**JAVASCRIPT FUNCTIONS**

# 1.Array.prototype.includes()

1. includes():

-------------------

The includes() method determines whether an array includes a certain value among its entries, returning true or false as appropriate.

syntax:

1.includes(searchElement) ---> search element rep the(The value to search for)

When comparing strings and characters, includes() is case-sensitive.

1. includes(searchElement, fromIndex) ---->The position in this array at which to begin searching for searchElement

[fromIndex]

ex:

let arr = ['a', 'b', 'c']

arr.includes('a',0) //true

arr.includes('c', 3) // false

arr.includes('c', 100) // false

examples

------------------------

const array1 = [1, 2, 3];

console.log(array1.includes(2));

// expected output: true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));

// expected output: true

console.log(pets.includes('at'));

// expected output: false

# Array.prototype.filter()

1. filter():

The filter() method **creates a new array** with all elements that pass the test implemented by the provided function.

example:

const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);

// expected output: Array ["exuberant", "destruction", "present"]

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter" \l "syntax" \o "Permalink to Syntax):

// Arrow function

filter((element) => { /\* ... \*/ } )

filter((element, index) => { /\* ... \*/ } )

filter((element, index, array) => { /\* ... \*/ } )

// Callback function

filter(callbackFn)

filter(callbackFn, thisArg)

// Inline callback function

filter(function(element) { /\* ... \*/ })

filter(function(element, index) { /\* ... \*/ })

filter(function(element, index, array){ /\* ... \*/ })

filter(function(element, index, array) { /\* ... \*/ }, thisArg)

callbackFn is invoked with three arguments:

1. the value of the element
2. the index of the element
3. the Array object being traversed

### [Filtering out all small values](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter" \l "filtering_out_all_small_values" \o "Permalink to Filtering out all small values)

The following example uses filter() to create a filtered array that has all elements with values less than 10 removed.

function isBigEnough(value) {

return value >= 10}

let filtered = [12, 5, 8, 130, 44].filter(isBigEnough)

// filtered is [12, 130, 44]

### [Filtering invalid entries from JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter" \l "filtering_invalid_entries_from_json" \o "Permalink to Filtering invalid entries from JSON)

The following example uses filter() to create a filtered JSON of all elements with non-zero, numeric id.

let arr = [

{ id: 15 },

{ id: -1 },

{ id: 0 },

{ id: 3 },

{ id: 12.2 },

{ },

{ id: null },

{ id: NaN },

{ id: 'undefined' }]

let invalidEntries = 0

function filterByID(item) {

if (Number.isFinite(item.id) && item.id !== 0) {

return true }

invalidEntries++

return false;

}let arrByID = arr.filter(filterByID)

console.log('Filtered Array\n', arrByID)

// Filtered Array

// [{ id: 15 }, { id: -1 }, { id: 3 }, { id: 12.2 }]

console.log('Number of Invalid Entries = ', invalidEntries)

// Number of Invalid Entries = 5

# isFinite()

The global isFinite() function determines whether the passed value is a finite number. If needed, the parameter is first converted to a number.

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/isFinite" \l "syntax" \o "Permalink to Syntax)

isFinite(testValue)

testValue==>(The value to be tested for finiteness.)

### [Using isFinite](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/isFinite" \l "using_isfinite" \o "Permalink to Using isFinite)

isFinite(Infinity); // false

isFinite(NaN); // false

isFinite(-Infinity); // false

isFinite(0); // true

isFinite(2e64); // true

isFinite(910); // true

isFinite(null); // true would've been false with the

// more robust Number.isFinite(null)

isFinite('0'); // true, would've been false with the

// more robust Number.isFinite("0")

# Array.prototype.pop()

The pop() method removes the **last** element from an array and returns that element. This method changes the length of the array.

const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

console.log(plants.pop());

// expected output: "tomato"

console.log(plants);

// expected output: Array ["broccoli", "cauliflower", "cabbage", "kale"]

plants.pop();

console.log(plants);

// expected output: Array ["broccoli", "cauliflower", "cabbage"]

## [Syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop" \l "syntax" \o "Permalink to Syntax):

pop()

## [Examples](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop" \l "examples" \o "Permalink to Examples)

### [Removing the last element of an array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop" \l "removing_the_last_element_of_an_array" \o "Permalink to Removing the last element of an array)

The following code creates the myFish array containing four elements, then removes its last element.

const myFish = ['angel', 'clown', 'mandarin', 'sturgeon'];

const popped = myFish.pop();

console.log(myFish); // ['angel', 'clown', 'mandarin' ]

console.log(popped); // 'sturgeon'

### [Using apply( ) or call ( ) on array-like objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/pop" \l "using_apply_or_call_on_array-like_objects" \o "Permalink to Using apply( ) or call ( ) on array-like objects)

The following code creates the myFish array-like object containing four elements and a length parameter, then removes its last element and decrements the length parameter.

const myFish = {0:'angel', 1:'clown', 2:'mandarin', 3:'sturgeon', length: 4};

const popped = Array.prototype.pop.call(myFish); //same syntax for using apply( )

console.log(myFish); // {0:'angel', 1:'clown', 2:'mandarin', length: 3}

console.log(popped); // 'sturgeon'