JavaScript Array Functions Cheat Sheet

May 21, 2020

concat() > 96%

merge two or more arrays

example

example

```
[ 1, 2 ].concat([5], [7, 9]) // [ 1, 2, 5, 7, 9 ]
```

syntax

```
const new_array = old_array.concat([value1[, value2[, ...[, va
```

copyWithin() > 92%

copies part of array to another location

example

```
[ 1, 2, 3, 4, 5 ].copyWithin(0,2)  // [ 3, 4, 5, 4, 5 ]
```

syntax

```
arr.copyWithin(target[, start[, end]])
```

entries() > 94%

Array Iterator with key/value pairs for each index

example

```
['a', 'b', 'c']
    .entries() // Array Iterator {  }
    .next() // { value: (2) […], done: false }
    .value // Array [ 0, "a" ]
```

syntax

```
arr.entries()
```

every() > 95%

tests if all elements in the array pass the test

example

example

```
[1, 30, 40].every(val => val > 0) // true
```

syntax

```
arr.every(callback(element[, index[, array]])[, thisArg])
```

fill() > 92%

changes elements in an array to a static value

example

```
[1, 2, 3, 4].fill('x', 1, 3) // [ 1, "x", "x", 4 ]
```

syntax

```
arr.fill(value[, start[, end]])
```

filter() > 95%

creates new array with elements that pass test

example

example

```
[1, 10, 5, 6].filter(val => val > 5) // [ 10, 6 ]
```

syntax

```
let newArray = arr.filter(callback(element[, index, [array]])[
```

find() > 95%

returns the value of the first element, that matches test

example

```
[1, 10, 5, 6].find(val => val > 5) // 10
```

syntax

```
arr.find(callback(element[, index[, array]])[, thisArg])
```

findIndex() > 94%

returns index of the first element, that matches test

example

example

```
[1, 4, 5, 6].findIndex(val => val > 5) // 3
```

syntax

```
arr.findIndex(callback( element[, index[, array]] )[, thisArg]
```

flat() > 87%

creates a new array with sub-array elements flattened by specified depth.

example

```
[1, [2, [3, [4]]]].flat(2) // [ 1, 2, 3, [4] ]
```

syntax

```
const new_array = arr.flat([depth]);
```

flatMap() > 87%

creates a new array with sub-array elements flattened by specified depth.

example

example

```
[[2], [4], [6], [8]].flatMap(val => val/2) // [ 1, 2, 3, 4 ]
```

syntax

```
var new_array = arr.flatMap(function callback(currentValue[, i
    // return element for new_array
}[, thisArg])
```

forEach() > 95%

executes provided function once for each array element

example

```
[ 1, 2, 3 ].forEach(val => console.log(val)) // 1 // 2 // 3
```

syntax

```
arr.forEach(callback(currentValue [, index [, array]])[, thisA
```

includes() > 95%

determines if array includes a certain value

example

example

```
[ 1, 2, 3 ].includes(3) // true
```

syntax

```
arr.includes(valueToFind[, fromIndex])
```

indexOf() > 95%

returns the first index at which element can be found

example

```
[ 1, 2, 3 ].indexOf(3) // 2
```

syntax

```
arr.indexOf(searchElement[, fromIndex])
```

join() > 96%

returns string by concatenating all elements in array

example

example

```
[ "x", "y", "z" ].join(" - ") // "x - y - z"
```

syntax

```
arr.join([separator])
```

keys() > 93%

returns Array Iterator that contains keys for each index

example

```
['a', 'b', 'c']
    .keys() // Array Iterator {  }
    .next() // { value: 0, done: false }
    .value // 0
```

syntax

```
arr.keys()
```

lastIndexOf() > 95%

returns last index at which given element can be found

example

example

```
[ 1, 2, 3, 1, 0].lastIndexOf(1) // 3
```

syntax

```
arr.lastIndexOf(searchElement[, fromIndex])
```

map() > 95%

creates new array with results of provided function

example

```
[ 2, 3, 4 ].map(val => val * 2) // [ 4, 6, 8 ]
```

syntax

```
let new_array = arr.map(function callback( currentValue[, inde
    // return element for new_array
}[, thisArg])
```

pop() > 96%

removes last element from array and returns that element

example

example

```
const arr = [ 1, 2, 3 ]
arr.pop() // returns: 3 // arr is [ 1, 2 ]
```

syntax

```
arr.pop()
```

push() > 96%

adds one or more elements to end of array and returns new length

example

```
const arr = [ 1, 2, 3 ]
arr.push(1) // returns: 4 // arr is [ 1, 2, 3, 4 ]
```

syntax

```
arr.push(element1[, ...[, elementN]])
```

reduce() > 95%

executes a reducer function, resulting in single output value

example

example

```
[ 'a', 'b', 'c' ].reduce((acc, curr) => acc + curr, 'd') // "d
```

syntax

```
arr.reduce(callback( accumulator, currentValue[, index[, array
```

reduceRight() > 95%

executes a reducer function from right to left, resulting in single output value

example

```
[ 'a', 'b', 'c' ].reduceRight((acc, curr) => acc + curr, 'd')
```

syntax

```
arr.reduceRight(callback(accumulator, currentValue[, index[, a
```

reverse() > 96%

reverses an array

example

example

```
[ 1, 2, 3 ].reverse() // [ 3, 2, 1 ]
```

syntax

```
arr.reverse()
```

shift() > 96%

removes the first element from array and returns that element

example

```
const arr = [ 1, 2, 3 ]
arr.shift() // returns: 1 // arr is [ 2, 3 ]
```

syntax

```
arr.shift()
```

slice() > 96%

returns a copy of part of array, while original array is not modified

example

example

```
[ 1, 2, 3, 4 ].slice(1, 3) // [ 2, 3 ]
```

syntax

```
arr.slice([begin[, end]])
```

some() > 95%

tests whether at least one element in array passes the test

example

```
[ 1, 2, 3, 4 ].some(val => val > 3) // true
```

syntax

```
arr.some(callback(element[, index[, array]])[, thisArg])
```

sort() > 96%

sorts the elements of array in place

example

example

```
[ 1, 2, 3, 4 ].sort((a, b) => b - a) // [ 4, 3, 2, 1 ]
```

syntax

```
arr.sort([compareFunction])
```

splice() > 96%

changes contents of array by removing, replacing and/or adding elements

example

```
const arr = [ 1, 2, 3, 4 ]
arr.splice(1, 2, 'a') // returns [ 2, 3 ] // arr is [ 1, "a",
```

syntax

```
let arrDeletedItems = array.splice(start[, deleteCount[, item1
```

toLocaleString() > 95%

elements are converted to Strings using toLocaleString and are separated by locale-specific String (eg. ",")

example

example

```
[1.1, 'a', new Date()].toLocaleString('EN') // "1.1,a,5/18/202
```

syntax

```
arr.toLocaleString([locales[, options]]);
```

toString() > 95%

returns a string representing the specified array

example

```
[ 'a', 2, 3 ].toString() // "a,2,3"
```

syntax

```
arr.toString()
```

unshift() > 96%

adds one or more elements to beginning of array and returns new length

example

example

```
const arr = [ 1, 2, 3 ]
arr.unshift(0, 99) // returns 5 // arr is [ 0, 99, 1, 2, 3 ]
```

syntax

```
arr.unshift(element1[, ...[, elementN]])
```

values() > 91%

returns Array Iterator object that contains values for each index in array

example

```
['a', 'b', 'c']
    .values() // Array Iterator {  }
    .next() // { value: "a", done: false }
    .value // "a"
```

syntax

```
arr.values()
```

Other Projects

- ImmoRadar
- OpenMailer
- bsky-embed
- CSS Speedrun

More

- Donate
- About
- Privacy Policy