```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Adding elements with high indexes can create undefined "holes" in an array.
</p>

<p id="demo"></p>

<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon";

fLen = fruits.length;
text = "";
for (i = 0; i < fLen; i++) {
  text += fruits[i] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

Example

```
var person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
var x = person.length;     // person.length will return 3
var y = person[0];         // person[0] will return "John"
```

Converting Arrays to Strings

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result:

Banana,Orange,Apple,Mango

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>toString()</h2>

<p>The toString() method returns an array as a comma separated string:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
</script>

</body>
</html>
```

The `join()` method also joins all array elements into a string.

It behaves just like `toString()`, but in addition you can specify the separator:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result:

Banana * Orange * Apple * Mango

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>join()</h2>

<p>The join() method joins array elements into a string.</p>

<p>It this example we have used " * " as a separator between the elements:</p

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
</script>

</body>
</html>
```

Popping

The `pop()` method removes the last element from an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();                 // Removes the last element ("Mango") from fruits
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>pop()</h2>

<p>The pop() method removes the last element from an array.</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.pop();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

The `pop()` method returns the value that was "popped out":

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.pop();       // the value of x is "Mango"
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>pop()</h2>

<p>The pop() method removes the last element from an array.</p>

<p>The return value of the pop() method is the removed item.</p>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
document.getElementById("demo2").innerHTML = fruits.pop();
document.getElementById("demo3").innerHTML = fruits;
</script>

</body>
</html>
```

Pushing

The `push()` method adds a new element to an array (at the end):

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");        //  Adds a new element ("Kiwi") to fruits
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>push()</h2>

<p>The push() method appends a new element to an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.push("Kiwi");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

The `push()` method returns the new array length:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");   //  the value of x is 5
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>push()</h2>

<p>The push() method returns the new array length.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;

function myFunction() {
  document.getElementById("demo2").innerHTML = fruits.push("Kiwi");
  document.getElementById("demo1").innerHTML = fruits;
}
</script>

</body>
</html>
```

Shifting Elements

Shifting is equivalent to popping, working on the first element instead of the last.

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();            // Removes the first element "Banana" from fruits
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>shift()</h2>

<p>The shift() method removes the first element of an array (and "shifts" all
other elements to the left):</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
fruits.shift();
document.getElementById("demo2").innerHTML = fruits;
</script>

</body>
</html>
```

The `shift()` method returns the string that was "shifted out":

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.shift();    // the value of x is "Banana"
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>shift()</h2>

<p>The shift() method returns the element that was shifted out.</p>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
document.getElementById("demo2").innerHTML = fruits.shift();
document.getElementById("demo3").innerHTML = fruits;
</script>

</body>
</html>
```

The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Adds a new element "Lemon" to fruits
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>unshift()</h2>

<p>The unshift() method adds new elements to the beginning of an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.unshift("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

The `unshift()` method returns the new array length.

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");    // Returns 5
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>unshift()</h2>

<p>The unshift() method returns the length of the new array:</p>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = fruits;
document.getElementById("demo2").innerHTML = fruits.unshift("Lemon");
document.getElementById("demo3").innerHTML = fruits;
</script>

<p><b>Note:</b> The unshift() method does not work properly in Internet
Explorer 8 and earlier, the values will be inserted, but the return value
will be <em>undefined</em>.</p>

</body>
</html>
```

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";         // Changes the first element of fruits to "Kiwi"
```

Try it Yourself »

he `length` property provides an easy way to append a new element to an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";          // Appends "Kiwi" to fruits
```

Try it Yourself »

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<p>The length property provides an easy way to append new elements to an array
without using the push() method.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits[fruits.length] = "Kiwi";
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Deleting Elements

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator `delete`:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];           // Changes the first element in fruits to undefined
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<p>Deleting elements leaves undefined holes in an array.</p>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML =
"The first fruit is: " + fruits[0];
delete fruits[0];
document.getElementById("demo2").innerHTML =
"The first fruit is: " + fruits[0];
</script>

</body>
</html>
```

Splicing an Array

The `splice()` method can be used to add new items to an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>splice()</h2>

<p>The splice() method adds new elements to an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo1"></p>
<p id="demo2"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = "Original Array:<br>" + fruits;
function myFunction() {
  fruits.splice(2, 0, "Lemon", "Kiwi");
  document.getElementById("demo2").innerHTML = "New Array:<br>" + fruits;
}
</script>

</body>
</html>
```

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The `splice()` method returns an array with the deleted items:

```
<h2>splice()</h2>

<p>The splice() method adds new elements to an array, and returns an array
with the deleted elements (if any).</p>

<button onclick="myFunction()">Try it</button>

<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo1").innerHTML = "Original Array:<br> " +
fruits;

function myFunction() {
  var removed = fruits.splice(2, 2, "Lemon", "Kiwi");
  document.getElementById("demo2").innerHTML = "New Array:<br>" + fruits;
  document.getElementById("demo3").innerHTML = "Removed Items:<br> " +
removed;
}
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The splice() method adds new elements to an array, and returns an array with the deleted elements (if any).

Try it

Original Array:
Banana,Orange,Apple,Mango

New Array:
Banana,Orange,Lemon,Kiwi

Removed Items:
Apple,Mango

Using splice() to Remove Elements

With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);          // Removes the first element of fruits
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>splice()</h2>

<p>The splice() methods can be used to remove array elements.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
  fruits.splice(0, 1);
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Array Methods

splice()

The splice() methods can be used to remove array elements.

Try it

Orange,Apple,Mango

Merging (Concatenating) Arrays

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);   // Concatenates (joins) myGirls and myBoys
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>concat()</h2>

<p>The concat() method is used to merge (concatenate) arrays:</p>

<p id="demo"></p>

<script>
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);

document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

JavaScript Array Methods

concat()

The concat() method is used to merge (concatenate) arrays:

Cecilie,Lone,Emil,Tobias,Linus

The `concat()` method does not change the existing arrays. It always returns a new array.

The `concat()` method can take any number of array arguments:

Example (Merging Three Arrays)

```
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];
var myChildren = arr1.concat(arr2, arr3);   // Concatenates arr1 with arr2 and arr3
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>concat()</h2>

<p>The concat() method is used to merge (concatenate) arrays:</p>

<p id="demo"></p>

<script>
var arr1 = ["Cecilie", "Lone"];
var arr2 = ["Emil", "Tobias", "Linus"];
var arr3 = ["Robin", "Morgan"];

var myChildren = arr1.concat(arr2, arr3);

document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

The `concat()` method can also take values as arguments:

Example (Merging an Array with Values)

```
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>concat()</h2>

<p>The concat() method can also merge values to arrays:</p>

<p id="demo"></p>

<script>
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);
document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

JavaScript Array Methods

concat()

The concat() method can also merge values to arrays:

Cecilie,Lone,Emil,Tobias,Linus

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>concat()</h2>

<p>The concat() method can also merge values to arrays:</p>

<p id="demo"></p>

<script>
var arr1 = ["Cecilie", "Lone"];
var myChildren = arr1.concat(["Emil", "Tobias", "Linus"]);
document.getElementById("demo").innerHTML = myChildren;
</script>

</body>
</html>
```

JavaScript Array Methods

concat()

The concat() method can also merge values to arrays:

Cecilie,Lone,Emil,Tobias,Linus

Slicing an Array

The `slice()` method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>slice()</h2>

<p>This example slices out a part of an array starting from array element 1
("Orange"):</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

This example slices out a part of an array starting from array element 1 ("Orange"):

Banana,Orange,Lemon,Apple,Mango

Orange,Lemon,Apple,Mango

The `slice()` method creates a new array. It does not remove any elements from the source array.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>slice()</h2>

<p>This  example slices out a part of an array starting from array element 3
("Apple")</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(3);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

This example slices out a part of an array starting from array element 3 ("Apple")

Banana,Orange,Lemon,Apple,Mango

Apple,Mango

The `slice()` method can take two arguments like `slice(1, 3)`.

The method then selects elements from the start argument, and up to (but not including) the end argument.

Example

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1, 3);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>slice()</h2>

<p>When the slice() method is given two arguments, it selects array elements
from the start argument, and up to (but not included) the end argument:</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1,3);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

When the slice() method is given two arguments, it selects array elements from the start argument, and up to (but not included) the end argument:

Banana,Orange,Lemon,Apple,Mango

Orange,Lemon

If the end argument is omitted, like in the first examples, the `slice()` method slices out the rest of the array.

Example

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(2);
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Methods</h2>

<h2>slice()</h2>

<p>This example slices out a part of an array starting from array element 2
("Lemon"):</p>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(2);
document.getElementById("demo").innerHTML = fruits + "<br><br>" + citrus;
</script>

</body>
</html>
```

JavaScript Array Methods

slice()

This example slices out a part of an array starting from array element 2 ("Lemon"):

Banana,Orange,Lemon,Apple,Mango

Lemon,Apple,Mango

JavaScript Sorting Arrays

‹ Previous

Sorting an Array

The `sort()` method sorts an array alphabetically:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();          // Sorts the elements of fruits
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort</h2>

<p>The sort() method sorts an array alphabetically.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.sort();
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

JavaScript Array Sort

The sort() method sorts an array alphabetically.

Try it

Apple,Banana,Mango,Orange

Reversing an Array

The `reverse()` method reverses the elements in an array.

You can use it to sort an array in descending order:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();          // First sort the elements of fruits
fruits.reverse();       // Then reverse the order of the elements
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort Reverse</h2>

<p>The reverse() method reverses the elements in an array.</p>
<p>By combining sort() and reverse() you can sort an array in descending
order.</p>
<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
// Create and display an array:
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  // First sort the array
  fruits.sort();
  // Then reverse it:
  fruits.reverse();
  document.getElementById("demo").innerHTML = fruits;
}
</script>
```

JavaScript Array Sort Reverse

The reverse() method reverses the elements in an array.

By combining sort() and reverse() you can sort an array in descending order.

Try it

Orange,Mango,Banana,Apple

Numeric Sort

By default, the `sort()` function sorts values as **strings**.

This works well for strings ("Apple" comes before "Banana").

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the `sort()` method will produce incorrect result when sorting numbers.

You can fix this by providing a **compare function**:

Example

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort</h2>

<p>Click the button to sort the array in ascending order.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo").innerHTML = points;

function myFunction() {
  points.sort(function(a, b){return a - b});
  document.getElementById("demo").innerHTML = points;
}
</script>

</body>
</html>
```

JavaScript Array Sort

Click the button to sort the array in ascending order.

Try it

1,5,10,25,40,100

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort</h2>

<p>Click the button to sort the array in descending order.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo").innerHTML = points;

function myFunction() {
  points.sort(function(a, b){return b - a});
  document.getElementById("demo").innerHTML = points;
}
</script>

</body>
</html>
```

The Compare Function

The purpose of the compare function is to define an alternative sort order.

The compare function should return a negative, zero, or positive value, depending on the arguments:

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

When the `sort()` function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.

Example:

When comparing 40 and 100, the `sort()` method calls the compare function(40,100).

The function calculates 40-100, and returns -60 (a negative value).

The sort function will sort 40 as a value lower than 100.

You can use this code snippet to experiment with numerically and alphabetically sorting:

Find the Highest (or Lowest) Array Value

There are no built-in functions for finding the max or min value in an array.

However, after you have sorted an array, you can use the index to obtain the highest and lowest values.

Sorting ascending:

Example

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});
// now points[0] contains the lowest value
// and points[points.length-1] contains the highest value
```

Try it Yourself »

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort</h2>

<p>The lowest number is <span id="demo"></span>.</p>

<script>
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
document.getElementById("demo").innerHTML = points[0];
</script>

</body>
</html>
```

Using Math.max() on an Array

You can use `Math.max.apply` to find the highest number in an array:

Example

```
function myArrayMax(arr) {
  return Math.max.apply(null, arr);
}
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array Sort</h2>

<p>The highest number is <span id="demo"></span>.</p>

<script>
var points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo").innerHTML = myArrayMax(points);

function myArrayMax(arr) {
  return Math.max.apply(null, arr);
}
</script>

</body>
</html>
```

Using Math.min() on an Array

You can use `Math.min.apply` to find the lowest number in an array:

Example

```
function myArrayMin(arr) {
  return Math.min.apply(null, arr);
}
```

`Math.max.apply(null, [1, 2, 3])` is equivalent to `Math.max(1, 2, 3)`.

`Math.min.apply(null, [1, 2, 3])` is equivalent to `Math.min(1, 2, 3)`.

Sorting Object Arrays

JavaScript arrays often contain objects:

Example

```
var cars = [
  {type:"Volvo", year:2016},
  {type:"Saab", year:2001},
  {type:"BMW", year:2010}
];
```

Example

```
cars.sort(function(a, b){return a.year - b.year});
```

```
<button onclick="myFunction()">Sort</button>

<p id="demo"></p>

<script>
var cars = [
  {type:"Volvo", year:2016},
  {type:"Saab", year:2001},
  {type:"BMW", year:2010}
];

displayCars();

function myFunction() {
  cars.sort(function(a, b){return a.year - b.year});
  displayCars();
}

function displayCars() {
  document.getElementById("demo").innerHTML =
  cars[0].type + " " + cars[0].year + "<br>" +
  cars[1].type + " " + cars[1].year + "<br>" +
  cars[2].type + " " + cars[2].year;
}
</script>

</body>
```

JavaScript Array Sort

Click the buttons to sort car objects on age.

Sort

Saab 2001
BMW 2010
Volvo 2016

```
<h2>JavaScript Array Sort</h2>

<p>Click the buttons to sort car objects on type.</p>

<button onclick="myFunction()">Sort</button>

<p id="demo"></p>

<script>
var cars = [
  {type:"Volvo", year:2016},
  {type:"Saab", year:2001},
  {type:"BMW", year:2010}
];

displayCars();

function myFunction() {
  cars.sort(function(a, b){
    var x = a.type.toLowerCase();
    var y = b.type.toLowerCase();
    if (x < y) {return -1;}
    if (x > y) {return 1;}
    return 0;
  });
  displayCars();
}
```

JavaScript Array Sort

Click the buttons to sort car objects on type.

Sort

Volvo 2016
Saab 2001
BMW 2010

JavaScript Array Iteration Methods

‹ Previous

Array iteration methods operate on every array item.

Array.forEach()

The `forEach()` method calls a function (a callback function) once for each array element.

Example

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.forEach()</h2>

<p>Calls a function once for each array element.</p>

<p id="demo"></p>

<script>
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = txt;

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.forEach()</h2>

<p>Calls a function once for each array element.</p>

<p id="demo"></p>

<script>
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = txt;

function myFunction(value, index, array) {
  txt = txt + value + "<br>";
}
</script>

</body>
</html>
```

JavaScript Array.forEach()

Calls a function once for each array element.

45
4
9
16
25

Array.map()

The `map()` method creates a new array by performing a function on each array element.

The `map()` method does not execute the function for array elements without values.

The `map()` method does not change the original array.

This example multiplies each array value by 2:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) {
  return value * 2;
}
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.map()</h2>

<p>Creates a new array by performing a function on each array element.</p>

<p id="demo"></p>

<script>
var numbers1 = [45, 4, 9, 16, 25];
var numbers2 = numbers1.map(myFunction);

document.getElementById("demo").innerHTML = numbers2;

function myFunction(value, index, array) {
  return value * 2;
}
</script>

</body>
</html>
```

Array.filter()

The `filter()` method creates a new array with array elements that passes a test.

This example creates a new array from elements with a value larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.filter()</h2>

<p>Creates a new array with all array elements that passes a test.</p>

<p id="demo"></p>

<script>
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

document.getElementById("demo").innerHTML = over18;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.filter()</h2>

<p>Creates a new array with all array elements that passes a test.</p>

<p id="demo"></p>

<script>
var numbers = [45, 4, 9, 16, 25];
var over18 = numbers.filter(myFunction);

document.getElementById("demo").innerHTML = over18;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

JavaScript Array.filter()

Creates a new array with all array elements that passes a test.

45,25

Array.reduce()

The `reduce()` method runs a function on each array element to produce (reduce it to) a single value.

The `reduce()` method works from left-to-right in the array. See also `reduceRight()`.

> The `reduce()` method does not reduce the original array.

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.reduce()</h2>

<p>This example finds the sum of all numbers in an array:</p>

<p id="demo"></p>

<script>
var numbers = [45, 4, 9, 16, 25];
var sum = numbers.reduce(myFunction);

document.getElementById("demo").innerHTML = "The sum is " + sum;

function myFunction(total, value, index, array) {
  return total + value;
}
</script>

</body>
</html>
```

JavaScript Array.reduce()

This example finds the sum of all numbers in an array:

The sum is 99

The `reduce()` method can accept an initial value:

Example

```
var numbers1 = [45, 4, 9, 16, 25];
var sum = numbers1.reduce(myFunction, 100);

function myFunction(total, value) {
  return total + value;
}
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.reduce()</h2>

<p>This example finds the sum of all numbers in an array:</p>

<p id="demo"></p>

<script>
var numbers = [45, 4, 9, 16, 25];
var sum = numbers.reduce(myFunction, 100);

document.getElementById("demo").innerHTML = "The sum is " + sum;

function myFunction(total, value) {
  return total + value;
}
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.reduce()</h2>

<p>This example finds the sum of all numbers in an array:</p>

<p id="demo"></p>

<script>
var numbers = [45, 4, 9, 16, 25];
var sum = numbers.reduce(myFunction);

document.getElementById("demo").innerHTML = "The sum is " + sum;

function myFunction(total, value, index, array) {
  return total + value;
}
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.reduceRight()</h2>

<p>This example finds the sum of all numbers in an array:</p>

<p id="demo"></p>

<script>
var numbers = [45, 4, 9, 16, 25];
var sum = numbers.reduceRight(myFunction);

document.getElementById("demo").innerHTML = "The sum is " + sum;

function myFunction(total, value, index, array) {
  return total + value;
}
</script>

</body>
</html>
```

Array.reduceRight()

The `reduceRight()` method runs a function on each array element to produce (reduce it to) a single value.

The `reduceRight()` works from right-to-left in the array. See also `reduce()`.

> The `reduceRight()` method does not reduce the original array.

Array.every()

The `every()` method check if all array values pass a test.

This example check if all array values are larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.every()</h2>

<p>The every() method checks if all array values pass a test.</p>

<p id="demo"></p>

<script>
var numbers = [45, 4, 9, 16, 25];
var allOver18 = numbers.every(myFunction);

document.getElementById("demo").innerHTML = "All over 18 is " + allOver18;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

JavaScript Array.every()

The every() method checks if all array values pass a test.

All over 18 is false

Array.indexOf()

The `indexOf()` method searches an array for an element value and returns its position.

Note: The first item has position 0, the second item has position 1, and so on.

Example

Search an array for the item "Apple":

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.indexOf()</h2>


<p id="demo"></p>

<script>
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");
document.getElementById("demo").innerHTML = "Apple is found in position " + (a
+ 1);
</script>

<p>The indexOf() does not work in Internet Explorer 8 and earlier versions.</p>

</body>
</html>
```

JavaScript Array.indexOf()

Apple is found in position 1

The indexOf() does not work in Internet Explorer 8 and earlier versions.

Syntax

```
array.indexOf(item, start)
```

item	Required. The item to search for.
start	Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the end.

`Array.indexOf()` returns -1 if the item is not found.

If the item is present more than once, it returns the position of the first occurrence.

Array.lastIndexOf()

`Array.lastIndexOf()` is the same as `Array.indexOf()`, but searches from the end of the array.

Example

Search an array for the item "Apple":

```
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.lastIndexOf("Apple");
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.lastIndexOf()</h2>


<p id="demo"></p>

<script>
var fruits = ["Apple", "Orange", "Apple", "Mango"];
var a = fruits.lastIndexOf("Apple");
document.getElementById("demo").innerHTML = "Apple is found in position " + (a
+ 1);
</script>

<p>The lastIndexOf() does not work in Internet Explorer 8 and earlier versions.
</p>

</body>
</html>
```

Array.some()

The `some()` method check if some array values pass a test.

This example check if some array values are larger than 18:

Example

```
var numbers = [45, 4, 9, 16, 25];
var someOver18 = numbers.some(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Array.some()</h2>

<p>The some() method checks if some array values pass a test.</p>

<p id="demo"></p>

<script>
var numbers = [45, 4, 9, 16, 25];
var someOver18 = numbers.some(myFunction);

document.getElementById("demo").innerHTML = "Some over 18 is " + someOver18;

function myFunction(value, index, array) {
  return value > 18;
}
</script>

</body>
</html>
```

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

Use the following syntax to create an **Array** object –

```
var fruits = new Array( "apple", "orange", "mango" );
```

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows –

```
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

```
fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element
```

Its syntax is as follows −

```
object.prototype.name = value
```

Example

Try the following example.

Live Demo

```
<html>
   <head>
      <title>User-defined objects</title>
      <script type = "text/javascript">
         function book(title, author) {
            this.title = title;
            this.author  = author;
         }
      </script>
   </head>
   <body>
      <script type = "text/javascript">
         var myBook = new book("Perl", "Mohtashim");
         book.prototype.price = null;
         myBook.price = 100;

         document.write("Book title is : " + myBook.title + "<br>");
         document.write("Book author is : " + myBook.author + "<br>");
         document.write("Book price is : " + myBook.price + "<br>");
      </script>
   </body>
</html>
```

JavaScript Object - prototype

Previous Page Next Page

Description

The prototype property allows you to add properties and methods to any object (Number, Boolean, String and Date etc.).

Note – Prototype is a global property which is available with almost all the objects.

Array Properties

Here is a list of the properties of the Array object along with their description.

Sr.No.	Property & Description
1	**constructor** Returns a reference to the array function that created the object.
2	**index** The property represents the zero-based index of the match in the string
3	**input** This property is only present in arrays created by regular expression matches.
4	**length** Reflects the number of elements in an array.
5	**prototype** The prototype property allows you to add properties and methods to an object.

Array Methods

Here is a list of the methods of the Array object along with their description.

Sr.No.	Method & Description
1	**concat()** Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	**every()** Returns true if every element in this array satisfies the provided testing function.
3	**filter()** Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	**forEach()** Calls a function for each element in the array.
5	**indexOf()** Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.

6	**join()** Joins all elements of an array into a string.
7	**lastIndexOf()** Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	**map()** Creates a new array with the results of calling a provided function on every element in this array.
9	**pop()** Removes the last element from an array and returns that element.
10	**push()** Adds one or more elements to the end of an array and returns the new length of the array.
11	**reduce()** Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.

12	**reduceRight()** Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13	**reverse()** Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	**shift()** Removes the first element from an array and returns that element.
15	**slice()** Extracts a section of an array and returns a new array.
16	**some()** Returns true if at least one element in this array satisfies the provided testing function.
17	**toSource()** Represents the source code of an object

18	**sort()** Sorts the elements of an array
19	**splice()** Adds and/or removes elements from an array.
20	**toString()** Returns a string representing the array and its elements.
21	**unshift()** Adds one or more elements to the front of an array and returns the new length of the array.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
var array_name = [item1, item2, ...];
```

Example

```
var cars = ["Saab", "Volvo", "BMW"];
```

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

```
var cars = new Array("Saab", "Volvo", "BMW");
```

Access the Elements of an Array

You access an array element by referring to the **index number**.

This statement accesses the value of the first element in `cars`:

```
var name = cars[0];
```

You access an array element by referring to the **index number**.

This statement accesses the value of the first element in `cars`:

```
var name = cars[0];
```

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

Changing an Array Element

This statement changes the value of the first element in `cars`:

```
cars[0] = "Opel";
```

Example

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p id="demo"></p>

<script>
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>

</body>
</html>
```

Arrays are Objects

Arrays are a special type of objects. The `typeof` operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, `person[0]` returns John:

Array:

```
var person = ["John", "Doe", 46];
```

Object:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

Try it Yourself »

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

The length Property

The `length` property of an array returns the length of an array (the number of array elements).

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;   // the length of fruits is 4
```

Try it Yourself »

Accessing the First Array Element

Example

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
```

Accessing the Last Array Element

Example

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var last = fruits[fruits.length - 1];
```

Looping Array Elements

The safest way to loop through an array, is using a `for` loop:

Example

```
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The best way to loop through an array is using a standard for loop:</p>

<p id="demo"></p>

<script>
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

```
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>Array.forEach() calls a function for each array element.</p>

<p>Array.forEach() is not supported in Internet Explorer 8 or earlier.</p>

<p id="demo"></p>

<script>
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";
document.getElementById("demo").innerHTML = text;

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
</script>

</body>
</html>
```

Adding Array Elements

The easiest way to add a new element to an array is using the `push()` method:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");    // adds a new element (Lemon) to fruits
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The push method appends a new element to an array.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.push("Lemon");
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The length property provides an easy way to append new elements to an array
without using the push() method.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits[fruits.length] = "Lemon";
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```