JavaScript Array Cheatsheet

@kealanparr

Code snippets included, and an example return detailed in the comment

Finding

- **indexOf()**
  Returns the index of the first occurence of an element in an array
  Returns -1 if the element's not in the array

```
['green'].indexOf('green') // 0
['red'].indexOf('green')     // -1
```

- **lastIndexOf()**
  Returns the index of the last occurence of an element in an array
  Returns -1 if the element's not in the array

```
['green', 'red', 'green'].lastIndexOf('green') // 2
['green', 'red', 'green'].lastIndexOf('red') // 1
```

- **length()**
  Returns how many elements are in an array

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

- **at()**
  Returns an element from an array at a certain index
  Negative values start from the end of the array

```
['first', 'last'].at(0)   // 'first'
['first', 'last'].at(-1)  // 'last'
```

- **find()**
  Returns the first element that matches a condition

```
[1, 4, 8].find( item => item % 2 === 0) // 4
```

- **findIndex()**
  Returns the first index of an element that matches a condition

```
[1, 4, 8].findIndex( item => item % 2 === 0) // 1
```

- **includes()**
  Returns whether an array contains an item.

```
['green'].includes('green')  // true
['green'].includes('red')     // false
```

Creating

- **slice()**
  Copies an array between a start and end index
  The cloned array will contain the start index
  But it stops at the last index (and won't be copied)
  If no indices arguments are passed - it clones the entire array
  If a negative index is passed, it clones items from the end of the array

```
[1, 2, 3, 4].slice() // [1, 2, 3, 4] - clone whole array
['red', 'green', 'blue', 'black'].slice(1, 3)  // [2, 3] - clone 1st and 2nd item
[1, 2, 3, 4].slice(-2)  // [3, 4] - gets the last 2 items
```

- **of()**
  Receives a variable amount of arguments, and makes a new array

```
Array.of(5) // [5]
Array.of('ok', 'still', 'working')  // ['ok', 'still', 'working']
```

- **flat()**
  Flattens a variable level of nested arrays, by default 1
  - Can turn 2 dimensional array into 1 dimensional array
  - Can turn 3 dimensional array into 2 dimensional array
  - etc

```
[0, 1, 2, [3, 4]].flat() // Default argument of 1 returns [0, 1, 2, 3, 4, 5]
[[ 0, 1, 2, 3, 4, 5 ]]     // Default arg of 1 returns [0, 1, 2, 3, 4, 5]
[[[[[[[[[[1]]]]]]]]]].flat(12) // [1]
```

- **flatMap()**
  Combines flat() and map()

```
[
   {name: 'Kealan', likes: ['chess', 'reading'] },
   {name: 'Jake', likes: ['films', 'coding']}
].flatMap(item => item.likes)

// ['chess', 'reading', 'films', 'coding']
```

Adding

- **splice()**
  Modifies the elements of an array
  More examples found here

```
// Remove 0 elements, before index 1 and add "black"
['green', 'red'].splice(1, 0, 'black') // ['green', 'black', 'red']
```

- **push()**
  Adds one or more elements to the end of an array
  and returns the new length

```
[0, 1].push(2)            // 3 which is new length of array [0, 1, 2]
['Jake'].push('Mike') // 2
```

- **copyWithin()**
  Shallow copies a part of an array to another position in the same array (never changes length of array)

```
// copy to index 0 the element at index 3
['a', 'b', 'c', 'd', 'e'].copyWithin(0, 3, 4) // ['d', 'b', 'c', 'd', 'e']
// copy the first two elements where the last 2 elements are
[1, 2, 3, 4, 5].copyWithin(-2) // [1, 2, 3, 1, 2]
```

- **fill()**
  Populates an array with a static value between two indices

```
// Fill from index 2 until index 4, with 0's
[1, 2, 3, 4].fill(0, 2, 4); // [1, 2, 0, 0]
[1, 2, 3, 4].fill(5, 1);     // [1, 5, 5, 5]
```

- **concat()**
  Merges two or more arrays together

```
[1, 2].concat(3, 4);            // [1, 2, 3, 4]
[1, 2].concat([3, 4], [5, 6]) // [1, 2, 3, 4, 5, 6]
```

- **unshift()**
  Adds one or more elements to the start of an array and returns the new length

```
[1, 2].push(0) // 3, new length of array [0, 1, 2]
['Kealan', 'Jake'].push('Mike') // 3
```

Removing

- **shift()**
  Removes and returns the first element from an array

```
const array1 = [1, 2, 3]
const firstElement = array1.shift()
console.log(array1) // Array [2, 3]
```

- **pop()**
  Removes and returns the last element from an array

```
[1, 2].pop() // 2 - array now is [1]
```

- **splice()**
  Modifies the items of an array
  Examples + more found here

```
// Remove 0 elements, before index 1 and add 1 item
['green', 'red'].splice(1, 0, 'black')
// ['green', 'black', 'red']

// Remove 0 elements, before index 1 and add 2 items
['green', 'red'].splice(1, 0, 'black', 'blue')
// ['green', 'black', 'red']
```

Re-arranging

- **reverse()**
  Reverses the order of the elements in an array

```
[1, 2, 3].reverse()   // [3, 2, 1]
['a', 'b', 'c'].reverse() // ['c', 'b', 'a']
```

- **sort()**
  Sorts an array by it's Unicode code point by default
  Can also receive a sorting function

```
[2, 1, 4, 31, 50].sort() // [1, 2, 31, 4, 50]

[2, 1, 4, 31, 50].sort((a, b)=> a - b) // [1, 2, 4, 31, 50]
```

Misc

- **join()**
  Returns a string by concatenating all the array elements together

```
['Fire', 'Air', 'Water'].join())   // "Fire,Air,Water"
['Fire', 'Air', 'Water'].join('')  // "FireAirWater"
['Fire', 'Air', 'Water'].join('-') // "Fire-Air-Water"
```

- **toString()**
  Returns a string representation of an array

```
['An', 'example', 1, 2].toString() // 'An,example,1,2'
```

- **isArray()**
  Returns whether the argument is an array

```
Array.isArray([])   // true
Array.isArray({})   // false
```

- **indexOf()**
  Returns how many elements are in an array

```
array.indexOf('green') // 1
array.includes('red')    // -1
```

Looping

- **filter()**
  Creates a new array with all elements that pass an assertion

```
const nums = [0, 1, 2, 3, 4]
const result = [0, 1, 2, 3, 4].filter(num => num > 3)

// nums is un-touched, [0, 1, 2, 3, 4]
// result is [4]
```

- **map()**
  Creates a new array by calling a function on all elements of the array

```
const nums = [0, 1, 2, 3, 4]
const result = [0, 1, 2, 3, 4].map(num => num * 2)

// nums is un-touched, [0, 1, 2, 3, 4]
// result is [0, 2, 4, 6, 8]
```

- **reduce()**
  Runs a user supplied "reducer" function on each element of the array, passing the return value into the reducer call on the next item

```
const nums = [0, 1, 2, 3, 4]
const result = [0, 1, 2, 3, 4].reduce((prev, curr) => prev + curr)

// nums is un-touched, [0, 1, 2, 3, 4]
// result is 10
```

- **forEach()**
  Runs a user supplied function on each element of the array

```
['First', 'Second'].forEach(function (e) {
   console.log(e);
});
// 'First'
// 'Second'
```

- **reduceRight()**
  Works exactly like reduce - but starts at the end of the array

```
const nums = [[0, 1], [2, 3], [4, 5]]

const result = nums.reduceRight((acc, curr) => acc.concat(curr))

console.log(result)

// nums is un-touched, [0, 1, 2, 3, 4]
// result is 10
```

- **some()**
  Runs a user supplied function on each element of the array, returns true if at least one element passes the assertion (otherwise returns false)

```
[1, 2, 3, 4, 5].some((item) => { item % 2 === 0 })

// returns true
```

- **every()**
  Runs a user supplied function on each element of the array, returns true if every element passes the assertion (otherwise returns false)

```
[1, 2, 3, 4, 5].every((item) => { item % 2 === 0 })

// returns false
```

- **entries()**
  Returns an array iterator to return key, value pairs of the array element & index

```
const array1 = ['a', 'b', 'c']

const iterator1 = array1.entries()

console.log(iterator1.next().value)
// expected output: Array [0, 'a']

console.log(iterator1.next().value)
// expected output: Array [1, 'b']
```