Array.prototype.sort()

Related Topics

Standard built-in objects

Array
- Properties
  - Array.length
  - Array.prototype
  - Array.prototype[@@unscopables]
- Methods
  - Array.from()
  - Array.isArray()
  - Array.observe()
  - Array.of()
  - Array.prototype.concat()
  - Array.prototype.copyWithin()
  - Array.prototype.entries()
  - Array.prototype.every()
  - Array.prototype.fill()
  - Array.prototype.filter()
  - Array.prototype.find()
  - Array.prototype.findIndex()
  - Array.prototype.flat()
  - Array.prototype.flatMap()
  - Array.prototype.forEach()
  - Array.prototype.includes()
  - Array.prototype.indexOf()
  - Array.prototype.join()
  - Array.prototype.keys()
  - Array.prototype.lastIndexOf()
  - Array.prototype.map()
  - Array.prototype.pop()
  - Array.prototype.push()
  - Array.prototype.reduce()
  - Array.prototype.reduceRight()
  - Array.prototype.reverse()
  - Array.prototype.shift()
  - Array.prototype.slice()
  - Array.prototype.some()
  - Array.prototype.sort()
  - Array.prototype.splice()
  - Array.prototype.toLocaleString()
  - Array.prototype.toSource()
  - Array.prototype.toString()
  - Array.prototype.unshift()
  - Array.prototype.values()
  - Array.prototype[@@iterator]()
  - Array.unobserve()
  - get Array[@@species]

Inheritance:

Function
- Properties
- Methods

Object
- Properties
- Methods

The `sort()` method sorts the elements of an array *in place* and returns the array. The sort is not necessarily stable. The default sort order is according to string Unicode code points.

The time and space complexity of the sort cannot be guaranteed as it is implementation dependent.

JavaScript Demo: Array.sort()

```
var months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months);
// expected output: Array ["Dec", "Feb", "Jan", "March"]

var array1 = [1, 30, 4, 21];
array1.sort();
console.log(array1);
// expected output: Array [1, 21, 30, 4]

```

Run ›

Reset

Syntax

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

Parameters

`compareFunction` Optional
: Specifies a function that defines the sort order. If omitted, the array is sorted according to each character's Unicode code point value, according to the string conversion of each element.

Return value

The sorted array. Note that the array is sorted *in place*, and no copy is made.

Description

If `compareFunction` is not supplied, all non-`undefined` array elements are sorted by converting them to strings and comparing strings in Unicode code point order. For example, "Banana" comes before "cherry". In a numeric sort, 9 comes before 80, but because numbers are converted to strings, "80" comes before "9" in Unicode order. All `undefined` elements are sorted to the end of the array.

If `compareFunction` is supplied, all non-`undefined` array elements are sorted according to the return value of the compare function (all `undefined` elements are sorted to the end of the array, with no call to `compareFunction`). If `a` and `b` are two elements being compared, then:

- If `compareFunction(a, b)` is less than 0, sort `a` to an index lower than `b`, i.e. `a` comes first.
- If `compareFunction(a, b)` returns 0, leave `a` and `b` unchanged with respect to each other, but sorted with respect to all different elements. Note: the ECMAscript standard does not guarantee this behaviour, and thus not all browsers (e.g. Mozilla versions dating back to at least 2003) respect this.
- If `compareFunction(a, b)` is greater than 0, sort `b` to an index lower than `a`, i.e. `b` comes first.
- `compareFunction(a, b)` must always return the same value when given a specific pair of elements a and b as its two arguments. If inconsistent results are returned then the sort order is undefined.

So, the compare function has the following form:

```
function compare(a, b) {
  if (a is less than b by some ordering criterion) {
    return -1;
  }
  if (a is greater than b by the ordering criterion) {
    return 1;
  }
  // a must be equal to b
  return 0;
}
```

To compare numbers instead of strings, the compare function can simply subtract `b` from `a`. The following function will sort the array ascending (if it doesn't contain `Infinity` and `NaN`):

```
function compareNumbers(a, b) {
  return a - b;
}
```

The `sort` method can be conveniently used with function expressions (and closures):

```
var numbers = [4, 2, 5, 1, 3];
numbers.sort(function(a, b) {
  return a - b;
});
console.log(numbers);

// [1, 2, 3, 4, 5]
```

Objects can be sorted given the value of one of their properties.

```
var items = [
  { name: 'Edward', value: 21 },
  { name: 'Sharpe', value: 37 },
  { name: 'And', value: 45 },
  { name: 'The', value: -12 },
  { name: 'Magnetic', value: 13 },
  { name: 'Zeros', value: 37 }
];

// sort by value
items.sort(function (a, b) {
  return a.value - b.value;
});

// sort by name
items.sort(function(a, b) {
  var nameA = a.name.toUpperCase(); // ignore upper and lowercase
  var nameB = b.name.toUpperCase(); // ignore upper and lowercase
  if (nameA < nameB) {
    return -1;
  }
  if (nameA > nameB) {
    return 1;
  }

  // names must be equal
  return 0;
});
```

Examples

Creating, displaying, and sorting an array

The following example creates four arrays and displays the original array, then the sorted arrays. The numeric arrays are sorted without, then with, a compare function.

```
var stringArray = ['Blue', 'Humpback', 'Beluga'];
var numericStringArray = ['80', '9', '700'];
var numberArray = [40, 1, 5, 200];
var mixedNumericArray = ['80', '9', '700', 40, 1, 5, 200];

function compareNumbers(a, b) {
  return a - b;
}

console.log('stringArray:', stringArray.join());
console.log('Sorted:', stringArray.sort());

console.log('numberArray:', numberArray.join());
console.log('Sorted without a compare function:', numberArray.sort());
console.log('Sorted with compareNumbers:', numberArray.sort(compareNumbers));

console.log('numericStringArray:', numericStringArray.join());
console.log('Sorted without a compare function:', numericStringArray.sort());
console.log('Sorted with compareNumbers:', numericStringArray.sort(compareNum

console.log('mixedNumericArray:', mixedNumericArray.join());
console.log('Sorted without a compare function:', mixedNumericArray.sort());
console.log('Sorted with compareNumbers:', mixedNumericArray.sort(compareNumb
```

This example produces the following output. As the output shows, when a compare function is used, numbers sort correctly whether they are numbers or numeric strings.

```
stringArray: Blue,Humpback,Beluga
Sorted: Beluga,Blue,Humpback

numberArray: 40,1,5,200
Sorted without a compare function: 1,200,40,5
Sorted with compareNumbers: 1,5,40,200

numericStringArray: 80,9,700
Sorted without a compare function: 700,80,9
Sorted with compareNumbers: 9,80,700

mixedNumericArray: 80,9,700,40,1,5,200
Sorted without a compare function: 1,200,40,5,700,80,9
Sorted with compareNumbers: 1,5,9,40,80,200,700
```

Sorting non-ASCII characters

For sorting strings with non-ASCII characters, i.e. strings with accented characters (e, é, è, a, ä, etc.), strings from languages other than English: use `String.localeCompare`. This function can compare those characters so they appear in the right order.

```
var items = ['réservé', 'premier', 'cliché', 'communiqué', 'café', 'adieu'];
items.sort(function (a, b) {
  return a.localeCompare(b);
});

// items is ['adieu', 'café', 'cliché', 'communiqué', 'premier', 'réservé']
```

Sorting with map

The `compareFunction` can be invoked multiple times per element within the array. Depending on the `compareFunction`'s nature, this may yield a high overhead. The more work a `compareFunction` does and the more elements there are to sort, the wiser it may be to consider using a map for sorting. The idea is to walk the array once to extract the actual values used for sorting into a temporary array, sort the temporary array and then walk the temporary array to achieve the right order.

```
// the array to be sorted
var list = ['Delta', 'alpha', 'CHARLIE', 'bravo'];

// temporary array holds objects with position and sort-value
var mapped = list.map(function(el, i) {
  return { index: i, value: el.toLowerCase() };
})

// sorting the mapped array containing the reduced values
mapped.sort(function(a, b) {
  if (a.value > b.value) {
    return 1;
  }
  if (a.value < b.value) {
    return -1;
  }
  return 0;
});

// container for the resulting order
var result = mapped.map(function(el){
  return list[el.index];
});
```

Specifications

Specification	Status	Comment
ECMAScript 1st Edition (ECMA-262)	ST Standard	Initial definition.
ECMAScript 5.1 (ECMA-262) The definition of 'Array.prototype.sort' in that specification.	ST Standard	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Array.prototype.sort' in that specification.	ST Standard	
ECMAScript Latest Draft (ECMA-262) The definition of 'Array.prototype.sort' in that specification.	D Draft	

Browser compatibility

New compatibility tables are in beta ▾

Basic support	1	Yes	1	5.5	Yes	Yes	Yes	Yes	Yes	4	Yes	Yes	Yes	Yes

Full support

See also

- `Array.prototype.reverse()`
- `String.prototype.localeCompare()`

Tags: Array JavaScript Method Prototype

Contributors to this page: chrisdavidmills, yeerkkiller1, diablero13, vedala, janderson215, rwaldron, Fl4m3Ph03n1x, wbamberg, davidjb, fscholz, BlueSkyEngineer, AmericanKulak, erikadoyle, Daniel Hug, mauroporrasp, chrisveness, KurtHarlandLarson, crueschenberg, SevenOutman, TWiStErRob, jenson555, eduardoboucas, DoomyTheFroomy, claudiahdz, izaakrogan, rutsky, studiowangfei, chris.johnson, jsx, JacksonGL, xixixao, hbkdsm, aurelianofs, joyously, tiffon, Mingun, ppseafield, AntonNiklasson, felixrabe, Glutnix, Granjow, rcaracaus, icarot, roryokane, nebulon, dbruant, robatron, Sheppy, madarche, ethertank, fkling, rodneyrehm, evilpie, dotnetCarpenter, Brendan, Brettz9, normanr, Husky, quotemstr, Neil, Trinithis, Mgjbot, Kris.kowal, Waldo, H3h, 1212jtraceur, Yuichirou, Nickolay, Ecmanaut, Maian, Dria

Last updated by: chrisdavidmills, May 12, 2018, 9:03:37 AM