JavaScript Arrays

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

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

An array is a special variable, which can hold more than one value:

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

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:

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

It is a common practice to declare arrays with the `const` keyword.

Example

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

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

Example

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

You can also create an array, and then provide the elements:

Example

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Using the JavaScript Keyword new

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

The two examples above do exactly the same. There is no need to use `new Array()`.

For simplicity, readability and execution speed, use the array literal method.

Accessing Array Elements

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

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

Note: Array indexes start with 0. [0] is the first element. [1] is the second element, and so on.

Changing an Array Element

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

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

Example

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

Converting an Array to a String

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

Example

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

Result:

Banana,Orange,Apple,Mango

Access the Full Array

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

Example

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

Arrays are Objects

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

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

Array:

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

Object:

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

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

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:

```
cars.length   // Returns the number of elements
cars.sort()   // Sorts the array
```

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

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

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

Accessing the First Array Element

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0];
```

Accessing the Last Array Element

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

Looping Array Elements

1. One way to loop through an array, is using a `for` loop:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

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

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

Example

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

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

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

Adding Array Elements

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

Example

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

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

Example

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[fruits.length] = "Lemon";  // Adds "Lemon" to fruits
```

WARNING !

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

Example

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[6] = "Lemon";  // Creates undefined "holes" in fruits
```

Associative Arrays – Not Supported

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

```
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
person.length;     // Will return 3
person[0];         // Will return "John"
```

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.

- Use **objects** when you want the element names to be **strings (text)**.
- Use **arrays** when you want the element names to be **numbers**.

JavaScript new Array()

JavaScript has a built-in array constructor `new Array()`. But we can safely use `[]` instead. These two different statements below create a new empty array named points:

```
const points = new Array();
const points = [];
```

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

```
const points = new Array(40, 100, 1, 5, 25, 10);
const points = [40, 100, 1, 5, 25, 10];
```

The `new` keyword can produce some unexpected results:

```
// Create an array with three elements:
const points = new Array(40, 100, 1);
```

```
// Create an array with two elements:
const points = new Array(40, 100);
```

```
// Create an array with one element ???
const points = new Array(40);
```

Note

```
// Create an array with one element:
const points = [40];
```

```
// Create an array with 40 undefined elements:
const points = new Array(40);
```

How to Recognize an Array

The problem is that the JavaScript operator `typeof` returns "`object`". The typeof operator returns object because a JavaScript array is an object.

```
const fruits = ["Banana", "Orange", "Apple"];
let type = typeof fruits;
```

Solution 1:

Use `Array.isArray(“<array_name>”)`:

Example:

```
Array.isArray(fruits);
```

Solution 2:

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

```
const fruits = ["Banana", "Orange", "Apple"];

(fruits instanceof Array);
```

Nested Arrays and Objects

Values in objects can be arrays, and values in arrays can be objects:

Example

```
const myObj = {
  name: "John",
  age: 30,
  cars: [
    {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
    {name:"BMW", models:["320", "X3", "X5"]},
    {name:"Fiat", models:["500", "Panda"]}
  ]
}
```

To access arrays inside arrays, use a for-in loop for each array:

Example

```
for (let i in myObj.cars) {
  x += "<h1>" + myObj.cars[i].name + "</h1>";
  for (let j in myObj.cars[i].models) {
    x += myObj.cars[i].models[j];
  }
}
```