JavaScript Arrays

An array is a special variable, which can hold more than one value:
Ex: const cities = ["Mumbai", "Karnataka", "Chennai"];

```
<script>
const cities = ["Mumbai", "Karnataka", "Chennai"];
document.getElementById("demo").innerHTML = cities;
</script>
```

Output: Mumbai, Karnataka, Chennai

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, ...];

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

```
<script>
const cities = [];
cities [0]= "Mumbai";
cities [1]= "Karnataka";
cities [2]= "Chennai";
document.getElementById("demo").innerHTML = cities;
</script>
```

Output: Mumbai, Karnataka, Chennai

Using the JavaScript Keyword new

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

```
<script>
const cities= new Array("Mumbai", "Karnataka", "Chennai");
document.getElementById("demo").innerHTML = cities;
</script>
```

Output: Mumbai, Karnataka, Chennai

Accessing Array Elements

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

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

```
<script>
const cities= new Array("Mumbai", "Karnataka", "Chennai");
document.getElementById("demo").innerHTML = cities[0];
</script>
```

Output: Mumbai

Changing an Array Element

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

```
<script>
const cities= new Array("Mumbai", "Karnataka", "Chennai");
cities[0] = "Hyderabad";
document.getElementById("demo").innerHTML = cities;
</script>
```

Output: Hyderabad, Karnataka, Chennai

Converting an Array to a String

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

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

Output: Banana, Orange, Apple, Mango

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 James:

```
<script>
const person = ["James", "Kint", 32];
document.getElementById("demo").innerHTML = person[0];
</script>
```

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

Output: James

```
<script>
const person = {firstName:"James", lastName:"Kint", age:46};
document.getElementById("demo").innerHTML = person.firstName;
</script>
```

Output: James

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:

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

The length Property

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

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let size = fruits.length;
document.getElementById("demo").innerHTML = size;
</script>
```

Output: 4

Accessing first array element:

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits[0];
</script>
```

Output: Banana

Accessing last array element:

```
<script>
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits[fruits.length-1];
</script>
```

Output : Mango

Looping an array element:

One way to loop through an array, is **using a for loop:**

```
<script>
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>";

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

Output:

Banana
Orange

Apple
Mango

One way to loop through an array, is **using a forEach loop:**

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

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

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

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

Output:
Banana
Orange
Apple
Mango

Adding Array Elements

- The easiest way to add a new element to an array is using the **push() method:**
- The push method appends a new element to an array.

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

<p id="demo"></p>
<script>
const fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = fruits;

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

Output:
Banana,Orange,Apple

Banana,Orange,Apple,Lemon

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

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

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

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

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

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

Output:
Banana,Orange,Apple

Try it

Banana,Orange,Apple,Lemon

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.

```
<script>
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>
```

Output: John 3

WARNING !!

If you use a named index when accessing an array, JavaScript will redefine the array to a standard object, and some array methods and properties will produce undefined or incorrect results.

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

<script>
const person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
document.getElementById("demo").innerHTML =
person[0] + " " + person.length;
</script>
```

Output: undefined

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.

JavaScript new Array()

JavaScript has a built-in array constructor new Array().
But you can safely use [] instead.

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

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

Note: Avoid using new Array(). Use [] instead.

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

<script>
//const points = new Array(40, 100, 1, 5, 25, 10);
const points = [40, 100, 1, 5, 25, 10];
document.getElementById("demo").innerHTML = points[0];
</script>
```

Output: 40

Array with three elements

```
<script>
var points = new Array(40, 100, 1);
document.getElementById("demo").innerHTML = points;
</script>
```

Output: 40, 100, 1

Array with one elements

```
<script>
var points = new Array(40);
document.getElementById("demo").innerHTML = points;
</script>
```

Output: ,,,,,,

Solution: Array with one elements

```
<script>
var points = [40];
```

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

Output: 40

Array with undefined

```
<script>
var points = new Array(40);
document.getElementById("demo").innerHTML = points[0];
</script>
```

Output: undefined

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":

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

Output: object

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

Solution 1:

To solve this problem ECMAScript 5 (JavaScript 2009) defined a new method **Array.isArray():**

```
<script>
const fruits = ["Banana", "Orange", "Apple"];
document.getElementById("demo").innerHTML = Array.isArray(fruits);
</script>
```

Output: true

Solution 2:

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

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

Output: true

Exercise 1:

Get the value "Volvo" from the cars array.
const cars = ["Saab", "Volvo", "BMW"];
let x=**cars[1];**

Exercise 2:

Change the first item of cars to "Ford".
const cars = ["Volvo", "Jeep", "Mercedes"];
cars[0] =”Ford”;

Exercise 3:

Alert the number of items in an array, using the correct Array property.
const cars = ["Volvo", "Jeep", "Mercedes"];
alert(**cars.length**);