Arrays

Quite often we find that we need an *ordered collection*, where we have a 1st, a 2nd, a 3rd element and so on to store data. There exists a special data structure named Array, to store ordered collections. JavaScript arrays are used to store multiple values in a single variable. An array can hold many values under a single name, and you can access the values by referring to an index number.

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

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

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

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

```
var fruits = ( "apple", "orange", "banana" );
```

Arrays use **numbers** to access its "elements". You access an array element by referring to the **index number**.

Built in methods to use with arrays :

** concat()

Returns a new array comprised of this array joined with two or more arrays. This method returns an array object representing the resultant array, a concatenation of the current and given array(s).

array.concat(value1, value2, ..., valueN);

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. The concat() method can also take strings as arguments.

toString()

converts an array to a string of (comma separated) array values.

array.toString();

The **join()** method also joins all array elements into a string. It behaves just like toString() but in addition you can specify the separator.

Javascript array **reverse()** method reverses the element of an array. The first array element becomes the last and the last becomes the first.

array.reverse();

Javascript array **sort()** method sorts the elements of an array.

array.sort(compareFunction);

compareFunction − Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically. Returns a sorted array.

Sorting of numeric values may cause problem in this case

Javascript array **slice()** method extracts a section of an array and returns a new array.

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

array.slice(begin [,end]);

- **begin** − Zero-based index at which to begin extraction. As a negative index, start indicates an offset from the end of the sequence.
- **end** − Zero-based index at which to end extraction.

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

Elements can be deleted by using the JavaScript operator **delete()**

Eg .

```
var arr1 = ["a", "b"];
var arr2 = ["c", "d", "e"];
var arr3 = ["f", "g"];
var arr4 = arr1.concat(arr2, arr3);   // Concatenates arr1 with arr2 and
arr3
```

Eg.

```
var arr1 = ["a", "b", "c"];
var arr2 = arr1.concat("newval");
```

Eg.

```
<html>
   <head>
      <title>JavaScript Array concat Method</title>
   </head>

   <body>
      <script type = "text/javascript">

 var arr1 = ["a", "b", "c"];
var arr2 = [1, 2, 3];

//CONCAT

         var newarr = arr1.concat(arr2);
         document.write("new array is : " + newarr );

//toString

         var str = arr1.toString();
         document.write("Returned string is : " + str );

//join

       var arr=["Abha","Alka","Seema"]
       var result=arr.join('-')
       document.write(result);

//reverse

         var arr3 = [0, 1, 2, 3].reverse();
```

```
         document.write("Reversed array is : " + arr3 );

//sort
         var arr4 = new Array("puja", "smita", "diya", "arti");
         var sorted = arr4.sort();
         document.write("Returned string is : " + sorted );

// slice
       var result = arr4.slice(1);
        document.write("resulting array is : " + result );
   document.write("<br />arr4.slice( 1, 3) : " + arr4.slice( 1, 3)
);

  delete arr4[0];
  // Changes the first element in arr4 to undefined
      </script>
   </body>
</html>
```

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.

compare function:

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

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.

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.
Example:
The compare function compares all the values in the array, two values at a time (a, b).
When comparing 40 and 100, the sort() method calls the compare function(40, 100).
The function calculates 40 - 100 (a - b), and since the result is negative (-60), the sort function will sort 40 as a value lower than 100.

The **isArray()** method is used to test whether the value passed is an array. If it finds the passed value is an array, it returns True. Otherwise, it returns False.

EXAMPLE:

```
<html>
<head> <h5> JavaScript Array Methods </h5> </head>
<body>
<script>

document.write(Array.isArray(1,2,3,4)); //Testing the passed values.

var arr=new Array(1,2,34,4,5);
document.write(Array.isArray(arr)); //It will return true.


document.write(Array.isArray(null)); //It will return false.

</script>
</body>
</html>
```

The JavaScript array **pop()** method removes the last element from the given array and return that element. This method changes the length of the original array.

array.pop()

The JavaScript array **push()** method adds one or more elements to the end of the given array. This method changes the length of the original array.

array.push(element1,element2....elementn)

EXAMPLE

```
<html>  <body>
<script>
var arr=["Abha","alka","Seema"];
document.writeln("Orginal array: "+arr+"<br>");
document.writeln("Extracted element: "+arr.pop()+"<br>");
document.writeln("Remaining elements: "+ arr);
document.writeln("<br>");

//to pop all elements out of array

var arr=["Abha","Alka","Seema"];
var len=arr.length;
  for(var x=1;x<=len;x++)
  {
document.writeln("Extracted element: "+arr.pop()+"<br>");
  }

//to push an element

var arr=["Abha","alka"];
arr.push("Pooja");
document.writeln(arr);
document.writeln("<br>");

//length after push

var arr=["Abha","alka"];
document.writeln("Length before invoking push(): "+arr.length+"<br>");
arr.push("Pooja","Seema");
document.writeln("Length after invoking push(): "+arr.length+"<br>");
document.writeln("Update array: "+arr);
document.writeln("<br>");
</script>
```

</body>
</html>