Javascript Arrays

Arrays

- Arrays are lists of elements indexed by a numerical value
- Array indexes in JavaScript begin at 0 Madhu Bhan
- Arrays can be modified in size even after they have been created

- new Array with one parameter creates an empty array of the specified number of elements
  new Array(10)
- new Array with two or more parameters creates an array with the specified parameters as elements
  new Array(10, 20)
- Literal arrays can be specified using square brackets to include a list of elements
  var alist = [1, “ii”, “gamma”, “4”];
- Elements of an array do not have to be of the same type

Characteristics of `Array` Objects

- The length of an array is one more than the highest index to which a value has been assigned or the initial size (using Array with one argument), whichever is larger
- Assignment to an index greater than or equal to the current length simply increases the length of the array
- Only assigned elements of an array occupy space
- Suppose an array were created using new Array(200)
- Suppose only elements 150 through 174 were assigned values
- Only the 25 assigned elements would be allocated storage, the other 175 would not be allocated storage

Arrays

JavaScript array is an object that represents a collection of similar type of elements. An array can hold many values under a single name, and you can access the values by referring to an index number.

There are 3 ways to construct array in JavaScript

- By array literal
- By creating instance of Array directly (using new keyword)
- By using an Array constructor (using new keyword)

1) JavaScript array literal

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

Syntax:

var arrayname=[value1,value2.....valueN];

Values are contained inside [ ] and separated by , (comma).

Example code:

```
<script>
var emp=["Sonoo","Vimal","Ratan"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}  </script>
```

JavaScript Array Const

It has become a common practice to declare arrays using const. An array declared with const must be initialized when it is declared

Example:

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

An array declared with const cannot be reassigned

Example

```
const cars = ["Saab", "Volvo", "BMW"];
cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

The constant defines a constant reference to an array.

Because of this, we can still change the elements of a constant array.

Example

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Toyota"; // You can change an element:
cars.push("Audi"); // You can add an element:
```

Exercise 1

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript const</h2>
<p>You can NOT reassign a constant array:</p>
<p id="demo"></p>
<script>
  const cars = ["Saab", "Volvo", "BMW"];
  cars = ["Toyota", "Volvo", "Audi"];
  for (i=0;i<cars.length;i++){
document.write(cars[i] + "<br/>");
}
</script>
</body>
</hml>
```

Madhu Bhan

2) Using the JavaScript Keyword new

Here, **new keyword** is used to create instance of array :

Syntax:

var arrayname=new Array();

Example Code

```
<script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";
  for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");  }  </script>
```

3)JavaScript array constructor (new keyword)

Create instance of array by passing arguments in constructor so that there is no need to provide value explicitly.


```
<script>

var emp=new Array("Jai","Vijay","Smith");

for (i=0;i<emp.length;i++){

document.write(emp[i] + "<br>");

}

</script>
```

forEach():

The **forEach()** is a built-in **JavaScript** method of **arrays** that allows you to loop over the elements of an array and execute a callback function for each element.

Syntax: *array.forEach(function(currentValue, index, arr), thisValue*)

The **currentValue** is the current item in an array that is iterating.
The **index** is an optional parameter, which is the current element index of an array.
The **arr** is an optional parameter, an array object to which the current element belongs.
The **thisValue** is an optional parameter that is the value to be passed to the function to be used as its "this" value.

```
var arr = [1, 2, 3, 4, 5];
var newArray = [];
arr.forEach(function(item) { newArray.push(item + 21); });
 document.write(newArray);
```

Output: [22,23,24,25,26]

JavaScript Array Iteration

JavaScript Array forEach():

The forEach() method calls a function once for each array element. It invokes the provided function once for each element of an array.

Example Code:

```
<script>
var sum = 0;
var arr = [10,18,12,20];

arr.forEach(function myFunction(element) {
    sum= sum+element;
  document.writeln(sum);
});
</script>
```

Output: 10 28 40 60

Exercise 2-Display sum

```
<script>
let sum = 0;
const numbers = [65, 44, 12, 4];
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = sum;
function myFunction(item) {
  sum += item;}
</script>
```

`Array` Methods

- join
- reverse
- sort
- concat
- slice

Madhu Bhan

join

The `join()` method returns an array as a string.
The `join()` method does not change the original array.
Any separator can be specified. The default is comma (,).

Syntax

array.join(*separator*)

Exercise 1

```
<script>
var arr=["AngularJs","Node.js","JQuery"]
var result=arr.join('-')
document.write(result);
</script>
```

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

reverse()

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

The `reverse()` method overwrites the original array.

Syntax: *array*.reverse()

Exercise 2

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

```
<body> <script type = "text/javascript">
var arr = [0, 1, 2, 3].reverse();
 document.write("Reversed array is : " + arr );
</script>
```

sort()

- The `sort()` sorts the elements of an array.
- The `sort()` overwrites the original array.
- The `sort()` sorts the elements as strings in alphabetical and ascending order.

Syntax:*array*.sort(*compareFunction*)

Exercise 3

```
<script>
let numbers = [0, 1 , 2, 3, 10, 20, 30 ];
numbers.sort();
document.write(numbers);
</script>
```

```
<script>
let animals = [ 'cat', 'dog', 'elephant', 'bee', 'ant' ];
animals.sort();
document.write(animals);
</script>
```

Sort Compare Function

- Sorting alphabetically works well for strings ("Apple" comes before "Banana").
- But, sorting numbers can produce incorrect results.
- [0, 1 , 2, 3, 10, 20, 30];
- 0, 1, 10, 2, 20, 3, 30
- You can fix this by providing a "compare function"
  - ***array*.sort(*compareFunction*)**

Optional.
A function that defines a sort order. The 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.
If the result is negative, `a` is sorted before `b`.
If the result is positive, `b` is sorted before `a`.
If the result is 0, no changes are done with the sort order of the two values.

Sort Compare Function

Sort the array in ascending order:sort()

Madhu Bhan

40,100,1,5,25,10

1,10,100,25,40,5

Sort the array in ascending order: sort(function(a, b){return a - b})

40,100,1,5,25,10

1,5,10,25,40,100

ort the array in descending order: sort(function(a, b){return b - a})

40,100,1,5,25,10

100,40,25,10,5,1

concat()

The `concat()` method concatenates (joins) two or more arrays.
The `concat()` method returns a new array, containing the joined arrays.
The `concat()` method does not change the existing arrays.

Syntax:*array1*.concat(*array2*, *array3*, ..., *arrayX*)

Exercise 4

```
<script>
const arr1 = ["Cecilie", "Lone"];
const arr2 = [1, 2, 3];
const arr3 = arr1.concat(arr2);
document.getElementById("demo").innerHTML = arr3;
</script>
```

```
<script>
var arr1=["C","C++","Python"];
var arr2=["Java","JavaScript","Android"];
var arr3=["Ruby","Kotlin"];
var result=arr1.concat(arr2,arr3);
document.writeln(result);
</script>
```

slice()

The `slice()` method returns selected elements in an array, as a new array.
The `slice()` method selects from a given *start*, up to a (not inclusive) given *end*.
The `slice()` method does not change the original array. Returns a new array.

Syntax:*array*.slice(*start*, *end*)

Start: Start position. Default is 0.Negative numbers select from the end of the array.
End: End position. Default is last element.Negative numbers select from the end of the array.

Exercise 6

```
<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1, 3);
document.getElementById("demo").innerHTML = citrus;
</script>
---------------------------------------
<script>
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const myBest = fruits.slice(-4, -2);
document.getElementById("demo").innerHTML = myBest;
</script>
```