JavaScript Arrays

‹ Previous Next ›

JavaScript arrays are used to store multiple values in a single variable.

Example

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

Try it Yourself »

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
var car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

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"];
```

Try it Yourself »

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

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

Try it Yourself »

Putting a comma after the last element (like "BMW",) is inconsistent across browsers.

IE 8 and earlier will fail.

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");
```

Try it Yourself »

The two examples above do exactly the same. There is no need to use new Array(). For simplicity, readability and execution speed, use the first one (the array literal method).

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];
```

Example

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

Try it Yourself »

Array indexes start with 0.

[0] is the first element. [1] is the second element.

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];
```

Try it Yourself »

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;
```

Try it Yourself »

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];
```

Try it Yourself »

Objects use **names** to access its "members". In this example, **person.firstName** returns John:

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;
```

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

Examples

```
var x = cars.length;   // The length property returns the number of
elements
var y = cars.sort();   // The sort() method sorts arrays
```

Array methods are covered in the next chapters.

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 »

The length property is always one more than the highest array index.

Accessing the First Array Element

Example

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

Try it Yourself »

Accessing the Last Array Element

Example

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

Try it Yourself »

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>";
```

Try it Yourself »

You can also use the Array.forEach() function:

Example

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

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
```

```
    text += "<li>" + value + "</li>";
}
```

Try it Yourself »

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
```

Try it Yourself »

New element can also be added to an array using the length property:

Example

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

Try it Yourself »

WARNING !

Adding elements with high indexes can create undefined "holes" in an array:

Example

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon";                  // adds a new element (Lemon) to
fruits
```

Try it Yourself »

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"
```

Try it Yourself »

WARNING !!
If you use named indexes, JavaScript will redefine the array to a standard object.
After that, some array methods and properties will produce **incorrect results**.

Example:

```
var person = [];
person["firstName"] = "John";
```

```
person["lastName"] = "Doe";
person["age"] = 46;
var x = person.length;         // person.length will return 0
var y = person[0];             // person[0] will return undefined
```

Try it Yourself »

The Difference Between Arrays and Objects

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

In JavaScript, **objects** use **named indexes**.

> Arrays are a special kind of objects, with numbered indexes.

When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

Avoid new Array()

There is no need to use the JavaScript's built-in array constructor **new** Array().

Use [] instead.

These two different statements both create a new empty array named points:

```
var points = new Array();          // Bad
var points = [];                   // Good
```

These two different statements both create a new array containing 6 numbers:

```
var points = new Array(40, 100, 1, 5, 25, 10); // Bad
var points = [40, 100, 1, 5, 25, 10];          // Good
```

Try it Yourself »

The **new** keyword only complicates the code. It can also produce some unexpected results:

```
var points = new Array(40, 100);  // Creates an array with two elements
(40 and 100)
```

What if I remove one of the elements?

```
var points = new Array(40);       // Creates an array with 40 undefined
elements !!!!!
```

Try it Yourself »

How to Recognize an Array

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator **typeof** returns "object":

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

typeof fruits;             // returns object
```

Try it Yourself »

The typeof operator returns object because a JavaScript array is an object.

Solution 1:

To solve this problem ECMAScript 5 defines a new method **Array.isArray()**:

```
Array.isArray(fruits);     // returns true
```

Try it Yourself »

The problem with this solution is that ECMAScript 5 is **not supported in older browsers**.

Solution 2:

To solve this problem you can create your own isArray() function:

```
function isArray(x) {
    return x.constructor.toString().indexOf("Array") > -1;
}
```

Try it Yourself »

The function above always returns true if the argument is an array.

Or more precisely: it returns true if the object prototype contains the word "Array".

Solution 3:

The **instanceof** operator returns true if an object is created by a given constructor:

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

fruits instanceof Array     // returns true
```

Try it Yourself »

Test Yourself with Exercises!

Exercise 1 » Exercise 2 » Exercise 3 »

‹ Previous Next ›