JavaScript Strings

❮ Previous Next ❯

JavaScript strings are used for storing and manipulating text.

JavaScript Strings

A JavaScript string is zero or more characters written inside quotes.

Example

```
var x = "John Doe";
```

Try it Yourself »

You can use single or double quotes:

Example

```
var carname = "Volvo XC60";  // Double quotes
var carname = 'Volvo XC60';  // Single quotes
```

Try it Yourself »

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
var answer = "It's alright";
var answer = "He is called 'Johnny'";
var answer = 'He is called "Johnny"';
```

Try it Yourself »

String Length

The length of a string is found in the built in property **length**:

Example

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

Try it Yourself »

Special Characters

Because strings must be written within quotes, JavaScript will misunderstand this string:

```
var x = "We are the so-called "Vikings" from the north.";
```

The string will be chopped to "We are the so-called ".

The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote

\\	\	Backslash

The sequence **\"** inserts a double quote in a string:

Example

```
var x = "We are the so-called \"Vikings\" from the north.";
```

Try it Yourself »

The sequence **\'** inserts a single quote in a string:

Example

```
var x = 'It\'s alright.';
```

Try it Yourself »

The sequence \\ inserts a backslash in a string:

Example

```
var x = "The character \\ is called backslash.";
```

Try it Yourself »

Six other escape sequences are valid in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return

\t	Horizontal Tabulator
\v	Vertical Tabulator

The 6 escape characters above were originally designed to control typewriters, teletypes, and fax machines. They do not make any sense in HTML.

Breaking Long Code Lines

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

Try it Yourself »

You can also break up a code line **within a text string** with a single backslash:

Example

```
document.getElementById("demo").innerHTML = "Hello \
Dolly!";
```

Try it Yourself »

The \ method is not the preferred method. It might not have universal support. Some browsers do not allow spaces behind the \ character.

A safer way to break up a string, is to use string addition:

Example

```
document.getElementById("demo").innerHTML = "Hello " +
"Dolly!";
```

Try it Yourself »

You cannot break up a code line with a backslash:

Example

```
document.getElementById("demo").innerHTML = \
"Hello Dolly!";
```

Try it Yourself »

Strings Can be Objects

Normally, JavaScript strings are primitive values, created from literals:

var firstName = "John";

But strings can also be defined as objects with the keyword new:

var firstName = new String("John");

Example

```
var x = "John";
var y = new String("John");

// typeof x will return string
// typeof y will return object
```

Try it Yourself »

Don't create strings as objects. It slows down execution speed.
The **new** keyword complicates the code. This can produce some unexpected results:

When using the == operator, equal strings are equal:

Example

```
var x = "John";
var y = new String("John");

// (x == y) is true because x and y have equal values
```

Try it Yourself »

When using the === operator, equal strings are not equal, because the === operator expects equality in both type and value.

Example

```
var x = "John";
var y = new String("John");

// (x === y) is false because x and y have different types (string and
object)
```

Try it Yourself »

Or even worse. Objects cannot be compared:

Example

```
var x = new String("John");
var y = new String("John");

// (x == y) is false because x and y are different objects
```

Try it Yourself »

Example

```
var x = new String("John");
var y = new String("John");

// (x === y) is false because x and y are different objects
```

Try it Yourself »

Note the difference between (x==y) and (x===y).
Comparing two JavaScript objects will **always** return false.

Test Yourself with Exercises!

Exercise 1 » Exercise 2 » Exercise 3 »

❮ Previous Next ❯