MDN Web Docs
moz://a

Array

The JavaScript `Array` class is a global object that is used in the construction of arrays; which are high-level, list-like objects.

Description

Arrays are list-like objects whose prototype has methods to perform traversal and mutation operations. Neither the length of a JavaScript array nor the types of its elements are fixed. Since an array's length can change at any time, and data can be stored at non-contiguous locations in the array, JavaScript arrays are not guaranteed to be dense; this depends on how the programmer chooses to use them. In general, these are convenient characteristics; but if these features are not desirable for your particular use, you might consider using typed arrays.

Arrays cannot use strings as element indexes (as in an associative array) but must use integers. Setting or accessing via non-integers using bracket notation (or dot notation) will not set or retrieve an element from the array list itself, but will set or access a variable associated with that array's object property collection. The array's object properties and list of array elements are separate, and the array's traversal and mutation operations cannot be applied to these named properties.

Common operations

Create an Array

```
let fruits = ['Apple', 'Banana']

console.log(fruits.length)
// 2
```

Access an Array item using the index position

```
let first = fruits[0]
```

```
// Apple

let last = fruits[fruits.length - 1]
// Banana
```

Loop over an Array

```
fruits.forEach(function(item, index, array) {
  console.log(item, index)
})
// Apple 0
// Banana 1
```

Add an item to the end of an Array

```
let newLength = fruits.push('Orange')
// ["Apple", "Banana", "Orange"]
```

Remove an item from the end of an Array

```
let last = fruits.pop() // remove Orange (from the end)
// ["Apple", "Banana"]
```

Remove an item from the beginning of an Array

```
let first = fruits.shift() // remove Apple from the front
// ["Banana"]
```

Add an item to the beginning of an Array

```
let newLength = fruits.unshift('Strawberry') // add to the front
// ["Strawberry", "Banana"]
```

Find the index of an item in the Array

```
fruits.push('Mango')
// ["Strawberry", "Banana", "Mango"]

let pos = fruits.indexOf('Banana')
// 1
```

Remove an item by index position

```
let removedItem = fruits.splice(pos, 1) // this is how to remove an item

// ["Strawberry", "Mango"]
```

Remove items from an index position

```
let vegetables = ['Cabbage', 'Turnip', 'Radish', 'Carrot']
console.log(vegetables)
// ["Cabbage", "Turnip", "Radish", "Carrot"]

let pos = 1
let n = 2

let removedItems = vegetables.splice(pos, n)
// this is how to remove items, n defines the number of items to be removed,
// starting at the index position specified by pos and progressing toward the e

console.log(vegetables)
// ["Cabbage", "Carrot"] (the original array is changed)

console.log(removedItems)
// ["Turnip", "Radish"]
```

Copy an Array

```
let shallowCopy = fruits.slice() // this is how to make a copy
// ["Strawberry", "Mango"]
```

Accessing array elements

JavaScript arrays are zero-indexed. The first element of an array is at index `0`, and the last element is at the index value equal to the value of the array's `length` property minus `1`.

Using an invalid index number returns `undefined`.

```
let arr = ['this is the first element', 'this is the second element', 'thi  s
console.log(arr[0])              // logs 'this is the first element'
console.log(arr[1])              // logs 'this is the second element'
```

```
console.log(arr[arr.length - 1]) // logs 'this is the last element'
```

Array elements are object properties in the same way that `toString` is a property (to be specific, however, `toString()` is a method). Nevertheless, trying to access an element of an array as follows throws a syntax error because the property name is not valid:

```
console.log(arr.0) // a syntax error
```

There is nothing special about JavaScript arrays and the properties that cause this. JavaScript properties that begin with a digit cannot be referenced with dot notation and must be accessed using bracket notation.

For example, if you had an object with a property named `3d`, it can only be referenced using bracket notation.

```
let years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
console.log(years.0)   // a syntax error
console.log(years[0])  // works properly
```

```
renderer.3d.setTexture(model, 'character.png')     // a syntax error
renderer['3d'].setTexture(model, 'character.png')  // works properly
```

In the `3d` example, `'3d'` *had* to be quoted (because it begins with a digit). But it's also possible to quote the array indexes as well (e.g., `years['2']` instead of `years[2]`), although it's not necessary.

The `2` in `years[2]` is coerced into a string by the JavaScript engine through an implicit `toString` conversion. As a result, `'2'` and `'02'` would refer to two different slots on the `years` object, and the following example could be `true`:

```
console.log(years['2'] != years['02'])
```

Relationship between length and numerical properties

A JavaScript array's `length` property and numerical properties are connected.

Several of the built-in array methods (e.g., `join()`, `slice()`, `indexOf()`, etc.) take into account the value of an array's `length` property when they're called.

Other methods (e.g., `push()`, `splice()`, etc.) also result in updates to an array's `length` property.

```
const fruits = []
fruits.push('banana', 'apple', 'peach')

console.log(fruits.length) // 3
```

When setting a property on a JavaScript array when the property is a valid array index and that index is outside the current bounds of the array, the engine will update the array's `length` property accordingly:

```
fruits[5] = 'mango'
console.log(fruits[5])            // 'mango'
console.log(Object.keys(fruits))  // ['0', '1', '2', '5']
console.log(fruits.length)        // 6
```

Increasing the `length`.

```
fruits.length = 10
console.log(fruits)               // ['banana', 'apple', 'peach', empty x 2
console.log(Object.keys(fruits)) // ['0', '1', '2', '5']
console.log(fruits.length)       // 10
console.log(fruits[8])           // undefined
```

Decreasing the `length` property does, however, delete elements.

```
fruits.length = 2
console.log(Object.keys(fruits)) // ['0', '1']
console.log(fruits.length)       // 2
```

This is explained further on the `Array.length` page.

Creating an array using the result of a match

The result of a match between a `RegExp` and a string can create a JavaScript array. This array has properties and elements which provide information about the match. Such an array is returned by `RegExp.exec()`, `String.match()`, and `String.replace()`.

To help explain these properties and elements, see this example and then refer to the table below:

```
// Match one d followed by one or more b's followed by one d
// Remember matched b's and the following d
// Ignore case

const myRe = /d(b+)(d)/i
const myArray = myRe.exec('cdbBdbsbz')
```

The properties and elements returned from this match are as follows:

Property/Element	Description	Example
`input` Read only	The original string against which the regular expression was matched.	`"cdbBdbsbz"`
`index` Read only	The zero-based index of the match in the string.	`1`
`[0]` Read only	The last matched characters.	`"dbBd"`
`[1], ...[n]` Read only	Elements that specify the parenthesized substring matches (if included) in the regular expression. The number of possible parenthesized substrings is unlimited.	`[1]: "bB"` `[2]: "d"`

Constructor

`Array()`

Creates a new `Array` object.

Static properties

`get Array[@@species]`

The constructor function is used to create derived objects.

Static methods

`Array.from()`

Creates a new `Array` instance from an array-like or iterable object.

`Array.isArray()`

Returns `true` if the argument is an array, or `false` otherwise.

`Array.of()`

Creates a new `Array` instance with a variable number of arguments, regardless of number or type of the arguments.

Instance properties

`Array.prototype.length`

Reflects the number of elements in an array.

`Array.prototype[@@unscopables]`

A symbol containing property names to exclude from a `with` binding scope.

Instance methods

`Array.prototype.at()` 

Returns the array item at the given index. Accepts negative integers, which count back from the last item.

`Array.prototype.concat()`

Returns a new array that is this array joined with other array(s) and/or value(s).

`Array.prototype.copyWithin()`

Copies a sequence of array elements within the array.

`Array.prototype.entries()`

Returns a new *array iterator* object that contains the key/value pairs for each index in the array.

`Array.prototype.every()`

Returns `true` if every element in this array satisfies the testing function.

`Array.prototype.fill()`

Fills all the elements of an array from a start index to an end index with a static value.

`Array.prototype.filter()`

Returns a new array containing all elements of the calling array for which the provided filtering function returns `true`.

`Array.prototype.find()`

Returns the found `element` in the array, if some element in the array satisfies the testing function, or `undefined` if not found.

`Array.prototype.findIndex()`

Returns the found index in the array, if an element in the array satisfies the testing function, or `-1` if not found.

`Array.prototype.flat()`

Returns a new array with all sub-array elements concatenated into it recursively up to the specified depth.

`Array.prototype.flatMap()`

Returns a new array formed by applying a given callback function to each element of the array, and then flattening the result by one level.

`Array.prototype.forEach()`

Calls a function for each element in the array.

`Array.prototype.includes()`

Determines whether the array contains a value, returning `true` or `false` as appropriate.

`Array.prototype.indexOf()`

Returns the first (least) index of an element within the array equal to an element, or `-1` if none is found.

`Array.prototype.join()`

Joins all elements of an array into a string.

`Array.prototype.keys()`

Returns a new *array iterator* that contains the keys for each index in the array.

`Array.prototype.lastIndexOf()`

Returns the last (greatest) index of an element within the array equal to an element, or `-1` if none is found.

`Array.prototype.map()`

Returns a new array containing the results of calling a function on every element in this array.

`Array.prototype.pop()`

Removes the last element from an array and returns that element.

`Array.prototype.push()`

Adds one or more elements to the end of an array, and returns the new `length` of the array.

`Array.prototype.reduce()`

Apply a function against an accumulator and each value of the array (from left-to-right) as to reduce it to a single value.

`Array.prototype.reduceRight()`

Apply a function against an accumulator and each value of the array (from right-to-left) as to reduce it to a single value.

`Array.prototype.reverse()`

Reverses the order of the elements of an array *in place*. (First becomes the last, last becomes first.)

`Array.prototype.shift()`

Removes the first element from an array and returns that element.

`Array.prototype.slice()`

Extracts a section of the calling array and returns a new array.

`Array.prototype.some()`

Returns `true` if at least one element in this array satisfies the provided testing function.

`Array.prototype.sort()`

Sorts the elements of an array in place and returns the array.

`Array.prototype.splice()`

Adds and/or removes elements from an array.

`Array.prototype.toLocaleString()`

Returns a localized string representing the array and its elements. Overrides the `Object.prototype.toLocaleString()` method.

`Array.prototype.toString()`

Returns a string representing the array and its elements. Overrides the `Object.prototype.toString()` method.

`Array.prototype.unshift()`

Adds one or more elements to the front of an array, and returns the new `length` of the array.

`Array.prototype.values()`

Returns a new *array iterator* object that contains the values for each index in the array.

`Array.prototype[@@iterator]()`

Returns a new *array iterator* object that contains the values for each index in the array.

Examples

Creating an array

The following example creates an array, `msgArray`, with a length of `0`, then assigns values to `msgArray[0]` and `msgArray[99]`, changing the `length` of the array to `100`.

```
let msgArray = []
msgArray[0] = 'Hello'
msgArray[99] = 'world'


if (msgArray.length === 100) {
  console.log('The length is 100.')
}
```

Creating a two-dimensional array

The following creates a chessboard as a two-dimensional array of strings. The first move is made by copying the `'p'` in `board[6][4]` to `board[4][4]`. The old position at `[6][4]` is made blank.

```
let board = [
  ['R','N','B','Q','K','B','N','R'],
  ['P','P','P','P','P','P','P','P'],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  [' ',' ',' ',' ',' ',' ',' ',' '],
  ['p','p','p','p','p','p','p','p'],
  ['r','n','b','q','k','b','n','r'] ]

console.log(board.join('\n') + '\n\n')

// Move King's Pawn forward 2
board[4][4] = board[6][4]
board[6][4] = ' '
console.log(board.join('\n'))
```

Here is the output:

```
R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , ,
 , , , , , , ,
 , , , , , , ,
 , , , , , , ,
p,p,p,p,p,p,p,p
```

```
r,n,b,q,k,b,n,r

R,N,B,Q,K,B,N,R
P,P,P,P,P,P,P,P
 , , , , , , ,

 , , , , , , ,
 , , , ,p, , ,
 , , , , , , ,
p,p,p,p, ,p,p,p
r,n,b,q,k,b,n,r
```

Using an array to tabulate a set of values

```
values = []
for (let x = 0; x < 10; x++){
 values.push([
  2 ** x,
  2 * x ** 2
 ])
}
console.table(values)
```

Results in

```
// The first column is the index
0       1       0
1       2       2
2       4       8
3       8       18
4       16      32
5       32      50
6       64      72
7       128     98
8       256     128
9       512     162
```

Specifications

Specification
ECMAScript Language Specification (ECMAScript) # sec-array-objects

Browser compatibility

Report problems with this compatibility data on GitHub

`Array`	
Chrome	1
Edge	12
Firefox	1
Internet Explorer	4
Opera	4
Safari	1
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`Array()` **constructor**	
Chrome	1
Edge	12
Firefox	1
Internet Explorer	4

Opera	4
Safari	1
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`at`

Chrome	92
Edge	92
Firefox	90
Internet Explorer	**No**
Opera	**No**
Safari	**No**
WebView Android	92
Chrome Android	92
Firefox for Android	90
Opera Android	**No**
Safari on iOS	**No**
Samsung Internet	16.0
Deno	1.12
Node.js	16.6.0

`concat`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`copyWithin`

Chrome	45
Edge	12
Firefox	32
Internet Explorer	**No**
Opera	32
Safari	9
WebView Android	45
Chrome Android	45
Firefox for Android	32

Opera Android	32
Safari on iOS	9
Samsung Internet	5.0
Deno	1.0
Node.js	4.0.0

`entries`

Chrome	38
Edge	12
Firefox	28
Internet Explorer	**No**
Opera	25
Safari	8
WebView Android	38
Chrome Android	38
Firefox for Android	28
Opera Android	25
Safari on iOS	8
Samsung Internet	3.0
Deno	1.0
Node.js	0.12.0

`every`

Chrome	1
Edge	12
Firefox	1.5
Internet Explorer	9

Opera	9.5
Safari	3
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`fill`

Chrome	45
Edge	12
Firefox	31
Internet Explorer	**No**
Opera	32
Safari	8
WebView Android	45
Chrome Android	45
Firefox for Android	31
Opera Android	32
Safari on iOS	8
Samsung Internet	5.0
Deno	1.0
Node.js	4.0.0

`filter`

Chrome	1
Edge	12
Firefox	1.5
Internet Explorer	9
Opera	9.5
Safari	3
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`find`

Chrome	45
Edge	12
Firefox	25
Internet Explorer	**No**
Opera	32
Safari	8
WebView Android	45
Chrome Android	45
Firefox for Android	4

Opera Android	32
Safari on iOS	8
Samsung Internet	5.0
Deno	1.0
Node.js	4.0.0

`findIndex`

Chrome	45
Edge	12
Firefox	25
Internet Explorer	**No**
Opera	32
Safari	8
WebView Android	45
Chrome Android	45
Firefox for Android	4
Opera Android	32
Safari on iOS	8
Samsung Internet	5.0
Deno	1.0
Node.js	4.0.0

`flat`

Chrome	69
Edge	79
Firefox	62
Internet Explorer	**No**

Opera	56
Safari	12
WebView Android	69
Chrome Android	69
Firefox for Android	62
Opera Android	48
Safari on iOS	12
Samsung Internet	10.0
Deno	1.0
Node.js	11.0.0

`flatMap`

Chrome	69
Edge	79
Firefox	62
Internet Explorer	**No**
Opera	56
Safari	12
WebView Android	69
Chrome Android	69
Firefox for Android	62
Opera Android	48
Safari on iOS	12
Samsung Internet	10.0
Deno	1.0
Node.js	11.0.0

`forEach`

Chrome	1
Edge	12
Firefox	1.5
Internet Explorer	9
Opera	9.5
Safari	3
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`from`

Chrome	45
Edge	12
Firefox	32
Internet Explorer	**No**
Opera	32
Safari	9
WebView Android	45
Chrome Android	45
Firefox for Android	32

Opera Android	32
Safari on iOS	9
Samsung Internet	5.0
Deno	1.0
Node.js	4.0.0

`includes`

Chrome	47
Edge	14
Firefox	43
Internet Explorer	**No**
Opera	34
Safari	9
WebView Android	47
Chrome Android	47
Firefox for Android	43
Opera Android	34
Safari on iOS	9
Samsung Internet	5.0
Deno	1.0
Node.js	6.0.0

`indexOf`

Chrome	1
Edge	12
Firefox	1.5
Internet Explorer	9

Opera	9.5
Safari	3
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`isArray`

Chrome	5
Edge	12
Firefox	4
Internet Explorer	9
Opera	10.5
Safari	5
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	14
Safari on iOS	5
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`join`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`keys`

Chrome	38
Edge	12
Firefox	28
Internet Explorer	**No**
Opera	25
Safari	8
WebView Android	38
Chrome Android	38
Firefox for Android	28

Opera Android	25
Safari on iOS	8
Samsung Internet	3.0
Deno	1.0
Node.js	0.12.0

`lastIndexOf`

Chrome	1
Edge	12
Firefox	1.5
Internet Explorer	9
Opera	9.5
Safari	3
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`length`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	4

Opera	4
Safari	1
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`map`

Chrome	1
Edge	12
Firefox	1.5
Internet Explorer	9
Opera	9.5
Safari	3
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

of

Chrome	45
Edge	12
Firefox	25
Internet Explorer	**No**
Opera	26
Safari	9
WebView Android	39
Chrome Android	39
Firefox for Android	25
Opera Android	26
Safari on iOS	9
Samsung Internet	4.0
Deno	1.0
Node.js	4.0.0

pop

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4

Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`push`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`reduce`

Chrome	3
Edge	12
Firefox	3
Internet Explorer	9

Opera	10.5
Safari	5
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	14
Safari on iOS	4
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`reduceRight`

Chrome	3
Edge	12
Firefox	3
Internet Explorer	9
Opera	10.5
Safari	5
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	14
Safari on iOS	4
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`reverse`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`shift`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4

Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`slice`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	4
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`some`

Chrome	1
Edge	12
Firefox	1.5
Internet Explorer	9

Opera	9.5
Safari	3
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`sort`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

Stable sorting	
Chrome	70
Edge	79
Firefox	3
Internet Explorer	**No**
Opera	57
Safari	10.1
WebView Android	70
Chrome Android	70
Firefox for Android	4
Opera Android	49
Safari on iOS	10.3
Samsung Internet	10.0
Deno	1.0
Node.js	12.0.0

`splice`	
Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4

Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`toLocaleString`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	37
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`locales`
parameter

Chrome	24
Edge	79
Firefox	52

Internet Explorer	**No**
Opera	15
Safari	6.1
WebView Android	37
Chrome Android	25
Firefox for Android	56
Opera Android	14
Safari on iOS	6.1
Samsung Internet	2.0
Deno	1.8
Node.js	13.0.0

`options`
parameter

Chrome	24
Edge	79
Firefox	52
Internet Explorer	**No**
Opera	15
Safari	6.1
WebView Android	37
Chrome Android	25
Firefox for Android	56
Opera Android	14
Safari on iOS	6.1
Samsung Internet	2.0
Deno	1.0

Node.js	0.12.0

`toSource`

Chrome	**No**
Edge	**No**
Firefox	**1–74**
Internet Explorer	**No**
Opera	**No**
Safari	**No**
WebView Android	**No**
Chrome Android	**No**
Firefox for Android	4
Opera Android	**No**
Safari on iOS	**No**
Samsung Internet	**No**
Deno	**No**
Node.js	**No**

`toString`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	4
Opera	4
Safari	1
WebView Android	37

Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`unshift`

Chrome	1
Edge	12
Firefox	1
Internet Explorer	5.5
Opera	4
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0
Deno	1.0
Node.js	0.10.0

`values`

Chrome	66
Edge	12

Firefox	60
Internet Explorer	**No**
Opera	53
Safari	9
WebView Android	66
Chrome Android	66
Firefox for Android	60
Opera Android	47
Safari on iOS	9
Samsung Internet	9.0
Deno	1.0
Node.js	10.9.0

`@@iterator`

Chrome	38
Edge	12
Firefox	36
Internet Explorer	**No**
Opera	25
Safari	10
WebView Android	38
Chrome Android	38
Firefox for Android	36
Opera Android	25
Safari on iOS	10
Samsung Internet	3.0

Deno	1.0
Node.js	0.12.0

`@@species`

Chrome	51
Edge	79
Firefox	48
Internet Explorer	**No**
Opera	38
Safari	10
WebView Android	51
Chrome Android	51
Firefox for Android	48
Opera Android	41
Safari on iOS	10
Samsung Internet	5.0
Deno	1.0
Node.js	6.5.0

`@@unscopables`

Chrome	38
Edge	12
Firefox	48
Internet Explorer	**No**
Opera	25
Safari	10
WebView Android	38

Chrome Android	38
Firefox for Android	48
Opera Android	25
Safari on iOS	10
Samsung Internet	3.0
Deno	1.0
Node.js	0.12.0

Full support

Partial support

No support

Non-standard. Check cross-browser support before using.

See implementation notes.

User must explicitly enable this feature.

Uses a non-standard name.

See also

- From the JavaScript Guide:
  - "Indexing object properties"

- “Indexing object properties”
- “Indexed collections: `Array` object”

- Typed Arrays

- RangeError: invalid array length

Last modified: Sep 3, 2021, by MDN contributors