', 'Amy']

We can also delete items without adding any new ones:

let arr = ["Danny", "Joe", "Amy"]

let spliced = arr.splice(1, 1) // ['Joe']

console.log(arr) // ['Danny', 'Amy']

Since splice() mutates the original array, it is often best to make a copy of it before splicing.

### .slice

slice() slices out a piece of an array, and returns it in a new array:

let arr = ["Danny", "Joe", "Amy"]

let slice = arr.slice(1) // ['Joe', 'Amy']

Above, we are slicing from the element at index 1. slice() does not mutate the original array.

We can provide a start and end index to splice from (up to but not including end index):

let arr = ["Danny", "Joe", "Amy"]

let slice = arr.slice(0, 2) // ['Danny', 'Joe']

### .indexOf

Find the first index that contains a certain value (searches from left to right):

let arr = ["Danny", "Joe", "Amy", "Joe"]

let index = arr.indexOf("Joe") // 1

### .lastIndexOf

Find the last index that contains a certain value (searches from right to left):

let arr = ["Danny", "Joe", "Amy", "Joe"]

let index = arr.lastIndexOf("Joe") // 3

### .forEach

The forEach method is basically just a shorter way of writing for(let i = 0; i < arr.length; i++) {...}.

It loops through the given array, and calls the given call-back function for each of the elements in the array.

The call-back passed to the forEach() function can accept any of the three arguments:

* the item value
* the item index
* the array itself

Example:

let numbers = [1, 2, 3, 4]

numbers.forEach(n => console.log(n))

// 1

// 2

// 3

// 4

It's better to use map if you want to return a new array with return value of each call-back run.

### .map

The map function takes in a call-back function as argument, and executes that function on each element of the array it is working on. It maps each of the return values of the call-back into a new array. It does not mutate the original array.

The call-back passed to the map() function can accept any of the three arguments:

* the item value
* the item index
* the array itself

Examples:

let numbers = [1, 2, 3, 4]

// Double all numbers

let doubledNumbers = numbers.map(n => n \* 2) // [2, 4, 6, 8]

// Only double numbers at odd indexes

let doubledOddIndexNumbers = numbers.map((n, i) => {

if (i % 2 === 1) return n \* 2

else return n

}) // [1, 4, 3, 8]

### .filter

The filter method is used to filter out array elements that fail a boolean test. Only elements that pass the test are allowed through into the new return array.

The call-back passed to the filter() function can accept any of the three arguments:

* the item value
* the item index
* the array itself

A good use case for filter is a search bar:

let articles = [

{ title: "PHP classes", author: "Danny Adams" },

{ title: "Python arrays", author: "Amy Sanders" },

{ title: "Arrays in PHP", author: "Danny Adams" },

]

// Lets say the user searches for all articles with PHP in the title

let PHPArticles = articles.filter(a => a.title.includes("PHP"))

// [

// { title: 'PHP classes', author: 'Danny Adams' },

// { title: 'Arrays in PHP', author: 'Danny Adams' },

// ];

Another use case is filtering via size:

let cities = [

{ name: "Stokington", rivers: 3 },

{ name: "Phillydelfia", rivers: 6 },

{ name: "New Ports", rivers: 2 },

]

let moreThanTwoRivers = cities.filter(c => c.rivers > 2)

// [

// { name: 'Stokington', rivers: 3 },

// { name: 'Phillydelfia', rivers: 6 },

// ];

### .reduce

The reduce method runs the call-back function on each array element, and reduces the array down into a single value.

The reduce function itself takes two arguments:

* A call-back function
* An initial value

reduce(callback, initialVal)

The call-back function passed into reduce can take up to four arguments:

* total or "accumulator"
* current item value
* current item index
* the array itself

Example:

let numbers = [1, 2, 3, 4]

let total = numbers.reduce((total, currentNum) => total + currentNum) // 10

In the above example, total is initially the first value in the array (1), and currentNum is the second (2).

If we wanted to start from a different value, we can pass in a second initialVal argument to reduce. Let's say we wanted to find the total, starting from 5:

let numbers = [1, 2, 3, 4]

let total = numbers.reduce((total, currentNum) => total + currentNum, 5) // 15

Above, total will now start out as 5, and currentNum will be the first element in the array (1).

Another good use case of reduce is to find the max or min value in an array:

let arr = [1, 2, 3]

let max = arr.reduce((a, b) => {

return Math.max(a, b)

}, -Infinity)

// 3

### .some

The some method checks if some array values pass a test. It returns either true or false.

The call-back takes 3 arguments:

* The item value
* The item index
* The array itself

Example:

let numbers = [4, 6, 14, 16]

let isSomeGreaterThan6 = numbers.some(n => n > 6) // true

let isSomeLessThan4 = numbers.some(n => n < 4) // false

### .every

every is similar to the some method, but checks if every value in the array passes a certain test, rather than just some.

Example:

let numbers = [4, 6, 14, 16]

let isEverythingGreaterThan6 = numbers.every(n => n > 6) // false

let isEverythingLessThan20 = numbers.some(n => n < 20) // true

### .flat

The flat() method creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

Example:

let arr = [1, 2, 3, [1, 2], 4]

let flatArr = arr.flat() // [1, 2, 3, 1, 2, 4]

flat takes one optional depth parameter. depth specifies how deep a nested array structure should be flattened. It defaults to 1.

Example:

let arr = [1, 2, 3, [1, 2], [1, [1, 2]]]

let flatArr1 = arr.flat() // [1, 2, 3, 1, 2, 1, [1, 2]]

let flatArr2 = arr.flat(2) // [1, 2, 3, 1, 2, 1, 1, 2]

### .find

The find method returns the first element in the array that passes a certain test.

The call-back passed to find takes 3 arguments:

* The item value
* The item index
* The array itself

Example

let stock = [

{ item: "ketchup", quantity: 32 },

{ item: "mayo", quantity: 9 },

{ item: "hot sauce", quantity: 12 },

]

let mayo = stock.find(s => s.item === "mayo")

// { item: 'mayo', quantity: 9 }

### .findIndex

Same as find, but returns the index instead of the value:

let stock = [

{ item: "ketchup", quantity: 32 },

{ item: "mayo", quantity: 9 },

{ item: "hot sauce", quantity: 12 },

]

let mayoIndex = stock.findIndex(s => s.item === "mayo")

// 1

### .sort

sort puts an array's elements in ascending order. It is an "in-place" sorting algorithm - meaning that it mutates the original array and returns it.

By default, sort works on strings:

let names = ["Zoe", "Adam", "Dan"]

names.sort()

console.log(names) // ['Adam', 'Dan', 'Zoe']

For numbers, we need to pass a comparison call-back function:

let numbers = [3, 1, 7, 2]

numbers.sort((a, b) => a - b)

console.log(numbers) // [1, 2, 3, 7]

Initially, a is 3 and b is 1. If a - b is negative, it knows that b is larger, and therefore should come after a. If positive, b should come before a.

# Local Storage

### Definition

* Data is shared between all tabs and windows from the same origin.
* The data will not expire. It will remain even after browser restart and survive OS reboot too.
* Limits the size of data you can store (~5MB across all major browsers).

### Storing : setItem()

It takes two parameters: a key and a value. The key can be referenced later to fetch the value attached to it.

      window.localStorage.setItem('name', 'Obaseki Nosa');

To store arrays or objects, you would have to convert them to strings.

To do this, we use the JSON.stringify() method before passing to setItem().

const person = {

        name: "Obaseki Nosa",

        location: "Lagos",

      };

window.localStorage.setItem("user", JSON.stringify(person));

### Retrieving : getItem()

getItem() accepts only one parameter, which is the key, and returns the value as a string.

To retrieve a user key:

window.localStorage.getItem('user');

If the key has been stored as a JSON then it will return a string with value as:

“{“name”:”Obaseki Nosa”,”location”:”Lagos”}”

To use this value, you would have to convert it back to an object.

To do this, we make use of the JSON.parse() method, which converts a JSON string into a JavaScript object.

JSON.parse(window.localStorage.getItem('user'));

### Deleting

//Delete specifice obeject from local storege

localStorage.removeItem(name);

//Delete all from local storege

localStorage.clear();

# Promises