**String Methods :**

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

The length property returns the length of a string:

var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
var sln = txt.length;

The indexOf() method returns the index of (the position of) the first occurrence of a specified text in a string:

var str = "Please locate where 'locate' occurs!";  
var pos = str.indexOf("locate");

The lastIndexOf() method returns the index of the **last** occurrence of a specified text in a string:

var str = "Please locate where 'locate' occurs!";  
var pos = str.lastIndexOf("locate");

Both indexOf(), and lastIndexOf() return -1 if the text is not found.Try it

var str = "Please locate where 'locate' occurs!";  
var pos = str.lastIndexOf("John");

Both methods accept a second parameter as the starting position for the search:

var str = "Please locate where 'locate' occurs!";  
var pos = str.indexOf("locate", 15);

The lastIndexOf() methods searches backwards (from the end to the beginning), meaning: if the second parameter is 15, the search starts at position 15, and searches to the beginning of the string.

var str = "Please locate where 'locate' occurs!";  
var pos = str.lastIndexOf("locate", 15);

The search() method searches a string for a specified value and returns the position of the match:

var str = "Please locate where 'locate' occurs!";  
var pos = str.search("locate");

The two methods, indexOf() and search(), are **equal?**

They accept the same arguments (parameters), and return the same value?

The two methods are **NOT** equal. These are the differences:

* The search() method cannot take a second start position argument.
* The indexOf() method cannot take powerful search values (regular expressions).

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: the start position, and the end position (end not included). This example slices out a portion of a string from position 7 to position 12 (13-1):

var str = "Apple, Banana, Kiwi";  
var res = str.slice(7, 13);

substring() is similar to slice().

The difference is that substring() cannot accept negative indexes.

substr() is similar to slice().

The difference is that the second parameter specifies the **length** of the extracted part.

The replace() method replaces a specified value with another value in a string:

By default, the replace() method replaces **only the first** match:

A string is converted to upper case with toUpperCase():

A string is converted to lower case with toLowerCase():

concat() joins two or more strings:

The trim() method removes whitespace from both sides of a string

**Array Methods :**

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

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.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:

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

The pop() method removes the last element from an array:

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.pop();              // Removes the last element ("Mango") from fruits

The push() method adds a new element to an array (at the end):

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Kiwi");       //  Adds a new element ("Kiwi") to fruits

Shifting is equivalent to popping, working on the first element instead of the last.

The shift() method removes the first array element and "shifts" all other elements to a lower index.

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift();            // Removes the first element "Banana" from fruits

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements:

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

The length property provides an easy way to append a new element to an array:

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[fruits.length] = "Kiwi";          // Appends "Kiwi" to fruits

Since JavaScript arrays are objects, elements can be deleted by using the JavaScript operator delete:

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
delete fruits[0];           // Changes the first element in fruits to **undefined**

The splice() method can be used to add new items to an array:

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2, 0, "Lemon", "Kiwi");

With clever parameter setting, you can use splice() to remove elements without leaving "holes" in the array:

The concat() method creates a new array by merging (concatenating) existing arrays:

The slice() method slices out a piece of an array into a new array.

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

Try it Yourself »Try it Yourself »