JavaScript Tutorial

<http://www.w3schools.com/js/default.asp>

# Table des matières

[Table des matières 1](#_Toc433221868)

# JS Tutorial

## JS HOME

JavaScript is the programming language of HTML and the Web.

Programming makes computers do what you want them to do.

JavaScript is easy to learn.

This tutorial will teach you JavaScript from basic to advanced.

### Examples in Each Chapter

With our "Try it Yourself" editor, you can change all examples and view the results.

#### Example

### My First JavaScript

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_myfirst)

|  |  |
| --- | --- |
| **Note** | We recommend reading this tutorial, in the sequence listed in the left menu. |

### Learn by Examples

Examples are better than 1000 words. Examples are often easier to understand than text explanations.

This tutorial supplements all explanations with clarifying "Try it Yourself" examples.

|  |  |
| --- | --- |
| **Note** | If you try all the examples, you will learn a lot about JavaScript, in a very short time! |

### Why Study JavaScript?

JavaScript is one of the **3 languages** all web developers **must** learn:

   1. **HTML** to define the content of web pages

   2. **CSS** to specify the layout of web pages

   3. **JavaScript** to program the behavior of web pages

This tutorial is about JavaScript, and how JavaScript works with HTML and CSS.

### Learning Speed

In this tutorial, the learning speed is your choice.

Everything is up to you.

If you are struggling, take a break, or reread the material.

**Always** make sure you understand the "Try-it-Yourself" examples and exercises.

### JavaScript References

W3Schools maintains a complete JavaScript reference, including all HTML DOM, and browser objects.

The reference contains examples for all objects, properties, and methods, and is continuously updated according to the latest web standards.

[Complete JavaScript Reference](http://www.w3schools.com/jsref/default.asp)

### JavaScript Quiz Test

Test your JavaScript skills at W3Schools!

[Start JavaScript Quiz!](http://www.w3schools.com/js/js_quiz.asp)

### JavaScript Exam - Get Your Diploma!

[](http://www.w3schools.com/cert/default.asp)

### W3Schools' Online Certification

The perfect solution for professionals who need to balance work, family, and career building.

More than 10 000 certificates already issued!

[Get Your Certificate »](http://www.w3schools.com/cert/default.asp)

The [HTML Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of HTML.

The [HTML5 Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of advanced HTML5.

The [CSS Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of advanced CSS.

The [JavaScript Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of JavaScript and HTML DOM.

The [jQuery Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of jQuery.

The [PHP Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of PHP and SQL (MySQL).

The [XML Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of XML, XML DOM and XSLT.

## JavaScript Introduction

JavaScript is the most popular programming language in the world.

This page contains some examples of what JavaScript can do.

### JavaScript Can Change HTML Content

One of many HTML methods is **getElementById()**.

This example uses the method to "find" an HTML element (with id="demo"**)**, and changes the element content (**innerHTML**) to "Hello JavaScript":

#### Example

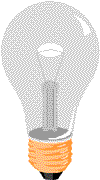
document.getElementById("demo").innerHTML = "Hello JavaScript";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_inner_html)

### JavaScript Can Change HTML Attributes

This example changes an HTML image, by changing the src attribute of an <img> tag:

#### The Light bulb



Click the light bulb to turn on/off the light

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_lightbulb)

### JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

#### Example

document.getElementById("demo").style.fontSize = "25px";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_style)

### JavaScript Can Validate Data

JavaScript is often used to validate input:

Please input a number between 1 and 10



[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_validate)

### Did You Know?

|  |  |
| --- | --- |
| **Note** | JavaScript and Java are completely different languages, both in concept and design. JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997. ECMA-262 is the official name. ECMAScript 6 (released in June 2015) is the latest official version of JavaScript. |

## JavaScript Where To

JavaScript can be placed in the <body> and the <head> sections of an HTML page.

### The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

#### Example

<script>  
document.getElementById("demo").innerHTML = "My First JavaScript";  
</script>

|  |  |
| --- | --- |
| **Note** | Older examples may use a type attribute: <script type="text/javascript">. The type attribute is not required. JavaScript is the default scripting language in HTML. |

### JavaScript Functions and Events

A JavaScript **function** is a block of JavaScript code, that can be executed when "asked" for.

For example, a function can be executed when an **event** occurs, like when the user clicks a button.

You will learn much more about functions and events in later chapters.

### JavaScript in <head> or <body>

You can place any number of scripts in an HTML document.

Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

|  |  |
| --- | --- |
| **Note** | Keeping all code in one place, is always a good habit. |

### JavaScript in <head>

In this example, a JavaScript function is placed in the <head> section of an HTML page.

The function is invoked (called) when a button is clicked:

#### Example

<!DOCTYPE html>  
<html>

<head>  
<script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = "Paragraph changed.";  
}  
</script>  
</head>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph</p>

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

</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_head)

### JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

#### Example

<!DOCTYPE html>  
<html>  
<body>   
  
<h1>My Web Page</h1>  
  
<p id="demo">A Paragraph</p>  
  
<button type="button" onclick="myFunction()">Try it</button>  
  
<script>  
function myFunction() {  
   document.getElementById("demo").innerHTML = "Paragraph changed.";  
}  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_body)

|  |  |
| --- | --- |
| **Note** | It is a good idea to place scripts at the bottom of the <body> element. This can improve page load, because HTML display is not blocked by scripts loading. |

### External JavaScript

Scripts can also be placed in external files.

External scripts are practical when the same code is used in many different web pages.

JavaScript files have the **file extension .js**.

To use an external script, put the name of the script file in the src (source) attribute of the <script> tag:

#### Example

<!DOCTYPE html>  
<html>  
<body>  
<script src="myScript.js"></script>  
</body>  
</html>

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_external)

You can place an external script reference in <head> or <body> as you like.

The script will behave as if it was located exactly where the <script> tag is located.

|  |  |
| --- | --- |
| **Note** | External scripts cannot contain <script> tags. |

### External JavaScript Advantages

Placing JavaScripts in external files has some advantages:

* It separates HTML and code
* It makes HTML and JavaScript easier to read and maintain
* Cached JavaScript files can speed up page loads

## JavaScript Output

JavaScript does NOT have any built-in print or display functions.

### JavaScript Display Possibilities

JavaScript can "display" data in different ways:

* Writing into an alert box, using **window.alert()**.
* Writing into the HTML output using **document.write()**.
* Writing into an HTML element, using **innerHTML**.
* Writing into the browser console, using **console.log()**.

### Using window.alert()

You can use an alert box to display data:

#### Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<script>  
window.alert(5 + 6);  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_alert)

### Using document.write()

For testing purposes, it is convenient to use **document.write()**:

#### Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<script>  
document.write(5 + 6);  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_write)

Using document.write() after an HTML document is fully loaded, will **delete all existing HTML**:

#### Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<button onclick="document.write(5 + 6)">Try it</button>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_write_over)

|  |  |
| --- | --- |
| **Note** | The document.write() method should be used only for testing. |

### Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

#### Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My First Paragraph</p>  
  
<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = 5 + 6;  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_dom)

|  |  |
| --- | --- |
| **Note** | To "display data" in HTML, (in most cases) you will set the value of an innerHTML property. |

### Using console.log()

In your browser, you can use the **console.log()** method to display data.

Activate the browser console with F12, and select "Console" in the menu.

#### Example

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
<p>My first paragraph.</p>  
  
<script>  
console.log(5 + 6);  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_console)

## JavaScript Syntax

JavaScript **syntax** is the set of rules, how JavaScript programs are constructed.

### JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by the computer.

In a programming language, these program instructions are called **statements**.

JavaScript is a **programming language**.

JavaScript statements are separated by **semicolons**.

#### Example

var x = 5;  
var y = 6;  
var z = x + y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_statements)

|  |  |
| --- | --- |
| **Note** | In HTML, JavaScript programs can be executed by the web browser. |

### JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

### JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values.

Fixed values are called **literals**. Variable values are called **variables**.

### JavaScript Literals

The most important rules for writing fixed values are:

**Numbers** are written with or without decimals:

10.50  
  
1001

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_numbers)

**Strings** are text, written within double or single quotes:

"John Doe"  
  
'John Doe'

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_strings)

### JavaScript Variables

In a programming language, **variables** are used to **store** data values.

JavaScript uses the **var** keyword to **define** variables.

An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

var x;  
  
x = 6;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_variables)

### JavaScript Operators

JavaScript uses an **assignment operator** ( = ) to **assign** values to variables:

var x = 5;  
var y = 6;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_assign)

JavaScript uses **arithmetic operators** ( + - \*  / ) to **compute** values:

(5 + 6) \* 10

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_operators)

### JavaScript Expressions

An expression is a combination of values, variables, and operators, which computes to a value.

The computation is called an evaluation.

For example, 5 \* 10 evaluates to 50:

5 \* 10

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_expressions)

Expressions can also contain variable values:

x \* 10

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_expressions_variables)

The values can be of various types, such as numbers and strings.

For example, "John" + " " + "Doe", evaluates to "John Doe":

"John" + " " + "Doe"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_expressions_strings)

### JavaScript Keywords

JavaScript **keywords** are used to identify actions to be performed.

The **var** keyword tells the browser to create a new variable:

var x = 5 + 6;  
var y = x \* 10;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_keywords)

### JavaScript Comments

Not all JavaScript statements are "executed".

Code after double slashes **//** or between **/\*** and **\*/** is treated as a **comment**.

Comments are ignored, and will not be executed:

var x = 5;   // I will be executed  
  
// var x = 6;   I will NOT be executed

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_comments)

### JavaScript Identifiers

Identifiers are names.

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels).

The rules for legal names are much the same in most programming languages.

In JavaScript, the first character must be a letter, an underscore (\_), or a dollar sign ($).

Subsequent characters may be letters, digits, underscores, or dollar signs.

|  |  |
| --- | --- |
| **Note** | Numbers are not allowed as the first character. This way JavaScript can easily distinguish identifiers from numbers. |

### JavaScript is Case Sensitive

All JavaScript identifiers are **case sensitive**.

The variables **lastName** and **lastname**, are two different variables.

lastName = "Doe";  
lastname = "Peterson";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_case)

JavaScript does not interpret **VAR** or **Var** as the keyword **var**.

### JavaScript and Camel Case

Historically, programmers have used three ways of joining multiple words into one variable name:

**Hyphens:**

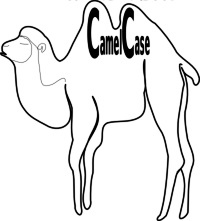
first-name, last-name, master-card, inter-city.

**Underscore:**

first\_name, last\_name, master\_card, inter\_city.

**Camel Case:**

FirstName, LastName, MasterCard, InterCity.



In programming languages, especially in JavaScript, camel case often starts with a lowercase letter:

firstName, lastName, masterCard, interCity.

|  |  |
| --- | --- |
| **Note** | Hyphens are not allowed in JavaScript. It is reserved for subtractions. |

### JavaScript Character Set

JavaScript uses the **Unicode** character set.

Unicode covers (almost) all the characters, punctuations, and symbols in the world.

For a closer look, please study our [Complete Unicode Reference](http://www.w3schools.com/charsets/ref_html_utf8.asp).

## JavaScript Statements

In HTML, JavaScript statements are "instructions" to be "executed" by the web browser.

### JavaScript Statements

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statement)

### JavaScript Programs

Most JavaScript programs contain many JavaScript statements.

The statements are executed, one by one, in the same order as they are written.

In this example, x, y, and z is given values, and finally z is displayed:

#### Example

var x = 5;  
var y = 6;  
var z = x + y;  
document.getElementById("demo").innerHTML = z;

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements)

|  |  |
| --- | --- |
| **Note** | JavaScript programs (and JavaScript statements) are often called JavaScript code. |

### Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

a = 5;  
b = 6;  
c = a + b;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_semicolon1)

When separated by semicolons, multiple statements on one line are allowed:

a = 5; b = 6; c = a + b;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_semicolon2)

|  |  |
| --- | --- |
| **Note** | On the web, you might see examples without semicolons.  Ending statements with semicolon is not required, but highly recommended. |

### JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

var person = "Hege";  
var person="Hege";

A good practice is to put spaces around operators ( = + - \* / ):

var x = y + z;

### JavaScript Line Length and Line Breaks

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 »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_linebreak)

### JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, are in JavaScript functions:

#### Example

function myFunction() {  
    document.getElementById("demo").innerHTML = "Hello Dolly.";  
    document.getElementById("myDIV").innerHTML = "How are you?";  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_blocks)

|  |  |
| --- | --- |
| **Note** | In this tutorial we use 4 spaces of indentation for code blocks. You will learn more about functions later in this tutorial. |

### JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| break | Terminates a switch or a loop |
| continue | Jumps out of a loop and starts at the top |
| debugger | Stops the execution of JavaScript, and calls (if available) the debugging function |
| do ... while | Executes a block of statements, and repeats the block, while a condition is true |
| for | Marks a block of statements to be executed, as long as a condition is true |
| function | Declares a function |
| if ... else | Marks a block of statements to be executed, depending on a condition |
| return | Exits a function |
| switch | Marks a block of statements to be executed, depending on different cases |
| try ... catch | Implements error handling to a block of statements |
| var | Declares a variable |

|  |  |
| --- | --- |
| **Note** | JavaScript keywords are reserved words. Reserved words cannot be used as names for variables. |

### JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

### Single Line Comments

Single line comments start with //.

Any text between // and the end of the line, will be ignored by JavaScript (will not be executed).

This example uses a single line comment before each line, to explain the code:

#### Example

// Change heading:  
document.getElementById("myH").innerHTML = "My First Page";  
// Change paragraph:  
document.getElementById("myP").innerHTML = "My first paragraph.";

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments1)

This example uses a single line comment at the end of each line, to explain the code:

#### Example

var x = 5;      // Declare x, give it the value of 5  
var y = x + 2;  // Declare y, give it the value of x + 2

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments5)

### Multi-line Comments

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

#### Example

/\*  
The code below will change  
the heading with id = "myH"  
and the paragraph with id = "myP"  
in my web page:  
\*/  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments2)

|  |  |
| --- | --- |
| **Note** | It is most common to use single line comments. Block comments are often used for formal documentation. |

### Using Comments to Prevent Execution

Using comments to prevent execution of code, is suitable for code testing.

Adding // in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

#### Example

//document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments3)

This example uses a comment block to prevent execution of multiple lines:

#### Example

/\*  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first paragraph.";  
\*/

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments4)

## JavaScript Variables

### JavaScript Variables

JavaScript variables are containers for storing data values.

In this example, x, y, and z, are variables:

#### Example

var x = 5;  
var y = 6;  
var z = x + y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables)

From the example above, you can expect:

* x stores the value 5
* y stores the value 6
* z stores the value 11

### Much Like Algebra

In this example, price1, price2, and total, are variables:

#### Example

var price1 = 5;  
var price2 = 6;  
var total = price1 + price2;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_total)

In programming, just like in algebra, we use variables (like price1) to hold values.

In programming, just like in algebra, we use variables in expressions (total = price1 + price2).

From the example above, you can calculate the total to be 11.

|  |  |
| --- | --- |
| **Note** | JavaScript variables are containers for storing data values. |

### JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y), or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

* Names can contain letters, digits, underscores, and dollar signs.
* Names must begin with a letter
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case sensitive (y and Y are different variables)
* Reserved words (like JavaScript keywords) cannot be used as names

|  |  |
| --- | --- |
| **Note** | JavaScript identifiers are case-sensitive. |

### The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

x = x + 5

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

|  |  |
| --- | --- |
| **Note** | The "equal to" operator is written like == in JavaScript. |

### JavaScript Data Types

JavaScript variables can hold numbers like 100, and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put quotes around a number, it will be treated as a text string.

#### Example

var pi = 3.14;  
var person = "John Doe";  
var answer = 'Yes I am!';

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_types)

### Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the **var** keyword:

var carName;

After the declaration, the variable has no value. (Technically it has the value of **undefined**)

To **assign** a value to the variable, use the equal sign:

carName = "Volvo";

You can also assign a value to the variable when you declare it:

var carName = "Volvo";

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

#### Example

<p id="demo"></p>  
  
<script>  
var carName = "Volvo";  
document.getElementById("demo").innerHTML = carName;   
</script>

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_create)

|  |  |
| --- | --- |
| **Note** | It's a good programming practice to declare all variables at the beginning of a script. |

### One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with **var** and separate the variables by **comma**:

var person = "John Doe", carName = "Volvo", price = 200;

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multi)

A declaration can span multiple lines:

var person = "John Doe",  
carName = "Volvo",  
price = 200;

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multiline)

### Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value **undefined**.

The variable carName will have the value undefined after the execution of this statement:

#### Example

var carName;

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_undefined)

### Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable, it will not lose its value.

The variable carName will still have the value "Volvo" after the execution of these statements:

#### Example

var carName = "Volvo";  
var carName;

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_redefine)

### JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

#### Example

var x = 5 + 2 + 3;

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_numbers)

You can also add strings, but strings will be concatenated (added end-to-end):

#### Example

var x = "John" + " " + "Doe";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_strings)

Also try this:

#### Example

var x = "5" + 2 + 3;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_string_number)

|  |  |
| --- | --- |
| **Note** | If you add a number to a string, the number will be treated as string, and concatenated. |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_variables1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_variables2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_variables3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_variables4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_variables5)  [Exercise 6 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_variables6)

## JavaScript Operators

#### Example

Assign values to variables and add them together:

var x = 5;         // assign the value 5 to x  
var y = 2;         // assign the value 2 to y  
var z = x + y;     // assign the value 7 to z (x + y)

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper)

### JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers (literals or variables).

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| -- | Decrement |

The **addition** operator (+) adds numbers:

#### Adding

var x = 5;  
var y = 2;  
var z = x + y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_add)

The **multiplication** operator (\*) multiplies numbers.

#### Multiplying

var x = 5;  
var y = 2;  
var z = x \* y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mult)

|  |  |
| --- | --- |
| **Note** | You will learn more about JavaScript operators in the next chapters. |

### JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

The **assignment** operator (=) assigns a value to a variable.

#### Assignment

var x = 10;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_equal)

The **addition assignment** operator (+=) adds a value to a variable.

#### Assignment

var x = 10;  
x += 5;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_plusequal)

### JavaScript String Operators

The + operator can also be used to add (concatenate) strings.

|  |  |
| --- | --- |
| **Note** | When used on strings, the + operator is called the concatenation operator. |

#### Example

txt1 = "John";  
txt2 = "Doe";  
txt3 = txt1 + " " + txt2;

The result of **txt3** will be:

John Doe

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concatenate)

The += assignment operator can also be used to add (concatenate) strings:

#### Example

txt1 = "What a very ";  
txt1 += "nice day";

The result of **txt1** will be:

What a very nice day

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat4)

### Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

#### Example

x = 5 + 5;  
y = "5" + 5;  
z= "Hello" + 5;

The result of *x*, *y*, and *z* will be:

10  
55  
Hello5

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat5)

The rule is: **If you add a number and a string, the result will be a string!**

### JavaScript Comparison and Logical Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | | **Description** |
| == | | equal to |
| === | | equal value and equal type |
| != | | not equal |
| !== | | not equal value or not equal type |
| > | | greater than |
| < | | less than |
| >= | | greater than or equal to |
| <= | | less than or equal to |
| **Note** | Comparison and logical operators are described in the **JS Comparisons** chapter. | | |

## JavaScript Arithmetic

A typical thing to do with numbers is arithmetic.

### JavaScript Arithmetic Operators

Arithmetic operators perform arithmetic on numbers (literals or variables).

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |
| ++ | Increment |
| -- | Decrement |

### Arithmetic Operations

A typical arithmetic operation operates on two numbers.

The two numbers can be literals:

#### Example

var x = 100 + 50;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_operation)

or variables:

#### Example

var x = a + b;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_variables)

or expressions:

#### Example

var x = (100 + 50) \* a;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_expressions)

### Operators and Operands

The numbers (in an arithmetic operation) are called **operands**.

The operation (to be performed between the two operands) is defined by an **operator**.

|  |  |  |
| --- | --- | --- |
| **Operand** | **Operator** | **Operand** |
| 100 | + | 50 |

The **addition** operator (+) adds numbers:

#### Adding

var x = 5;  
var y = 2;  
var z = x + y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_add)

The **subtraction** operator (-) subtracts numbers.

#### Subtracting

var x = 5;  
var y = 2;  
var z = x - y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_sub)

The **multiplication** operator (\*) multiplies numbers.

#### Multiplying

var x = 5;  
var y = 2;  
var z = x \* y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mult)

The **division** operator (/) divides numbers.

#### Dividing

var x = 5;  
var y = 2;  
var z = x / y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_div)

The **modular** operator (%) returns the division remainder.

#### Modulus

var x = 5;  
var y = 2;  
var z = x % y;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mod)

The **increment** operator (++) increments numbers.

#### Incrementing

var x = 5;  
x++;  
var z = x;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_increment)

The **decrement** operator (--) decrements numbers.

#### Decrementing

var x = 5;  
x--;  
var z = x;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_decrement)

### Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

#### Example

var x = 100 + 50 \* 3;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence1)

Is the result of example above the same as 150 \* 3, or is it the same as 100 + 150?

Is the addition or the multiplication done first?

As in traditional school mathematics, the multiplication is done first.

Multiplication (\*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

And (as in school mathematics) the precedence can be changed by using parentheses:

#### Example

var x = (100 + 50) \* 3;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence2)

When using parentheses, the operations inside the parentheses are computed first.

When many operations have the same precedence (like addition and subtraction), they are computed from left to right:

#### Example

var x = 100 + 50 - 3;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_arithmetic_precedence3)

### JavaScript Operator Precedence Values

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Operator** | **Description** | **Example** |
| 19 | ( ) | Expression grouping | (3 + 4) |
|  |  |  |  |
| 18 | . | Member | person.name |
| 18 | [] | Member | person["name"] |
|  |  |  |  |
| 17 | () | Function call | myFunction() |
| 17 | new | Create | new Date() |
|  |  |  |  |
| 16 | ++ | Postfix Increment | ++i |
| 16 | -- | Postfix Decrement | --i |
|  |  |  |  |
| 15 | ++ | Prefix Increment | i++ |
| 15 | -- | Prefix Decrement | i-- |
| 15 | ! | Logical not | !(x==y) |
| 15 | typeof | Type | typeof x |
|  |  |  |  |
| 14 | \* | Multiplication | 10 \* 5 |
| 14 | / | Division | 10 / 5 |
| 14 | % | Modulo division | 10 % 5 |
| 14 | \*\* | Exponentiation | 10 \*\* 2 |
|  |  |  |  |
| 13 | + | Addition | 10 + 5 |
| 13 | - | Subtraction | 10 - 5 |
|  |  |  |  |
| 12 | << | Shift left | x << 2 |
| 12 | >> | Shift right | x >> 2 |
|  |  |  |  |
| 11 | < | Less than | x < y |
| 11 | <= | Less than or equal | x <= y |
| 11 | > | Greater than | x > y |
| 11 | >= | Greater than or equal | x >= y |
|  |  |  |  |
| 10 | == | Equal | x == y |
| 10 | === | Strict equal | x === y |
| 10 | != | Unequal | x != y |
| 10 | !== | Strict unequal | x !== y |
|  |  |  |  |
| 6 | && | And | x && y |
| 5 | || | Or | x || y |
|  |  |  |  |
| 3 | = | Assignment | x = y |
| 3 | += | Assignment | x += y |
| 3 | -= | Assignment | x -= y |
| 3 | \*= | Assignment | x \*= y |
| 3 | /= | Assignment | x /= y |

|  |  |
| --- | --- |
| **Note** | Expressions in parentheses are fully computed before the value is used in the rest of the expression. |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arithmetic1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arithmetic2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arithmetic3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arithmetic4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arithmetic5)

## JavaScript Assignment

### JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| \*= | x \*= y | x = x \* y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

The = assignment operator assigns a value to a variable.

#### Assignment

var x = 10;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_equal)

The += assignment operator adds a value to a variable.

#### Assignment

var x = 10;  
x += 5;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_plusequal)

The -= assignment operator subtracts a value from a variable.

#### Assignment

var x = 10;  
x -= 5;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_minequal)

The \*= assignment operator multiplies a variable.

#### Assignment

var x = 10;  
x \*= 5;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_multequal)

The /= assignment divides a variable.

#### Assignment

var x = 10;  
x /= 5;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_divequal)

The %= assignment operator assigns a remainder to a variable.

#### Assignment

var x = 10;  
x %= 5;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_assign_modequal)

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_assignment1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_assignment2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_assignment3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_assignment4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_assignment5)

## JavaScript Data Types

String, Number, Boolean, Array, Object.

### JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, arrays, objects and more:

var length = 16;                               // Number  
var lastName = "Johnson";                      // String  
var cars = ["Saab", "Volvo", "BMW"];           // Array  
var x = {firstName:"John", lastName:"Doe"};    // Object

### The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

var x = 16 + "Volvo";

Does it make any sense to add "Volvo" to sixteen? Will it produce a result? Will it produce an error?

JavaScript will treat the example above as:

var x = "16" + "Volvo";

|  |  |
| --- | --- |
| **Note** | If the second operand is a string, JavaScript will also treat the first operand as a string. |

#### Example

var x = 16 + "Volvo";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstring)

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

#### JavaScript:

var x = 16 + 4 + "Volvo";

#### Result:

20Volvo

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_1)

#### JavaScript:

var x = "Volvo" + 16 + 4;

#### Result:

Volvo164

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_2)

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

### JavaScript Has Dynamic Types

JavaScript has dynamic types. This means that the same variable can be used as different types:

#### Example

var x;               // Now x is undefined  
var x = 5;           // Now x is a Number  
var x = "John";      // Now x is a String

### JavaScript Strings

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

#### Example

var carName = "Volvo XC60";   // Using double quotes  
var carName = 'Volvo XC60';   // Using single quotes

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";             // Single quote inside double quotes  
var answer = "He is called 'Johnny'";    // Single quotes inside double quotes  
var answer = 'He is called "Johnny"';    // Double quotes inside single quotes

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string)

You will learn more about strings later in this tutorial.

### JavaScript Numbers

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

#### Example

var x1 = 34.00;     // Written with decimals  
var x2 = 34;        // Written without decimals

Extra large or extra small numbers can be written with scientific (exponential) notation:

#### Example

var y = 123e5;      // 12300000  
var z = 123e-5;     // 0.00123

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_numbers)

You will learn more about numbers later in this tutorial.

### JavaScript Booleans

Booleans can only have two values: true or false.

#### Example

var x = true;  
var y = false;

Booleans are often used in conditional testing.

You will learn more about conditional testing later in this tutorial.

### JavaScript Arrays

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

#### Example

var cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_array)

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You will learn more about arrays later in this tutorial.

### JavaScript Objects

JavaScript objects are written with curly braces.

Object properties are written as name:value pairs, separated by commas.

#### Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_object)

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

You will learn more about objects later in this tutorial.

### The typeof Operator

You can use the JavaScript **typeof** operator to find the type of a JavaScript variable:

#### Example

typeof "John"                // Returns string   
typeof 3.14                  // Returns number  
typeof false                 // Returns boolean  
typeof [1,2,3,4]             // Returns object  
typeof {name:'John', age:34} // Returns object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof)

|  |  |
| --- | --- |
| **Note** | In JavaScript, an array is a special type of object. Therefore typeof [1,2,3,4] returns object. |

### Undefined

In JavaScript, a variable without a value, has the value **undefined**. The typeof is also **undefined**.

#### Example

var person;                  // Value is undefined, type is undefined

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined)

Any variable can be emptied, by setting the value to **undefined**. The type will also be **undefined**.

#### Example

person = undefined;          // Value is undefined, type is undefined

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_2)

### Empty Values

An empty value has nothing to do with undefined.

An empty string variable has both a value and a type.

#### Example

var car = "";                // The value is "", the typeof is string

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_empty)

### Null

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

Unfortunately, in JavaScript, the data type of null is an object.

|  |  |
| --- | --- |
| **Note** | You can consider it a bug in JavaScript that typeof null is an object. It should be null. |

You can empty an object by setting it to null:

#### Example

var person = null;           // Value is null, but type is still an object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_null)

You can also empty an object by setting it to undefined:

#### Example

var person = undefined;     // Value is undefined, type is undefined

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined_1)

### Difference Between Undefined and Null

typeof undefined             // undefined  
typeof null                  // object  
null === undefined           // false  
null == undefined            // true

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_typeof_undefined_3)

## JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

#### Example

function myFunction(p1, p2) {  
    return p1 \* p2;              // The function returns the product of p1 and p2  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_return)

### JavaScript Function Syntax

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **(*parameter1,  parameter2, ...*)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

function *name*(*parameter1, parameter2, parameter3*) {  
    *code to be executed*  
}

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** received by the function when it is invoked.

Inside the function, the arguments behaves as local variables.

|  |  |
| --- | --- |
| **Note** | A Function is much the same as a Procedure or a Subroutine, in other programming languages. |

### Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

* When an event occurs (when a user clicks a button)
* When it is invoked (called) from JavaScript code
* Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

### Function Return

When JavaScript reaches a **return statement**, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

#### Example

Calculate the product of two numbers, and return the result:

var x = myFunction(4, 3);        // Function is called, return value will end up in x  
  
function myFunction(a, b) {  
    return a \* b;                // Function returns the product of a and b  
}

The result in x will be:

12

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_return)

### Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

#### Example

Convert Fahrenheit to Celsius:

function toCelsius(fahrenheit) {  
    return (5/9) \* (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius(77);

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius)

### The () Operator Invokes the Function

Using the example above, toCelsius refers to the function object, and toCelsius() refers to the function result.

#### Example

Accessing a function without () will return the function definition:

function toCelsius(fahrenheit) {  
    return (5/9) \* (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius_2)

### Functions Used as Variables

In JavaScript, you can use functions the same way as you use variables.

#### Example

You can use:

var text = "The temperature is " + toCelsius(77) + " Celsius";

Instead of:

var x = toCelsius(32);  
var text = "The temperature is " + x + " Celsius";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_variable)

|  |  |
| --- | --- |
| **Note** | You will learn a lot more about functions later in this tutorial. |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_functions1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_functions2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_functions3)   [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_functions4)   [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_functions5)

## JavaScript Objects

### Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

|  |  |  |
| --- | --- | --- |
| **Object** | **Properties** | **Methods** |
| http://www.w3schools.com/js/objectExplained.gif | car.name = Fiat  car.model = 500  car.weight = 850kg  car.color = white | car.start()  car.drive()  car.brake()   car.stop() |

All cars have the same **properties**, but the property values differ from car to car.

All cars have the same **methods**, but the methods are performed at different times.

### JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

var car = "Fiat";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_variable)

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

var car = {type:"Fiat", model:500, color:"white"};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_object)

The values are written as **name:value** pairs (name and value separated by a colon).

|  |  |
| --- | --- |
| **Note** | JavaScript objects are containers for **named values**. |

### Object Properties

The name:values pairs (in JavaScript objects) are called **properties**.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |

### Object Methods

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

|  |  |
| --- | --- |
| **Property** | **Property Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |
| fullName | function() {return this.firstName + " " + this.lastName;} |

|  |  |
| --- | --- |
| **Note** | JavaScript objects are containers for named values (called properties) and methods. |

### Object Definition

You define (and create) a JavaScript object with an object literal:

#### Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_create_1)

Spaces and line breaks are not important. An object definition can span multiple lines:

#### Example

var person = {  
    firstName:"John",  
    lastName:"Doe",  
    age:50,  
    eyeColor:"blue"  
};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_create_2)

### Accessing Object Properties

You can access object properties in two ways:

*objectName.propertyName*

or

*objectName["propertyName"]*

#### Example1

person.lastName;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_1)

#### Example2

person["lastName"];

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_2)

### Accessing Object Methods

You access an object method with the following syntax:

*objectName.methodName()*

#### Example

name = person.fullName();

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_method)

If you access the fullName **property**, without (), it will return the **function definition**:

#### Example

name = person.fullName;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_function)

### Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

var x = new String();        // Declares x as a String object  
var y = new Number();        // Declares y as a Number object  
var z = new Boolean();       // Declares z as a Boolean object

Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

|  |  |
| --- | --- |
| **Note** | You will learn more about objects later in this tutorial. |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_objects1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_objects2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_objects3)

## JavaScript Scope

Scope is the set of variables you have access to.

### JavaScript Scope

In JavaScript, objects and functions are also variables.

**In JavaScript, scope is the set of variables, objects, and functions you have access to.**

JavaScript has function scope: The scope changes inside functions.

### Local JavaScript Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables have **local scope**: They can only be accessed within the function.

#### Example

// code here can not use carName  
  
function myFunction() {  
    var carName = "Volvo";  
  
    // code here can use carName  
  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_scope_local)

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

### Global JavaScript Variables

A variable declared outside a function, becomes **GLOBAL**.

A global variable has **global scope**: All scripts and functions on a web page can access it.

#### Example

var carName = " Volvo";  
  
// code here can use carName  
  
function myFunction() {  
  
    // code here can use carName   
  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_scope_global)

### Automatically Global

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.

This code example will declare carName as a global variable, even if it is executed inside a function.

#### Example

// code here can use carName  
  
function myFunction() {  
    carName = "Volvo";  
  
    // code here can use carName  
  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_local_global)

### The Lifetime of JavaScript Variables

The lifetime of a JavaScript variable starts when it is declared.

Local variables are deleted when the function is completed.

Global variables are deleted when you close the page.

### Function Arguments

Function arguments (parameters) work as local variables inside functions.

### Global Variables in HTML

With JavaScript, the global scope is the complete JavaScript environment.

In HTML, the global scope is the window object: All global variables belong to the window object.

#### Example

// code here can use window.carName  
  
function myFunction() {  
    carName = "Volvo";  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_scope_window)

#### Did You Know?

|  |  |
| --- | --- |
| **Note** | Your global variables (or functions) can overwrite window variables (or functions). Any function, including the window object, can overwrite your global variables and functions. |

## JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

### HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

* An HTML web page has finished loading
* An HTML input field was changed
* An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

<*some-HTML-element* *some-event*=**'*some JavaScript*'**>

With double quotes:

<*some-HTML-element* *some-event*=**"*some JavaScript*"**>

In the following example, an onclick attribute (with code), is added to a button element:

#### Example

<button onclick='getElementById("demo").innerHTML=Date()'>The time is?</button>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick1)

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using **this**.innerHTML):

#### Example

<button onclick="this.innerHTML=Date()">The time is?</button>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick)

|  |  |
| --- | --- |
| **Note** | JavaScript code is often several lines long. It is more common to see event attributes calling functions: |

#### Example

<button onclick="displayDate()">The time is?</button>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events1)

### Common HTML Events

Here is a list of some common HTML events:

|  |  |
| --- | --- |
| **Event** | **Description** |
| onchange | An HTML element has been changed |
| onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

The list is much longer: [W3Schools JavaScript Reference HTML DOM Events](http://www.w3schools.com/jsref/dom_obj_event.asp).

### What can JavaScript Do?

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

* Things that should be done every time a page loads
* Things that should be done when the page is closed
* Action that should be performed when a user clicks a button
* Content that should be verified when a user inputs data
* And more ...

Many different methods can be used to let JavaScript work with events:

* HTML event attributes can execute JavaScript code directly
* HTML event attributes can call JavaScript functions
* You can assign your own event handler functions to HTML elements
* You can prevent events from being sent or being handled
* And more ...

|  |  |
| --- | --- |
| **Note** | You will learn a lot more about events and event handlers in the HTML DOM chapters. |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_events1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_events2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_events3)

## JavaScript Strings

JavaScript strings are used for storing and manipulating text.

### JavaScript Strings

A JavaScript string simply stores a series of characters like "John Doe".

A string can be any text inside quotes. You can use single or double quotes:

#### Example

var carname = "Volvo XC60";  
var carname = 'Volvo XC60';

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strings)

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 »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strings_quotes)

### 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 »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_length)

### Special Characters

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

var y = "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 **\ escape character**.

The backslash escape character turns special characters into string characters:

#### Example

var x = 'It\'s alright';  
var y = "We are the so-called \"Vikings\" from the north."

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strings_escape)

The escape character (\) can also be used to insert other special characters in a string.

This is the list of special characters that can be added to a text string with the backslash sign:

|  |  |
| --- | --- |
| **Code** | **Outputs** |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \n | new line |
| \r | carriage return |
| \t | tab |
| \b | backspace |
| \f | form feed |

### 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 »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_linebreak)

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 »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_break)

|  |  |
| --- | --- |
| **Note** | The \ method is not a ECMAScript (JavaScript) standard. Some browsers do not allow spaces behind the \ character. |

The safest (but a little slower) way to break a long string is to use string addition:

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_break_ok)

You cannot break up a code line with a backslash:

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strings_codebreak)

### 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 »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object)

|  |  |
| --- | --- |
| **Note** | Don't create strings as objects. It slows down execution speed, and produces nasty side effects: |

When using the == equality operator, equal strings looks equal:

#### Example

var x = "John";               
var y = new String("John");  
  
// (x == y) is true because x and y have equal values

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object1)

When using the === equality 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 »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object2)

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  
// (x == x) is true because both are the same object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_object3)

|  |  |
| --- | --- |
| **Note** | JavaScript objects cannot be compared. |

### String Properties and 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.

**String methods are covered in next chapter.**

### String Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| constructor | Returns the function that created the String object's prototype |
| length | Returns the length of a string |
| prototype | Allows you to add properties and methods to an object |

### String Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| charAt() | Returns the character at the specified index (position) |
| charCodeAt() | Returns the Unicode of the character at the specified index |
| concat() | Joins two or more strings, and returns a copy of the joined strings |
| fromCharCode() | Converts Unicode values to characters |
| indexOf() | Returns the position of the first found occurrence of a specified value in a string |
| lastIndexOf() | Returns the position of the last found occurrence of a specified value in a string |
| localeCompare() | Compares two strings in the current locale |
| match() | Searches a string for a match against a regular expression, and returns the matches |
| replace() | Searches a string for a value and returns a new string with the value replaced |
| search() | Searches a string for a value and returns the position of the match |
| slice() | Extracts a part of a string and returns a new string |
| split() | Splits a string into an array of substrings |
| substr() | Extracts a part of a string from a start position through a number of characters |
| substring() | Extracts a part of a string between two specified positions |
| toLocaleLowerCase() | Converts a string to lowercase letters, according to the host's locale |
| toLocaleUpperCase() | Converts a string to uppercase letters, according to the host's locale |
| toLowerCase() | Converts a string to lowercase letters |
| toString() | Returns the value of a String object |
| toUpperCase() | Converts a string to uppercase letters |
| trim() | Removes whitespace from both ends of a string |
| valueOf() | Returns the primitive value of a String object |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_strings1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_strings2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_strings3)   [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_strings4)

## JavaScript String Methods

String methods help you to work with strings.

### Finding a String in a String

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_indexof)

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_lastindexof)

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

|  |  |
| --- | --- |
| **Note** | JavaScript counts positions from zero. 0 is the first position in a string, 1 is the second, 2 is the third ... |

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

### Searching for a String in a String

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_search_locate)

|  |  |
| --- | --- |
| **Note** | **Did You Notice?** |

The two methods, indexOf() and search(), are equal.

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

The two methods are equal, but the search() method can take much more powerful search values.

You will learn more about powerful search values in the chapter about regular expressions.

### Extracting String Parts

There are 3 methods for extracting a part of a string:

* slice(start, end)
* substring(start, end)
* substr(start, length)

### The slice() Method

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

The method takes 2 parameters: the starting index (position), and the ending index (position).

This example slices out a portion of a string from position 7 to position 13:

#### Example

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

The result of res will be:

Banana

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice)

If a parameter is negative, the position is counted from the end of the string.

This example slices out a portion of a string from position -12 to position -6:

#### Example

var str = "Apple, Banana, Kiwi";  
var res = str.slice(-12,-6);

The result of res will be:

Banana

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice_negative)

If you omit the second parameter, the method will slice out the rest of the string:

#### Example

var res = str.slice(7);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice_rest)

or, counting from the end:

#### Example

var res = str.slice(-12);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_slice_rest_negative)

|  |  |
| --- | --- |
| **Note** | Negative positions does not work in Internet Explorer 8 and earlier. |

### The substring() Method

**substring()** is similar to slice().

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

#### Example

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

The result of *res* will be:

Banana

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_substring)

If you omit the second parameter, substring() will slice out the rest of the string.

### The substr() Method

**substr()** is similar to slice().

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

#### Example

var str = "Apple, Banana, Kiwi";  
var res = str.substr(7,6);

The result of res will be:

Banana

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_substr)

If the first parameter is negative, the position counts from the end of the string.

The second parameter can not be negative, because it defines the length.

If you omit the second parameter, substr() will slice out the rest of the string.

### Replacing String Content

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

#### Example

str = "Please visit Microsoft!";  
var n = str.replace("Microsoft","W3Schools");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace)

The replace() method can also take a regular expression as the search value.

### Converting to Upper and Lower Case

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

#### Example

var text1 = "Hello World!";       // String  
var text2 = text1.toUpperCase();  // text2 is text1 converted to upper

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_toupper)

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

#### Example

var text1 = "Hello World!";       // String  
var text2 = text1.toLowerCase();  // text2 is text1 converted to lower

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_tolower)

### The concat() Method

**concat()** joins two or more strings:

#### Example

var text1 = "Hello";  
var text2 = "World";  
text3 = text1.concat(" ",text2);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_concat)

The **concat()** method can be used instead of the plus operator. These two lines do the same:

#### Example

var text = "Hello" + " " + "World!";  
var text = "Hello".concat(" ","World!");

|  |  |
| --- | --- |
| **Note** | All string methods return a new string. They don't modify the original string. Formally said: Strings are immutable: Strings cannot be changed, only replaced. |

### Extracting String Characters

There are 2 **safe** methods for extracting string characters:

* charAt(position)
* charCodeAt(position)

### The charAt() Method

The **charAt()** method returns the character at a specified index (position) in a string:

#### Example

var str = "HELLO WORLD";  
str.charAt(0);            // returns H

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_charat)

### The charCodeAt() Method

The **charCodeAt()** method returns the unicode of the character at a specified index in a string:

#### Example

var str = "HELLO WORLD";  
  
str.charCodeAt(0);         // returns 72

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_charcodeat)

### Accessing a String as an Array is Unsafe

You might have seen code like this, accessing a string as an array:

var str = "HELLO WORLD";  
  
str[0];                   // returns H

This is **unsafe** and **unpredictable:**

* It does not work in all browsers (not in IE5, IE6, IE7)
* It makes strings look like arrays (but they are not)
* str[0] = "H" does not give an error (but does not work)

If you want to read a string as an array, convert it to an array first.

### Converting a String to an Array

A string can be converted to an array with the **split()** method:

#### Example

var txt = "a,b,c,d,e";   // String  
txt.split(",");          // Split on commas  
txt.split(" ");          // Split on spaces  
txt.split("|");          // Split on pipe

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_split)

If the separator is omitted, the returned array will contain the whole string in index [0].

If the separator is "", the returned array will be an array of single characters:

#### Example

var txt = "Hello";       // String  
txt.split("");           // Split in characters

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_split_char)

### Complete String Reference

For a complete reference, go to our [Complete JavaScript String Reference](http://www.w3schools.com/jsref/jsref_obj_string.asp).

The reference contains descriptions and examples of all string properties and methods.

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_stringsmet1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_stringsmet2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_stringsmet3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_stringsmet4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_stringsmet5)  [Exercise 6 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_stringsmet6)

## JavaScript Numbers

JavaScript has only one type of number.

Numbers can be written with, or without, decimals.

### JavaScript Numbers

JavaScript numbers can be written with, or without decimals:

#### Example

var x = 34.00;    // A number with decimals  
var y = 34;       // A number without decimals

Extra large or extra small numbers can be written with scientific (exponent) notation:

#### Example

var x = 123e5;    // 12300000  
var y = 123e-5;   // 0.00123

### JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.   
  
This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

|  |  |  |
| --- | --- | --- |
| **Value (aka Fraction/Mantissa)** | **Exponent** | **Sign** |
| 52 bits (0 - 51) | 11 bits (52 - 62) | 1 bit (63) |

### Precision

Integers (numbers without a period or exponent notation) are considered accurate up to 15 digits.

#### Example

var x = 999999999999999;   // x will be 999999999999999  
var y = 9999999999999999;  // y will be 10000000000000000

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_inaccurate1)

The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate:

#### Example

var x = 0.2 + 0.1;         // x will be 0.30000000000000004

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_inaccurate2)

To solve the problem above, it helps to multiply and divide:

#### Example

var x = (0.2 \* 10 + 0.1 \* 10) / 10;       // x will be 0.3

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_inaccurate3)

### Hexadecimal

JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

#### Example

var x = 0xFF;             // x will be 255

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_hex)

|  |  |
| --- | --- |
| **Note** | Never write a number with a leading zero (like 07). Some JavaScript versions interpret numbers as octal if they are written with a leading zero. |

By default, Javascript displays numbers as base 10 decimals.

But you can use the toString() method to output numbers as base 16 (hex), base 8 (octal), or base 2 (binary).

#### Example

var myNumber = 128;  
myNumber.toString(16);     // returns 80  
myNumber.toString(8);      // returns 200  
myNumber.toString(2);      // returns 10000000

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_tostring)

### Infinity

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

#### Example

var myNumber = 2;  
while (myNumber != Infinity) {          // Execute until Infinity  
    myNumber = myNumber \* myNumber;  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_infinity)

Division by 0 (zero) also generates Infinity:

#### Example

var x =  2 / 0;          // x will be Infinity  
var y = -2 / 0;          // y will be -Infinity

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_infinity_zero)

Infinity is a number: typeOf Infinity returns number.

#### Example

typeof Infinity;        // returns "number"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_infinity_number)

### NaN - Not a Number

NaN is a JavaScript reserved word indicating that a value is not a number.

Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):

#### Example

var x = 100 / "Apple";  // x will be NaN (Not a Number)

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nan_string)

However, if the string contains a numeric value , the result will be a number:

#### Example

var x = 100 / "10";     // x will be 10

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nan_number)

You can use the global JavaScript function isNaN() to find out if a value is a number.

#### Example

var x = 100 / "Apple";  
isNaN(x);               // returns true because x is Not a Number

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_isnan_true)

Watch out for NaN. If you use NaN in a mathematical operation, the result will also be NaN:

#### Example

var x = NaN;  
var y = 5;  
var z = x + y;         // z will be NaN

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nan_math)

 Or the result might be a concatenation:

#### Example

var x = NaN;  
var y = "5";  
var z = x + y;         // z will be NaN5

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nan_concat)

NaN is a number, and typeof NaN returns number:

#### Example

typeof NaN;             // returns "number"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_typeof_nan)

### Numbers Can be Objects

Normally JavaScript numbers are primitive values created from literals: **var x = 123**

But numbers can also be defined as objects with the keyword new: **var y = new Number(123)**

#### Example

var x = 123;  
var y = new Number(123);  
  
// typeof x returns number  
// typeof y returns object

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_number_type)

|  |  |
| --- | --- |
| **Note** | Don't create Number objects. They slow down execution speed, and produce nasty side effects: |

When using the == equality operator, equal numbers looks equal:

#### Example

var x = 500;               
var y = new Number(500);  
  
// (x == y) is true because x and y have equal values

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object1)

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

#### Example

var x = 500;               
var y = new Number(500);  
  
// (x === y) is false because x and y have different types

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object2)

Or even worse. Objects cannot be compared:

#### Example

var x = new Number(500);               
var y = new Number(500);  
  
// (x == y) is false because objects cannot be compared

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_object3)

|  |  |
| --- | --- |
| **Note** | JavaScript objects cannot be compared. |

### Number Properties and Methods

Primitive values (like 3.14 or 2014), cannot have properties and 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.

### Number Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| MAX\_VALUE | Returns the largest number possible in JavaScript |
| MIN\_VALUE | Returns the smallest number possible in JavaScript |
| NEGATIVE\_INFINITY | Represents negative infinity (returned on overflow) |
| NaN | Represents a "Not-a-Number" value |
| POSITIVE\_INFINITY | Represents infinity (returned on overflow) |

#### Example

var x = Number.MAX\_VALUE;

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_max)

Number properties belongs to the JavaScript's number object wrapper called **Number**.

These properties can only be accessed as **Number**.MAX\_VALUE.

Using *myNumber*.MAX\_VALUE, where *myNumber* is a variable, expression, or value, will return undefined:

#### Example

var x = 6;  
var y = x.MAX\_VALUE;    // y becomes undefined

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_max_undefined)

|  |  |
| --- | --- |
| **Note** | Number methods are covered in the next chapter |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_numbers1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_numbers2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_numbers3)   [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_numbers4)

## JavaScript Number Methods

Number methods help you to work with numbers.

### Global Methods

JavaScript global functions can be used on all JavaScript data types.

These are the most relevant methods, when working with numbers:

|  |  |
| --- | --- |
| **Method** | **Description** |
| Number() | Returns a number, converted from its argument. |
| parseFloat() | Parses its argument and returns a floating point number |
| parseInt() | Parses its argument and returns an integer |

### Number Methods

JavaScript number methods are methods that can be used on numbers:

|  |  |
| --- | --- |
| **Method** | **Description** |
| toString() | Returns a number as a string |
| toExponential() | Returns a string, with a number rounded and written using exponential notation. |
| toFixed() | Returns a string, with a number rounded and written with a specified number of decimals. |
| toPrecision() | Returns a string, with a number written with a specified length |
| valueOf() | Returns a number as a number |

|  |  |
| --- | --- |
| **Note** | All number methods return a new value. They do not change the original variable. |

### The toString() Method

**toString()** returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

#### Example

var x = 123;  
x.toString();            // returns 123 from variable x  
(123).toString();        // returns 123 from literal 123  
(100 + 23).toString();   // returns 123 from expression 100 + 23

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_tostring)

### The toExponential() Method

**toExponential()** returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point:

#### Example

var x = 9.656;  
x.toExponential(2);     // returns 9.66e+0  
x.toExponential(4);     // returns 9.6560e+0  
x.toExponential(6);     // returns 9.656000e+0

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_toexponential)

The parameter is optional. If you don't specify it, JavaScript will not round the number.

### The toFixed() Method

**toFixed()** returns a string, with the number written with a specified number of decimals:

#### Example

var x = 9.656;  
x.toFixed(0);           // returns 10  
x.toFixed(2);           // returns 9.66  
x.toFixed(4);           // returns 9.6560  
x.toFixed(6);           // returns 9.656000

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_tofixed)

|  |  |
| --- | --- |
| **Note** | toFixed(2) is perfect for working with money. |

### The toPrecision() Method

**toPrecision()** returns a string, with a number written with a specified length:

#### Example

var x = 9.656;  
x.toPrecision();        // returns 9.656  
x.toPrecision(2);       // returns 9.7  
x.toPrecision(4);       // returns 9.656  
x.toPrecision(6);       // returns 9.65600

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_toprecision)

### Converting Variables to Numbers

There are 3 JavaScript functions that can be used to convert variables to numbers:

* The Number() method
* The parseInt() method
* The parseFloat() method

These methods are not **number** methods, but **global** JavaScript methods.

### The Number() Method

**Number()**, can be used to convert JavaScript variables to numbers:

#### Example

x = true;  
Number(x);        // returns 1  
x = false;       
Number(x);        // returns 0  
x = new Date();  
Number(x);        // returns 1404568027739  
x = "10"  
Number(x);        // returns 10  
x = "10 20"  
Number(x);        // returns NaN

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_global_number)

|  |  |
| --- | --- |
| **Note** | Used on Date(), the Number() method returns the number of milliseconds since 1.1.1970. |

### The parseInt() Method

**parseInt()** parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

#### Example

parseInt("10");         // returns 10  
parseInt("10.33");      // returns 10  
parseInt("10 20 30");   // returns 10  
parseInt("10 years");   // returns 10  
parseInt("years 10");   // returns NaN

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_global_parseint)

If the number cannot be converted, NaN (Not a Number) is returned.

### The parseFloat() Method

**parseFloat()** parses a string and returns a number. Spaces are allowed. Only the first number is returned:

#### Example

parseFloat("10");        // returns 10  
parseFloat("10.33");     // returns 10.33  
parseFloat("10 20 30");  // returns 10  
parseFloat("10 years");  // returns 10  
parseFloat("years 10");  // returns NaN

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_global_parsefloat)

If the number cannot be converted, NaN (Not a Number) is returned.

### The valueOf() Method

**valueOf()** returns a number as a number.

#### Example

var x = 123;  
x.valueOf();            // returns 123 from variable x  
(123).valueOf();        // returns 123 from literal 123  
(100 + 23).valueOf();   // returns 123 from expression 100 + 23

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_valueof)

In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).

The valueOf() method is used internally in JavaScript to convert Number objects to primitive values.

There is no reason to use it in your code.

|  |  |
| --- | --- |
| **Note** | In JavaScript, all data types have a valueOf() and a toString() method. |

### Complete JavaScript Number Reference

For a complete reference, go to our [Complete JavaScript Number Reference](http://www.w3schools.com/jsref/jsref_obj_number.asp).

The reference contains descriptions and examples of all Number properties and methods.

## JavaScript Math Object

The Math object allows you to perform mathematical tasks on numbers.

### The Math Object

The Math object allows you to perform mathematical tasks.

The Math object includes several mathematical methods.

One common use of the Math object is to create a random number:

#### Example

Math.random();       // returns a random number

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_random)

|  |  |
| --- | --- |
| **Note** | Math has no constructor. No methods have to create a Math object first. |

### Math.min() and Math.max()

Math.min() and Math.max() can be used to find the lowest or highest value in a list of arguments:

#### Example

Math.min(0, 150, 30, 20, -8, -200);      // returns -200

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_min)

#### Example

Math.max(0, 150, 30, 20, -8, -200);      // returns 150

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_max)

### Math.random()

Math.random() returns a random number between 0 (inclusive),  and 1 (exclusive):

#### Example

Math.random();              // returns a random number

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_random)

|  |  |
| --- | --- |
| **Note** | Math.random() always returns a number lower than 1. |

### Math.round()

Math.round() rounds a number to the nearest integer:

#### Example

Math.round(4.7);            // returns 5  
Math.round(4.4);            // returns 4

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_round)

## Math.ceil()

Math.ceil() rounds a number **up** to the nearest integer:

#### Example

Math.ceil(4.4);             // returns 5

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_ceil)

### Math.floor()

Math.floor() rounds a number **down** to the nearest integer:

#### Example

Math.floor(4.7);            // returns 4

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_floor)

Math.floor() and Math.random() can be used together to return a random number between 0 and 10:

#### Example

Math.floor(Math.random() \* 11);   // returns a random number between 0 and 10

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_floor_random)

### Math Constants

JavaScript provides 8 mathematical constants that can be accessed with the Math object:

#### Example

Math.E          // returns Euler's number  
Math.PI         // returns PI  
Math.SQRT2      // returns the square root of 2  
Math.SQRT1\_2    // returns the square root of 1/2  
Math.LN2        // returns the natural logarithm of 2  
Math.LN10       // returns the natural logarithm of 10  
Math.LOG2E      // returns base 2 logarithm of E  
Math.LOG10E     // returns base 10 logarithm of E

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_constants)

### Math Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| abs(x) | Returns the absolute value of x |
| acos(x) | Returns the arccosine of x, in radians |
| asin(x) | Returns the arcsine of x, in radians |
| atan(x) | Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians |
| atan2(y,x) | Returns the arctangent of the quotient of its arguments |
| ceil(x) | Returns x, rounded upwards to the nearest integer |
| cos(x) | Returns the cosine of x (x is in radians) |
| exp(x) | Returns the value of Ex |
| floor(x) | Returns x, rounded downwards to the nearest integer |
| log(x) | Returns the natural logarithm (base E) of x |
| max(x,y,z,...,n) | Returns the number with the highest value |
| min(x,y,z,...,n) | Returns the number with the lowest value |
| pow(x,y) | Returns the value of x to the power of y |
| random() | Returns a random number between 0 and 1 |
| round(x) | Rounds x to the nearest integer |
| sin(x) | Returns the sine of x (x is in radians) |
| sqrt(x) | Returns the square root of x |
| tan(x) | Returns the tangent of an angle |

### Complete Math Reference

For a complete reference, go to our [complete Math object reference](http://www.w3schools.com/jsref/jsref_obj_math.asp).

The reference contains descriptions and examples of all Math properties and methods.

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_math1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_math2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_math3)   [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_math4)

## JavaScript Dates

The Date object lets you work with dates (years, months, days, hours, minutes, seconds, and milliseconds)

### JavaScript Date Formats

A JavaScript date can be written as a string:

**Wed Oct 21 2015 15:26:43 GMT+0200**

or as a number:

**1445434003777**

Dates written as numbers, specifies the number of milliseconds since January 1, 1970, 00:00:00.

### Displaying Dates

In this tutorial we use a script to display dates inside a <p> element with id="demo":

#### Example

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = Date();  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_current)

The script above says: assign the value of Date() to the content (innerHTML) of the element with id="demo".

|  |  |
| --- | --- |
| **Note** | You will learn how to display a date, in a more readable format, at the bottom of this page. |

### Creating Date Objects

The Date object lets us work with dates.

A date consists of a year, a month, a day, an hour, a minute, a second, and milliseconds.

Date objects are created with the **new Date()** constructor.

There are **4 ways** of initiating a date:

new Date()  
new Date(milliseconds)  
new Date(dateString)  
new Date(year, month, day, hours, minutes, seconds, milliseconds)

Using new Date(), creates a new date object with the **current date and time**:

#### Example

<script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_new)

Using new Date(**date string**), creates a new date object from the **specified date and time**:

#### Example

<script>  
var d = new Date("October 13, 2014 11:13:00");  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_new_string)

|  |  |
| --- | --- |
| **Note** | Valid date strings (date formats) are described in the next chapter. |

Using new Date(**number**), creates a new date object as **zero time plus the number**.

Zero time is 01 January 1970 00:00:00 UTC. The number is specified in milliseconds:

#### Example

<script>  
var d = new Date(86400000);  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_new_millisec)

|  |  |
| --- | --- |
| **Note** | JavaScript dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC). One day contains 86,400,000 millisecond. |

Using new Date(**7 numbers**), creates a new date object with the **specified date and time**:

The 7 numbers specify the year, month, day, hour, minute, second, and millisecond, in that order:

#### Example

<script>  
var d = new Date(99,5,24,11,33,30,0);  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_new_all)

Variants of the example above let us omit any of the last 4 parameters:

#### Example

<script>  
var d = new Date(99,5,24);  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_new_numbers)

|  |  |
| --- | --- |
| **Note** | JavaScript counts months from 0 to 11. January is 0. December is 11. |

### Date Methods

When a Date object is created, a number of **methods** allow you to operate on it.

Date methods allow you to get and set the year, month, day, hour, minute, second, and millisecond of objects, using either local time or UTC (universal, or GMT) time.

|  |  |
| --- | --- |
| **Note** | Date methods are covered in a later chapter. |

### Displaying Dates

When you display a date object in HTML, it is automatically converted to a string, with the **toString()** method.

#### Example

<p id="demo"></p>  
  
<script>  
d = new Date();  
document.getElementById("demo").innerHTML = d;  
</script>

Is the same as:

<p id="demo"></p>  
  
<script>  
d = new Date();  
document.getElementById("demo").innerHTML = d.toString();  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_tostring)

The **toUTCString()** method converts a date to a UTC string (a date display standard).

#### Example

<script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.toUTCString();  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_toutcstring)

The **toDateString()** method converts a date to a more readable format:

#### Example

<script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.toDateString();  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_todatestring)

|  |  |
| --- | --- |
| **Note** | Date objects are static, not dynamic. The computer time is ticking, but date objects, once created, are not. |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dates1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dates2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dates3)

## JavaScript Date Formats

### Valid JavaScript Date Strings

There are generally three types of valid JavaScript date formats:

* ISO Dates
* Long Dates
* Short Dates

### JavaScript ISO Dates

The ISO 8601 syntax (YYYY-MM-DD) is the newest (and preferred) JavaScript date format:

#### Example

var d = new Date("2015-03-25");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_iso1)

Or as: "2015-12" (YYYY-MM):

#### Example

var d = new Date("2015-03");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_iso2)

Or as "2015" (YYYY):

#### Example

var d = new Date("2015");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_iso3)

Or as "2015-12-24T12:00:00":

#### Example

var d = new Date("2015-03-25T12:00:00");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_iso4)

The T in the date string, between the date and time, indicates UTC time.

|  |  |
| --- | --- |
| **Note** | UTC (Universal Time Coordinated)  is the same as GMT (Greenwich Mean Time). |

### JavaScript Long Dates.

Long dates are most often written with a "MMM DD YYYY" syntax like this:

#### Example

var d = new Date("Mar 25 2015");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_1)

But, year, month, and day can be in any order:

#### Example

var d = new Date("25 Mar 2015");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_2)

#### Example

var d = new Date("2015 Mar 25");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_3)

And, month can be written in full (January), or abbreviated (Jan):

#### Example

var d = new Date("January 25 2015");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_4)

#### Example

var d = new Date("Jan 25 2015");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_5)

Commas are ignored. Names are case insensitive:

#### Example

var d = new Date("2015, JANUARY, 25");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_6)

### JavaScript Short Dates.

Short dates are most often written with an "MM/DD/YYYY" syntax like this:

#### Example

var d = new Date("03/25/2015");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_7)

Either "/" or "-" can be used as a separator:

#### Example

var d = new Date("03-25-2015");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_8)

JavaScript will also accept "YYYY/MM/DD":

#### Example

var d = new Date("2015/03/25");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_9)

|  |  |
| --- | --- |
| **Note** | Month is written before day in all short date and ISO date formats. |

### Full Format

JavaScript will accept date strings in "full JavaScript format":

#### Example

var d = new Date("Wed Mar 25 2015 09:56:24 GMT+0100 (W. Europe Standard Time)");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_10)

JavaScript will ignore errors both in the day name and in the time parentheses:

#### Example

var d = new Date("Fri Mar 25 2015 09:56:24 GMT+0100 (Tokyo Time)");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_string_11)

## JavaScript Date Methods

Date methods let you get and set date values (years, months, days, hours, minutes, seconds, milliseconds)

### Date Get Methods

Get methods are used for getting a part of a date. Here are the most common (alphabetically):

|  |  |
| --- | --- |
| **Method** | **Description** |
| getDate() | Get the day as a number (1-31) |
| getDay() | Get the weekday as a number (0-6) |
| getFullYear() | Get the four digit year (yyyy) |
| getHours() | Get the hour (0-23) |
| getMilliseconds() | Get the milliseconds (0-999) |
| getMinutes() | Get the minutes (0-59) |
| getMonth() | Get the month (0-11) |
| getSeconds() | Get the seconds (0-59) |
| getTime() | Get the time (milliseconds since January 1, 1970) |

### The getTime() Method

**getTime()** returns the number of milliseconds since January 1, 1970:

#### Example

<script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getTime();  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_gettime)

### The getFullYear() Method

**getFullYear()** returns the year of a date as a four digit number:

#### Example

<script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getFullYear();  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_getfullyear)

### The getDay() Method

**getDay()** returns the weekday as a number (0-6):

#### Example

<script>  
var d = new Date();  
document.getElementById("demo").innerHTML = d.getDay();  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_getday)

|  |  |
| --- | --- |
| **Note** | In JavaScript, the first of the week (0) means "Sunday", even if some countries in the world consider the first day of the week to be "Monday". |

You can use an array of names, and getDay() to return the weekday as a name:

#### Example

<script>  
var d = new Date();  
var days = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"];  
document.getElementById("demo").innerHTML = days[d.getDay()];  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_weekday)

### Date Set Methods

Set methods are used for setting a part of a date. Here are the most common (alphabetically):

|  |  |
| --- | --- |
| **Method** | **Description** |
| setDate() | Set the day as a number (1-31) |
| setFullYear() | Set the year (optionally month and day) |
| setHours() | Set the hour (0-23) |
| setMilliseconds() | Set the milliseconds (0-999) |
| setMinutes() | Set the minutes (0-59) |
| setMonth() | Set the month (0-11) |
| setSeconds() | Set the seconds (0-59) |
| setTime() | Set the time (milliseconds since January 1, 1970) |

### The setFullYear() Method

**setFullYear()** sets a date object to a specific date. In this example, to January 14, 2020:

#### Example

<script>  
var d = new Date();  
d.setFullYear(2020, 0, 14);  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_setfullyear)

### The setDate() Method

**setDate()** sets the day of the month (1-31):

#### Example

<script>  
var d = new Date();  
d.setDate(20);  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_setdate)

The setDate() method can also be used to **add days** to a date:

#### Example

<script>  
var d = new Date();  
d.setDate(d.getDate() + 50);  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_ahead)

|  |  |
| --- | --- |
| **Note** | If adding days, shifts the month or year, the changes are handled automatically by the Date object. |

### Date Input - Parsing Dates

If you have a valid date string, you can use the **Date.parse()** method to convert it to milliseconds.

**Date.parse()** returns the number of milliseconds between the date and January 1, 1970:

#### Example

<script>  
var msec = Date.parse("March 21, 2012");  
document.getElementById("demo").innerHTML = msec;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_parse)

You can then use the number of milliseconds to **convert it to a date** object:

#### Example

<script>  
var msec = Date.parse("March 21, 2012");  
var d = new Date(msec);  
document.getElementById("demo").innerHTML = d;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_convert)

### Compare Dates

Dates can easily be compared.

The following example compares today's date with January 14, 2100:

#### Example

var today, someday, text;  
today = new Date();  
someday = new Date();  
someday.setFullYear(2100, 0, 14);  
  
if (someday > today) {  
    text = "Today is before January 14, 2100.";  
} else {  
    text = "Today is after January 14, 2100.";  
}  
document.getElementById("demo").innerHTML = text;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_compare)

|  |  |
| --- | --- |
| **Note** | JavaScript counts months from 0 to 11. January is 0. December is 11. |

### Complete JavaScript Date Reference

For a complete reference, go to our [Complete JavaScript Date Reference](http://www.w3schools.com/jsref/jsref_obj_date.asp).

The reference contains descriptions and examples of all Date properties and methods.

## JavaScript Arrays

JavaScript arrays are used to store multiple values in a single variable.

### Displaying Arrays

In this tutorial we will use a script to display arrays inside a <p> element with id="demo":

#### Example

<p id="demo"></p>  
  
<script>  
var cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;  
</script>

The first line (in the script) creates an array named cars.

The second line "finds" the element with id="demo", and "displays" the array in the "innerHTML" of it.

### Try it Yourself

Create an array, and assign values to it:

#### Example

var cars = ["Saab", "Volvo", "BMW"];

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array)

Spaces and line breaks are not important. A declaration can span multiple lines:

#### Example

var cars = [  
    "Saab",  
    "Volvo",  
    "BMW"  
];

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_newlines)

|  |  |
| --- | --- |
| **Note** | Never put a comma after the last element (like "BMW",). The effect is inconsistent across browsers. |

### What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

var car1 = "Saab";  
var car2 = "Volvo";  
var car3 = "BMW";

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

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:

var *array-name* = [*item1*, *item2*, ...];

Example:

var cars = ["Saab", "Volvo", "BMW"];

### Using the JavaScript Keyword new

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

#### Example

var cars = new Array("Saab", "Volvo", "BMW");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new)

|  |  |
| --- | --- |
| **Note** | The two examples above do exactly the same. There is no need to use new Array(). For simplicity, readability and execution speed, use the first one (the array literal method). |

### Access the Elements of an Array

You refer to an array element by referring to the **index number**.

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

var name = cars[0];

This statement modifies the first element in cars:

cars[0] = "Opel";

|  |  |
| --- | --- |
| **Note** | [0] is the first element in an array. [1] is the second. Array indexes start with 0. |

### You Can Have Different Objects in One Array

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;

### 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 John:

#### Array:

var person = ["John", "Doe", 46];

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_array)

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

#### Object:

var person = {firstName:"John", lastName:"Doe", age:46};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_object)

### Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

#### Examples

var x = cars.length;         // The length property returns the number of elements in cars  
var y = cars.sort();         // The sort() method sort cars in alphabetical order

Array methods are covered in the next chapter.

### The length Property

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

#### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.length;                       // the length of fruits is 4

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_length)

|  |  |
| --- | --- |
| **Note** | The length property is always one more than the highest array index. |

### Adding Array Elements

The easiest way to add a new element to an array is using the push method:

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add_push)

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_add)

Adding elements with high indexes can create undefined "holes" in an array:

#### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[10] = "Lemon";                // adds a new element (Lemon) to fruits

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_holes)

### Looping Array Elements

The best way to loop through an array, is using a "for" loop:

#### Example

var index;  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
for (index = 0; index < fruits.length; index++) {  
    text += fruits[index];  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_loop)

### 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**.

#### Example

var person = [];  
person[0] = "John";  
person[1] = "Doe";  
person[2] = 46;  
var x = person.length;         // person.length will return 3  
var y = person[0];             // person[0] will return "John"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_1)

|  |  |
| --- | --- |
| **Note** | **WARNING !!** If you use a named index, JavaScript will redefine the array to a standard object. After that, **all array methods and properties will produce incorrect results**. |

#### Example:

var person = [];  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
var x = person.length;         // person.length will return 0  
var y = person[0];             // person[0] will return undefined

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_2)

### The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

|  |  |
| --- | --- |
| **Note** | 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**.

### Avoid new Array()

There is no need to use the JavaScript's built-in array constructor **new** Array().

**Use [] instead.**

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

var points = new Array();         // Bad  
var points = [];                  // Good

These two different statements both create a new array containing 6 numbers:

var points = new Array(40, 100, 1, 5, 25, 10)  // Bad  
var points = [40, 100, 1, 5, 25, 10];          // Good

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_literal)

The **new** keyword complicates your code and produces nasty side effects:

var points = new Array(40, 100);  // Creates an array with two elements (40 and 100)

What if I remove one of the elements?

var points = new Array(40);       // Creates an array with 40 undefined elements !!!!!

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_new_error)

### 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":

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
  
typeof fruits;             // typeof returns object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_typeof)

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

To solve this problem you can create your own isArray() function:

function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_isarray)

The function above always returns true if the argument is an array.

Or more precisely: it returns true if the object prototype of the argument is "[object array]".

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arrays1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arrays2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arrays3)   [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arrays4)   [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arrays5)

## JavaScript Array Methods

The strength of JavaScript arrays lies in the array methods.

### Converting Arrays to Strings

In JavaScript, all objects have the valueOf() and toString() methods.

The **valueOf()** method is the default behavior for an array. It returns an array as a string:

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_valueof)

For JavaScript arrays, valueOf() and **toString()** are equal.

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_tostring)

The **join()** method also joins all array elements into a string.

It behaves just like toString(), but you can specify the separator:

#### Example

<p id="demo"></p>  
  
<script>  
var fruits = ["Banana", "Orange","Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.join(" \* ");  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_join)

### Popping and Pushing

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.

### Popping

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_pop)

The pop() method returns the value that was "popped out":

#### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
var x = fruits.pop();      // the value of x is "Mango"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_pop_out)

### Pushing

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

|  |  |
| --- | --- |
| **Note** | Remember: [0] is the first element in an array. [1] is the second. Array **indexes** start with 0. |

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_push)

The push() method returns the new array length:

#### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
var x = fruits.push("Kiwi");   //  the value of x is 5

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_push_length)

### Shifting Elements

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

The **shift()** method removes the first element of an array, and "shifts" all other elements one place up.

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_shift)

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_unshift)

The shift() method returns the string that was "shifted out".

The unshift() method returns the new array length.

### Changing Elements

Array elements are accessed using their **index number**:

#### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[0] = "Kiwi";        // Changes the first element of fruits to "Kiwi"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_change)

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_change_add)

### Deleting Elements

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_delete)

|  |  |
| --- | --- |
| **Note** | Using **delete** on array elements leaves undefined holes in the array. Use pop() or shift() instead. |

### Splicing an Array

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

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_splice)

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

### Using splice() to Remove Elements

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

#### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(0, 1);        // Removes the first element of fruits

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_remove)

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

### Sorting an Array

The **sort()** method sorts an array alphabetically:

#### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();            // Sorts the elements of fruits

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort)

### Reversing an Array

The **reverse()** method reverses the elements in an array.

You can use it to sort an array in descending order:

#### Example

var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();            // Sorts the elements of fruits   
fruits.reverse();         // Reverses the order of the elements

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_reverse)

### 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.

You can fix this by providing a **compare function**:

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort2)

Use the same trick to sort an array descending:

#### Example

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort3)

### The Compare Function

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.

**Example:**

When comparing 40 and 100, the sort() method calls the compare function(40,100).

The function calculates 40-100, and returns -60 (a negative value).

The sort function will sort 40 as a value lower than 100.

**Fin**d **the Highest (or Lowest) Value**

How to find the highest value in an array?

#### Example

var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b-a});  
// now points[0] contains the highest value

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_high)

And the lowest:

#### Example

var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a-b});  
// now points[0] contains the lowest value

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_low)

### Joining Arrays

The **concat()** method creates a new array by concatenating two arrays:

#### Example

var myGirls = ["Cecilie", "Lone"];  
var myBoys = ["Emil", "Tobias","Linus"];  
var myChildren = myGirls.concat(myBoys);     // Concatenates (joins) myGirls and myBoys

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat)

The concat() method can take any number of array arguments:

#### Example

var arr1 = ["Cecilie", "Lone"];  
var arr2 = ["Emil", "Tobias","Linus"];  
var arr3 = ["Robin", "Morgan"];  
var myChildren = arr1.concat(arr2, arr3);     // Concatenates arr1 with arr2 and arr3

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat2)

### Slicing an Array

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

#### Example

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var citrus = fruits.slice(1, 3);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice)

The slice() method selects elements starting at the start argument, and ends at, but does not include, the end argument.

If the end argument is omitted, the slice() method slices out the rest of the array:

#### Example

var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
var citrus = fruits.slice(1);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice2)

### Complete Array Reference

For a complete reference, go to our [Complete JavaScript Array Reference](http://www.w3schools.com/jsref/jsref_obj_array.asp).

The reference contains descriptions and examples of all Array properties and methods.

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arraysmet1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arraysmet2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arraysmet4)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arraysmet5)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_arraysmet6)

## JavaScript Booleans

A JavaScript Boolean represents one of two values: **true** or **false**.

### Boolean Values

Very often, in programming, you will need a data type that can only have one of two values, like

* YES / NO
* ON / OFF
* TRUE / FALSE

For this, JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

### The Boolean() Function

You can use the Boolean() function to find out if an expression (or a variable) is true:

#### Example

Boolean(10 > 9)        // returns true

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_expression1)

Or even easier:

#### Example

(10 > 9)              // also returns true  
10 > 9                // also returns true

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_expression2)

### Comparisons and Conditions

The chapter JS Comparisons gives a full overview of comparison operators.

The chapter JS Conditions gives a full overview of conditional statements.

Here are some examples:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | equal to | if (day == "Monday") |
| > | greater than | if (salary > 9000) |
| < | less than | if (age < 18) |

|  |  |
| --- | --- |
| **Note** | The Boolean value of an expression is the fundament for JavaScript comparisons and conditions. |

### Everything With a "Real" Value is True

#### Examples

100  
  
3.14  
  
-15  
  
"Hello"  
  
"false"  
  
7 + 1 + 3.14  
  
5 < 6

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean)

### Everything Without a "Real" is False

The Boolean value of **0** (zero) is **false**:

var x = 0;  
Boolean(x);       // returns false

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_zero)

The Boolean value of **-0** (minus zero) is **false**:

var x = -0;  
Boolean(x);       // returns false

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_minus)

The Boolean value of **""** (empty string) is **false**:

var x = "";  
Boolean(x);       // returns false

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_empty)

The Boolean value of **undefined** is **false**:

var x;  
Boolean(x);       // returns false

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_undefined)

The Boolean value of **null** is **false**:

var x = null;  
Boolean(x);       // returns false

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_null)

The Boolean value of **false** is (you guessed it) **false**:

var x = false;  
Boolean(x);       // returns false

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_false)

The Boolean value of **NaN** is **false**:

var x = 10 / "H";  
Boolean(x);       // returns false

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_nan)

### Boolean Properties and Methods

Primitive values, like true and false, 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.

### Complete Boolean Reference

For a complete reference, go to our [Complete JavaScript Boolean Reference](http://www.w3schools.com/jsref/jsref_obj_boolean.asp).

The reference contains descriptions and examples of all Boolean properties and methods.

## JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for *true* or *false*.

### Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x = 5**, the table below explains the comparison operators:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Comparing** | **Returns** | **Try it** |
| == | equal to | x == 8 | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison1) |
| x == 5 | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison2) |
| x == "5" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison12) |
| === | equal value and equal type | x === 5 | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison4) |
| x === "5" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison3) |
| != | not equal | x != 8 | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison5) |
| !== | not equal value or not equal type | x !== "5" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison6) |
| x !== 5 | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison7) |
| > | greater than | x > 8 | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison8) |
| < | less than | x < 8 | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison9) |
| >= | greater than or equal to | x >= 8 | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison10) |
| <= | less than or equal to | x <= 8 | *true* | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison11) |

### How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

if (age < 18) text = "Too young";

You will learn more about the use of conditional statements in the next chapter of this tutorial.

### Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that **x = 6** and **y = 3**, the table below explains the logical operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| && | and | (x < 10 && y > 1) is true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_and) |
| || | or | (x == 5 || y == 5) is false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_or) |
| ! | not | !(x == y) is true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_not) |

### Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

#### Syntax

*variablename* = (*condition*) ? *value1*:*value2*

#### Example

var voteable = (age < 18) ? "Too young":"Old enough";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison)

If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough".

### Comparing Different Types

Comparing data of different types may give unexpected results.

When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison. An empty string converts to 0. A non-numeric string converts to NaN which is always false.

|  |  |  |
| --- | --- | --- |
| **Case** | **Value** | **Try** |
| 2 < 12 | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_20) |
| 2 < "12" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_21) |
| 2 < "John" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_23) |
| 2 > "John" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_24) |
| 2 == "John" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_25) |
| "2" < "12" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_26) |
| "2" > "12" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_27) |
| "2" == "12" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_28) |

When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2.

To secure a proper result, variables should be converted to the proper type before comparison:

age = Number(age);  
if (isNaN(age)) {  
    voteable = "Error in input";  
} else {  
    voteable = (age < 18) ? "Too young" : "Old enough";  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison_12)

### JavaScript Bitwise Operators

Bit operators work on 32-bit numbers.

Any numeric operand in the operation is converted into a 32-bit number.

The result is converted back to a JavaScript number.

#### Example

x = 5 & 1;

The result in x:

1

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_bitwise1)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Operator** | | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | | AND | x = 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | | OR | x = 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | | NOT | x = ~ 5 | ~0101 | 1010 | 10 |
| ^ | | XOR | x = 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | | Left shift | x = 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | | Right shift | x = 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| **Note** | The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.  Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.  ~00000000000000000000000000000101 will return 11111111111111111111111111111010 | | | | | | |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_comparisons1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_comparisons2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_comparisons3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_comparisons4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_comparisons5)  [Exercise 6 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_comparisons6)

## JavaScript If...Else Statements

Conditional statements are used to perform different actions based on different conditions.

### Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

* Use **if** to specify a block of code to be executed, if a specified condition is true
* Use **else** to specify a block of code to be executed, if the same condition is false
* Use **else if** to specify a new condition to test, if the first condition is false
* Use **switch** to specify many alternative blocks of code to be executed

### The if Statement

Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

#### Syntax

if (*condition*) {  
*block of code to be executed if the condition is true*}

|  |  |
| --- | --- |
| **Note** | Note that **if** is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error. |

#### Example

Make a "Good day" greeting if the hour is less than 18:00:

if (hour < 18) {  
    greeting = "Good day";  
}

The result of greeting will be:

Good day

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_ifthen)

### The else Statement

Use the **else** statement to specify a block of code to be executed if the condition is false.

if (*condition*) {  
*block of code to be executed if the condition is true*} else {   
*block of code to be executed if the condition is false*}

#### Example

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

if (hour < 18) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}

The result of greeting will be:

Good day

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_ifthenelse)

### The else if Statement

Use the **else if** statement to specify a new condition if the first condition is false.

#### Syntax

if (*condition1*) {  
*block of code to be executed if condition1 is true*} else if (*condition2*) {  
*block of code to be executed if the condition1 is false and condition2 is true*  
} else {  
*block of code to be executed if the condition1 is false and condition2 is false*}

#### Example

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

if (time < 10) {  
    greeting = "Good morning";  
} else if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}

The result of greeting will be:

Good day

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_elseif)

Examples

### More Examples

[Random link](http://www.w3schools.com/js/tryit.asp?filename=tryjs_randomlink)  
This example will write a link to either W3Schools or to the World Wildlife Foundation (WWF). By using a random number, there is a 50% chance for each of the links.

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_conditions1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_conditions2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_conditions3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_conditions4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_conditions5)  [Exercise 6 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_conditions6)

## JavaScript Switch Statement

The switch statement is used to perform different actions based on different conditions.

### The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

#### Syntax

switch(*expression*) {  
    case *n*:  
*code block*        break;  
    case *n*:  
*code block*        break;  
    default:  
        *default code block*  
}

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.

#### Example

The getDay() method returns the weekday as a number between 0 and 6. (Sunday=0, Monday=1, Tuesday=2 ..)

Use the weekday number to calculate weekday name:

switch (new Date().getDay()) {  
    case 0:  
        day = "Sunday";  
        break;  
    case 1:  
        day = "Monday";  
        break;  
    case 2:  
        day = "Tuesday";  
        break;  
    case 3:  
        day = "Wednesday";  
        break;  
    case 4:  
        day = "Thursday";  
        break;  
    case 5:  
        day = "Friday";  
        break;  
    case 6:  
        day = "Saturday";  
        break;  
}

The result of day will be:

Wednesday

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_switch)

### The break Keyword

When the JavaScript code interpreter reaches a **break** keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

|  |  |
| --- | --- |
| **Note** | When a match is found, and the job is done, it's time for a break. There is no need for more testing. |

### The default Keyword

The **default** keyword specifies the code to run if there is no case match:

#### Example

The getDay() method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message:

switch (new Date().getDay()) {  
    case 6:  
        text = "Today is Saturday";  
        break;   
    case 0:  
        text = "Today is Sunday";  
        break;   
    default:   
        text = "Looking forward to the Weekend";  
}

The result of text will be:

Looking forward to the Weekend

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_switch2)

### Common Code and Fall-Through

Sometimes, in a switch block, you will want different cases to use the same code, or fall-through to a common default.

Note from the next example, that cases can share the same code block, and that the default case does not have to be the last case in a switch block:

#### Example

switch (new Date().getDay()) {  
    case 1:  
    case 2:  
    case 3:  
    default:   
        text = "Looking forward to the Weekend";  
        break;   
    case 4:  
    case 5:  
       text = "Soon it is Weekend";  
        break;   
    case 0:  
    case 6:  
       text = "It is Weekend";  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_switch3)

|  |  |
| --- | --- |
| **Note** | If default is not the last case in the switch block, remember to end it with a break. |

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_switch1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_switch2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_switch3)   [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_switch4)

## JavaScript For Loop

Loops can execute a block of code a number of times.

### JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

#### Instead of writing:

text += cars[0] + "<br>";   
text += cars[1] + "<br>";   
text += cars[2] + "<br>";   
text += cars[3] + "<br>";   
text += cars[4] + "<br>";   
text += cars[5] + "<br>";

#### You can write:

for (i = 0; i < cars.length; i++) {   
    text += cars[i] + "<br>";  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for)

### Different Kinds of Loops

JavaScript supports different kinds of loops:

* **for** - loops through a block of code a number of times
* **for/in** - loops through the properties of an object
* **while** - loops through a block of code while a specified condition is true
* **do/while** - also loops through a block of code while a specified condition is true

### The For Loop

The for loop is often the tool you will use when you want to create a loop.

The for loop has the following syntax:

for (*statement 1*; *statement 2*; *statement 3*) {  
    *code block to be executed*  
}

**Statement 1** is executed before the loop (the code block) starts.

**Statement 2** defines the condition for running the loop (the code block).

**Statement 3** is executed each time after the loop (the code block) has been executed.

#### Example

for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_ex)

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

### Statement 1

Normally you will use statement 1 to initiate the variable used in the loop (i = 0).

This is not always the case, JavaScript doesn't care. Statement 1 is optional.

You can initiate many values in statement 1 (separated by comma):

#### Example

for (i = 0, len = cars.length, text = ""; i < len; i++) {   
    text += cars[i] + "<br>";  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_om1)

And you can omit statement 1 (like when your values are set before the loop starts):

#### Example

var i = 2;  
var len = cars.length;  
var text = "";  
for (; i < len; i++) {   
    text += cars[i] + "<br>";  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_om2)

### Statement 2

Often statement 2 is used to evaluate the condition of the initial variable.

This is not always the case, JavaScript doesn't care. Statement 2 is also optional.

If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.

|  |  |
| --- | --- |
| **Note** | If you omit statement 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial. |

### Statement 3

Often statement 3 increases the initial variable.

This is not always the case, JavaScript doesn't care, and statement 3 is optional.

Statement 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Statement 3 can also be omitted (like when you increment your values inside the loop):

#### Example

var i = 0;  
var len = cars.length;  
for (; i < len; ) {   
    text += cars[i] + "<br>";  
    i++;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_om3)

### The For/In Loop

The JavaScript for/in statement loops through the properties of an object:

#### Example

var person = {fname:"John", lname:"Doe", age:25};   
  
var text = "";  
var x;  
for (x in person) {  
    text += person[x];  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_for_in)

### The While Loop

The while loop and the do/while loop will be explained in the next chapter.

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_for1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_for2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_for3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_for4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_for5)  [Exercise 6 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_for6)

## JavaScript While Loop

Loops can execute a block of code as long as a specified condition is true.

### The While Loop

The while loop loops through a block of code as long as a specified condition is true.

#### Syntax

while (*condition*) {  
*code block to be executed*  
}

#### Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

#### Example

while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_while)

|  |  |
| --- | --- |
| **Note** | If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser. |

### The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

#### Syntax

do {  
*code block to be executed*}  
while (*condition*);

#### Example

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

#### Example

do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dowhile)

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

### Comparing For and While

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a **for loop** to collect the car names from the cars array:

#### Example

var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
for (;cars[i];) {  
    text += cars[i] + "<br>";  
    i++;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_for_cars)

The loop in this example uses a **while loop** to collect the car names from the cars array:

#### Example

var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
while (cars[i]) {  
    text += cars[i] + "<br>";  
    i++;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loop_while_cars)

### Test Yourself with Exercises!

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_while1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_while2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_while3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_while4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_while5)

## JavaScript Break and Continue

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

### The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

The break statement can also be used to jump out of a loop.

The **break statement** breaks the loop and continues executing the code after the loop (if any):

#### Example

for (i = 0; i < 10; i++) {  
    if (i === 3) { break; }  
    text += "The number is " + i + "<br>";  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_break)

**The Continue Statement**

The **continue statement** breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of 3:

**Example**

for (i = 0; i < 10; i++) {  
    if (i === 3) { continue; }  
    text += "The number is " + i + "<br>";  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_continue)

**JavaScript Labels**

To label JavaScript statements you precede the statements with a label name and a colon:

label:  
statements

The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.

Syntax:

break *labelname*;   
  
continue *labelname*;

The continue statement (with or without a label reference) can only be used to **skip one loop iteration**.

The break statement, without a label reference, can only be used to **jump out of a loop or a switch**.

With a label reference, the break statement can be used to **jump out of any code block**:

**Example**

var cars = ["BMW", "Volvo", "Saab", "Ford"];  
list: {  
    text += cars[0] + "<br>";   
    text += cars[1] + "<br>";   
    text += cars[2] + "<br>";   
    text += cars[3] + "<br>";   
    break list;  
    text += cars[4] + "<br>";   
    text += cars[5] + "<br>";   
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_break_list)

|  |  |
| --- | --- |
| **Note** | A code block is a block of code between { and }. |

**Test Yourself with Exercises!**

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_break1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_break2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_break3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_break4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_break5)

**JavaScript Type Conversion**

[« Previous](http://www.w3schools.com/js/js_break.asp)

[Next Chapter »](http://www.w3schools.com/js/js_regexp.asp)

Number() converts to a Number, String() converts to a String, Boolean() converts to a Boolean.

**JavaScript Data Types**

In JavaScript there are 5 different data types that can contain values:

* string
* number
* boolean
* object
* function

There are 3 types of objects:

* Object
* Date
* Array

And 2 data types that cannot contain values:

* null
* undefined

**The typeof Operator**

You can use the **typeof** operator to find the data type of a JavaScript variable.

**Example**

typeof "John"                 // Returns string   
typeof 3.14                   // Returns number  
typeof NaN                    // Returns number  
typeof false                  // Returns boolean  
typeof [1,2,3,4]              // Returns object  
typeof {name:'John', age:34}  // Returns object  
typeof new Date()             // Returns object  
typeof function () {}         // Returns function  
typeof myCar                  // Returns undefined (if myCar is not declared)  
typeof null                   // Returns object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_typeof_all)

Please observe:

* The data type of NaN is number
* The data type of an array is object
* The data type of a date is object
* The data type of null is object
* The data type of an undefined variable is undefined

|  |  |
| --- | --- |
| **Note** | You cannot use **typeof** to determine if a JavaScript object is an array (or a date). |

**The Data Type of typeof**

The typeof operator is not a variable. It is an operator. Operators ( + - \* / ) do not have any data type.

But, the typeof operator always **returns a string** containing the type of the operand.

**The constructor Property**

The **constructor** property returns the constructor function for all JavaScript variables.

**Example**

"John".constructor                 // Returns function String()  { [native code] }  
(3.14).constructor                 // Returns function Number()  { [native code] }  
false.constructor                  // Returns function Boolean() { [native code] }  
[1,2,3,4].constructor              // Returns function Array()   { [native code] }  
{name:'John', age:34}.constructor  // Returns function Object()  { [native code] }  
new Date().constructor             // Returns function Date()    { [native code] }  
function () {}.constructor         // Returns function Function(){ [native code] }

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_constructor_all)

You can check the constructor property to find out if an object is an Array (contains the word "Array"):

**Example**

function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_isarray)

You can check the constructor property to find out if an object is a Date (contains the word "Date"):

**Example**

function isDate(myDate) {  
    return myDate.constructor.toString().indexOf("Date") > -1;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_isdate)

**JavaScript Type Conversion**

JavaScript variables can be converted to a new variable and another data type:

* By the use of a JavaScript function
* **Automatically** by JavaScript itself

**Converting Numbers to Strings**

The global method **String()** can convert numbers to strings.

It can be used on any type of numbers, literals, variables, or expressions:

**Example**

String(x)         // returns a string from a number variable x  
String(123)       // returns a string from a number literal 123  
String(100 + 23)  // returns a string from a number from an expression

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_string)

The Number method **toString()** does the same.

**Example**

x.toString()  
(123).toString()  
(100 + 23).toString()

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_tostring)

In the chapter [Number Methods](http://www.w3schools.com/js/js_number_methods.asp), you will find more methods that can be used to convert numbers to strings:

|  |  |
| --- | --- |
| **Method** | **Description** |
| toExponential() | Returns a string, with a number rounded and written using exponential notation. |
| toFixed() | Returns a string, with a number rounded and written with a specified number of decimals. |
| toPrecision() | Returns a string, with a number written with a specified length |

**Converting Booleans to Strings**

The global method **String()** can convert booleans to strings.

String(false)        // returns "false"  
String(true)         // returns "true"

The Boolean method **toString()** does the same.

false.toString()     // returns "false"  
true.toString()      // returns "true"

**Converting Dates to Strings**

The global method **String()** can convert dates to strings.

String(Date())      // returns Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

The Date method **toString()** does the same.

**Example**

Date().toString()   // returns Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)

In the chapter [Date Methods](http://www.w3schools.com/js/js_date_methods.asp), you will find more methods that can be used to convert dates to strings:

|  |  |
| --- | --- |
| **Method** | **Description** |
| getDate() | Get the day as a number (1-31) |
| getDay() | Get the weekday a number (0-6) |
| getFullYear() | Get the four digit year (yyyy) |
| getHours() | Get the hour (0-23) |
| getMilliseconds() | Get the milliseconds (0-999) |
| getMinutes() | Get the minutes (0-59) |
| getMonth() | Get the month (0-11) |
| getSeconds() | Get the seconds (0-59) |
| getTime() | Get the time (milliseconds since January 1, 1970) |

**Converting Strings to Numbers**

The global method **Number()** can convert strings to numbers.

Strings containing numbers (like "3.14") convert to numbers (like 3.14).

Empty strings convert to 0.

Anything else converts to NaN (Not a number).

Number("3.14")    // returns 3.14  
Number(" ")       // returns 0   
Number("")        // returns 0  
Number("99 88")   // returns NaN

In the chapter [Number Methods](http://www.w3schools.com/js/js_number_methods.asp), you will find more methods that can be used to convert strings to numbers:

|  |  |
| --- | --- |
| **Method** | **Description** |
| parseFloat() | Parses a string and returns a floating point number |
| parseInt() | Parses a string and returns an integer |

**The Unary + Operator**

The **unary + operator** can be used to convert a variable to a number:

**Example**

var y = "5";      // y is a string  
var x = + y;      // x is a number

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_typeof3)

If the variable cannot be converted, it will still become a number, but with the value NaN (Not a number):

**Example**

var y = "John";   // y is a string  
var x = + y;      // x is a number (NaN)

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_typeof4)

**Converting Booleans to Numbers**

The global method **Number()** can also convert booleans to numbers.

Number(false)     // returns 0  
Number(true)      // returns 1

**Converting Dates to Numbers**

The global method **Number()** can be used to convert dates to numbers.

d = new Date();  
Number(d)          // returns 1404568027739

The date method **getTime()** does the same.

d = new Date();  
d.getTime()        // returns 1404568027739

**Automatic Type Conversion**

When JavaScript tries to operate on a "wrong" data type, it will try to convert the value to a "right" type.

The result is not always what you expect:

5 + null    // returns 5         because null is converted to 0  
"5" + null  // returns "5null"   because null is converted to "null"  
"5" + 2     // returns 52        because 2 is converted to "2"  
"5" - 2     // returns 3         because "5" is converted to 5  
"5" \* "2"   // returns 10        because "5" and "2" are converted to 5 and 2

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_auto)

**Automatic String Conversion**

JavaScript automatically calls the variable's toString() function when you try to "output" an object or a variable:

document.getElementById("demo").innerHTML = myVar;  
  
// if myVar = {name:"Fjohn"}  // toString converts to "[object Object]"  
// if myVar = [1,2,3,4]       // toString converts to "1,2,3,4"  
// if myVar = new Date()      // toString converts to "Fri Jul 18 2014 09:08:55 GMT+0200"

Numbers and booleans are also converted, but this is not very visible:

// if myVar = 123             // toString converts to "123"  
// if myVar = true            // toString converts to "true"  
// if myVar = false           // toString converts to "false"

**JavaScript Type Conversion Table**

This table shows the result of converting different JavaScript values to Number, String, and Boolean:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Original Value** | **Converted to Number** | **Converted to String** | **Converted to Boolean** | **Try it** |
| false | 0 | "false" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_false) |
| true | 1 | "true" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_true) |
| 0 | 0 | "0" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_number_0) |
| 1 | 1 | "1" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_number_1) |
| "0" | 0 | "0" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_0) |
| "1" | 1 | "1" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_1) |
| NaN | NaN | "NaN" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_nan) |
| Infinity | Infinity | "Infinity" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_infinity) |
| -Infinity | -Infinity | "-Infinity" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_infinity_minus) |
| "" | 0 | "" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_empty) |
| "20" | 20 | "20" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_number) |
| "twenty" | NaN | "twenty" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_string_text) |
| [ ] | 0 | "" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_empty) |
| [20] | 20 | "20" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_one_number) |
| [10,20] | NaN | "10,20" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_two_numbers) |
| ["twenty"] | NaN | "twenty" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_one_string) |
| ["ten","twenty"] | NaN | "ten,twenty" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_array_two_strings) |
| function(){} | NaN | "function(){}" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_function) |
| { } | NaN | "[object Object]" | true | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_object) |
| null | 0 | "null" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_null) |
| undefined | NaN | "undefined" | false | [Try it »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_type_convert_undefined) |

Values in quotes indicate string values.

Red values indicate values (some) programmers might not expect.

**JavaScript Regular Expressions**

[« Previous](http://www.w3schools.com/js/js_type_conversion.asp)

[Next Chapter »](http://www.w3schools.com/js/js_errors.asp)

A regular expression is a sequence of characters that forms a search pattern.

The search pattern can be used for text search and text replace operations.

**What Is a Regular Expression?**

A regular expression is a sequence of characters that forms a **search pattern**.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

**Syntax**

/*pattern*/*modifiers*;

**Example**

var patt = /w3schools/i;

Example explained:

**/w3schools/i**  is a regular expression.

**w3schools**  is a pattern (to be used in a search).

**i**  is a modifier (modifies the search to be case-insensitive).

**Using String Methods**

In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().

**The search() method** uses an expression to search for a match, and returns the position of the match.

**The replace() method** returns a modified string where the pattern is replaced.

**Using String search() With a Regular Expression**

**Example**

Use a regular expression to do a case-insensitive search for "w3schools" in a string:

var str = "Visit W3Schools";  
var n = str.search(/w3schools/i);

The result in n will be:

6

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_search_regexp)

**Using String search() With String**

The search method will also accept a string as search argument. The string argument will be converted to a regular expression:

**Example**

Use a string to do a search for "W3schools" in a string:

var str = "Visit W3Schools!";  
var n = str.search("W3Schools");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_search)

**Use String replace() With a Regular Expression**

**Example**

Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:

var str = "Visit Microsoft!";  
var res = str.replace(/microsoft/i, "W3Schools");

The result in res will be:

Visit W3Schools!

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace_regexp)

**Using String replace() With a String**

The replace() method will also accept a string as search argument:

var str = "Visit Microsoft!";  
var res = str.replace("Microsoft", "W3Schools");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace)

**Did You Notice?**

|  |  |
| --- | --- |
| **Note** | Regular expression arguments (instead of string arguments) can be used in the methods above. Regular expressions can make your search much more powerful (case insensitive for example). |

**Regular Expression Modifiers**

**Modifiers** can be used to perform case-insensitive more global searches:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| i | Perform case-insensitive matching |
| g | Perform a global match (find all matches rather than stopping after the first match) |
| m | Perform multiline matching |

**Regular Expression Patterns**

**Brackets** are used to find a range of characters:

|  |  |
| --- | --- |
| **Expression** | **Description** |
| [abc] | Find any of the characters between the brackets |
| [0-9] | Find any of the digits between the brackets |
| (x|y) | Find any of the alternatives separated with | |

**Metacharacters** are characters with a special meaning:

|  |  |
| --- | --- |
| **Metacharacter** | **Description** |
| \d | Find a digit |
| \s | Find a whitespace character |
| \b | Find a match at the beginning or at the end of a word |
| \uxxxx | Find the Unicode character specified by the hexadecimal number xxxx |

**Quantifiers** define quantities:

|  |  |
| --- | --- |
| **Quantifier** | **Description** |
| n+ | Matches any string that contains at least one *n* |
| n\* | Matches any string that contains zero or more occurrences of *n* |
| n? | Matches any string that contains zero or one occurrences of *n* |

**Using the RegExp Object**

In JavaScript, the RegExp object is a regular expression object with predefined properties and methods.

**Using test()**

The test() method is a RegExp expression method.

It searches a string for a pattern, and returns true or false, depending on the result.

The following example searches a string for the character "e":

**Example**

var patt = /e/;  
patt.test("The best things in life are free!");

Since there is an "e" in the string, the output of the code above will be:

true

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_test)

You don't have to put the regular expression in a variable first. The two lines above can be shortened to one:

/e/.test("The best things in life are free!");

**Using exec()**

The exec() method is a RegExp expression method.

It searches a string for a specified pattern, and returns the found text.

If no match is found, it returns *null.*

The following example searches a string for the character "e":

**Example 1**

/e/.exec("The best things in life are free!");

Since there is an "e" in the string, the output of the code above will be:

e

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_regexp_exec)

**Complete RegExp Reference**

For a complete reference, go to our [Complete JavaScript RegExp Reference](http://www.w3schools.com/jsref/jsref_obj_regexp.asp).

The reference contains descriptions and examples of all RegExp properties and methods.

**JavaScript Errors - Throw and Try to Catch**

[« Previous](http://www.w3schools.com/js/js_regexp.asp)

[Next Chapter »](http://www.w3schools.com/js/js_debugging.asp)

The **try** statement lets you test a block of code for errors.

The **catch** statement lets you handle the error.

The **throw** statement lets you create custom errors.

The **finally** statement lets you execute code, after try and catch, regardless of the result.

**Errors Will Happen!**

When executing JavaScript code, different errors can occur.

Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things:

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<p id="demo"></p>  
  
<script>  
try {  
    adddlert("Welcome guest!");  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.message;  
}  
</script>  
  
</body>  
</html>

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_try_catch)

In the example above we have made a typo in the code (in the **try block**).

The **catch block** catches the error, and executes code to handle it.

**JavaScript try and catch**

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements **try** and **catch** come in pairs:

try {  
    *Block of code to try*}  
catch(err) {  
    *Block of code to handle errors*}

**JavaScript can Raise Errors**

When an error occurs, JavaScript will normally stop, and generate an error message.

The technical term for this is: JavaScript will  **raise (or throw) an exception**.

**The throw Statement**

The **throw** statement allows you to create a custom error.

Technically you can **raise (throw) an exception**.

The exception can be a JavaScript String, a Number, a Boolean or an Object:

throw "Too big";    // throw a text  
throw 500;          // throw a number

If you use **throw** together with **try** and **catch**, you can control program flow and generate custom error messages.

**Input Validation Example**

This example examines input. If the value is wrong, an exception (err) is thrown.

The exception (err) is caught by the catch statement and a custom error message is displayed:

<!DOCTYPE html>  
<html>  
<body>  
  
<p>Please input a number between 5 and 10:</p>  
  
<input id="demo" type="text">  
<button type="button" onclick="myFunction()">Test Input</button>  
<p id="message"></p>  
  
<script>  
function myFunction() {  
    var message, x;  
    message = document.getElementById("message");  
    message.innerHTML = "";  
    x = document.getElementById("demo").value;  
    try {   
        if(x == "") throw "empty";  
        if(isNaN(x)) throw "not a number";  
        x = Number(x);  
        if(x < 5) throw "too low";  
        if(x > 10) throw "too high";  
    }  
    catch(err) {  
        message.innerHTML = "Input is " + err;  
    }  
}  
</script>  
  
</body>  
</html>

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_throw_error)

**HTML Validation**

The code above is just an example.

Modern browser will often use a combination of JavaScript and built-in HTML validation, using predefined validation rules defined in HTML attributes:

<input id="demo" type="number" min="5" max="10" step="1"

You can read more about forms validation in a later chapter of this tutorial.

**The finally Statement**

The **finally** statement lets you execute code, after try and catch, regardless of the result:

try {  
    *Block of code to try*}  
catch(err) {  
    *Block of code to handle errors*}   
finally {  
    *Block of code to be executed regardless of the try / catch result*}

**Example**

function myFunction() {  
    var message, x;  
    message = document.getElementById("message");  
    message.innerHTML = "";  
    x = document.getElementById("demo").value;  
    try {   
        if(x == "") throw "is empty";  
        if(isNaN(x)) throw "is not a number";  
        x = Number(x);  
        if(x > 10) throw "is too high";  
        if(x < 5) throw "is too low";  
    }  
    catch(err) {  
        message.innerHTML = "Error: " + err + ".";  
    }  
    finally {  
        document.getElementById("demo").value = "";  
    }  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_finally_error)

**JavaScript Debugging**

[« Previous](http://www.w3schools.com/js/js_errors.asp)

[Next Chapter »](http://www.w3schools.com/js/js_hoisting.asp)

You will soon get lost, writing JavaScript code without a debugger.

**JavaScript Debugging**

It is difficult to write JavaScript code without a debugger.

Your code might contain syntax errors, or logical errors, that are difficult to diagnose.

Often, when JavaScript code contains errors, nothing will happen. There are no error messages, and you will get no indications where to search for errors.

|  |  |
| --- | --- |
| **Note** | Normally, errors will happen, every time you try to write some new JavaScript code. |

**JavaScript Debuggers**

Searching for errors in programming code is called code debugging.

Debugging is not easy. But fortunately, all modern browsers have a built-in debugger.

Built-in debuggers can be turned on and off, forcing errors to be reported to the user.

With a debugger, you can also set breakpoints (places where code execution can be stopped), and examine variables while the code is executing.

Normally, otherwise follow the steps at the bottom of this page, you activate debugging in your browser with the F12 key, and select "Console" in the debugger menu.

**The console.log() Method**

If your browser supports debugging, you can use console.log() to display JavaScript values in the debugger window:

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My First Web Page</h1>  
  
<script>  
a = 5;  
b = 6;  
c = a + b;  
console.log(c);  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_console)

**Setting Breakpoints**

In the debugger window, you can set breakpoints in the JavaScript code.

At each breakpoint, JavaScript will stop executing, and let you examine JavaScript values.

After examining values, you can resume the execution of code (typically with a play button).

**The debugger Keyword**

The **debugger** keyword stops the execution of JavaScript, and calls (if available) the debugging function.

This has the same function as setting a breakpoint in the debugger.

If no debugging is available, the debugger statement has no effect.

With the debugger turned on, this code will stop executing before it executes the third line.

**Example**

var x = 15 \* 5;  
debugger;  
document.getElementbyId("demo").innerHTML = x;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_debugger)

**Major Browsers' Debugging Tools**

Normally, you activate debugging in your browser with F12, and select "Console" in the debugger menu.

Otherwise follow these steps:

**Chrome**

* Open the browser.
* From the menu, select tools.
* From tools, choose developer tools.
* Finally, select Console.

**Firefox Firebug**

* Open the browser.
* Go to the web page:  
  http://www.getfirebug.com
* Follow the instructions how to:  
  install Firebug

**Internet Explorer**

* Open the browser.
* From the menu, select tools.
* From tools, choose developer tools.
* Finally, select Console.

**Opera**

* Open the browser.
* Go to the webpage:  
  http://dev.opera.com
* Follow the instructions how to:  
  add a Developer Console button to your toolbar.

**Safari Firebug**

* Open the browser.
* Go to the webpage:  
  http://extensions.apple.com
* Follow the instructions how to:  
  install Firebug Lite.

**Safari Develop Menu**

* Go to Safari, Preferences, Advanced in the main menu.
* Check "Enable Show Develop menu in menu bar".
* When the new option "Develop" appears in the menu:  
  Choose "Show Error Console".

**Did You Know?**

|  |  |
| --- | --- |
| **Note** | Debugging is the process of testing, finding, and reducing bugs (errors) in computer programs. The first known computer bug was a real bug (an insect), stuck in the electronics. |

**JavaScript Hoisting**

[« Previous](http://www.w3schools.com/js/js_debugging.asp)

[Next Chapter »](http://www.w3schools.com/js/js_strict.asp)

Hoisting is JavaScript's default behavior of moving declarations to the top.

**JavaScript Declarations are Hoisted**

In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared.

**Example 1** gives the same result as **Example 2**:

**Example 1**

x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x;                     // Display x in the element  
  
var x; // Declare x

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_hoisting1)

**Example 2**

var x; // Declare x  
x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x;                     // Display x in the element

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_hoisting2)

To understand this, you have to understand the term "hoisting".

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

**JavaScript Initializations are Not Hoisted**

JavaScript only hoists declarations, not initializations.

**Example 1** does **not** give the same result as **Example 2**:

**Example 1**

var x = 5; // Initialize x  
var y = 7; // Initialize y  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_hoisting4)

**Example 2**

var x = 5; // Initialize x  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y  
  
var y = 7; // Initialize y

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_hoisting3)

Does it make sense that y is undefined in the last example?

This is because only the declaration (var y), not the initialization (=7) is hoisted to the top.

Because of hoisting, y has been declared before it is used, but because initializations are not hoisted, the value of y is undefined.

Example 2 is the same as writing:

**Example**

var x = 5; // Initialize x  
var y;     // Declare y  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y  
  
y = 7;    // Assign 7 to y

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_hoisting5)

**Declare Your Variables At the Top !**

Hoisting is (to many developers) an unknown or overlooked behavior of JavaScript.

If a developer doesn't understand hoisting, programs may contain bugs (errors).

To avoid bugs, always declare all variables at the beginning of every scope.

Since this is how JavaScript interprets the code, it is always a good rule.

|  |  |
| --- | --- |
| **Note** | JavaScript in strict mode does not allow variables to be used if they are not declared. Study **"use strict"** in the next chapter. |

**JavaScript Use Strict**

[« Previous](http://www.w3schools.com/js/js_hoisting.asp)

[Next Chapter »](http://www.w3schools.com/js/js_conventions.asp)

**"use strict";**  Defines that JavaScript code should be executed in "strict mode".

**The "use strict" Directive**

The "use strict" directive is new in JavaScript 1.8.5 (ECMAScript version 5).

It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.

The purpose of "use strict" is to indicate that the code should be executed in "strict mode".

With strict mode, you can not, for example, use undeclared variables.

|  |  |
| --- | --- |
| **Note** | Strict mode is supported in: Internet Explorer from version 10. Firefox from version 4. Chrome from version 13. Safari from version 5.1. Opera from version 12. |

**Declaring Strict Mode**

Strict mode is declared by adding "use strict"; to the beginning of a JavaScript file, or a JavaScript function.

Declared at the beginning of a JavaScript file, it has global scope (all code will execute in strict mode):

**Example**

"use strict";  
x = 3.14;       // This will cause an error

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strict_variable)

**Example**

"use strict";  
myFunction();  
  
function myFunction() {  
    y = 3.14;   // This will also cause an error  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strict_global)

Declared inside a function, it has local scope (only the code inside the function is in strict mode):

x = 3.14;       // This will not cause an error.   
myFunction();  
  
function myFunction() {  
   "use strict";  
    y = 3.14;   // This will cause an error  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strict_local)

**The "use strict"; Syntax**

The syntax, for declaring strict mode, was designed to be compatible with older versions of JavaScript.

Compiling a numeric literal (4 + 5;) or a string literal ("John Doe";) in a JavaScript program has no side effects. It simply compiles to a non existing variable and dies.

So "use strict"; only matters to new compilers that "understand" the meaning of it.

**Why Strict Mode?**

Strict mode makes it easier to write "secure" JavaScript.

Strict mode changes previously accepted "bad syntax" into real errors.

As an example, in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.

In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties.

In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.

**Not Allowed in Strict Mode**

Using a variable (property or object) without declaring it, is not allowed:

"use strict";  
x = 3.14;                 // This will cause an error (if x has not been declared)

Deleting a variable, a function, or an argument, is not allowed.

"use strict";  
x = 3.14;  
delete x;                 // This will cause an error

Defining a property more than once, is not allowed:

"use strict";  
var x = {p1:10, p1:20};   // This will cause an error

Duplicating a parameter name is not allowed:

"use strict";  
function x(p1, p1) {};    // This will cause an error

Octal numeric literals and escape characters are not allowed:

"use strict";  
var x = 010;             // This will cause an error  
var y = \010;            // This will cause an error

Writing to a read-only property is not allowed:

"use strict";  
var obj = {};  
obj.defineProperty(obj, "x", {value:0, writable:false});  
  
obj.x = 3.14;            // This will cause an error

Writing to a get-only property is not allowed:

"use strict";  
var obj = {get x() {return 0} };  
  
obj.x = 3.14;            // This will cause an error

Deleting an undeletable property is not allowed:

"use strict";  
delete Object.prototype; // This will cause an error

The string "eval" cannot be used as a variable:

"use strict";  
var eval = 3.14;         // This will cause an error

The string "arguments" cannot be used as a variable:

"use strict";  
var arguments = 3.14;    // This will cause an error

The with statement is not allowed:

"use strict";  
with (Math){x = cos(2)}; // This will cause an error

For security reasons, eval() is not allowed to create variables in the scope from which it was called:

"use strict";  
eval ("var x = 2");  
alert (x)                // This will cause an error

In function calls like f(), the this value was the global object. In strict mode, it is now undefined.

Future reserved keywords are not allowed. These are:

* implements
* interface
* package
* private
* protected
* public
* static
* field

**Watch Out!**

|  |  |
| --- | --- |
| **Note** | The "use strict" directive is only recognized at the **beginning** of a script or a function. |

**JavaScript Style Guide and Coding Conventions**

[« Previous](http://www.w3schools.com/js/js_strict.asp)

[Next Chapter »](http://www.w3schools.com/js/js_best_practices.asp)

Always use the same coding conventions for all your JavaScript projects.

**JavaScript Coding Conventions**

Coding conventions are **style guidelines for programming**. They typically cover:

* Naming and declaration rules for variables and functions.
* Rules for the use of white space, indentation, and comments.
* Programming practices and principles

Coding conventions **secure quality**:

* Improves code readability
* Make code maintenance easier

Coding conventions can be documented rules for teams to follow, or just be your individual coding practice.

|  |  |
| --- | --- |
| **Note** | This page describes the general JavaScript code conventions used by W3Schools. You should also read the next chapter "Best Practices", and learn how to avoid coding pitfalls. |

**Variable Names**

At W3schools we use **camelCase** for identifier names (variables and functions).

All names start with a **letter**.

At the bottom of this page, you will find a wider discussion about naming rules.

firstName = "John";  
lastName = "Doe";  
  
price = 19.90;  
tax = 0.20;  
  
fullPrice = price + (price \* tax);

**Spaces Around Operators**

Always put spaces around operators ( = + - \* / ), and after commas:

**Examples:**

var x = y + z;  
var values = ["Volvo", "Saab", "Fiat"];

**Code Indentation**

Always use 4 spaces for indentation of code blocks:

**Functions:**

function toCelsius(fahrenheit) {  
    return (5 / 9) \* (fahrenheit - 32);  
}

|  |  |
| --- | --- |
| **Note** | Do not use tabs (tabulators) for indentation. Different editors interpret tabs differently. |

**Statement Rules**

General rules for simple statements:

* Always end a simple statement with a semicolon.

**Examples:**

var values = ["Volvo", "Saab", "Fiat"];  
  
var person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50,  
    eyeColor: "blue"  
};

General rules for complex (compound) statements:

* Put the opening bracket at the end of the first line.
* Use one space before the opening bracket.
* Put the closing bracket on a new line, without leading spaces.
* Do not end a complex statement with a semicolon.

**Functions:**

function toCelsius(fahrenheit) {  
    return (5 / 9) \* (fahrenheit - 32);  
}

**Loops:**

for (i = 0; i < 5; i++) {  
    x += i;  
}

**Conditionals:**

if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}

**Object Rules**

General rules for object definitions:

* Place the opening bracket on the same line as the object name.
* Use colon plus one space between each property and its value.
* Use quotes around string values, not around numeric values.
* Do not add a comma after the last property-value pair.
* Place the closing bracket on a new line, without leading spaces.
* Always end  an object definition with a semicolon.

**Example**

var person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50,  
    eyeColor: "blue"  
};

Short objects can be written compressed, on one line, using spaces only between properties, like this:

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

**Line Length < 80**

For readability, avoid 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 or a comma.

**Example**

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

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_line_break)

**Naming Conventions**

Always use the same naming convention for all your code. For example:

* Variable and function names written as **camelCase**
* Global variable written in **UPPERCASE**
* Constants (like PI) written in **UPPERCASE**

Should you use **hyp-hens**, **camelCase**, or **under\_scores** in variable names?

This is a question programmers often discuss. The answer depends on who you ask:

**Hyphens in HTML and CSS:**

HTML5 attributes can start with data- (data-quantity, data-price).

CSS uses hyphens in property-names (font-size).

|  |  |
| --- | --- |
| **Note** | Hyphens can be mistaken as subtraction attempts. Hyphens are not allowed in JavaScript names. |

**Underscores:**

Many programmers prefer to use underscores (date\_of\_birth), especially in SQL databases.

Underscores are often used in PHP documentation.

**PascalCase:**

PascalCase is often preferred by C programmers.

**camelCase:**

camelCase is used by JavaScript itself, by jQuery, and other JavaScript libraries.

|  |  |
| --- | --- |
| **Note** | Don't start names with a $ sign. It will put you in conflict with many JavaScript library names. |

**Loading JavaScript in HTML**

Use simple syntax for loading external scripts (the type attribute is not necessary):

<script src="myscript.js">

**Accessing HTML Elements**

A consequence of using "untidy" HTML styles, might result in JavaScript errors.

These two JavaScript statements will produce different results:

var obj = getElementById("Demo")  
  
var obj = getElementById("demo")

If possible, use the same naming convention (as JavaScript) in HTML.

[Visit the HTML Style Guide](http://www.w3schools.com/html/html5_syntax.asp).

**File Extensions**

HTML files should have a **.html** extension (not **.htm**).

CSS files should have a **.css** extension.

JavaScript files should have a **.js** extension.

**Use Lower Case File Names**

Most web servers (Apache, Unix) are case sensitive about file names:

london.jpg cannot be accessed as London.jpg.

Other web servers (Microsoft, IIS) are not case sensitive:

london.jpg can be accessed as London.jpg or london.jpg.

If you use a mix of upper and lower case, you have to be extremely consistent.

If you move from a case insensitive, to a case sensitive server, even small errors can break your web site.

To avoid these problems, always use lower case file names (if possible).

**Performance**

Coding conventions are not used by computers. Most rules have little impact on the execution of programs.

Indentation and extra spaces are not significant in small scripts.

For code in development, readability should be preferred. Larger production scripts should be minified.

**JavaScript Best Practices**

[« Previous](http://www.w3schools.com/js/js_conventions.asp)

[Next Chapter »](http://www.w3schools.com/js/js_mistakes.asp)

Avoid global variables,  avoid new,  avoid  ==,  avoid eval()

**Avoid Global Variables**

Minimize the use of global variables.

This includes all data types, objects, and functions.

Global variables and functions can be overwritten by other scripts.

Use local variables instead, and learn how to use [closures](http://www.w3schools.com/js/js_function_closures.asp).

**Always Declare Local Variables**

All variables used in a function should be declared as **local** variables.

Local variables **must** be declared with the **var** keyword, otherwise they will become global variables.

|  |  |
| --- | --- |
| **Note** | Strict mode does not allow undeclared variables. |

**Declarations on Top**

It is a good coding practice to put all declarations at the top of each script or function.

This will:

* Give cleaner code
* Provide a single place to look for local variables
* Make it easier to avoid unwanted (implied) global variables
* Reduce the possibility of unwanted re-declarations

// Declare at the beginning  
var firstName, lastName, price, discount, fullPrice;  
  
// Use later  
firstName = "John";  
lastName = "Doe";  
  
price = 19.90;  
discount = 0.10;  
  
fullPrice = price \* 100 / discount;

This also goes for loop variables:

// Declare at the beginning  
var i;  
  
// Use later  
for (i = 0; i < 5; i++) {

|  |  |
| --- | --- |
| **Note** | By default, JavaScript moves all declarations to the top (JavaScript hoisting). |

**Initialize Variables**

It is a good coding practice to initialize variables when you declare them.

This will:

* Give cleaner code
* Provide a single place to initialize variables
* Avoid undefined values

// Declare and initiate at the beginning  
var firstName = "",  
    lastName = "",  
    price = 0,  
    discount = 0,  
    fullPrice = 0,  
    myArray = [],  
    myObject = {};

|  |  |
| --- | --- |
| **Note** | Initializing variables provides an idea of the intended use (and intended data type). |

**Never Declare Number, String, or Boolean Objects**

Always treat numbers, strings, or booleans as primitive values. Not as objects.

Declaring these types as objects, slows down execution speed, and produces nasty side effects:

**Example**

var x = "John";               
var y = new String("John");  
(x === y) // is false because x is a string and y is an object.

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_best_object_string1)

Or even worse:

**Example**

var x = new String("John");               
var y = new String("John");  
(x == y) // is false because you cannot compare objects.

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_best_object_string2)

**Don't Use new Object()**

* Use {} instead of new Object()
* Use "" instead of new String()
* Use 0 instead of new Number()
* Use false instead of new Boolean()
* Use [] instead of new Array()
* Use /()/ instead of new RegExp()
* Use function (){} instead of new function()

**Example**

var x1 = {};           // new object  
var x2 = "";           // new primitive string  
var x3 = 0;            // new primitive number  
var x4 = false;        // new primitive boolean  
var x5 = [];           // new array object  
var x6 = /()/;         // new regexp object  
var x7 = function(){}; // new function object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_best_constructors)

**Beware of Automatic Type Conversions**

Beware that numbers can accidentally be converted to strings or NaN (Not a Number).

JavaScript is loosely typed. A variable can contain different data types, and a variable can change its data type:

**Example**

var x = "Hello";     // typeof x is a string  
x = 5;               // changes typeof x to a number

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_best_typeof)

When doing mathematical operations, JavaScript can convert numbers to strings:

**Example**

var x = 5 + 7;       // x.valueOf() is 12,  typeof x is a number  
var x = 5 + "7";     // x.valueOf() is 57,  typeof x is a string  
var x = "5" + 7;     // x.valueOf() is 57,  typeof x is a string  
var x = 5 - 7;       // x.valueOf() is -2,  typeof x is a number  
var x = 5 - "7";     // x.valueOf() is -2,  typeof x is a number  
var x = "5" - 7;     // x.valueOf() is -2,  typeof x is a number  
var x = 5 - "x";     // x.valueOf() is NaN, typeof x is a number

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_best_valueof)

Subtracting a string from a string, does not generate an error but returns NaN (Not a Number):

**Example**

"Hello" - "Dolly"    // returns NaN

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_best_string_op1)

**Use === Comparison**

The == comparison operator always converts (to matching types) before comparison.

The === operator forces comparison of values and type:

**Example**

0 == "";        // true  
1 == "1";       // true  
1 == true;      // true  
  
0 === "";       // false  
1 === "1";      // false  
1 === true;     // false

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_best_comparison)

**Use Parameter Defaults**

If a function is called with a missing argument, the value of the missing argument is set to **undefined**.

Undefined values can break your code. It is a good habit to assign default values to arguments.

**Example**

function myFunction(x, y) {  
    if (y === undefined) {  
        y = 0;  
    }  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_best_parameter_default)

Read more about function parameters and arguments at [Function Parameters](http://www.w3schools.com/js/js_function_parameters.asp)

**End Your Switches with Defaults**

Always end your switch statements with a default. Even if you think there is no need for it.

**Example**

switch (new Date().getDay()) {  
    case 0:  
        day = "Sunday";  
        break;  
    case 1:  
        day = "Monday";  
        break;  
    case 2:  
        day = "Tuesday";  
        break;  
    case 3:  
        day = "Wednesday";  
        break;  
    case 4:  
        day = "Thursday";  
        break;  
    case 5:  
        day = "Friday";  
        break;  
    case 6:  
        day = "Saturday";  
        break;  
    default:  
        day = "Unknown";  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_break_switch)

**Avoid Using eval()**

The eval() function is used to run text as code. In almost all cases, it should not be necessary to use it.

Because it allows arbitrary code to be run, it also represents a security problem.

**JavaScript Common Mistakes**

[« Previous](http://www.w3schools.com/js/js_best_practices.asp)

[Next Chapter »](http://www.w3schools.com/js/js_performance.asp)

This chapter points out some common JavaScript mistakes.

**Accidentally Using the Assignment Operator**

JavaScript programs may generate unexpected results if a programmer accidentally uses an assignment operator (=), instead of a comparison operator (==) in an if statement.

This **if** statement returns **false** (as expected) because x is not equal to 10:

var x = 0;  
if (x == 10)

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_equal_1)

This **if** statement returns **true** (maybe not as expected), because 10 is true:

var x = 0;  
if (x = 10)

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_equal_2)

This **if** statement returns **false** (maybe not as expected), because 0 is false:

var x = 0;  
if (x = 0)

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_equal_3)

|  |  |
| --- | --- |
| **Note** | An assignment always returns the value of the assignment. |

**Expecting Loose Comparison**

In regular comparison, data type does not matter. This if statement returns true:

var x = 10;  
var y = "10";  
if (x == y)

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_loose_1)

In strict comparison, data type does matter. This if statement returns false:

var x = 10;  
var y = "10";  
if (x === y)

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_loose_2)

It is a common mistake to forget that switch statements use strict comparison:

This case switch will display an alert:

var x = 10;  
switch(x) {  
    case 10: alert("Hello");  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_loose_3)

This case switch will not display an alert:

var x = 10;  
switch(x) {  
    case "10": alert("Hello");  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_loose_4)

**Confusing Addition & Concatenation**

**Addition** is about adding **numbers**.

**Concatenation** is about adding **strings**.

In JavaScript both operations use the same + operator.

Because of this, adding a number as a number will produce a different result from adding a number as a string:

var x = 10 + 5;          // the result in x is 15  
var x = 10 + "5";        // the result in x is "105"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_add_1)

When adding two variables, it can be difficult to anticipate the result:

var x = 10;  
var y = 5;  
var z = x + y;           // the result in z is 15  
  
var x = 10;  
var y = "5";  
var z = x + y;           // the result in z is "105"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_add_2)

**Misunderstanding Floats**

All numbers in JavaScript are stored as 64-bits **Floating point numbers** (Floats).

All programming languages, including JavaScript, have difficulties with precise floating point values:

var x = 0.1;  
var y = 0.2;  
var z = x + y            // the result in z will not be 0.3  
if (z == 0.3)            // this if test will fail

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_floats)

To solve the problem above, it helps to multiply and divide:

**Example**

var z = (x \* 10 + y \* 10) / 10;       // z will be 0.3

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_floats_ok)

**Breaking a JavaScript String**

JavaScript will allow you to break a statement into two lines:

**Example 1**

var x =  
"Hello World!";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_string_1)

But, breaking a statement in the middle of a string will not work:

**Example 2**

var x = "Hello  
World!";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_string_2)

You must use a "backslash" if you must break a statement in a string:

**Example 3**

var x = "Hello \  
World!";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_string_3)

**Misplacing Semicolon**

Because of a misplaced semicolon, this code block will execute regardless of the value of x:

if (x == 19);  
{  
    // code block    
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_semicolon)

**Breaking a Return Statement**

It is a default JavaScript behavior to close a statement automatically at the end of a line.

Because of this, these two examples will return the same result:

**Example 1**

function myFunction(a) {  
    var power = 10    
    return a \* power  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_return_1)

**Example 2**

function myFunction(a) {  
    var power = 10;  
    return a \* power;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_return_2)

JavaScript will also allow you to break a statement into two lines.

Because of this, example 3 will also return the same result:

**Example 3**

function myFunction(a) {  
    var  
    power = 10;    
    return a \* power;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_return_3)

But, what will happen if you break the return statement in two lines like this:

**Example 4**

function myFunction(a) {  
    var  
    power = 10;    
    return  
    a \* power;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_return_4)

The function will return undefined!

Why? Because JavaScript thinks you meant:

**Example 5**

function myFunction(a) {  
    var  
    power = 10;    
    return;  
    a \* power;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_return_5)

**Explanation**

If a statement is incomplete like:

var

JavaScript will try to complete the statement by reading the next line:

power = 10;

But since this statement is complete:

return

JavaScript will automatically close it like this:

return;

This happens because closing (ending) statements with semicolon is optional in JavaScript.

JavaScript will close the return statement at the end of the line, because it is a complete statement.

|  |  |
| --- | --- |
| **Note** | Never break a return statement. |

**Accessing Arrays with Named Indexes**

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** use **numbered indexes**:

**Example**

var person = [];  
person[0] = "John";  
person[1] = "Doe";  
person[2] = 46;  
var x = person.length;         // person.length will return 3  
var y = person[0];             // person[0] will return "John"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_1)

In JavaScript, **objects** use **named indexes**.

If you use a named index, when accessing an array, JavaScript will redefine the array to a standard object.

After the automatic redefinition, array methods and properties will produce undefined or incorrect results:

**Example:**

var person = [];  
person["firstName"] = "John";  
person["lastName"] = "Doe";  
person["age"] = 46;  
var x = person.length;         // person.length will return 0  
var y = person[0];             // person[0] will return undefined

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_associative_2)

**Ending an Array Definition with a Comma**

**Incorrect:**

points = [40, 100, 1, 5, 25, 10,];

Some JSON and JavaScript engines will fail, or behave unexpectedly.

**Correct:**

points = [40, 100, 1, 5, 25, 10];

**Ending an Object Definition with a Comma**

**Incorrect:**

person = {firstName:"John", lastName:"Doe", age:46,}

Some JSON and JavaScript engines will fail, or behave unexpectedly.

**Correct:**

person = {firstName:"John", lastName:"Doe", age:46}

**Undefined is Not Null**

With JavaScript, **null** is for objects, **undefined** is for variables, properties, and methods.

To be null, an object has to be defined, otherwise it will be undefined.

If you want to test if an object exists, this will throw an error if the object is undefined:

**Incorrect:**

if (myObj !== null && typeof myObj !== "undefined")

Because of this, you must test typeof() first:

**Correct:**

if (typeof myObj !== "undefined" && myObj !== null)

**Expecting Block Level Scope**

JavaScript **does not** create a new scope for each code block.

It is true in many programming languages, but **not true** in JavaScript.

It is a common mistake, among new JavaScript developers, to believe that this code returns undefined:

**Example**

for (var i = 0; i < 10; i++) {  
    // some code  
}  
return i;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_mistakes_scope)

**JavaScript Performance**

[« Previous](http://www.w3schools.com/js/js_mistakes.asp)

[Next Chapter »](http://www.w3schools.com/js/js_reserved.asp)

How to speed up your JavaScript code.

**Reduce Activity in Loops**

Loops are often used in programming.

Each statement in a loop, including the for statement, is executed for each iteration of the loop.

Search for statements or assignments that can be placed outside the loop.

**Bad Code:**

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

**Better Code:**

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

The bad code accesses the length property of an array each time the loop is iterated.

The better code accesses the length property outside the loop, and makes the loop run faster.

**Reduce DOM Access**

Accessing the HTML DOM is very slow, compared to other JavaScript statements.

If you expect to access a DOM element several times, access it once, and use it as a local variable:

**Example**

obj = document.getElementById("demo");  
obj.innerHTML = "Hello";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_element_reference)

**Reduce DOM Size**

Keep the number of elements in the HTML DOM small.

This will always improve page loading, and speed up rendering (page display), especially on smaller devices.

Every attempt to search the DOM (like getElementsByTagName) will benefit from a smaller DOM.

**Avoid Unnecessary Variables**

Don't create new variables if you don't plan to save values.

Often you can replace code like this:

var fullName = firstName + " " + lastName;  
document.getElementById("demo").innerHTML = fullName;

With this:

document.getElementById("demo").innerHTML = firstName + " " + lastName

**Delay JavaScript Loading**

Putting your scripts at the bottom of the page body, lets the browser load the page first.

While a script is downloading, the browser will not start any other downloads. In addition all parsing and rendering activity might be blocked.

|  |  |
| --- | --- |
| **Note** | The HTTP specification defines that browsers should not download more than two components in parallel. |

An alternative is to use **defer="true"** in the script tag. The defer attribute specifies that the script should be executed after the page has finished parsing, but it only works for external scripts.

If possible, you can add your script to the page by code, after the page has loaded:

**Example**

<script>  
window.onload = downScripts;  
  
function downScripts() {  
    var element = document.createElement("script");  
    element.src = "myScript.js";  
    document.body.appendChild(element);  
}  
</script>

**Avoid Using with**

Avoid using the **with keyword**. It has a negative effect on speed. It also clutters up JavaScript scopes.

The with keyword is **not allowed** in strict mode.

**JavaScript Reserved Words**

[« Previous](http://www.w3schools.com/js/js_performance.asp)

[Next Chapter »](http://www.w3schools.com/js/js_json.asp)

In JavaScript, some identifiers are reserved words and cannot be used as variables or function names.

**JavaScript Standards**

ECMAScript 3 (ES3) was released in December 1999.

ECMAScript 4 (ES4) was abandoned.

ECMAScript 5 (ES5) was released in December 2009.

ECMAScript 6 (ES6) was released in June 2015, and is the latest official version of JavaScript.

Time passes, and we are now beginning to see complete support for ES5/ES6 in all modern browsers.

**JavaScript Reserved Words**

In JavaScript you cannot use these reserved words as variables, labels, or function names:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | arguments | boolean | break | byte |
| case | catch | char | class\* | const |
| continue | debugger | default | delete | do |
| double | else | enum\* | eval | export\* |
| extends\* | false | final | finally | float |
| for | function | goto | if | implements |
| import\* | in | instanceof | int | interface |
| let | long | native | new | null |
| package | private | protected | public | return |
| short | static | super\* | switch | synchronized |
| this | throw | throws | transient | true |
| try | typeof | var | void | volatile |
| while | with | yield |  |  |

Words marked with\* are new in ECMAScript5

**JavaScript Objects, Properties, and Methods**

You should also avoid using the name of JavaScript built-in objects, properties, and methods:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | Date | eval | function | hasOwnProperty |
| Infinity | isFinite | isNaN | isPrototypeOf | length |
| Math | NaN | name | Number | Object |
| prototype | String | toString | undefined | valueOf |

**Java Reserved Words**

JavaScript is often used together with Java. You should avoid using some Java objects and properties as JavaScript identifiers:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| getClass | java | JavaArray | javaClass | JavaObject | JavaPackage |

**Windows Reserved Words**

JavaScript can be used outside HTML. It can be used as the programming language in many other applications.

In HTML you must (for portability you should) avoid using the name of HTML and Windows objects and properties:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| alert | all | anchor | anchors | area |
| assign | blur | button | checkbox | clearInterval |
| clearTimeout | clientInformation | close | closed | confirm |
| constructor | crypto | decodeURI | decodeURIComponent | defaultStatus |
| document | element | elements | embed | embeds |
| encodeURI | encodeURIComponent | escape | event | fileUpload |
| focus | form | forms | frame | innerHeight |
| innerWidth | layer | layers | link | location |
| mimeTypes | navigate | navigator | frames | frameRate |
| hidden | history | image | images | offscreenBuffering |
| open | opener | option | outerHeight | outerWidth |
| packages | pageXOffset | pageYOffset | parent | parseFloat |
| parseInt | password | pkcs11 | plugin | prompt |
| propertyIsEnum | radio | reset | screenX | screenY |
| scroll | secure | select | self | setInterval |
| setTimeout | status | submit | taint | text |
| textarea | top | unescape | untaint | window |

**HTML Event Handlers**

In addition you should avoid using the name of all HTML event handlers.

Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| onblur | onclick | onerror | onfocus |
| onkeydown | onkeypress | onkeyup | onmouseover |
| onload | onmouseup | onmousedown | onsubmit |

**JavaScript JSON**

[« Previous](http://www.w3schools.com/js/js_reserved.asp)

[Next Chapter »](http://www.w3schools.com/js/js_validation.asp)

JSON is a format for storing and transporting data.

JSON is often used when data is sent from a server to a web page.

**What is JSON?**

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is lightweight data interchange format
* JSON is language independent **\***
* JSON is "self-describing" and easy to understand

\* The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

**JSON Example**

This JSON syntax defines an employees object: an array of 3 employee records (objects):

**JSON Example**

{  
"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},  
    {"firstName":"Peter", "lastName":"Jones"}  
]  
}

**The JSON Format Evaluates to JavaScript Objects**

The JSON format is syntactically identical to the code for creating JavaScript objects.

Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

**JSON Syntax Rules**

* Data is in name/value pairs
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

**JSON Data - A Name and a Value**

JSON data is written as name/value pairs, just like JavaScript object properties.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"firstName":"John"

|  |  |
| --- | --- |
| **Note** | JSON names require double quotes. JavaScript names don't. |

**JSON Objects**

JSON objects are written inside curly braces.

Just like in JavaScript, objects can contain multiple name/value pairs:

{"firstName":"John", "lastName":"Doe"}

**JSON Arrays**

JSON arrays are written inside square brackets.

Just like in JavaScript, an array can contain objects:

"employees":[  
    {"firstName":"John", "lastName":"Doe"},   
    {"firstName":"Anna", "lastName":"Smith"},   
    {"firstName":"Peter", "lastName":"Jones"}  
]

In the example above, the object "employees" is an array. It contains three objects.

Each object is a record of a person (with a first name and a last name).

**Converting a JSON Text to a JavaScript Object**

A common use of JSON is to read data from a web server, and display the data in a web page.

For simplicity, this can be demonstrated using a string as input (or read more in our [JSON tutorial](http://www.w3schools.com/json/default.asp)):

First, create a JavaScript string containing JSON syntax:

var text = '{ "employees" : [' +  
'{ "firstName":"John" , "lastName":"Doe" },' +  
'{ "firstName":"Anna" , "lastName":"Smith" },' +  
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';

Then, use the JavaScript built-in function JSON.parse() to convert the string into a JavaScript object:

var obj = JSON.parse(text);

Finally, use the new JavaScript object in your page:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML =  
obj.employees[1].firstName + " " + obj.employees[1].lastName;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_json_parse)

You can read more about JSON in our [JSON tutorial](http://www.w3schools.com/json/default.asp).

**JavaScript Validation API**

[« Previous](http://www.w3schools.com/js/js_validation.asp)

[Next Chapter »](http://www.w3schools.com/js/js_object_definition.asp)

**Constraint Validation DOM Methods**

|  |  |
| --- | --- |
| **Property** | **Description** |
| checkValidity() | Returns true if an input element contains valid data. |
| setCustomValidity() | Sets the validationMessage property of an input element. |

If an input field contains invalid data, display a message:

**The checkValidity() Method**

<input id="id1" type="number" min="100" max="300">  
<button onclick="myFunction()">OK</button>  
  
<p id="demo"></p>  
  
<script>  
function myFunction() {  
    var inpObj = document.getElementById("id1");  
    if (inpObj.checkValidity() == false) {  
        document.getElementById("demo").innerHTML = inpObj.validationMessage;  
    }  
}  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_validation_check)

**Constraint Validation DOM Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| validity | Contains boolean properties related to the validity of an input element. |
| validationMessage | Contains the message a browser will display when the validity is false. |
| willValidate | Indicates if an input element will be validated. |

**Validity Properties**

The **validity property** of an input element contains a number of properties related to the validity of data:

|  |  |
| --- | --- |
| **Property** | **Description** |
| customError | Set to true, if a custom validity message is set. |
| patternMismatch | Set to true, if an element's value does not match its pattern attribute. |
| rangeOverflow | Set to true, if an element's value is greater than its max attribute. |
| rangeUnderflow | Set to true, if an element's value is less than its min attribute. |
| stepMismatch | Set to true, if an element's value is invalid per its step attribute. |
| tooLong | Set to true, if an element's value exceeds its maxLength attribute. |
| typeMismatch | Set to true, if an element's value is invalid per its type attribute. |
| valueMissing | Set to true, if an element (with a required attribute) has no value. |
| valid | Set to true, if an element's value is valid. |

**Examples**

If the number in an input field is greater than 100 (the input's max attribute), display a message:

**The rangeOverflow Property**

<input id="id1" type="number" max="100">  
<button onclick="myFunction()">OK</button>  
  
<p id="demo"></p>  
  
<script>  
function myFunction() {  
    var txt = "";  
    if (document.getElementById("id1").validity.rangeOverflow) {  
       txt = "Value too large";  
    }  
    document.getElementById("demo").innerHTML = txt;  
}  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_validation_rangeOverflow)

If the number in an input field is less than 100 (the input's min attribute), display a message:

**The rangeUnderflow Property**

<input id="id1" type="number" min="100">  
<button onclick="myFunction()">OK</button>  
  
<p id="demo"></p>  
  
<script>  
function myFunction() {  
    var txt = "";  
    if (document.getElementById("id1").validity.rangeUnderflow) {  
       txt = "Value too small";  
    }  
    document.getElementById("demo").innerHTML = txt;  
}  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_validation_rangeUnderflow)

**JavaScript Objects**

[« Previous](http://www.w3schools.com/js/js_validation_api.asp)

[Next Chapter »](http://www.w3schools.com/js/js_properties.asp)

**JavaScript Objects**

|  |  |
| --- | --- |
| **Note** | In JavaScript, objects are king. If you understand objects, you understand JavaScript. |

In JavaScript, almost "everything" is an object.

* Booleans can be objects (or primitive data treated as objects)
* Numbers can be objects (or primitive data treated as objects)
* Strings can be objects (or primitive data treated as objects)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects
* Arrays are always objects
* Functions are always objects
* Objects are objects

In JavaScript, all values, except primitive values, are objects.

Primitive values are: strings ("John Doe"), numbers (3.14), true, false, null, and undefined.

**Objects are Variables Containing Variables**

JavaScript variables can contain single values:

**Example**

var person = "John Doe";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_variable)

Objects are variables too. But objects can contain many values.

The values are written as **name : value** pairs (name and value separated by a colon).

**Example**

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_object)

|  |  |
| --- | --- |
| **Note** | A JavaScript object is an unordered collection of variables called **named values**. |

**Object Properties**

The named values, in JavaScript objects, are called **properties**.

|  |  |
| --- | --- |
| **Property** | **Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |

Objects written as name value pairs are similar to:

* Associative arrays in PHP
* Dictionaries in Python
* Hash tables in C
* Hash maps in Java
* Hashes in Ruby and Perl

**Object Methods**

Methods are **actions** that can be performed on objects.

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.

|  |  |
| --- | --- |
| **Property** | **Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |
| fullName | function() {return this.firstName + " " + this.lastName;} |

|  |  |
| --- | --- |
| **Note** | JavaScript objects are containers for named values, called properties and methods. |

You will learn more about methods in the next chapters.

**Creating a JavaScript Object**

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

* Define and create a single object, using an object literal.
* Define and create a single object, with the keyword new.
* Define an object constructor, and then create objects of the constructed type.

|  |  |
| --- | --- |
| **Note** | In ECMAScript 5, an object can also be created with the function Object.create(). |

**Using an Object Literal**

This is the easiest way to create a JavaScript Object.

Using an object literal, you both define and create an object in one statement.

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

**Example**

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_create_1)

Spaces and line breaks are not important. An object definition can span multiple lines:

**Example**

var person = {  
    firstName:"John",  
    lastName:"Doe",  
    age:50,  
    eyeColor:"blue"  
};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_create_2)

**Using the JavaScript Keyword new**

The following example also creates a new JavaScript object with four properties:

**Example**

var person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_create_new)

|  |  |
| --- | --- |
| **Note** | The two examples above do exactly the same. There is no need to use new Object(). For simplicity, readability and execution speed, use the first one (the object literal method). |

**Using an Object Constructor**

The examples above are limited in many situations. They only create a single object.

Sometimes we like to have an "object type" that can be used to create many objects of one type.

The standard way to create an "object type" is to use an object constructor function:

**Example**

function person(first, last, age, eye) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eye;  
}  
var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor)

The above function (person) is an object constructor.

Once you have an object constructor, you can create new objects of the same type:

var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");

**The *this* Keyword**

In JavaScript, the thing called **this**, is the object that "owns" the JavaScript code.

The value of **this**, when used in a function, is the object that "owns" the function.

The value of **this**, when used in an object, is the object itself.

The **this** keyword in an object constructor does not have a value. It is only a substitute for the new object.

The value of **this** will become the new object when the constructor is used to create an object.

|  |  |
| --- | --- |
| **Note** | Note that **this** is not a variable. It is a keyword. You cannot change the value of **this**. |

**Built-in JavaScript Constructors**

JavaScript has built-in constructors for native objects:

**Example**

var x1 = new Object();    // A new Object object  
var x2 = new String();    // A new String object  
var x3 = new Number();    // A new Number object  
var x4 = new Boolean()    // A new Boolean object  
var x5 = new Array();     // A new Array object  
var x6 = new RegExp();    // A new RegExp object  
var x7 = new Function();  // A new Function object  
var x8 = new Date();      // A new Date object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_builtin)

The Math() object is not in the list. Math is a global object. The new keyword cannot be used on Math.

|  |  |
| --- | --- |
| **Note** | **Did You Know?** |

As you can see, JavaScript has object versions of the primitive data types String, Number, and Boolean.

There is no reason to create complex objects. Primitive values execute much faster.

And there is no reason to use new Array(). Use array literals instead: []

And there is no reason to use new RegExp(). Use pattern literals instead: /()/

And there is no reason to use new Function(). Use function expressions instead: function () {}.

And there is no reason to use new Object(). Use object literals instead: {}

**Example**

var x1 = {};            // new object  
var x2 = "";            // new primitive string  
var x3 = 0;             // new primitive number  
var x4 = false;         // new primitive boolean  
var x5 = [];            // new array object  
var x6 = /()/           // new regexp object  
var x7 = function(){};  // new function object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_best)

**JavaScript Objects are Mutable**

Objects are mutable: They are addressed by reference, not by value.

If y is an object, the following statement will not create a copy of y:

var x = y;  // This will not create a copy of y.

The object x is not a **copy** of y. It **is** y. Both x and y points to the same object.

Any changes to y will also change x, because x and y are the same object.

**Example**

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}  
  
var x = person;  
x.age = 10;           // This will change both x.age and person.age

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_mutable)

**JavaScript Object Properties**

[« Previous](http://www.w3schools.com/js/js_object_definition.asp)

[Next Chapter »](http://www.w3schools.com/js/js_object_methods.asp)

Properties are the most important part of any JavaScript object.

**JavaScript Properties**

Properties are the values associated with a JavaScript object.

A JavaScript object is a collection of unordered properties.

Properties can usually be changed, added, and deleted, but some are read only.

**Accessing JavaScript Properties**

The syntax for accessing the property of an object is:

*objectName.property* // person.age

or

*objectName*["*property*"]       // person["age"]

or

*objectName*[*expression*]       // x = "age"; person[x]

|  |  |
| --- | --- |
| **Note** | The expression must evaluate to a property name. |

**Example 1**

person.firstname + " is " + person.age + " years old.";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties1)

**Example 2**

person["firstname"] + " is " + person["age"] + " years old.";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties2)

**JavaScript for...in Loop**

The JavaScript for...in statement loops through the properties of an object.

**Syntax**

for (*variable* in *object*) {  
*code to be executed*  
}

The block of code inside of the for...in loop will be executed once for each property.

Looping through the properties of an object:

**Example**

var person = {fname:"John", lname:"Doe", age:25};   
  
for (x in person) {  
    txt += person[x];  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties_for_in)

**Adding New Properties**

You can add new properties to an existing object by simply giving it a value.

Assume that the person object already exists - you can then give it new properties:

**Example**

person.nationality = "English";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties3)

|  |  |
| --- | --- |
| **Note** | You cannot use reserved words for property (or method) names. JavaScript naming rules apply. |

**Deleting Properties**

The **delete** keyword deletes a property from an object:

**Example**

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
delete person.age;   // or delete person["age"];

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties4)

The delete keyword deletes both the value of the property and the property itself.

After deletion, the property cannot be used before it is added back again.

The delete operator is designed to be used on object properties. It has no effect on variables or functions.

The delete operator should not be used on predefined JavaScript object properties. It can crash your application.

**Property Attributes**

All properties have a name. In addition they also have a value.

The value is one of the property's attributes.

Other attributes are: enumerable, configurable, and writable.

These attributes define how the property can be accessed (is it readable?, is it writable?)

In JavaScript, all attributes can be read, but only the value attribute can be changed (and only if the property is writable).

( ECMAScript 5 has methods for both getting and setting all property attributes)

**Prototype Properties**

JavaScript objects inherit the properties of their prototype.

The delete keyword does not delete inherited properties, but if you delete a prototype property, it will affect all objects inherited from the prototype.

**JavaScript Object Methods**

[« Previous](http://www.w3schools.com/js/js_properties.asp)

[Next Chapter »](http://www.w3schools.com/js/js_object_prototypes.asp)

**JavaScript Methods**

JavaScript methods are the actions that can be performed on objects.

A JavaScript **method** is a property containing a **function definition**.

|  |  |
| --- | --- |
| **Property** | **Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | blue |
| fullName | function() {return this.firstName + " " + this.lastName;} |

|  |  |
| --- | --- |
| **Note** | Methods are functions stored as object properties. |

**Accessing Object Methods**

You create an object method with the following syntax:

*methodName : function() { code lines }*

You access an object method with the following syntax:

*objectName.methodName()*

You will typically describe fullName() as a method of the person object, and fullName as a property.

The fullName property will execute (as a function) when it is invoked with ().

This example accesses the fullName() **method** of a person object:

**Example**

name = person.fullName();

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_method)

If you access the fullName **property**, without (), it will return the **function definition**:

**Example**

name = person.fullName;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_function)

**Using Built-In Methods**

This example uses the toUpperCase() method of the String object, to convert a text to uppercase:

var message = "Hello world!";  
var x = message.toUpperCase();

The value of x, after execution of the code above will be:

HELLO WORLD!

**Adding New Methods**

Defining methods to an object is done inside the constructor function:

**Example**

function person(firstName, lastName, age, eyeColor) {  
    this.firstName = firstName;    
    this.lastName = lastName;  
    this.age = age;  
    this.eyeColor = eyeColor;  
    this.changeName = function (name) {  
        this.lastName = name;  
    }  
}

The changeName() function assigns the value of name to the person's lastName property.

**Now You Can Try:**

myMother.changeName("Doe");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_create_object3)

JavaScript knows which person you are talking about by "substituting" **this** with **myMother**.

**JavaScript Object Prototypes**

[« Previous](http://www.w3schools.com/js/js_object_methods.asp)

[Next Chapter »](http://www.w3schools.com/js/js_function_definition.asp)

Every JavaScript object has a prototype. The prototype is also an object.

All JavaScript objects inherit their properties and methods from their prototype.

**JavaScript Prototypes**

All JavaScript objects inherit the properties and methods from their prototype.

Objects created using an object literal, or with new Object(), inherit from a prototype called Object.prototype.

Objects created with new Date() inherit the Date.prototype.

The Object.prototype is on the top of the prototype chain.

All JavaScript objects (Date, Array, RegExp, Function, ....) inherit from the Object.prototype.

**Creating a Prototype**

The standard way to create an object prototype is to use an object constructor function:

**Example**

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}

With a constructor function, you can use the **new** keyword to create new objects from the same prototype:

**Example**

var myFather = new person("John", "Doe", 50, "blue");  
var myMother = new person("Sally", "Rally", 48, "green");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype1)

|  |  |
| --- | --- |
| **Note** | The constructor function is the prototype for your person objects. |

**Adding Properties and Methods to Objects**

Sometimes you want to add new properties (or methods) to an existing object.

Sometimes you want to add new properties (or methods) to all existing objects of a given type.

Sometimes you want to add new properties (or methods) to an object prototype.

**Adding a Property to an Object**

Adding a new property to an existing object is easy:

**Example**

myFather.nationality = "English";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype2)

The property will be added to myFather. Not to myMother. Not to any other person objects.

**Adding a Method to an Object**

Adding a new method to an existing object is also easy:

**Example**

myFather.name = function () {  
    return this.firstName + " " + this.lastName;  
};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype8)

The method will be added to myFather. Not to myMother.

**Adding Properties to a Prototype**

You cannot add a new property to a prototype the same way as you add a new property to an existing object, because the prototype is not an existing object.

**Example**

person.nationality = "English";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype3)

To add a new property to a constructor, you must add it to the constructor function:

**Example**

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
    this.nationality = "English"  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype4)

|  |  |
| --- | --- |
| **Note** | Prototype properties can have prototype values (default values). |

**Adding Methods to a Prototype**

Your constructor function can also define methods:

**Example**

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
    this.name = function() {return this.firstName + " " + this.lastName;};  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype7)

**Using the prototype Property**

The JavaScript prototype property allows you to add new properties to an existing prototype:

**Example**

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}  
person.prototype.nationality = "English";

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype5)

The JavaScript prototype property also allows you to add new methods to an existing prototype:

**Example**

function person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}  
person.prototype.name = function() {  
    return this.firstName + " " + this.lastName;  
};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype6)

|  |  |
| --- | --- |
| **Note** | Only modify your **own** prototypes. Never modify the prototypes of standard JavaScript objects. |

**JavaScript Function Definitions**

[« Previous](http://www.w3schools.com/js/js_object_prototypes.asp)

[Next Chapter »](http://www.w3schools.com/js/js_function_parameters.asp)

JavaScript functions are **defined** with the **function** keyword.

You can use a function **declaration** or a function **expression**.

**Function Declarations**

Earlier in this tutorial, you learned that functions are **declared** with the following syntax:

function *functionName*(*parameters*) {  
  *code to be executed*  
}

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

**Example**

function myFunction(a, b) {  
    return a \* b;  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_return)

|  |  |
| --- | --- |
| **Note** | Semicolons are used to separate executable JavaScript statements. Since a function **declaration** is not an executable statement, it is not common to end it with a semicolon. |

**Function Expressions**

A JavaScript function can also be defined using an **expression**.

A function expression can be stored in a variable:

**Example**

var x = function (a, b) {return a \* b};

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_expression)

After a function expression has been stored in a variable, the variable can be used as a function:

**Example**

var x = function (a, b) {return a \* b};  
var z = x(4, 3);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_expression_variable)

The function above is actually an **anonymous function** (a function without a name).

Functions stored in variables do not need function names. They are always invoked (called) using the variable name.

|  |  |
| --- | --- |
| **Note** | The function above ends with a semicolon because it is a part of an executable statement. |

**The Function() Constructor**

As you have seen in the previous examples, JavaScript functions are defined with the **function** keyword.

Functions can also be defined with a built-in JavaScript function constructor called Function().

**Example**

var myFunction = new Function("a", "b", "return a \* b");  
  
var x = myFunction(4, 3);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_constructor)

You actually don't have to use the function constructor. The example above is the same as writing:

**Example**

var myFunction = function (a, b) {return a \* b};  
  
var x = myFunction(4, 3);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_constructor2)

|  |  |
| --- | --- |
| **Note** | Most of the time, you can avoid using the **new** keyword in JavaScript. |

**Function Hoisting**

Earlier in this tutorial, you learned about "hoisting".

Hoisting is JavaScript's default behavior of moving **declarations** to the top of the current scope.

Hoisting applies to variable declarations and to function declarations.

Because of this, JavaScript functions can be called before they are declared:

myFunction(5);  
  
function myFunction(y) {  
    return y \* y;  
}

Functions defined using an expression are not hoisted.

**Self-Invoking Functions**

Function expressions can be made "self-invoking".

A self-invoking expression is invoked (started) automatically, without being called.

Function expressions will execute automatically if the expression is followed by ().

You cannot self-invoke a function declaration.

You have to add parentheses around the function to indicate that it is a function expression:

**Example**

(function () {  
    var x = "Hello!!";      // I will invoke myself  
})();

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_expression_self)

The function above is actually an **anonymous self-invoking function** (function without name).

**Functions Can Be Used as Values**

JavaScript functions can be used as values:

**Example**

function myFunction(a, b) {  
    return a \* b;  
}  
  
var x = myFunction(4, 3);

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_value)

JavaScript functions can be used in expressions:

**Example**

function myFunction(a, b) {  
    return a \* b;  
}  
  
var x = myFunction(4, 3) \* 2;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_value2)

**Functions are Objects**

The **typeof** operator in JavaScript returns "function" for functions.

But, JavaScript functions can best be described as objects.

JavaScript functions have both **properties** and **methods**.

The arguments.length property returns the number of arguments received when the function was invoked:

**Example**

function myFunction(a, b) {  
    return arguments.length;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_length)

The toString() method returns the function as a string:

**Example**

function myFunction(a, b) {  
    return a \* b;  
}  
  
var txt = myFunction.toString();

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_tostring)

|  |  |
| --- | --- |
| **Note** | A function defined as the property of an object, is called a method to the object. A function designed to create new objects, is called an object constructor. |

**JavaScript Function Parameters**

[« Previous](http://www.w3schools.com/js/js_function_definition.asp)

[Next Chapter »](http://www.w3schools.com/js/js_function_invocation.asp)

A JavaScript function does not perform any checking on parameter values (arguments).

**Function Parameters and Arguments**

Earlier in this tutorial, you learned that functions can have **parameters**:

*functionName*(*parameter1, parameter2, parameter3*) {  
    *code to be executed*  
}

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** passed to (and received by) the function.

**Parameter Rules**

JavaScript function definitions do not specify data types for parameters.

JavaScript functions do not perform type checking on the passed arguments.

JavaScript functions do not check the number of arguments received.

**Parameter Defaults**

If a function is called with **missing arguments** (less than declared), the missing values are set to: **undefined**

Sometimes this is acceptable, but sometimes it is better to assign a default value to the parameter:

**Example**

function myFunction(x, y) {  
    if (y === undefined) {  
          y = 0;  
    }   
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_undefined)

If a function is called with **too many arguments** (more than declared), these arguments cannot be referred, because they don't have a name. They can only be reached in the arguments object.

**The Arguments Object**

JavaScript functions have a built-in object called the arguments object.

The argument object contains an array of the arguments used when the function was called (invoked).

This way you can simply use a function to find (for instance) the highest value in a list of numbers:

**Example**

x = findMax(1, 123, 500, 115, 44, 88);  
  
function findMax() {  
    var i;  
    var max = -Infinity;  
    for (i = 0; i < arguments.length; i++) {  
        if (arguments[i] > max) {  
            max = arguments[i];  
        }  
    }  
    return max;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_arguments)

Or create a function to summarize all input values:

**Example**

x = sumAll(1, 123, 500, 115, 44, 88);  
  
function sumAll() {  
    var i, sum = 0;  
    for (i = 0; i < arguments.length; i++) {  
        sum += arguments[i];  
    }  
    return sum;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_arguments_sum)

**Arguments are Passed by Value**

The parameters, in a function call, are the function's arguments.

JavaScript arguments are passed by **value**: The function only gets to know the values, not the argument's locations.

If a function changes an argument's value, it does not change the parameter's original value.

**Changes to arguments are not visible (reflected) outside the function.**

**Objects are Passed by Reference**

In JavaScript, object references are values.

Because of this, objects will behave like they are passed by **reference:**

If a function changes an object property, it changes the original value.

**Changes to object properties are visible (reflected) outside the function.**

**JavaScript Function Invocation**

[« Previous](http://www.w3schools.com/js/js_function_parameters.asp)

[Next Chapter »](http://www.w3schools.com/js/js_function_closures.asp)

JavaScript functions can be invoked in 4 different ways.

Each method differs in how **this** is initialized.

**The *this* Keyword**

In JavaScript, the thing called **this**, is the object that "owns" the current code.

The value of this, when used in a function, is the object that "owns" the function.

|  |  |
| --- | --- |
| **Note** | Note that **this** is not a variable. It is a keyword. You cannot change the value of **this**. |

**Invoking a JavaScript Function**

You have already learned that the code inside a JavaScript function will execute when "something" invokes it.

The code in a function is not executed when the function is **defined**. It is executed when the function is **invoked**.

Some people use the term "**call a function**" instead of "**invoke a function**".

It is also quite common to say "call upon a function", "start a function", or "execute a function".

In this tutorial, we will use **invoke**, because a JavaScript function can be invoked without being called.

**Invoking a Function as a Function**

**Example**

function myFunction(a, b) {  
    return a \* b;  
}  
myFunction(10, 2);           // myFunction(10, 2) will return 20

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_invoke_function)

The function above does not belong to any object. But in JavaScript there is always a default global object.

In HTML the default global object is the HTML page itself, so the function above "belongs" to the HTML page.

In a browser the page object is the browser window. The function above automatically becomes a window function.

myFunction() and window.myFunction() is the same function:

**Example**

function myFunction(a, b) {  
    return a \* b;  
}  
window.myFunction(10, 2);    // window.myFunction(10, 2) will also return 20

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_invoke_function_2)

|  |  |
| --- | --- |
| **Note** | This is a common way to invoke a JavaScript function, but not a good practice in computer programming. Global variables, methods, or functions can easily create name conflicts and bugs in the global object. |

**The Global Object**

When a function is called without an owner object, the value of **this** becomes the global object.

In a web browser the global object is the browser window.

This example returns the window object as the value of **this**:

**Example**

function myFunction() {  
    return this;  
}  
myFunction();                // Will return the window object

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_invoke_function_3)

|  |  |
| --- | --- |
| **Note** | Invoking a function as a global function, causes the value of **this** to be the global object. Using the window object as a variable can easily crash your program. |

**Invoking a Function as a Method**

In JavaScript you can define function as object methods.

The following example creates an object (**myObject**), with two properties (**firstName** and **lastName**), and a method (**fullName**):

**Example**

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this.firstName + " " + this.lastName;  
    }  
}  
myObject.fullName();         // Will return "John Doe"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_invoke_method)

The **fullName** method is a function. The function belongs to the object. **myObject** is the owner of the function.

The thing called **this**, is the object that "owns" the JavaScript code. In this case the value of **this** is **myObject**.

Test it! Change the **fullName** method to return the value of **this**:

**Example**

var myObject = {  
    firstName:"John",  
    lastName: "Doe",  
    fullName: function () {  
        return this;  
    }  
}  
myObject.fullName();          // Will return [object Object] (the owner object)

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_invoke_method_2)

|  |  |
| --- | --- |
| **Note** | Invoking a function as an object method, causes the value of **this** to be the object itself. |

**Invoking a Function with a Function Constructor**

If a function invocation is preceded with the **new** keyword, it is a constructor invocation.

It looks like you create a new function, but since JavaScript functions are objects you actually create a new object:

**Example**

// This is a function constructor:  
function myFunction(arg1, arg2) {  
    this.firstName = arg1;  
    this.lastName  = arg2;  
}  
  
// This creates a new object  
var x = new myFunction("John","Doe");  
x.firstName;                             // Will return "John"

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_invoke_constructor)

A constructor invocation creates a new object. The new object inherits the properties and methods from its constructor.

|  |  |
| --- | --- |
| **Note** | The **this** keyword in the constructor does not have a value. The value of **this** will be the new object created when the function is invoked. |

**Invoking a Function with a Function Method**

In JavaScript, functions are objects. JavaScript functions have properties and methods.

**call()** and **apply()** are predefined JavaScript function methods. Both methods can be used to invoke a function, and both methods must have the owner object as first parameter.

**Example**

function myFunction(a, b) {  
    return a \* b;  
}  
myObject = myFunction.call(myObject, 10, 2);     // Will return 20

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_invoke_call)

**Example**

function myFunction(a, b) {  
    return a \* b;  
}  
myArray = [10, 2];  
myObject = myFunction.apply(myObject, myArray);  // Will also return 20

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_invoke_apply)

Both methods takes an owner object as the first argument. The only difference is that call() takes the function arguments separately, and apply() takes the function arguments in an array.

In JavaScript strict mode, the first argument becomes the value of **this** in the invoked function, even if the argument is not an object.

In "non-strict" mode, if the value of the first argument is null or undefined, it is replaced with the global object.

|  |  |
| --- | --- |
| **Note** | With call() or apply() you can set the value of **this**, and invoke a function as a new method of an existing object. |

**JavaScript Closures**

[« Previous](http://www.w3schools.com/js/js_function_invocation.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom.asp)

JavaScript variables can belong to the **local** or **global** scope.

Private variables can be made possible with **closures**.

**Global Variables**

A function can access all variables defined **inside** the function, like this:

**Example**

function myFunction() {  
    var a = 4;  
    return a \* a;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_scope2)

But a function can also access variables defined **outside** the function, like this:

**Example**

var a = 4;  
function myFunction() {  
    return a \* a;  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_scope1)

In the last example, **a** is a **global** variable.

In a web page, global variables belong to the window object.

Global variables can be used (and changed) by all scripts in the page (and in the window).

In the first example, **a** is a **local** variable.

A local variable can only be used inside the function where it is defined. It is hidden from other functions and other scripting code.

Global and local variables with the same name are different variables. Modifying one, does not modify the other.

|  |  |
| --- | --- |
| **Note** | Variables created **without** the keyword **var**, are always global, even if they are created inside a function. |

**Variable Lifetime**

Global variables live as long as your application (your window / your web page) lives.

Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

**A Counter Dilemma**

Suppose you want to use a variable for counting something, and you want this counter to be available to all functions.

You could use a global variable, and a function to increase the counter:

**Example**

var counter = 0;  
  
function add() {  
    counter += 1;  
}  
  
add();  
add();  
add();  
  
// the counter is now equal to 3

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_counter)

The counter should only be changed by the add() function.

The problem is, that any script on the page can change the counter, without calling add().

If I declare the counter inside the function, nobody will be able to change it without calling add():

**Example**

function add() {  
    var counter = 0;  
    counter += 1;  
}  
  
add();  
add();  
add();  
  
// the counter should now be 3, but it does not work !

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_counter2)

It did not work! Every time I call the add() function, the counter is set to 1.

**A JavaScript inner function can solve this.**

**JavaScript Nested Functions**

All functions have access to the global scope.

In fact, in JavaScript, all functions have access to the scope "above" them.

JavaScript supports nested functions. Nested functions have access to the scope "above" them.

In this example, the inner function **plus()** has access to the **counter** variable in the parent function:

**Example**

function add() {  
    var counter = 0;  
    function plus() {counter += 1;}  
    plus();      
    return counter;   
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_counter1)

This could have solved the counter dilemma, if we could reach the **plus()** function from the outside.

We also need to find a way to execute **counter = 0** only once.

**We need a closure.**

**JavaScript Closures**

Remember self-invoking functions? What does this function do?

**Example**

var add = (function () {  
    var counter = 0;  
    return function () {return counter += 1;}  
})();  
  
add();  
add();  
add();  
  
// the counter is now 3

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_counter3)

**Example Explained**

The variable **add** is assigned the return value of a self-invoking function.

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.

This is called a JavaScript **closure.** It makes it possible for a function to have "**private**" variables.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

|  |  |
| --- | --- |
| **Note** | A closure is a function having access to the parent scope, even after the parent function has closed. |

**JavaScript HTML DOM**

[« Previous](http://www.w3schools.com/js/js_function_closures.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_methods.asp)

With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

**The HTML DOM (Document Object Model)**

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

**The HTML DOM Tree of Objects**



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

* JavaScript can change all the HTML elements in the page
* JavaScript can change all the HTML attributes in the page
* JavaScript can change all the CSS styles in the page
* JavaScript can remove existing HTML elements and attributes
* JavaScript can add new HTML elements and attributes
* JavaScript can react to all existing HTML events in the page
* JavaScript can create new HTML events in the page

**What You Will Learn**

In the next chapters of this tutorial you will learn:

* How to change the content of HTML elements
* How to change the style (CSS) of HTML elements
* How to react to HTML DOM events
* How to add and delete HTML elements

**What is the DOM?**

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

*"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

The W3C DOM standard is separated into 3 different parts:

* Core DOM - standard model for all document types
* XML DOM - standard model for XML documents
* HTML DOM - standard model for HTML documents

**What is the HTML DOM?**

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

* The HTML elements as **objects**
* The **properties** of all HTML elements
* The **methods** to access all HTML elements
* The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

**JavaScript - HTML DOM Methods**

[« Previous](http://www.w3schools.com/js/js_htmldom.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_document.asp)

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

**The DOM Programming Interface**

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as **objects**.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

**Example**

The following example changes the content (the innerHTML) of the <p> element with id="demo":

**Example**

<html>  
<body>  
  
<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = "Hello World!";  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_method)

In the example above, getElementById is a **method**, while innerHTML is a **property**.

**The getElementById Method**

The most common way to access an HTML element is to use the id of the element.

In the example above the getElementById method used id="demo" to find the element.

**The innerHTML Property**

The easiest way to get the content of an element is by using the **innerHTML** property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

|  |  |
| --- | --- |
| **Note** | The innerHTML property can be used to get or change any HTML element, including <html> and <body>. |

**JavaScript HTML DOM Document**

[« Previous](http://www.w3schools.com/js/js_htmldom_methods.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_elements.asp)

The HTML DOM document object is the owner of all other objects in your web page.

**The HTML DOM Document Object**

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

**Finding HTML Elements**

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.getElementById() | Find an element by element id |
| document.getElementsByTagName() | Find elements by tag name |
| document.getElementsByClassName() | Find elements by class name |

**Changing HTML Elements**

|  |  |
| --- | --- |
| **Method** | **Description** |
| *element*.innerHTML= | Change the inner HTML of an element |
| *element*.*attribute=* | Change the attribute of an HTML element |
| *element*.setAttribute*(attribute,value)* | Change the attribute of an HTML element |
| *element*.style.*property=* | Change the style of an HTML element |

**Adding and Deleting Elements**

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.createElement() | Create an HTML element |
| document.removeChild() | Remove an HTML element |
| document.appendChild() | Add an HTML element |
| document.replaceChild() | Replace an HTML element |
| document.write(*text*) | Write into the HTML output stream |

**Adding Events Handlers**

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.getElementById(*id*).onclick=function(){*code*} | Adding event handler code to an onclick event |

**Finding HTML Objects**

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| document.anchors | Returns all <a> elements that have a name attribute | 1 |
| document.applets | Returns all <applet> elements (Deprecated in HTML5) | 1 |
| document.baseURI | Returns the absolute base URI of the document | 3 |
| document.body | Returns the <body> element | 1 |
| document.cookie | Returns the document's cookie | 1 |
| document.doctype | Returns the document's doctype | 3 |
| document.documentElement | Returns the <html> element | 3 |
| document.documentMode | Returns the mode used by the browser | 3 |
| document.documentURI | Returns the URI of the document | 3 |
| document.domain | Returns the domain name of the document server | 1 |
| document.domConfig | Obsolete. Returns the DOM configuration | 3 |
| document.embeds | Returns all <embed> elements | 3 |
| document.forms | Returns all <form> elements | 1 |
| document.head | Returns the <head> element | 3 |
| document.images | Returns all <img> elements | 1 |
| document.implementation | Returns the DOM implementation | 3 |
| document.inputEncoding | Returns the document's encoding (character set) | 3 |
| document.lastModified | Returns the date and time the document was updated | 3 |
| document.links | Returns all <area> and <a> elements that have a href attribute | 1 |
| document.readyState | Returns the (loading) status of the document | 3 |
| document.referrer | Returns the URI of the referrer (the linking document) | 1 |
| document.scripts | Returns all <script> elements | 3 |
| document.strictErrorChecking | Returns if error checking is enforced | 3 |
| document.title | Returns the <title> element | 1 |
| document.URL | Returns the complete URL of the document | 1 |

**JavaScript HTML DOM Elements**

[« Previous](http://www.w3schools.com/js/js_htmldom_document.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_html.asp)

This page teaches you how to find and access HTML elements in an HTML page.

**Finding HTML Elements**

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

* Finding HTML elements by id
* Finding HTML elements by tag name
* Finding HTML elements by class name
* Finding HTML elements by CSS selectors
* Finding HTML elements by HTML object collections

**Finding HTML Element by Id**

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id="intro":

**Example**

var myElement = document.getElementById("intro");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_getelementbyid)

If the element is found, the method will return the element as an object (in myElement).

If the element is not found, myElement will contain null.

**Finding HTML Elements by Tag Name**

This example finds all <p> elements:

**Example**

var x = document.getElementsByTagName("p");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_getelementsbytagname2)

This example finds the element with id="main", and then finds all <p> elements inside "main":

**Example**

var x = document.getElementById("main");  
var y = x.getElementsByTagName("p");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_getelementsbytagname)

**Finding HTML Elements by Class Name**

If you want to find all HTML elements with the same class name, use getElementsByClassName().

This example returns a list of all elements with class="intro".

**Example**

var x = document.getElementsByClassName("intro");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_getelementsbyclassname)

|  |  |
| --- | --- |
| **Note** | Finding elements by class name does not work in Internet Explorer 8 and earlier versions. |

**Finding HTML Elements by CSS Selectors**

If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

This example returns a list of all <p> elements with class="intro".

**Example**

var x = document.querySelectorAll("p.intro");

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_queryselectorall)

|  |  |
| --- | --- |
| **Note** | The querySelectorAll() method does not work in Internet Explorer 8 and earlier versions. |

**Finding HTML Elements by HTML Object Collections**

This example finds the form element with id="frm1", in the forms collection, and displays all element values:

**Example**

var x = document.forms["frm1"];  
var text = "";  
var i;  
for (i = 0; i < x.length; i++) {  
    text += x.elements[i].value + "<br>";  
}  
document.getElementById("demo").innerHTML = text;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_form_elements)

The following HTML objects (and object collections) are also accessible:

* [document.anchors](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_anchors)
* [document.body](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_body)
* [document.documentElement](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_element)
* [document.embeds](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_embeds)
* [document.forms](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_forms)
* [document.head](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_head)
* [document.images](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_images)
* [document.links](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_links)
* [document.scripts](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_scripts)
* [document.title](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_title)

**Test Yourself with Exercises!**

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_elements1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_elements2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_elements3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_elements4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_elements5)

**JavaScript HTML DOM - Changing HTML**

[« Previous](http://www.w3schools.com/js/js_htmldom_elements.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_css.asp)

The HTML DOM allows JavaScript to change the content of HTML elements.

**Changing the HTML Output Stream**

JavaScript can create dynamic HTML content:

**Date: Wed Oct 21 2015 15:34:51 GMT+0200**

In JavaScript, document.write() can be used to write directly to the HTML output stream:

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<script>  
document.write(Date());  
</script>  
  
</body>  
</html>

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date)

|  |  |
| --- | --- |
| **Note** | Never use document.write() after the document is loaded. It will overwrite the document. |

**Changing HTML Content**

The easiest way to modify the content of an HTML element is by using the **innerHTML** property.

To change the content of an HTML element, use this syntax:

document.getElementById(*id*).innerHTML = *new HTML*

This example changes the content of a <p> element:

**Example**

<html>  
<body>  
  
<p id="p1">Hello World!</p>  
  
<script>  
document.getElementById("p1").innerHTML = "New text!";  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_change_innerhtml)

This example changes the content of an <h1> element:

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1 id="header">Old Header</h1>  
  
<script>  
var element = document.getElementById("header");  
element.innerHTML = "New Header";  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_innerhtml)

Example explained:

* The HTML document above contains an <h1> element with id="header"
* We use the HTML DOM to get the element with id="header"
* A JavaScript changes the content (innerHTML) of that element

**Changing the Value of an Attribute**

To change the value of an HTML attribute, use this syntax:

document.getElementById(*id*).*attribute=new value*

This example changes the value of the src attribute of an <img> element:

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<img id="myImage" src="smiley.gif">  
  
<script>  
document.getElementById("myImage").src = "landscape.jpg";  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_image)

Example explained:

* The HTML document above contains an <img> element with id="myImage"
* We use the HTML DOM to get the element with id="myImage"
* A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg"

**Test Yourself with Exercises!**

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_change1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_change2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_change3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_change4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_change5)

**JavaScript HTML DOM - Changing CSS**

[« Previous](http://www.w3schools.com/js/js_htmldom_html.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_events.asp)

The HTML DOM allows JavaScript to change the style of HTML elements.

**Changing HTML Style**

To change the style of an HTML element, use this syntax:

document.getElementById(*id*).style.*property*=*new style*

The following example changes the style of a <p> element:

**Example**

<html>  
<body>  
  
<p id="p2">Hello World!</p>  
  
<script>  
document.getElementById("p2").style.color = "blue";  
</script>  
  
<p>The paragraph above was changed by a script.</p>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_change_style)

**Using Events**

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

* An element is clicked on
* The page has loaded
* Input fields are changed

You will learn more about events in the next chapter of this tutorial.

This example changes the style of the HTML element with id="id1", when the user clicks a button:

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1 id="id1">My Heading 1</h1>  
  
<button type="button"   
onclick="document.getElementById('id1').style.color = 'red'">  
Click Me!</button>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_color2)

**More Examples**

[Visibility](http://www.w3schools.com/js/tryit.asp?filename=tryjs_visibility) How to make an element invisible. Do you want to show the element or not?

**HTML DOM Style Object Reference**

For all HTML DOM style properties, look at our complete [HTML DOM Style Object Reference](http://www.w3schools.com/jsref/dom_obj_style.asp).

**Test Yourself with Exercises!**

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_css1)  [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_css2)  [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_css3)  [Exercise 4 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_css4)  [Exercise 5 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_css5)

**JavaScript HTML DOM Events**

[« Previous](http://www.w3schools.com/js/js_htmldom_css.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_eventlistener.asp)

HTML DOM allows JavaScript to react to HTML events:

**Mouse Over Me**

**Click Me**

**Reacting to Events**

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

Examples of HTML events:

* When a user clicks the mouse
* When a web page has loaded
* When an image has been loaded
* When the mouse moves over an element
* When an input field is changed
* When an HTML form is submitted
* When a user strokes a key

In this example, the content of the <h1> element is changed when a user clicks on it:

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick2)

In this example, a function is called from the event handler:

**Example**

<!DOCTYPE html>  
<html>  
<body>  
  
<h1 onclick="changeText(this)">Click on this text!</h1>  
  
<script>  
function changeText(id) {   
    id.innerHTML = "Ooops!";  
}  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick3)

**HTML Event Attributes**

To assign events to HTML elements you can use event attributes.

**Example**

Assign an onclick event to a button element:

<button onclick="displayDate()">Try it</button>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events1)

In the example above, a function named *displayDate* will be executed when the button is clicked.

**Assign Events Using the HTML DOM**

The HTML DOM allows you to assign events to HTML elements using JavaScript:

**Example**

Assign an onclick event to a button element:

<script>  
document.getElementById("myBtn").onclick = displayDate;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events2)

In the example above, a function named *displayDate* is assigned to an HTML element with the id="myBtn".

The function will be executed when the button is clicked.

**The onload and onunload Events**

The onload and onunload events are triggered when the user enters or leaves the page.

The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload and onunload events can be used to deal with cookies.

**Example**

<body onload="checkCookies()">

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onload)

**The onchange Event**

The onchange event are often used in combination with validation of input fields.

Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

**Example**

<input type="text" id="fname" onchange="upperCase()">

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_onchange)

**The onmouseover and onmouseout Events**

The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element:

Mouse Over Me

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_mouseover)

**The onmousedown, onmouseup and onclick Events**

The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

Click Me

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_mousedown)

**More Examples**

[onmousedown and onmouseup](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onmousedown)  
Change an image when a user holds down the mouse button.

[onload](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onload)  
Display an alert box when the page has finished loading.

[onfocus](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onfocus)  
Change the background-color of an input field when it gets focus.

[Mouse Events](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onmouse)  
Change the color of an element when the cursor moves over it.

**HTML DOM Event Object Reference**

For a list of all HTML DOM events, look at our complete [HTML DOM Event Object Reference](http://www.w3schools.com/jsref/dom_obj_event.asp).

**Test Yourself with Exercises!**

[Exercise 1 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_events1)   [Exercise 2 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_events2)   [Exercise 3 »](http://www.w3schools.com/js/exercise.asp?filename=exercise_dom_events3)

**JavaScript HTML DOM EventListener**

[« Previous](http://www.w3schools.com/js/js_htmldom_events.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_navigation.asp)

**The addEventListener() method**

**Example**

Add an event listener that fires when a user clicks a button:

document.getElementById("myBtn").addEventListener("click", displayDate);

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_displaydate)

The addEventListener() method attaches an event handler to the specified element.

The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The addEventListener() method makes it easier to control how the event reacts to bubbling.

When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the removeEventListener() method.

**Syntax**

*element*.addEventListener(*event, function, useCapture*);

The first parameter is the type of the event (like "click" or "mousedown").

The second parameter is the function we want to call when the event occurs.

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

|  |  |
| --- | --- |
| **Note** | Note that you don't use the "on" prefix for the event; use "click" instead of "onclick". |

**Add an Event Handler to an Element**

**Example**

Alert "Hello World!" when the user clicks on an element:

*element*.addEventListener("click", function(){ alert("Hello World!"); });

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_add)

You can also refer to an external "named" function:

**Example**

Alert "Hello World!" when the user clicks on an element:

*element*.addEventListener("click", myFunction);  
  
function myFunction() {  
    alert ("Hello World!");  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_add2)

**Add Many Event Handlers to the Same Element**

The addEventListener() method allows you to add many events to the same element, without overwriting existing events:

**Example**

*element*.addEventListener("click", myFunction);  
*element*.addEventListener("click", mySecondFunction);

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_add_many)

You can add events of different types to the same element:

**Example**

*element*.addEventListener("mouseover", myFunction);  
*element*.addEventListener("click", mySecondFunction);  
*element*.addEventListener("mouseout", myThirdFunction);

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_add_many2)

**Add an Event Handler to the Window Object**

The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that supports events, like the xmlHttpRequest object.

**Example**

Add an event listener that fires when a user resizes the window:

window.addEventListener("resize", function(){  
    document.getElementById("demo").innerHTML = *sometext*;  
});

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_dom)

**Passing Parameters**

When passing parameter values, use an "anonymous function" that calls the specified function with the parameters:

**Example**

*element*.addEventListener("click", function(){ myFunction(p1, p2); });

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_parameters)

**Event Bubbling or Event Capturing?**

There are two ways of event propagation in the HTML DOM, bubbling and capturing.

Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?

In *bubbling* the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.

In *capturing* the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

addEventListener(*event*, *function*, *useCapture*);

The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

**Example**

document.getElementById("myP").addEventListener("click", myFunction, true);  
document.getElementById("myDiv").addEventListener("click", myFunction, true);

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_usecapture)

**The removeEventListener() method**

The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:

**Example**

*element*.removeEventListener("mousemove", myFunction);

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_remove)

**Browser Support**

The numbers in the table specifies the first browser version that fully supports these methods.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Method** |  |  |  |  |  |
| addEventListener() | 1.0 | 9.0 | 1.0 | 1.0 | 7.0 |
| removeEventListener() | 1.0 | 9.0 | 1.0 | 1.0 | 7.0 |

**Note:** The addEventListener() and removeEventListener() methods are not supported in IE 8 and earlier versions and Opera 6.0 and earlier versions. However, for these specific browser versions, you can use the attachEvent() method to attach an event handlers to the element, and the detachEvent() method to remove it:

*element.*attachEvent*(event, function);  
element.*detachEvent*(event, function);*

**Example**

Cross-browser solution:

var x = document.getElementById("myBtn");  
if (x.addEventListener) {                    // For all major browsers, except IE 8 and earlier  
    x.addEventListener("click", myFunction);  
} else if (x.attachEvent) {                  // For IE 8 and earlier versions  
    x.attachEvent("onclick", myFunction);  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_addeventlistener_crossbrowser)

**HTML DOM Event Object Reference**

For a list of all HTML DOM events, look at our complete [HTML DOM Event Object Reference](http://www.w3schools.com/jsref/dom_obj_event.asp).

**JavaScript HTML DOM Navigation**

[« Previous](http://www.w3schools.com/js/js_htmldom_eventlistener.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_nodes.asp)

With the HTML DOM, you can navigate the node tree using node relationships.

**DOM Nodes**

According to the W3C HTML DOM standard, everything in an HTML document is a node:

* The entire document is a document node
* Every HTML element is an element node
* The text inside HTML elements are text nodes
* Every HTML attribute is an attribute node
* All comments are comment nodes



With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.

New nodes can be created, and all nodes can be modified or deleted.

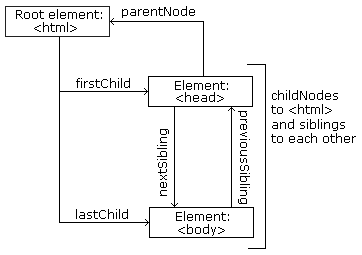
**Node Relationships**

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

* In a node tree, the top node is called the root (or root node)
* Every node has exactly one parent, except the root (which has no parent)
* A node can have a number of children
* Siblings (brothers or sisters) are nodes with the same parent

<html>  
  
  <head>  
      <title>DOM Tutorial</title>  
  </head>  
  
  <body>  
      <h1>DOM Lesson one</h1>  
      <p>Hello world!</p>  
  </body>  
  
</html>



From the HTML above you can read:

* <html> is the root node
* <html> has no parents
* <html> is the parent of <head> and <body>
* <head> is the first child of <html>
* <body> is the last child of <html>

and:

* <head> has one child: <title>
* <title> has one child (a text node): "DOM Tutorial"
* <body> has two children: <h1> and <p>
* <h1> has one child: "DOM Lesson one"
* <p> has one child: "Hello world!"
* <h1> and <p> are siblings

**Navigating Between Nodes**

You can use the following node properties to navigate between nodes with JavaScript:

* parentNode
* childNodes[*nodenumber*]
* firstChild
* lastChild
* nextSibling
* previousSibling

**Warning !**

A common error in DOM processing is to expect an element node to contain text.

In this example: **<title>DOM Tutorial</title>**, the element node <title> does not contain text. It contains a **text node** with the value "DOM Tutorial".

The value of the text node can be accessed by the node's **innerHTML** property, or the **nodeValue**.

**Child Nodes and Node Values**

In addition to the innerHTML property, you can also use the childNodes and nodeValue properties to get the content of an element.

The following example collects the node value of an <h1> element and copies it into a <p> element:

**Example**

<html>  
<body>  
  
<h1 id="intro">My First Page</h1>  
  
<p id="demo">Hello!</p>  
  
<script>  
var myText = document.getElementById("intro").childNodes[0].nodeValue;  
document.getElementById("demo").innerHTML = myText;  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_nodevalue)

In the example above, getElementById is a method, while childNodes and nodeValue are properties.

In this tutorial we use the innerHTML property. However, learning the method above is useful for understanding the tree structure and the navigation of the DOM.

Using the firstChild property is the same as using childNodes[0]:

**Example**

<html>  
<body>  
  
<h1 id="intro">My First Page</h1>  
  
<p id="demo">Hello World!</p>  
  
<script>  
myText = document.getElementById("intro").firstChild.nodeValue;  
document.getElementById("demo").innerHTML = myText;  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_firstchild)

**DOM Root Nodes**

There are two special properties that allow access to the full document:

* document.body - The body of the document
* document.documentElement - The full document

**Example**

<html>  
<body>  
  
<p>Hello World!</p>  
<div>  
<p>The DOM is very useful!</p>  
<p>This example demonstrates the <b>document.body</b> property.</p>  
</div>  
  
<script>  
alert(document.body.innerHTML);  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_body)

**Example**

<html>  
<body>  
  
<p>Hello World!</p>  
<div>  
<p>The DOM is very useful!</p>  
<p>This example demonstrates the <b>document.documentElement</b> property.</p>  
</div>  
  
<script>  
alert(document.documentElement.innerHTML);  
</script>  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_document)

**The nodeName Property**

The nodeName property specifies the name of a node.

* nodeName is read-only
* nodeName of an element node is the same as the tag name
* nodeName of an attribute node is the attribute name
* nodeName of a text node is always #text
* nodeName of the document node is always #document

**Note:** nodeName always contains the uppercase tag name of an HTML element.

**The nodeValue Property**

The nodeValue property specifies the value of a node.

* nodeValue for element nodes is undefined
* nodeValue for text nodes is the text itself
* nodeValue for attribute nodes is the attribute value

**The nodeType Property**

The nodeType property returns the type of node. nodeType is read only.

The most important node types are:

|  |  |
| --- | --- |
| **Element type** | **NodeType** |
| Element | 1 |
| Attribute | 2 |
| Text | 3 |
| Comment | 8 |
| Document | 9 |

**JavaScript HTML DOM Elements (Nodes)**

[« Previous](http://www.w3schools.com/js/js_htmldom_navigation.asp)

[Next Chapter »](http://www.w3schools.com/js/js_htmldom_nodelist.asp)

Adding and Removing Nodes (HTML Elements)

**Creating New HTML Elements (Nodes)**

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

**Example**

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
var para = document.createElement("p");  
var node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
var element = document.getElementById("div1");  
element.appendChild(para);  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_elementcreate)

**Example Explained**

This code creates a new <p> element:

var para = document.createElement("p");

To add text to the <p> element, you must create a text node first. This code creates a text node:

var node = document.createTextNode("This is a new paragraph.");

Then you must append the text node to the <p> element:

para.appendChild(node);

Finally you must append the new element to an existing element.

This code finds an existing element:

var element = document.getElementById("div1");

This code appends the new element to the existing element:

element.appendChild(para);

**Creating new HTML Elements - insertBefore()**

The appendChild() method in the previous example, appended the new element as the last child of the parent.

If you don't want that you can use the insertBefore() method:

**Example**

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
var para = document.createElement("p");  
var node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
var element = document.getElementById("div1");  
var child = document.getElementById("p1");  
element.insertBefore(para,child);  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_elementcreate2)

**Removing Existing HTML Elements**

To remove an HTML element, you must know the parent of the element:

**Example**

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
var parent = document.getElementById("div1");  
var child = document.getElementById("p1");  
parent.removeChild(child);  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_elementremove)

**Example Explained**

This HTML document contains a <div> element with two child nodes (two <p> elements):

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>

Find the element with id="div1":

var parent = document.getElementById("div1");

Find the <p> element with id="p1":

var child = document.getElementById("p1");

Remove the child from the parent:

parent.removeChild(child);

|  |  |
| --- | --- |
| **Note** | It would be nice to be able to remove an element without referring to the parent. But sorry. The DOM needs to know both the element you want to remove, and its parent. |

Here is a common workaround: Find the child you want to remove, and use its parentNode property to find the parent:

var child = document.getElementById("p1");  
child.parentNode.removeChild(child);

**Replacing HTML Elements**

To replace an element to the HTML DOM, use the replaceChild() method:

**Example**

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
var para = document.createElement("p");  
var node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
var parent = document.getElementById("div1");  
var child = document.getElementById("p1");  
parent.replaceChild(para,child);  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_elementreplace)

**JavaScript HTML DOM Node List**

[« Previous](http://www.w3schools.com/js/js_htmldom_nodes.asp)

[Next Chapter »](http://www.w3schools.com/js/js_window.asp)

A node list is a collection of nodes

**HTML DOM Node List**

The getElementsByTagName() method returns a **node list**. A node list is an array-like collection of nodes.

The following code selects all <p> nodes in a document:

**Example**

var x = document.getElementsByTagName("p");

The nodes can be accessed by an index number. To access the second <p> node you can write:

y = x[1];

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_nodelist)

**Note:** The index starts at 0.

**HTML DOM Node List Length**

The length property defines the number of nodes in a node list:

**Example**

var myNodelist = document.getElementsByTagName("p");  
document.getElementById("demo").innerHTML = myNodelist.length;

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_nodelist_length)

Example explained:

1. Get all <p> elements in a node list
2. Display the length of the node list

The length property is useful when you want to loop through the nodes in a node list:

**Example**

Change the background color of all <p> elements in a node list:

var myNodelist = document.getElementsByTagName("p");  
var i;  
for (i = 0; i < myNodelist.length; i++) {  
    myNodelist[i].style.backgroundColor = "red";  
}

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_nodelist_loop)

|  |  |
| --- | --- |
| **Note** | **A node list is not an array!**  A node list may look like an array, but it is not. You can loop through the node list and refer to its nodes like an array. However, you cannot use Array Methods, like valueOf() or join() on the node list. |

**JavaScript Window - The Browser Object Model**

[« Previous](http://www.w3schools.com/js/js_htmldom_nodelist.asp)

[Next Chapter »](http://www.w3schools.com/js/js_window_screen.asp)

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.

**The Browser Object Model (BOM)**

There are no official standards for the **B**rowser **O**bject **M**odel (BOM).

Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

**The Window Object**

The **window** object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

Global variables are properties of the window object.

Global functions are methods of the window object.

Even the document object (of the HTML DOM) is a property of the window object:

window.document.getElementById("header");

is the same as:

document.getElementById("header");

**Window Size**

Three different properties can be used to determine the size of the browser window (the browser viewport, NOT including toolbars and scrollbars).

For Internet Explorer, Chrome, Firefox, Opera, and Safari:

* window.innerHeight - the inner height of the browser window
* window.innerWidth - the inner width of the browser window

For Internet Explorer 8, 7, 6, 5:

* document.documentElement.clientHeight
* document.documentElement.clientWidth
* or
* document.body.clientHeight
* document.body.clientWidth

A practical JavaScript solution (covering all browsers):

**Example**

var w = window.innerWidth  
|| document.documentElement.clientWidth  
|| document.body.clientWidth;  
  
var h = window.innerHeight  
|| document.documentElement.clientHeight  
|| document.body.clientHeight;

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_win_inner)

The example displays the browser window's height and width: (NOT including toolbars/scrollbars)

**Other Window Methods**

Some other methods:

* window.open() - open a new window
* window.close() - close the current window
* window.moveTo() -move the current window
* window.resizeTo() -resize the current window

**JavaScript Window Screen**

[« Previous](http://www.w3schools.com/js/js_window.asp)

[Next Chapter »](http://www.w3schools.com/js/js_window_location.asp)

The window.screen object contains information about the user's screen.

**Window Screen**

The **window.screen** object can be written without the window prefix.

Properties:

* screen.width
* screen.height
* screen.availWidth
* screen.availHeight
* screen.colorDepth
* screen.pixelDepth

**Window Screen Width**

The screen.width property returns the width of the visitor's screen in pixels.

**Example**

Display the width of the screen in pixels:

document.getElementById("demo").innerHTML =  
"Screen Width: " + screen.width;

Result will be:

Screen Width: 1280

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_width)

**Window Screen Height**

The screen.height property returns the height of the visitor's screen in pixels.

**Example**

Display the height of the screen in pixels:

document.getElementById("demo").innerHTML =  
"Screen Height: " + screen.height;

Result will be:

Screen Height: 1024

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_height)

**Window Screen Available Width**

The screen.availWidth property returns the width of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

**Example**

Display the available width of the screen in pixels:

document.getElementById("demo").innerHTML =  
"Available Screen Width: " + screen.availWidth;

Result will be:

Available Screen Width: 1280

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_availwidth)

**Window Screen Available Height**

The screen.availHeight property returns the height of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

**Example**

Display the available height of the screen in pixels:

document.getElementById("demo").innerHTML =  
"Available Screen Height: " + screen.availHeight;

Result will be:

Available Screen Height: 984

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_availheight)

**Window Screen Color Depth**

The screen.colorDepth property returns the number of bits used to display one color.

All modern computers use 24 bit or 32 bit hardware for color resolution:

* 24 bits =      16,777,216 different "True Colors"
* 32 bits = 4,294,967,296 different "Deep Colors"

Older computers used 16 bits: 65,536 different "High Colors" resolution.

Very old computers, and old cell phones used 8 bits: 256 different "VGA colors".

**Example**

Display the color depth of the screen in bits:

document.getElementById("demo").innerHTML =   
"Screen Color Depth: " + screen.colorDepth;

Result will be:

Screen Color Depth: 24

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_colorDepth)

|  |  |
| --- | --- |
| **Note** | The #rrggbb (rgb) values used in HTML represents "True Colors" (16,777,216 different colors) |

**Window Screen Pixel Depth**

The screen.pixelDepth property returns the pixel depth of the screen.

**Example**

Display the pixel depth of the screen in bits:

document.getElementById("demo").innerHTML =  
"Screen Pixel Depth: " + screen.pixelDepth;

Result will be:

Screen Pixel Depth: 24

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_pixelDepth)

|  |  |
| --- | --- |
| **Note** | For modern computers, Color Depth and Pixel Depth are equal. |

**JavaScript Window Location**

[« Previous](http://www.w3schools.com/js/js_window_screen.asp)

[Next Chapter »](http://www.w3schools.com/js/js_window_history.asp)

The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.

**Window Location**

The **window.location** object can be written without the window prefix.

Some examples:

* window.location.href returns the href (URL) of the current page
* window.location.hostname returns the domain name of the web host
* window.location.pathname returns the path and filename of the current page
* window.location.protocol returns the web protocol used (http:// or https://)
* window.location.assign loads a new document

**Window Location Href**

The **window.location.href** property returns the URL of the current page.

**Example**

Display the href (URL) of the current page:

document.getElementById("demo").innerHTML =  
"Page location is " + window.location.href;

Result is:

Page location is http://www.w3schools.com/js/js\_window\_location.asp

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_href)

**Window Location Hostname**

The **window.location.hostname** property returns the name of the internet host (of the current page).

**Example**

Display the name of the host:

document.getElementById("demo").innerHTML =  
"Page hostname is " + window.location.hostname;

Result is:

Page hostname is www.w3schools.com

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_host)

**Window Location Pathname**

The **window.location.pathname** property returns the pathname of the current page.

**Example**

Display the path name of the current URL:

document.getElementById("demo").innerHTML =  
"Page path is " + window.location.pathname;

Result is:

/js/js\_window\_location.asp

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_pathname)

**Window Location Protocol**

The **window.location.protocol** property returns the web protocol of the page.

**Example**

Display the web protocol:

document.getElementById("demo").innerHTML =  
"Page protocol is " + window.location.protocol;

Result is:

Page protocol is http:

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_protocol)

**Window Location Assign**

The **window.location.assign()** method loads a new document.

**Example**

Load a new document:

<html>  
<head>  
<script>  
function newDoc() {  
    window.location.assign("http://www.w3schools.com")  
}  
</script>  
</head>  
<body>  
  
<input type="button" value="Load new document" onclick="newDoc()">  
  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_assign)

**JavaScript Window History**

[« Previous](http://www.w3schools.com/js/js_window_location.asp)

[Next Chapter »](http://www.w3schools.com/js/js_window_navigator.asp)

The window.history object contains the browsers history.

**Window History**

The **window.history** object can be written without the window prefix.

To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

* history.back() - same as clicking back in the browser
* history.forward() - same as clicking forward in the browser

**Window History Back**

The history.back() method loads the previous URL in the history list.

This is the same as clicking the Back button in the browser.

**Example**

Create a back button on a page:

<html>  
<head>  
<script>  
function goBack() {  
    window.history.back()  
}  
</script>  
</head>  
<body>  
  
<input type="button" value="Back" onclick="goBack()">  
  
</body>  
</html>

The output of the code above will be:

**Window History Forward**

The history forward() method loads the next URL in the history list.

This is the same as clicking the Forward button in the browser.

**Example**

Create a forward button on a page:

<html>  
<head>  
<script>  
function goForward() {  
    window.history.forward()  
}  
</script>  
</head>  
<body>  
  
<input type="button" value="Forward" onclick="goForward()">  
  
</body>  
</html>

The output of the code above will be:

**JavaScript Window Navigator**

[« Previous](http://www.w3schools.com/js/js_window_history.asp)

[Next Chapter »](http://www.w3schools.com/js/js_popup.asp)

The window.navigator object contains information about the visitor's browser.

**Window Navigator**

The **window.navigator** object can be written without the window prefix.

Some examples:

* navigator.appName
* navigator.appCodeName
* navigator.platform

**Navigator Cookie Enabled**

The property cookieEnabled returns true if cookies are enabled, otherwise false:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML =  
"Cookies Enabled is " + navigator.cookieEnabled;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_cookieenabled)

**The Browser Names**

The properties **appName** and **appCodeName** return the name of the browser:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML =  
"Name is " + navigator.appName + ". Code name is " + navigator.appCodeName;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_appcodename)

|  |  |
| --- | --- |
| **Note** | Did you know?  IE11, Chrome, Firefox, and Safari return appName "Netscape".  Chrome, Firefox, IE, Safari, and Opera all return appCodeName "Mozilla". |

**The Browser Engine**

The property **product** returns the engine name of the browser:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = navigator.product;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_product)

**The Browser Version I**

The property **appVersion** returns version information about the browser:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = navigator.appVersion;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_appversion)

**The Browser Version II**

The property **userAgent also** returns version information about the browser:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = navigator.userAgent;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_userAgent)

**Warning !!!**

The information from the navigator object can often be misleading, and should not be used to detect browser versions because:

* Different browsers can use the same name
* The navigator data can be changed by the browser owner
* Some browsers misidentify themselves to bypass site tests
* Browsers cannot report new operating systems, released later than the browser

**The Browser Platform**

The property **platform** returns the browser platform (operating system):

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = navigator.platform;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_platform)

**The Browser Language**

The property **language** returns the browser's language:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = navigator.language;  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_language)

**Is Java Enabled?**

The method **javaEnabled()** returns true if Java is enabled:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = navigator.javaEnabled();  
</script>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_javaenabled)

**JavaScript Popup Boxes**

[« Previous](http://www.w3schools.com/js/js_window_navigator.asp)

[Next Chapter »](http://www.w3schools.com/js/js_timing.asp)

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

**Alert Box**

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

**Syntax**

window.alert("*sometext*");

The **window.alert()** method can be written without the window prefix.

**Example**

alert("I am an alert box!");

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_alert)

**Confirm Box**

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

**Syntax**

window.confirm("*sometext*");

The **window.confirm()** method can be written without the window prefix.

**Example**

var r = confirm("Press a button");  
if (r == true) {  
    x = "You pressed OK!";  
} else {  
    x = "You pressed Cancel!";  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_confirm)

**Prompt Box**

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

**Syntax**

window.prompt("*sometext*","*defaultText*");

The **window.prompt()** method can be written without the window prefix.

**Example**

var person = prompt("Please enter your name", "Harry Potter");  
if (person != null) {  
    document.getElementById("demo").innerHTML =  
    "Hello " + person + "! How are you today?";  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_prompt)

**Line Breaks**

To display line breaks inside a popup box, use a back-slash followed by the character n.

**Example**

alert("Hello\nHow are you?");

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_alert2)

**JavaScript Timing Events**

[« Previous](http://www.w3schools.com/js/js_popup.asp)

[Next Chapter »](http://www.w3schools.com/js/js_cookies.asp)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | JavaScript can be executed in time-intervals.  This is called timing events. |

**Timing Events**

The window object allows execution of code at specified time intervals.

These time intervals are called timing events.

The two key methods to use with JavaScript are:

* setTimeout(*function, milliseconds*)  
  Executes a function, after waiting a specified number of milliseconds.
* setInterval(*function, milliseconds*)  
  Same as setTimeout(), but repeats the execution of the function continuously.

|  |  |
| --- | --- |
| **Note** | The setTimeout() and setInterval() are both methods of the HTML DOM Window object. |

**The setTimeout() Method**

window.setTimeout(*function*, *milliseconds*);

The **window.setTimeout()** method can be written without the window prefix.

The first parameter is a function to be executed.

The second parameter indicates the number of milliseconds before execution.

**Example**

Click a button. Wait 3 seconds, and the page will alert "Hello":

<b<button onclick="setTimeout(myFunction, 3000)">Try it</button>  
  
<script>  
function myFunction() {  
    alert('Hello');  
}  
</script>

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing1)

**How to Stop the Execution?**

The clearTimeout() method stops the execution of the function specified in setTimeout().

window.clearTimeout(*timeoutVariable*)

The **window.clearTimeout()** method can be written without the window prefix.

The clearTimeout() method uses the variable returned from setTimeout():

myVar = setTimeout(*function*, *milliseconds*);  
clearTimeout(myVar);

If the function has not already been executed, you can stop the execution by calling the clearTimeout() method:

**Example**

Same example as above, but with an added "Stop" button:

<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>  
  
<button onclick="clearTimeout(myVar)">Stop it</button>

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_settimeout2)

**The setInterval() Method**

The setInterval() method repeats a given function at every given time-interval.

window.setInterval(*function*, *milliseconds*);

The **window.setInterval()** method can be written without the window prefix.

The first parameter is the function to be executed.

The second parameter indicates the length of the time-interval between each execution.

This example executes a function called "myTimer" once every second (like a digital watch).

**Example**

Display the current time:

var myVar = setInterval(myTimer, 1000);  
  
function myTimer() {  
    var d = new Date();  
    document.getElementById("demo").innerHTML = d.toLocaleTimeString();  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_setinterval2)

|  |  |
| --- | --- |
| **Note** | There are 1000 milliseconds in one second. |

**How to Stop the Execution?**

The clearInterval() method stops the executions of the function specified in the setInterval() method.

window.clearInterval(*timerVariable*)

The **window.clearInterval()** method can be written without the window prefix.

The clearInterval() method uses the variable returned from setInterval():

myVar = setInterval(*function*, *milliseconds*);  
clearInterval(myVar);

**Example**

Same example as above, but we have added a "Stop time" button:

<p id="demo"></p>  
  
<button onclick="clearInterval(myVar)">Stop time</button>  
  
<script>  
var myVar = setInterval(myTimer, 1000);  
function myTimer() {  
    var d = new Date();  
    document.getElementById("demo").innerHTML = d.toLocaleTimeString();  
}  
</script>

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_setinterval3)

Examples

**More Examples**

[Another simple timing](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing2)

[A clock created with a timing event](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing_clock)

**JavaScript Cookies**

[« Previous](http://www.w3schools.com/js/js_timing.asp)

[Next Chapter »](http://www.w3schools.com/js/js_libraries.asp)

Cookies let you store user information in web pages.

**What are Cookies?**

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

* When a user visits a web page, his name can be stored in a cookie.
* Next time the user visits the page, the cookie "remembers" his name.

Cookies are saved in name-value pairs like:

username=John Doe

When a browser request a web page from a server, cookies belonging to the page is added to the request. This way the server gets the necessary data to "remember" information about users.

**Create a Cookie with JavaScript**

JavaScript can create, read, and delete cookies with the **document.cookie** property.

With JavaScript, a cookie can be created like this:

document.cookie="username=John Doe";

You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";

With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

**Read a Cookie with JavaScript**

With JavaScript, cookies can be read like this:

var x = document.cookie;

|  |  |
| --- | --- |
| **Note** | document.cookie will return all cookies in one string much like: cookie1=value; cookie2=value; cookie3=value; |

**Change a Cookie with JavaScript**

With JavaScript, you can change a cookie the same way as you create it:

document.cookie="username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

The old cookie is overwritten.

**Delete a Cookie with JavaScript**

Deleting a cookie is very simple. Just set the expires parameter to a passed date:

document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC";

Note that you don't have to specify a cookie value when you delete a cookie.

**The Cookie String**

The document.cookie property looks like a normal text string. But it is not.

Even if you write a whole cookie string to document.cookie, when you read it out again, you can only see the name-value pair of it.

If you set a new cookie, older cookies are not overwritten. The new cookie is added to document.cookie, so if you read document.cookie again you will get something like:

cookie1=value; cookie2=value;

If you want to find the value of one specified cookie, you must write a JavaScript function that searches for the cookie value in the cookie string.

**JavaScript Cookie Example**

In the example to follow, we will create a cookie that stores the name of a visitor.

The first time a visitor arrives to the web page, he will be asked to fill in his name. The name is then stored in a cookie.

The next time the visitor arrives at the same page, he will get a welcome message.

For the example we will create 3 JavaScript functions:

1. A function to set a cookie value
2. A function to get a cookie value
3. A function to check a cookie value

**A Function to Set a Cookie**

First, we create a function that stores the name of the visitor in a cookie variable:

**Example**

function setCookie(cname, cvalue, exdays) {  
    var d = new Date();  
    d.setTime(d.getTime() + (exdays\*24\*60\*60\*1000));  
    var expires = "expires="+d.toUTCString();  
    document.cookie = cname + "=" + cvalue + "; " + expires;  
}

**Example explained:**

The parameters of the function above are the name of the cookie (cname), the value of the cookie (cvalue), and the number of days until the cookie should expire (exdays).

The function sets a cookie by adding together the cookiename, the cookie value, and the expires string.

**A Function to Get a Cookie**

Then, we create a function that returns the value of a specified cookie:

**Example**

function getCookie(cname) {  
    var name = cname + "=";  
    var ca = document.cookie.split(';');  
    for(var i=0; i<ca.length; i++) {  
        var c = ca[i];  
        while (c.charAt(0)==' ') c = c.substring(1);  
        if (c.indexOf(name) == 0) return c.substring(name.length,c.length);  
    }  
    return "";  
}

**Function explained:**

Take the cookiename as parameter (cname).

Create a variable (name) with the text to search for (cname + "=").

Split document.cookie on semicolons into an array called ca (ca = document.cookie.split(';')).

Loop through the ca array (i=0;i<ca.length;i++), and read out each value c=ca[i]).

If the cookie is found (c.indexOf(name) == 0), return the value of the cookie (c.substring(name.length,c.length).

If the cookie is not found, return "".

**A Function to Check a Cookie**

Last, we create the function that checks if a cookie is set.

If the cookie is set it will display a greeting.

If the cookie is not set, it will display a prompt box, asking for the name of the user, and stores the username cookie for 365 days, by calling the setCookie function:

**Example**

function checkCookie() {  
    var username=getCookie("username");  
    if (username!="") {  
        alert("Welcome again " + username);  
    }else{  
        username = prompt("Please enter your name:", "");  
        if (username != "" && username != null) {  
            setCookie("username", username, 365);  
        }  
    }  
}

**All Together Now**

**Example**

function setCookie(cname, cvalue, exdays) {  
    var d = new Date();  
    d.setTime(d.getTime() + (exdays\*24\*60\*60\*1000));  
    var expires = "expires="+d.toUTCString();  
    document.cookie = cname + "=" + cvalue + "; " + expires;  
}  
  
function getCookie(cname) {  
    var name = cname + "=";  
    var ca = document.cookie.split(';');  
    for(var i=0; i<ca.length; i++) {  
        var c = ca[i];  
        while (c.charAt(0)==' ') c = c.substring(1);  
        if (c.indexOf(name) == 0) return c.substring(name.length, c.length);  
    }  
    return "";  
}  
  
function checkCookie() {  
    var user = getCookie("username");  
    if (user != "") {  
        alert("Welcome again " + user);  
    } else {  
        user = prompt("Please enter your name:", "");  
        if (user != "" && user != null) {  
            setCookie("username", user, 365);  
        }  
    }  
}

[Try it yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_cookie_username)

**JavaScript Libraries**

[« Previous](http://www.w3schools.com/js/js_cookies.asp)

[Next Chapter »](http://www.w3schools.com/js/js_lib_jquery.asp)

JavaScript libraries - jQuery, Prototype, MooTools.

**JavaScript Frameworks (Libraries)**

Advanced JavaScript programming (especially the complex handling of browser differences), can often be very difficult and time-consuming to work with.

To deal with these difficulties, a lot of **JavaScript (helper) libraries** have been developed.

These JavaScript libraries are often called **JavaScript frameworks**.

In this tutorial, we will take a look at some of the most popular JavaScript frameworks:

* jQuery
* Prototype
* MooTools

All of these frameworks have functions for common JavaScript tasks like animations, DOM manipulation, and Ajax handling.

In this tutorial we will teach you how start using them, to make JavaScript programming easier, safer, and much more exciting.

**jQuery**

**jQuery** is the most popular JavaScript framework on the Internet today.

It uses CSS selectors to access and manipulate HTML elements (DOM Objects) on a web page.

jQuery also provides a companion UI (user interface) framework and numerous other plug-ins.

Many of the largest companies on the Web use jQuery:

* Google
* Microsoft
* IBM
* Netflix

You will find an excellent [jQuery Tutorial](http://www.w3schools.com/jquery/default.asp) here at W3Schools.

**Prototype**

**Prototype** is a JavaScript library that provides a simple API to perform common web tasks.

**API** is short for Application Programming Interface. It is a library of properties and methods for manipulating the HTML DOM.

Prototype enhances JavaScript by providing classes and inheritance.

**MooTools**

**MooTools** is also a framework that offers an API to make common JavaScript programming easier.

MooTools also includes some lightweight effects and animation functions.

**Other Frameworks**

Here are some other frameworks not covered in this short overview:

**YUI** - The Yahoo! User Interface Framework is a large library that covers a lot of functions, from simple JavaScript utilities to complete internet widgets.

**Ext JS** - Customizable widgets for building rich Internet applications.

**Dojo** - A toolkit designed around packages for DOM manipulation, events, widgets, and more.

**script.aculo.us** - Open-source JavaScript framework for visual effects and interface behaviors.

**UIZE** - Widgets, AJAX, DOM, templates, and more.

**CDN - Content Delivery Networks**

You always want your web pages to be as fast as possible. You want to keep the size of your pages as small as possible, and you want the browser to cache as much as possible.

If many different web sites use the same JavaScript framework, it makes sense to host the framework library in a common location for every web page to share.

A CDN (Content Delivery Network) solves this. A CDN is a network of servers containing shared code libraries.  
  
Google provides a free CDN for a number of JavaScript libraries, including:

* jQuery
* Prototype
* MooTools
* Dojo
* Yahoo! YUI

To use a JavaScript framework library in your web pages, just include the library in a <script> tag:

**Including jQuery**

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>

**Using Frameworks**

Before you decide to use a JavaScript framework for your web pages, it might be a good idea to test the framework first.

JavaScript frameworks are very easy to test. You don't have to install them on your computer, and there are no setup programs.

Normally you just have to reference a library file from your web page.

In the next chapter of this tutorial we will walk you through a test process for jQuery.

**JavaScript - Testing jQuery**

[« Previous](http://www.w3schools.com/js/js_libraries.asp)

[Next Chapter »](http://www.w3schools.com/js/js_lib_prototype.asp)

Testing JavaScript Framework Libraries - jQuery

**Including jQuery**

To test a JavaScript library, you need to include it in your web page.

To include a library, use the <script> tag with the src attribute set to the URL of the library:

**Including jQuery**

<!DOCTYPE html>  
<html>  
  
<head>  
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>  
</head>  
  
<body>  
.  
.  
.

**jQuery Described**

The main jQuery function is the $() function (the jQuery function). If you pass DOM objects to this function, it returns jQuery objects, with jQuery functionality added to them.

jQuery allows you to select elements by CSS selectors.

In JavaScript, you can assign a function to handle the window's load event:

**The JavaScript Way:**

function myFunction() {  
    var obj = document.getElementById("h01");  
    obj.innerHTML = "Hello jQuery";  
}  
onload = myFunction;

The jQuery equivalent is different:

**The jQuery Way:**

function myFunction() {  
    $("#h01").html("Hello jQuery");  
}  
$(document).ready(myFunction);

The last line of the code above, passes the HTML DOM document object to jQuery: $(document).

When you pass DOM objects to jQuery, jQuery returns new jQuery objects wrapped around the HTML DOM objects.

The jQuery function returns a new jQuery object, where ready() is a method.

Since functions are variables in jQuery, myFunction can be passed as a variable to the jQuery ready() method.

|  |  |
| --- | --- |
| **Note** | jQuery returns a jQuery object, different from the DOM object that was passed. The jQuery object has properties and methods, different from the DOM object. You cannot use HTML DOM properties and methods on jQuery objects. |

**Testing jQuery**

Try the following example:

**Example**

<!DOCTYPE html>  
<html>  
  
<head>  
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">  
</script>  
<script>  
function myFunction() {  
    $("#h01").html("Hello jQuery")  
}  
$(document).ready(myFunction);  
</script>  
</head>  
  
<body>  
<h1 id="h01"></h1>  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_lib_jquery)

Also try this one:

**Example**

<!DOCTYPE html>  
<html>  
  
<head>  
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">  
</script>  
<script>  
function myFunction() {  
    $("#h01").attr("style", "color:red").html("Hello jQuery")  
}  
$(document).ready(myFunction);  
</script>  
</head>  
  
<body>  
<h1 id="h01"></h1>  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_lib_jquery2)

As you can see from the example above, jQuery allows chaining.

Chaining is a handy way to perform multiple tasks on one object.

Want to learn more? You will find an excellent [jQuery Tutorial](http://www.w3schools.com/jquery/default.asp) here at W3Schools.

**JavaScript - Testing Prototype**

[« Previous](http://www.w3schools.com/js/js_lib_jquery.asp)

[Next Chapter »](http://www.w3schools.com/js/js_examples.asp)

Testing JavaScript Framework Libraries - Prototype

**Including Prototype**

To test a JavaScript library, you need to include it in your web page.

To include a library, use the <script> tag with the src attribute set to the URL of the library:

**Including Prototype**

<!DOCTYPE html>  
<html>  
  
<head>  
<script src="http://ajax.googleapis.com/ajax/libs/prototype/1.7.2.0/prototype.js"></script>  
</head>  
  
<body>  
.  
.  
.

**Prototype Described**

Prototype provides functions to make HTML DOM programming easier.

Like jQuery, Prototype has its $() function.

The $() function accepts HTML DOM element id values (or DOM elements), and adds new functionality to DOM objects.

Unlike jQuery, Prototype has no ready() method to take the place of window.onload(). Instead, Prototype adds extensions to the browser and the HTML DOM.

In JavaScript, you can assign a function to handle the window's load event:

**The JavaScript Way:**

function myFunction() {  
    var obj = document.getElementById("h01");  
    obj.innerHTML = "Hello Prototype";  
}  
onload = myFunction;

The Prototype equivalent is different:

**The Prototype Way:**

function myFunction() {  
    $("h01").insert("Hello Prototype!");  
}  
Event.observe(window, "load", myFunction);

Event.observe() accepts three arguments:

* The HTML DOM or BOM (Browser Object Model) object you want to handle
* The event you want to handle
* The function you want to call

**Testing Prototype**

Try the following example:

**Example**

<!DOCTYPE html>  
<html>  
  
<head>  
<script src="http://ajax.googleapis.com/ajax/libs/prototype/1.7.2.0/prototype.js"></script>  
<script>  
function myFunction() {  
    $("h01").insert("Hello Prototype!");  
}  
Event.observe(window, "load", myFunction);  
</script>  
</head>  
  
<body>  
<h1 id="h01"></h1>  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_lib_prototype)

Also try this one:

**Example**

<!DOCTYPE html>  
<html>  
  
<head>  
<script src="http://ajax.googleapis.com/ajax/libs/prototype/1.7.2.0/prototype.js"></script>  
<script>  
function myFunction() {  
    $("h01").writeAttribute("style", "color:red").insert("Hello Prototype!");  
}  
Event.observe(window, "load", myFunction);  
</script>  
</head>  
  
<body>  
<h1 id="h01"></h1>  
</body>  
</html>

[Try it Yourself »](http://www.w3schools.com/js/tryit.asp?filename=tryjs_lib_prototype2)

As you can see from the example above, like jQuery, Prototype allows chaining.

Chaining is a handy way to perform multiple tasks on one object.

**JavaScript Examples**

[« Previous](http://www.w3schools.com/js/js_lib_prototype.asp)

[Next Chapter »](http://www.w3schools.com/js/js_dom_examples.asp)

**What can JavaScript do?**

[JavaScript can change HTML content](http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_inner_html)  
[JavaScript can change HTML attributes](http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_lightbulb)  
[JavaScript can change CSS style](http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_style)  
[JavaScript can validate input](http://www.w3schools.com/js/tryit.asp?filename=tryjs_intro_validate)

**JavaScript Output**

[Writing into an window alert box](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_alert)  
[Writing into the HTML output](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_write)  
[Writing into an HTML element](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_dom)  
[Writing into the browser console](http://www.w3schools.com/js/tryit.asp?filename=tryjs_output_console)

[Output explained](http://www.w3schools.com/js/js_output.asp)

**Where to Insert JavaScript**

[JavaScript in <head>](http://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_head)  
[JavaScript in <body>](http://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_body)  
[JavaScript in an external file](http://www.w3schools.com/js/tryit.asp?filename=tryjs_whereto_external)

[Placement explained](http://www.w3schools.com/js/js_whereto.asp)

**JavaScript Syntax**

[JavaScript statements](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_statements)  
[JavaScript numbers](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_numbers)  
[JavaScript strings](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_strings)  
[JavaScript expressions](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_expressions)  
[JavaScript keywords](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_keywords)  
[JavaScript variables](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_variables)  
[JavaScript assignment](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_assign)  
[JavaScript operators](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_operators)  
[JavaScript comments](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_comments)  
[JavaScript is case sensitive](http://www.w3schools.com/js/tryit.asp?filename=tryjs_syntax_case)

[Syntax explained](http://www.w3schools.com/js/js_syntax.asp)

**JavaScript Statements**

[JavaScript statements are commands to the browser](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statement)  
[JavaScript code is a sequence of statements](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements)  
[JavaScript statements are separated with semicolon](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_semicolon1)  
[Multiple statement on one line is allowed](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_semicolon2)  
[JavaScript statements can be grouped together in code blocks](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_blocks)  
[You can break a code line after an operator or a comma.](http://www.w3schools.com/js/tryit.asp?filename=tryjs_statements_linebreak)

[Statements explained](http://www.w3schools.com/js/js_statements.asp)

**JavaScript Comments**

[Single line comments](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments1)  
[Single line comments at the end of a line](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments5)  
[Multiple lines comments](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments2)  
[Single line comment to prevent execution](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments3)  
[Multiple lines comment to prevent execution](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comments4)

[Comments explained](http://www.w3schools.com/js/js_comments.asp)

**JavaScript Variables**

[JavaScript variables](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables)  
[JavaScript variables as algebra](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_total)  
[JavaScript numbers and strings](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_types)  
[JavaScript var keyword.](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_create)  
[Declaring many variables in one statement](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multi)  
[Declaring many variables multiline](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_multiline)  
[A variable without a value returns the value undefined](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_undefined)  
[Re-declaring a variable will not destroy the value](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_redefine)  
[Adding JavaScript numbers](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_numbers)  
[Adding JavaScript strings](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_strings)  
[Adding strings and numbers](http://www.w3schools.com/js/tryit.asp?filename=tryjs_variables_add_string_number)

[Variables explained](http://www.w3schools.com/js/js_variables.asp)

**JavaScript Arithmetic**

[The addition (+) operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_add)  
[The subtraction (-) operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_sub)  
[The multiplication (\*) operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mult)  
[The division (/) operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_div)  
[The modulus (%) operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_mod)  
[The increment (++) operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_incr)  
[The decrement (--) operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_decr)

[Arithmetic explained](http://www.w3schools.com/js/js_operators.asp)

**JavaScript Assignment**

[The = assignment operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_equal)  
[The += assignment operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_plusequal)  
[The -= assignment operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_minequal)  
[The \*= assignment operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_multequal)  
[The /= assignment operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_divequal)  
[The %= assignment operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_modequal)

[Assignment explained](http://www.w3schools.com/js/js_operators.asp)

**JavaScript String Concatenation**

[Adding two strings together using the concatenating (+) operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat1)  
[Adding two strings together with a space in the first string](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat2)  
[Adding two strings together with a space in between](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat3)  
[Adding two strings together using using the += operator](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat4)  
[Adding strings and numbers](http://www.w3schools.com/js/tryit.asp?filename=tryjs_oper_concat5)

[Concatenation explained](http://www.w3schools.com/js/js_operators.asp)

**JavaScript Data Types**

[Declare (create) strings](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string)  
[Declare (create) numbers](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_numbers)  
[Declare (create) an array](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_array)  
[Declare (create) an object](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_object)  
[Find the type of a variable](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_typeof)  
[Adding two numbers and a string](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_1)  
[Adding a string and two numbers](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_addstrings_2)  
[An undefined variable](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_undefined)  
[An empty variable](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_empty)

[Data types explained](http://www.w3schools.com/js/js_datatypes.asp)

**JavaScript Objects**

[Create a JavaScript variable](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_variable)  
[Create a JavaScript object](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_object)  
[Create a person object (single line)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_create_1)  
[Create a person object (multiple lines)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_create_2)  
[Access object properties using .property](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_1)  
[Access object properties using [property]](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_properties_2)  
[Access a function property as a method](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_method)  
[Access a function property as a property](http://www.w3schools.com/js/tryit.asp?filename=tryjs_objects_function)

[Objects explained](http://www.w3schools.com/js/js_objects.asp)

**JavaScript Functions**

[A simple function](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function1)  
[A function with an argument](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function2)  
[A function with an argument 2](http://www.w3schools.com/js/tryit.asp?filename=tryjs_functionarg2)  
[A function that returns a value](http://www.w3schools.com/js/tryit.asp?filename=tryjs_function_return)  
[A function that converts Fahrenheit to Celsius](http://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius)  
[A function call without ()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_farenheit_to_celsius_2)

[Functions explained](http://www.w3schools.com/js/js_functions.asp)

**JavaScript Events**

[An onclick event changes an HTML element](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick1)  
[An onclick event changes its own element](http://www.w3schools.com/js/tryit.asp?filename=tryjs_event_onclick)  
[An onclick event calls a function](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events1)

[Events explained](http://www.w3schools.com/js/js_events.asp)

**JavaScript Strings**

[Strings can be written with single or double quotes.](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strings)  
[Show some string examples](http://www.w3schools.com/js/tryit.asp?filename=tryjs_datatypes_string)  
[Backslash before quotes accepts quotes as quotes.](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strings_escape)  
[Find the length of a string](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_length)  
[You can break text string with a backslash.](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_break)  
[You cannot break code with a backslash.](http://www.w3schools.com/js/tryit.asp?filename=tryjs_strings_codebreak)  
[Find the position of the first occurrence of a text in a string - indexOf()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_indexof)  
[Search for a text in a string and return the text if found - match()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_match)  
[Replace characters in a string - replace()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace)  
[Convert string to upper case - toUpperCase()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_toupper)  
[Convert string to lower case - toLowerCase()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_tolower)  
[Split a string into an array - split()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_split)

[Strings explained](http://www.w3schools.com/js/js_strings.asp)

**JavaScript Numbers**

[Number are considered accurate only up to 15 digits](http://www.w3schools.com/js/tryit.asp?filename=tryjs_inaccurate1)  
[Floating point arithmetic is not always 100% accurate](http://www.w3schools.com/js/tryit.asp?filename=tryjs_inaccurate2)  
[But it helps to multiply and divide by 10](http://www.w3schools.com/js/tryit.asp?filename=tryjs_inaccurate3)  
[Constants, preceded by 0x, are interpreted as hexadecimal](http://www.w3schools.com/js/tryit.asp?filename=tryjs_hex)  
[The toString() method can output numbers as hex, octal, and binary](http://www.w3schools.com/js/tryit.asp?filename=tryjs_tostring)  
[JavaScript will generate Infinity if you calculate a too large number](http://www.w3schools.com/js/tryit.asp?filename=tryjs_infinity)  
[Division by zero generates Infinity](http://www.w3schools.com/js/tryit.asp?filename=tryjs_infinity_zero)  
[A number divided by a string is not a number](http://www.w3schools.com/js/tryit.asp?filename=tryjs_isnan_false)

[Numbers explained](http://www.w3schools.com/js/js_numbers.asp)

**JavaScript Maths**

[Use Math.random() to return a random number between 0 and 1](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_random)  
[Use Math.round() to round a number](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_round)  
[Use Math.ceil() to round a number (up)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_ceil)  
[Use Math.floor() to round a number (down)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_floor)  
[Use Math.floor() and random() to return a random integer](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_floor_random)  
[Use Math.max() to return the number with the highest value](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_max)  
[Use Math.min() to return the number with the lowest value](http://www.w3schools.com/js/tryit.asp?filename=tryjs_math_min)  
[Convert Celsius to Fahrenheit](http://www.w3schools.com/js/tryit.asp?filename=tryjs_celsius)

[Maths explained](http://www.w3schools.com/js/js_math.asp)

**JavaScript Dates**

[Use Date() to display today's date and time](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_current)  
[Use getFullYear() display the year](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_getfullyear)  
[Use getTime() to calculate the number of milliseconds since 1970](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_gettime)  
[Use setFullYear() to set a specific date](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_setfullyear)  
[Use toUTCString() to convert today's date (according to UTC) to a string](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_toutcstring)  
[Use getDay() to display the weekday as a number](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_getday)  
[Use getDay() and an array to display the weekday as a name](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_weekday)  
[Display a clock](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing_clock)

[Dates explained](http://www.w3schools.com/js/js_dates.asp)

**JavaScript Arrays**

[Create an array](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array)  
[Join two arrays - concat()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat)  
[Join three arrays - concat()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_concat2)  
[Join all elements of an array into a string - join()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_join)  
[Remove the last element of an array - pop()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_pop)  
[Add new elements to the end of an array - push()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_push)  
[Reverse the order of the elements in an array - reverse()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_reverse)  
[Remove the first element of an array - shift()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_shift)  
[Select elements from an array - slice()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_slice)  
[Sort an array (alphabetically and ascending) - sort()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort)  
[Sort numbers (numerically and ascending) - sort()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort2)  
[Sort numbers (numerically and descending) - sort()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_sort3)  
[Add an element to position 2 in an array - splice()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_splice)  
[Convert an array to a string - toString()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_tostring)  
[Add new elements to the beginning of an array - unshift()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_unshift)

[Arrays explained](http://www.w3schools.com/js/js_arrays.asp)

**JavaScript Type Conversion**

[Display the typeof all variable types](http://www.w3schools.com/js/tryit.asp?filename=tryjs_typeof_all)  
[Display the constructor of all variable types](http://www.w3schools.com/js/tryit.asp?filename=tryjs_constructor_all)  
[Convert a number to a string using String()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_string)  
[Convert a number to a string using toString()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_number_tostring)  
[Find out if a variable is an array](http://www.w3schools.com/js/tryit.asp?filename=tryjs_array_isarray)  
[Find out if a variable is an date](http://www.w3schools.com/js/tryit.asp?filename=tryjs_date_isdate)

[Type Conversion Explained](http://www.w3schools.com/js/js_type_conversion.asp)

**JavaScript Booleans**

[Display the value of Boolean(10 > 9)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_expression1)  
[Display the value of 10 > 9](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_expression2)  
[Everything with a real value is true](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean)  
[The Boolean value of zero is false](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_zero)  
[The Boolean value of minus zero is false](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_minus)  
[The Boolean value of an empty string is false](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_empty)  
[The Boolean value of undefined is false](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_undefined)  
[The Boolean value of null is false](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_null)  
[The Boolean value of false is false](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_false)  
[The Boolean value of NaN is false](http://www.w3schools.com/js/tryit.asp?filename=tryjs_boolean_nan)

[Booleans explained](http://www.w3schools.com/js/js_booleans.asp)

**JavaScript Comparisons**

[Assign 5 to x, and display the value of (x == 8)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison1)  
[Assign 5 to x, and display the value of (x == 5)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison2)  
[Assign 5 to x, and display the value of (x === 5)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison4)  
[Assign 5 to x, and display the value of (x === "5")](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison3)  
[Assign 5 to x, and display the value of (x != 8)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison5)  
[Assign 5 to x, and display the value of (x !== 5)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison7)  
[Assign 5 to x, and display the value of (x !== "5")](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison6)  
[Assign 5 to x, and display the value of (x > 8)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison8)  
[Assign 5 to x, and display the value of (x < 8)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison9)  
[Assign 5 to x, and display the value of (x >= 8)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison10)  
[Assign 5 to x, and display the value of (x <= 8)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_comparison11)

[Comparisons explained](http://www.w3schools.com/js/js_comparisons.asp)

**JavaScript Conditionals**

[The if statement](http://www.w3schools.com/js/tryit.asp?filename=tryjs_ifthen)  
[The else statement](http://www.w3schools.com/js/tryit.asp?filename=tryjs_ifthenelse)  
[The else if statement](http://www.w3schools.com/js/tryit.asp?filename=tryjs_elseif)  
[Random link](http://www.w3schools.com/js/tryit.asp?filename=tryjs_randomlink)  
[Switch statement](http://www.w3schools.com/js/tryit.asp?filename=tryjs_switch)

[Conditionals explained](http://www.w3schools.com/js/js_if_else.asp)

**JavaScript Loops**

[For loop](http://www.w3schools.com/js/tryit.asp?filename=tryjs_fornext)  
[Looping through HTML headers](http://www.w3schools.com/js/tryit.asp?filename=tryjs_fornext_header)  
[While loop](http://www.w3schools.com/js/tryit.asp?filename=tryjs_while)  
[Do While loop](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dowhile)  
[Break a loop](http://www.w3schools.com/js/tryit.asp?filename=tryjs_break)  
[Break and continue a loop](http://www.w3schools.com/js/tryit.asp?filename=tryjs_continue)  
[Use a for...in statement to loop through the elements of an object](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_for_in)

[Loops explained](http://www.w3schools.com/js/js_loop_for.asp)

**JavaScript Error Handling**

[The try...catch statement](http://www.w3schools.com/js/tryit.asp?filename=tryjs_try_catch)  
[The try...catch statement with a confirm box](http://www.w3schools.com/js/tryit.asp?filename=tryjs_try_catch2)  
[The onerror event](http://www.w3schools.com/js/tryit.asp?filename=tryjs_onerror)

[Errors explained](http://www.w3schools.com/js/js_errors.asp)

**JavaScript Regular Expressions**

[Search for an expression in a string](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_search_regexp)  
[Search for an expression and replace it](http://www.w3schools.com/js/tryit.asp?filename=tryjs_string_replace_regexp)

[Regular Expressions Explained](http://www.w3schools.com/js/js_regexp.asp)

**JavaScript Objects**

[Creating a JavaScript variable](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_variable)  
[Creating a JavaScript object](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_object)  
[Creating a JavaScript object (single line)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_create_1)  
[Creating a JavaScript object (multiple lines)](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_create_2)  
[Creating a JavaScript object using new](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_create_new)  
[Creating JavaScript objects using a constructor](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_constructor)  
[Creating built-in JavaScript objects](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_builtin)  
[The best way to create JavaScript variables](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_best)  
[JavaScript objects are mutable](http://www.w3schools.com/js/tryit.asp?filename=tryjs_object_mutable)

[Objects explained](http://www.w3schools.com/js/js_object_definition.asp)

**JavaScript Object Properties**

[Accessing properties using .property](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties1)  
[Accessing properties using [property]](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties2)  
[Accessing properties using for in](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties_for_in)  
[Adding new properties to existing objects](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties3)  
[Deleting properties from objects](http://www.w3schools.com/js/tryit.asp?filename=tryjs_properties4)

[Object Properties explained](http://www.w3schools.com/js/js_properties.asp)

**JavaScript HTML DOM Examples**

[« Previous](http://www.w3schools.com/js/js_examples.asp)

[Next Chapter »](http://www.w3schools.com/js/js_input_examples.asp)

Examples of using JavaScript to access and manipulate DOM objects.

**The Document Object**

[Display all name/value pairs of cookies in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_cookie)  
[Display the domain name of the server that loaded the document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_domain)  
[Display the date and time the document was last modified](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_lastmodified)  
[Display the URL of the document that loaded the current document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_referrer)  
[Display the title of a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_title)  
[Display the full URL of a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_url)  
[Replace the content of a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_open)  
[Open a new window, and add some content](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_open2)  
[Display the number of elements with a specific name](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_getelementsbyname)  
[Display the number of elements with a specific tag name](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_getelementsbytagname)

[Document Object Explained](http://www.w3schools.com/js/js_htmldom.asp)

**The Anchors Collection**

[Find the number of anchors in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_anchors)  
[Find the innerHTML of the first anchor in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_anchors2)

**The Links Collection**

[Display the number of links in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_links)  
[Display the href attribute of the first link in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_links2)

**The Forms Collection**

[Find the number of forms in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_forms)  
[Find the name of the first form in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_forms2)

**The Images Collection**

[Return the number of images in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_images)  
[Return the id of the first image in a document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_doc_images2)

**CSS Manipulation**

[Change the visibility of an HTML element](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_visibility)  
[Change the background color of an HTML element](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_css_bgcolor)

**JavaScript HTML Input Examples**

[« Previous](http://www.w3schools.com/js/js_dom_examples.asp)

[Next Chapter »](http://www.w3schools.com/js/js_ex_dom.asp)

Examples of using JavaScript to access and manipulate HTML input objects.

**Button Object**

[Disable a button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_button_disabled)  
[Find the name of a button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_button_name)  
[Find the type of a button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_button_type)  
[Find the value of a button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_button_value)  
[Find the text displayed on a button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_button_text)  
[Find the id of the form a button belongs to](http://www.w3schools.com/js/tryit.asp?filename=tryjs_button_form)

**Form Object**

[Submit a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_submit)  
[Reset a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_reset)

[Find the value of each element in a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_elements)  
[Find the accepted character set of a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_acceptcharset)  
[Find the action attribute of a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_action)  
[Find the value of the enctype attribute in a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_enctype)  
[Find the number of elements in a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_length)  
[Find the method for sending form data](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_method)  
[Find the name of a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_name)  
[Find the target of a form](http://www.w3schools.com/js/tryit.asp?filename=tryjs_form_target)

**Option and Select Objects**

[Disable and enable a dropdown list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_select_disabled)  
[Get the id of the form that contains the dropdown list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_select_form)  
[Get the number of options in the dropdown list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_select_length)  
[Turn the dropdown list into a multiline list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_select_size)  
[Select multiple options in a dropdown list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_select_multiple)  
[Display the selected option in a dropdown list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_select_option)  
[Display all options from a dropdown list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_select_options)  
[Display the index of the selected option in a dropdown list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_option_index)  
[Change the text of the selected option](http://www.w3schools.com/js/tryit.asp?filename=try_dom_option_settext)  
[Remove options from a dropdown list](http://www.w3schools.com/js/tryit.asp?filename=try_dom_select_remove)

**JavaScript HTML Objects Examples**

[« Previous](http://www.w3schools.com/js/js_input_examples.asp)

[Next Chapter »](http://www.w3schools.com/js/js_events_examples.asp)

Examples of using JavaScript to access and manipulate HTML objects.

**Anchor Object**

[Find the value of the href attribute of a link](http://www.w3schools.com/js/tryit.asp?filename=tryjs_anchor_href)  
[Find the value of the hreflang attribute of a link](http://www.w3schools.com/js/tryit.asp?filename=tryjs_anchor_hreflang)  
[Find the value of the id attribute a link](http://www.w3schools.com/js/tryit.asp?filename=tryjs_anchor_id)  
[Find the value of the rel attribute of a link](http://www.w3schools.com/js/tryit.asp?filename=tryjs_anchor_rel)  
[Find the value of the target attribute of a link](http://www.w3schools.com/js/tryit.asp?filename=tryjs_anchor_target)  
[Find the value of the type attribute of a link](http://www.w3schools.com/js/tryit.asp?filename=tryjs_anchor_type)

**Area Object**

[Find the alternate text of an image-map area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_alt)  
[Find the coordinates of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_coords)  
[Find the shape of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_shape)  
[Find the href of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_href)  
[Find the protocol part of the href attribute of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_protocol)  
[Find the hostname part of the href attribute of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_hostname)  
[Find the port number part of the href attribute of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_port)  
[Find the pathname part of the href attribute of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_pathname)  
[Find the querystring part of the href attribute of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_search)  
[Find the hash part of the href attribute of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_hash)  
[Find the value of the target attribute of an area](http://www.w3schools.com/js/tryit.asp?filename=tryjs_area_target)

**Base Object**

[Find the base URL for all relative URLs on a page](http://www.w3schools.com/js/tryit.asp?filename=tryjs_base_href)  
[Find the base target for all links on a page](http://www.w3schools.com/js/tryit.asp?filename=tryjs_base_target)

**IFrame Object**

[Change the background color of an iframe](http://www.w3schools.com/js/tryit.asp?filename=tryjs_iframe_contentdocument)  
[Find the height of an iframe](http://www.w3schools.com/js/tryit.asp?filename=tryjs_iframe_height)  
[Find the value of the name attribute of an iframe](http://www.w3schools.com/js/tryit.asp?filename=tryjs_iframe_name)  
[Find the source (src) attribute of an iframe](http://www.w3schools.com/js/tryit.asp?filename=tryjs_iframe_src)  
[Change the source (src) attribute of an iframe](http://www.w3schools.com/js/tryit.asp?filename=tryjs_iframe_src2)

**Image Object**

[Find the alternate text of an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_img_alt)  
[Find the height of an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_img_height)  
[Click to display a high-resolution version of an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_img_lowsrc)  
[Find the source (src) of an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_img_src)  
[Change the source of an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_img_src_buttons)  
[Change the source of the lightbulb](http://www.w3schools.com/js/tryit.asp?filename=tryjs_lightbulb)  
[Find the value of the usemap attribute of an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_img_usemap)

**Table Objects**

[Change the width of a table border](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_table_border)  
[Change the padding of a table](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_table_padding)  
[Find the innerHTML of a cell](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_table_cells)  
[Create a caption for a table](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_table_createcaption)  
[Delete rows in a table](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_table_deleterow)  
[Add rows to a table](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_table_insertrow)  
[Change the content of a table cell](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_table_cellcontent)

**JavaScript HTML DOM Events Examples**

[« Previous](http://www.w3schools.com/js/js_ex_dom.asp)

[Next Chapter »](http://www.w3schools.com/js/js_ex_browser.asp)

Examples of using JavaScript to react to events

**Input Events**

[onblur - When a user leaves an input field](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onblur)  
[onchange - When a user changes the content of an input field](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onchange)  
[onchange - When a user selects a dropdown value](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_dropdown)  
[onfocus - When an input field gets focus](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onfocus)  
[onselect - When input text is selected](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onselect)  
[onsubmit - When a user clicks the submit button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onsubmit)  
[onreset - When a user clicks the reset button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onreset)  
[onkeydown - When a user is pressing/holding down a key](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onkeydown)  
[onkeypress - When a user is pressing/holding down a key](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onkeypress)  
[onkeyup - When the user releases a key](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onkeyup)  
[onkeyup - When the user releases a key](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onkeyup2)  
[onkeydown vs onkeyup - Both](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onkeydownup)

**Mouse Events**

[onmouseover/onmouseout - When the mouse passes over an element](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onmouse)  
[onmousedown/onmouseup - When pressing/releasing a mouse button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onmousedown)  
[onmousedown - When mouse is clicked: Alert which element](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_srcelement)  
[onmousedown - When mouse is clicked: Alert which button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onmousedown2)  
[onmousemove/onmouseout - When moving the mouse pointer over/out of an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onmousemove)  
[onmouseover/onmouseout - When moving the mouse over/out of an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onmouseover)  
[onmouseover an image map](http://www.w3schools.com/js/tryit.asp?filename=tryjs_imagemap)

**Click Events**

[Acting to the onclick event](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events)  
[onclick - When button is clicked](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onclick)  
[ondblclick - When a text is double-clicked](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_ondblclick)

**Load Events**

[onload - When the page has been loaded](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_body_onload)  
[onload - When an image has been loaded](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_img_onload)  
[onerror - When an error occurs when loading an image](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onerror)  
[onunload - When the browser closes the document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onunload)  
[onresize - When the browser window is resized](http://www.w3schools.com/js/tryit.asp?filename=tryjs_events_onresize)

**Others**

[What is the keycode of the key pressed?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_event_keycode)  
[What are the coordinates of the cursor?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_event_clientxy)  
[What are the coordinates of the cursor, relative to the screen?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_event_screenxy)  
[Was the shift key pressed?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_event_shiftkey)  
[Which event type occurred?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_event_type)

[Examples explained](http://www.w3schools.com/js/js_htmldom_events.asp)

**JavaScript Browser Objects Examples**

[« Previous](http://www.w3schools.com/js/js_events_examples.asp)

[Next Chapter »](http://www.w3schools.com/js/js_quiz.asp)

Examples of using JavaScript to access and manipulate the Browser objects.

**Window Object**

[Open a new window when clicking on a button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_openwindow)  
[Open a new window and control its appearance](http://www.w3schools.com/js/tryit.asp?filename=tryjs_openallwindow)  
[Blur and Focus a new window](http://www.w3schools.com/js/tryit.asp?filename=tryjs_win_blur)  
[Close the new window](http://www.w3schools.com/js/tryit.asp?filename=tryjs_win_close)  
[Checks whether the new window has been closed or not](http://www.w3schools.com/js/tryit.asp?filename=tryjs_win_closed)  
[Write some text to the source (parent) window](http://www.w3schools.com/js/tryit.asp?filename=tryjs_win_opener)  
[Move the new window relative to its current position](http://www.w3schools.com/js/tryit.asp?filename=tryjs_win_moveby)  
[Move the new window to the specified position](http://www.w3schools.com/js/tryit.asp?filename=tryjs_win_moveto)  
[Print the current page](http://www.w3schools.com/js/tryit.asp?filename=tryjs_print)  
[Resize a window by the specified pixels](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_window_resizeby)  
[Resize a window to a specified size](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_window_resizeto)  
[Scroll the content by the specified number of pixels](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_window_scrollby)  
[Scroll the content to a specified position](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_window_scrollto)

[Window explained](http://www.w3schools.com/js/js_window.asp)

**Screen Object**

[The visitor's screen: Width](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_width)  
[The visitor's screen: Height](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_height)  
[The visitor's screen: Available Width](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_availwidth)  
[The visitor's screen: Available Height](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_availheight)  
[The visitor's screen: Color Depth](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_colordepth)  
[The visitor's screen: Pixel Depth](http://www.w3schools.com/js/tryit.asp?filename=tryjs_screen_pixeldepth)

[Screen explained](http://www.w3schools.com/js/js_window_screen.asp)

**Location Object**

[Return the hostname and port of the current URL](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_host)  
[Return the entire URL of the current page](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_href)  
[Return the path name of the current URL](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_pathname)  
[Return the protocol portion of the current URL](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_protocol)  
[Load a new document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_assign)  
[Replace the current document](http://www.w3schools.com/js/tryit.asp?filename=tryjs_loc_replace)  
[Break out of a frame](http://www.w3schools.com/js/tryit.asp?filename=tryjs_breakout)

[Location explained](http://www.w3schools.com/js/js_window_location.asp)

**History Object**

[Display the number of URLs in the history list](http://www.w3schools.com/js/tryit.asp?filename=tryjs_history_length)  
[Create a back button on a page](http://www.w3schools.com/js/tryit.asp?filename=tryjs_history_back)  
[Create a forward button on a page](http://www.w3schools.com/js/tryit.asp?filename=tryjs_history_forward)  
[Load a specific URL from the history list](http://www.w3schools.com/js/tryit.asp?filename=tryjs_history_go)

[History explained](http://www.w3schools.com/js/js_window_history.asp)

**Navigator Object**

[Is cookies enabled in the visitor's browser?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_cookieenabled)  
[What is the name of the visitor's browser?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_appcodename)  
[What is the engine name of the visitor's browser?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_product)  
[What is the version information of the visitor's browser?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_appversion)  
[What is user agent information of the visitor's browser?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_useragent)  
[What is the platform of the visitor's browser?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_platform)  
[What is the language of the visitor's browser?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_language)  
[Is Java enabled in the visitor's browser?](http://www.w3schools.com/js/tryit.asp?filename=tryjs_nav_javaenabled)

[Navigator explained](http://www.w3schools.com/js/js_window_navigator.asp)

**Popup Boxes**

[Display an alert box](http://www.w3schools.com/js/tryit.asp?filename=tryjs_alert)  
[Demonstrate line breaks in an alert box](http://www.w3schools.com/js/tryit.asp?filename=tryjs_alert2)  
[Display a confirm box](http://www.w3schools.com/js/tryit.asp?filename=tryjs_confirm)  
[Display a prompt box](http://www.w3schools.com/js/tryit.asp?filename=tryjs_prompt)

[Popup explained](http://www.w3schools.com/js/js_popup.asp)

**Timing**

[Simple timing](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing1)  
[Another simple timing](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing2)  
[Timing event in an infinite loop](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing_infinite)  
[Timing event in an infinite loop - with a Stop button](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing_stop)  
[A clock created with a timing event](http://www.w3schools.com/js/tryit.asp?filename=tryjs_timing_clock)  
[Set and stop a timer with setInterval() and clearInterval()](http://www.w3schools.com/js/tryit.asp?filename=tryjs_setinterval)

[Timing explained](http://www.w3schools.com/js/js_timing.asp)

**Cookies**

[Create a welcome cookie](http://www.w3schools.com/js/tryit.asp?filename=tryjs_cookie_username)

[Cookies explained](http://www.w3schools.com/js/js_cookies.asp)

**JavaScript Quiz Test**

[« Previous](http://www.w3schools.com/js/js_ex_browser.asp)

[Next Chapter »](http://www.w3schools.com/js/js_exam.asp)

You can test your JavaScript skills with W3Schools' Quiz.

**The Test**

The test contains 25 questions and there is no time limit.

The test is not official, it's just a nice way to see how much you know, or don't know, about JavaScript.

**Count Your Score**

You will get 1 point for each correct answer. At the end of the Quiz, your total score will be displayed. Maximum score is 25 points.

**Start the Quiz**

Good luck!

[](http://www.w3schools.com/cert/default.asp)

**W3Schools' Online Certification**

The perfect solution for professionals who need to balance work, family, and career building.

More than 10 000 certificates already issued!

[Get Your Certificate »](http://www.w3schools.com/cert/default.asp)

The [HTML Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of HTML.

The [HTML5 Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of advanced HTML5.

The [CSS Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of advanced CSS.

The [JavaScript Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of JavaScript and HTML DOM.

The [jQuery Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of jQuery.

The [PHP Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of PHP and SQL (MySQL).

The [XML Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of XML, XML DOM and XSLT

**W3Schools JavaScript Certificate**

[« Previous](http://www.w3schools.com/js/js_quiz.asp)

[Next Chapter »](http://www.w3schools.com/js/js_summary.asp)

|  |  |  |
| --- | --- | --- |
| W3Schools Certified |  | W3Schools offers an Online Certification Program.  The perfect solution for busy professionals who need to balance work, family, and career building.  More than 15 000 certificates already issued! |

|  |  |
| --- | --- |
| **Document Your Skills**  Knowledge is power, especially in the current job market. Documentation of your skills enables you to advance your career, or help you to start a new one.  **Get a Certificate**  Getting a certificate proves your commitment to upgrade your skills, gives you the credibility needed for more responsibilities, larger projects, and a higher salary.  [Get Your Certificate »](http://www.w3schools.com/cert/default.asp) | W3Schools Certificate |

**How Does It Work?**

* Study for free at W3Schools.com
* Study at your own speed
* Test your skills with W3Schools online quizzes
* Apply for your certificate by paying an exam fee
* Take your exam online, at any time, and from any location
* **You Have Learned JavaScript, Now What?**
* [« Previous](http://www.w3schools.com/js/js_exam.asp)
* [JavaScript References »](http://www.w3schools.com/jsref/default.asp)
* **JavaScript Summary**
* This tutorial has taught you how to add JavaScript to your HTML pages, to make your web site more dynamic and interactive.
* You have learned how to create responses to events, validate forms and how to make different scripts run in response to different scenarios.
* You have also learned how to create and use objects, and how to use JavaScript's built-in objects.
* For more information on JavaScript, please look at our [JavaScript examples](http://www.w3schools.com/js/js_examples.asp) and our [JavaScript reference](http://www.w3schools.com/jsref/default.asp).
* **Now You Know JavaScript, What's Next?**
* The next step is to learn about the HTML DOM, jQuery, and AJAX.
* If you want to learn about server-side scripting, the next step is to learn ASP
* **jQuery**
* jQuery is a JavaScript Library.
* jQuery greatly simplifies JavaScript programming.
* If you want to learn more about jQuery, please visit our [jQuery tutorial](http://www.w3schools.com/jquery/default.asp).
* **AJAX**
* AJAX = Asynchronous JavaScript and XML.
* AJAX is not a new programming language, but a new way to use existing standards.
* AJAX is about exchanging data with a server, and update parts of a web page - without reloading the whole page.
* Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.
* If you want to learn more about AJAX, please visit our [AJAX tutorial](http://www.w3schools.com/ajax/default.asp).
* **Web Building**
* In this tutorial we have created dynamic web pages by using scripts on the client (in the browser).
* Web pages can also be made more dynamic by using scripts on the server.
* With server-side scripting you can edit, add, or change any web page content. You can respond to data submitted from HTML forms, access data or databases and return the results to a browser, and customize pages for individual users.
* [Web Building Tutorial](http://www.w3schools.com/website/default.asp)
* [](http://www.w3schools.com/cert/default.asp)
* **W3Schools' Online Certification**
* The perfect solution for professionals who need to balance work, family, and career building.
* More than 10 000 certificates already issued!
* [Get Your Certificate »](http://www.w3schools.com/cert/default.asp)
* The [HTML Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of HTML.
* The [HTML5 Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of advanced HTML5.
* The [CSS Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of advanced CSS.
* The [JavaScript Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of JavaScript and HTML DOM.
* The [jQuery Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of jQuery.
* The [PHP Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of PHP and SQL (MySQL).
* The [XML Certificate](http://www.w3schools.com/cert/default.asp) documents your knowledge of XML, XML DOM and XSLT.
* **JavaScript Reference**
* The references describe the properties and methods of all JavaScript objects, along with examples.
* [String](http://www.w3schools.com/jsref/jsref_obj_string.asp) [Number](http://www.w3schools.com/jsref/jsref_obj_number.asp) [Operators](http://www.w3schools.com/jsref/jsref_operators.asp) [Statements](http://www.w3schools.com/jsref/jsref_statements.asp) [Math](http://www.w3schools.com/jsref/jsref_obj_math.asp) [Date](http://www.w3schools.com/jsref/jsref_obj_date.asp) [Array](http://www.w3schools.com/jsref/jsref_obj_array.asp) [Boolean](http://www.w3schools.com/jsref/jsref_obj_boolean.asp) [RegExp](http://www.w3schools.com/jsref/jsref_obj_regexp.asp) [Global](http://www.w3schools.com/jsref/jsref_obj_global.asp) [Conversion](http://www.w3schools.com/jsref/jsref_type_conversion.asp)
* **Browser Objects Reference**
* The references describe the properties and methods of each object, along with examples.
* [Window](http://www.w3schools.com/jsref/obj_window.asp) [Navigator](http://www.w3schools.com/jsref/obj_navigator.asp) [Screen](http://www.w3schools.com/jsref/obj_screen.asp) [History](http://www.w3schools.com/jsref/obj_history.asp) [Location](http://www.w3schools.com/jsref/obj_location.asp)
* **HTML DOM Reference**
* The references describe the properties and methods of the HTML DOM, along with examples.
* [DOM Document](http://www.w3schools.com/jsref/dom_obj_document.asp) [DOM Elements](http://www.w3schools.com/jsref/dom_obj_all.asp) [DOM Attributes](http://www.w3schools.com/jsref/dom_obj_attributes.asp) [DOM Events](http://www.w3schools.com/jsref/dom_obj_event.asp)
* **HTML Element Objects Reference**
* The references describe the properties and methods of each HTML object, along with examples.
* [a](http://www.w3schools.com/jsref/dom_obj_anchor.asp) [abbr](http://www.w3schools.com/jsref/dom_obj_abbr.asp) [address](http://www.w3schools.com/jsref/dom_obj_address.asp) [area](http://www.w3schools.com/jsref/dom_obj_area.asp) [article](http://www.w3schools.com/jsref/dom_obj_article.asp) [aside](http://www.w3schools.com/jsref/dom_obj_aside.asp) [audio](http://www.w3schools.com/jsref/dom_obj_audio.asp) [b](http://www.w3schools.com/jsref/dom_obj_b.asp) [base](http://www.w3schools.com/jsref/dom_obj_base.asp) [bdo](http://www.w3schools.com/jsref/dom_obj_bdo.asp) [blockquote](http://www.w3schools.com/jsref/dom_obj_blockquote.asp) [body](http://www.w3schools.com/jsref/dom_obj_body.asp) [br](http://www.w3schools.com/jsref/dom_obj_br.asp) [button](http://www.w3schools.com/jsref/dom_obj_pushbutton.asp) [canvas](http://www.w3schools.com/jsref/dom_obj_canvas.asp) [caption](http://www.w3schools.com/jsref/dom_obj_caption.asp) [cite](http://www.w3schools.com/jsref/dom_obj_cite.asp) [code](http://www.w3schools.com/jsref/dom_obj_code.asp) [col](http://www.w3schools.com/jsref/dom_obj_col.asp) [colgroup](http://www.w3schools.com/jsref/dom_obj_colgroup.asp) [datalist](http://www.w3schools.com/jsref/dom_obj_datalist.asp) [dd](http://www.w3schools.com/jsref/dom_obj_dd.asp) [del](http://www.w3schools.com/jsref/dom_obj_del.asp) [details](http://www.w3schools.com/jsref/dom_obj_details.asp) [dfn](http://www.w3schools.com/jsref/dom_obj_dfn.asp) [dialog](http://www.w3schools.com/jsref/dom_obj_dialog.asp) [div](http://www.w3schools.com/jsref/dom_obj_div.asp) [dl](http://www.w3schools.com/jsref/dom_obj_dl.asp) [dt](http://www.w3schools.com/jsref/dom_obj_dt.asp) [em](http://www.w3schools.com/jsref/dom_obj_em.asp) [embed](http://www.w3schools.com/jsref/dom_obj_embed.asp) [fieldset](http://www.w3schools.com/jsref/dom_obj_fieldset.asp) [figcaption](http://www.w3schools.com/jsref/dom_obj_figcaption.asp) [figure](http://www.w3schools.com/jsref/dom_obj_figure.asp) [footer](http://www.w3schools.com/jsref/dom_obj_footer.asp) [form](http://www.w3schools.com/jsref/dom_obj_form.asp) [head](http://www.w3schools.com/jsref/dom_obj_head.asp) [header](http://www.w3schools.com/jsref/dom_obj_header.asp) [h1 - h6](http://www.w3schools.com/jsref/dom_obj_heading.asp) [hr](http://www.w3schools.com/jsref/dom_obj_hr.asp) [html](http://www.w3schools.com/jsref/dom_obj_html.asp) [i](http://www.w3schools.com/jsref/dom_obj_i.asp) [iframe](http://www.w3schools.com/jsref/dom_obj_frame.asp) [img](http://www.w3schools.com/jsref/dom_obj_image.asp) [ins](http://www.w3schools.com/jsref/dom_obj_ins.asp) [input button](http://www.w3schools.com/jsref/dom_obj_button.asp) [input checkbox](http://www.w3schools.com/jsref/dom_obj_checkbox.asp) [input color](http://www.w3schools.com/jsref/dom_obj_color.asp) [input date](http://www.w3schools.com/jsref/dom_obj_date.asp) [input datetime](http://www.w3schools.com/jsref/dom_obj_datetime.asp) [input datetime-local](http://www.w3schools.com/jsref/dom_obj_datetime-local.asp) [input email](http://www.w3schools.com/jsref/dom_obj_email.asp) [input file](http://www.w3schools.com/jsref/dom_obj_fileupload.asp) [input hidden](http://www.w3schools.com/jsref/dom_obj_hidden.asp) [input image](http://www.w3schools.com/jsref/dom_obj_input_image.asp) [input month](http://www.w3schools.com/jsref/dom_obj_month.asp) [input number](http://www.w3schools.com/jsref/dom_obj_number.asp) [input password](http://www.w3schools.com/jsref/dom_obj_password.asp) [input radio](http://www.w3schools.com/jsref/dom_obj_radio.asp) [input range](http://www.w3schools.com/jsref/dom_obj_range.asp) [input reset](http://www.w3schools.com/jsref/dom_obj_reset.asp) [input search](http://www.w3schools.com/jsref/dom_obj_search.asp) [input submit](http://www.w3schools.com/jsref/dom_obj_submit.asp) [input text](http://www.w3schools.com/jsref/dom_obj_text.asp) [input time](http://www.w3schools.com/jsref/dom_obj_input_time.asp) [input url](http://www.w3schools.com/jsref/dom_obj_url.asp) [input week](http://www.w3schools.com/jsref/dom_obj_week.asp) [kbd](http://www.w3schools.com/jsref/dom_obj_kbd.asp) [keygen](http://www.w3schools.com/jsref/dom_obj_keygen.asp) [label](http://www.w3schools.com/jsref/dom_obj_label.asp) [legend](http://www.w3schools.com/jsref/dom_obj_legend.asp) [li](http://www.w3schools.com/jsref/dom_obj_li.asp) [link](http://www.w3schools.com/jsref/dom_obj_link.asp) [map](http://www.w3schools.com/jsref/dom_obj_map.asp) [mark](http://www.w3schools.com/jsref/dom_obj_mark.asp) [menu](http://www.w3schools.com/jsref/dom_obj_menu.asp) [menuitem](http://www.w3schools.com/jsref/dom_obj_menuitem.asp) [meta](http://www.w3schools.com/jsref/dom_obj_meta.asp) [meter](http://www.w3schools.com/jsref/dom_obj_meter.asp) [nav](http://www.w3schools.com/jsref/dom_obj_nav.asp) [object](http://www.w3schools.com/jsref/dom_obj_object.asp) [ol](http://www.w3schools.com/jsref/dom_obj_ol.asp) [optgroup](http://www.w3schools.com/jsref/dom_obj_optgroup.asp) [option](http://www.w3schools.com/jsref/dom_obj_option.asp) [output](http://www.w3schools.com/jsref/dom_obj_output.asp) [p](http://www.w3schools.com/jsref/dom_obj_paragraph.asp) [param](http://www.w3schools.com/jsref/dom_obj_param.asp) [pre](http://www.w3schools.com/jsref/dom_obj_pre.asp) [progress](http://www.w3schools.com/jsref/dom_obj_progress.asp) [q](http://www.w3schools.com/jsref/dom_obj_quote.asp) [s](http://www.w3schools.com/jsref/dom_obj_s.asp) [samp](http://www.w3schools.com/jsref/dom_obj_samp.asp) [script](http://www.w3schools.com/jsref/dom_obj_script.asp) [section](http://www.w3schools.com/jsref/dom_obj_section.asp) [select](http://www.w3schools.com/jsref/dom_obj_select.asp) [small](http://www.w3schools.com/jsref/dom_obj_small.asp) [source](http://www.w3schools.com/jsref/dom_obj_source.asp) [span](http://www.w3schools.com/jsref/dom_obj_span.asp) [strong](http://www.w3schools.com/jsref/dom_obj_strong.asp) [style](http://www.w3schools.com/jsref/dom_obj_style.asp) [sub](http://www.w3schools.com/jsref/dom_obj_sub.asp) [summary](http://www.w3schools.com/jsref/dom_obj_summary.asp) [sup](http://www.w3schools.com/jsref/dom_obj_sup.asp) [table](http://www.w3schools.com/jsref/dom_obj_table.asp) [td](http://www.w3schools.com/jsref/dom_obj_tabledata.asp) [th](http://www.w3schools.com/jsref/dom_obj_tablehead.asp) [tr](http://www.w3schools.com/jsref/dom_obj_tablerow.asp) [textarea](http://www.w3schools.com/jsref/dom_obj_textarea.asp) [time](http://www.w3schools.com/jsref/dom_obj_time.asp) [title](http://www.w3schools.com/jsref/dom_obj_title.asp) [track](http://www.w3schools.com/jsref/dom_obj_track.asp) [u](http://www.w3schools.com/jsref/dom_obj_u.asp) [ul](http://www.w3schools.com/jsref/dom_obj_ul.asp) [var](http://www.w3schools.com/jsref/dom_obj_var.asp) [video](http://www.w3schools.com/jsref/dom_obj_video.asp)
* **JavaScript Strings**
* A JavaScript string stores a series of characters like "John Doe".
* A string can be any text inside double or single quotes:
* var carname = "Volvo XC60";  
  var carname = 'Volvo XC60';
* String indexes are zero-based: The first character is in position 0, the second in 1, and so on.
* For a tutorial about Strings, read our [JavaScript String Tutorial](http://www.w3schools.com/js/js_strings.asp).
* **String Properties and 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.
* **String Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](http://www.w3schools.com/jsref/jsref_constructor_string.asp) | Returns the string's constructor function |
| [length](http://www.w3schools.com/jsref/jsref_length_string.asp) | Returns the length of a string |
| [prototype](http://www.w3schools.com/jsref/jsref_prototype_string.asp) | Allows you to add properties and methods to an object |

* **String Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [charAt()](http://www.w3schools.com/jsref/jsref_charat.asp) | Returns the character at the specified index (position) |
| [charCodeAt()](http://www.w3schools.com/jsref/jsref_charcodeat.asp) | Returns the Unicode of the character at the specified index |
| [concat()](http://www.w3schools.com/jsref/jsref_concat_string.asp) | Joins two or more strings, and returns a new joined strings |
| [fromCharCode()](http://www.w3schools.com/jsref/jsref_fromcharcode.asp) | Converts Unicode values to characters |
| [indexOf()](http://www.w3schools.com/jsref/jsref_indexof.asp) | Returns the position of the first found occurrence of a specified value in a string |
| [lastIndexOf()](http://www.w3schools.com/jsref/jsref_lastindexof.asp) | Returns the position of the last found occurrence of a specified value in a string |
| [localeCompare()](http://www.w3schools.com/jsref/jsref_localecompare.asp) | Compares two strings in the current locale |
| [match()](http://www.w3schools.com/jsref/jsref_match.asp) | Searches a string for a match against a regular expression, and returns the matches |
| [replace()](http://www.w3schools.com/jsref/jsref_replace.asp) | Searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced |
| [search()](http://www.w3schools.com/jsref/jsref_search.asp) | Searches a string for a specified value, or regular expression, and returns the position of the match |
| [slice()](http://www.w3schools.com/jsref/jsref_slice_string.asp) | Extracts a part of a string and returns a new string |
| [split()](http://www.w3schools.com/jsref/jsref_split.asp) | Splits a string into an array of substrings |
| [substr()](http://www.w3schools.com/jsref/jsref_substr.asp) | Extracts the characters from a string, beginning at a specified start position, and through the specified number of character |
| [substring()](http://www.w3schools.com/jsref/jsref_substring.asp) | Extracts the characters from a string, between two specified indices |
| [toLocaleLowerCase()](http://www.w3schools.com/jsref/jsref_tolocalelowercase.asp) | Converts a string to lowercase letters, according to the host's locale |
| [toLocaleUpperCase()](http://www.w3schools.com/jsref/jsref_tolocaleuppercase.asp) | Converts a string to uppercase letters, according to the host's locale |
| [toLowerCase()](http://www.w3schools.com/jsref/jsref_tolowercase.asp) | Converts a string to lowercase letters |
| [toString()](http://www.w3schools.com/jsref/jsref_tostring_string.asp) | Returns the value of a String object |
| [toUpperCase()](http://www.w3schools.com/jsref/jsref_touppercase.asp) | Converts a string to uppercase letters |
| [trim()](http://www.w3schools.com/jsref/jsref_trim_string.asp) | Removes whitespace from both ends of a string |
| [valueOf()](http://www.w3schools.com/jsref/jsref_valueof_string.asp) | Returns the primitive value of a String object |

* **String HTML Wrapper Methods**
* The HTML wrapper methods return the string wrapped inside the appropriate HTML tag.
* These are not standard methods, and may not work as expected in all browsers.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [anchor()](http://www.w3schools.com/jsref/jsref_anchor.asp) | Creates an anchor |
| [big()](http://www.w3schools.com/jsref/jsref_big.asp) | Displays a string using a big font |
| [blink()](http://www.w3schools.com/jsref/jsref_blink.asp) | Displays a blinking string |
| [bold()](http://www.w3schools.com/jsref/jsref_bold.asp) | Displays a string in bold |
| [fixed()](http://www.w3schools.com/jsref/jsref_fixed.asp) | Displays a string using a fixed-pitch font |
| [fontcolor()](http://www.w3schools.com/jsref/jsref_fontcolor.asp) | Displays a string using a specified color |
| [fontsize()](http://www.w3schools.com/jsref/jsref_fontsize.asp) | Displays a string using a specified size |
| [italics()](http://www.w3schools.com/jsref/jsref_italics.asp) | Displays a string in italic |
| [link()](http://www.w3schools.com/jsref/jsref_link.asp) | Displays a string as a hyperlink |
| [small()](http://www.w3schools.com/jsref/jsref_small.asp) | Displays a string using a small font |
| [strike()](http://www.w3schools.com/jsref/jsref_strike.asp) | Displays a string with a strikethrough |
| [sub()](http://www.w3schools.com/jsref/jsref_sub.asp) | Displays a string as subscript text |
| [sup()](http://www.w3schools.com/jsref/jsref_sup.asp) | Displays a string as superscript text |

**JavaScript Number Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_obj_string.asp)

[Next Reference »](http://www.w3schools.com/jsref/jsref_operators.asp)

**JavaScript Numbers**

JavaScript has only one type of number.

Numbers can be written with, or without, decimals:

**Example**

var x = 3.14;     // A number with decimals  
var y = 34;       // A number without decimals

Extra large or extra small numbers can be written with scientific (exponent) notation:

**Example**

var x = 123e5;    // 12300000  
var y = 123e-5;   // 0.00123

For a tutorial about JavaScript numbers, read our [JavaScript Number Tutorial](http://www.w3schools.com/js/js_numbers.asp).

**Number Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](http://www.w3schools.com/jsref/jsref_constructor_number.asp) | Returns the function that created JavaScript's Number prototype |
| [MAX\_VALUE](http://www.w3schools.com/jsref/jsref_max_value.asp) | Returns the largest number possible in JavaScript |
| [MIN\_VALUE](http://www.w3schools.com/jsref/jsref_min_value.asp) | Returns the smallest number possible in JavaScript |
| [NEGATIVE\_INFINITY](http://www.w3schools.com/jsref/jsref_negative_infinity.asp) | Represents negative infinity (returned on overflow) |
| [NaN](http://www.w3schools.com/jsref/jsref_number_nan.asp) | Represents a "Not-a-Number" value |
| [POSITIVE\_INFINITY](http://www.w3schools.com/jsref/jsref_positive_infinity.asp) | Represents infinity (returned on overflow) |
| [prototype](http://www.w3schools.com/jsref/jsref_prototype_num.asp) | Allows you to add properties and methods to an object |

**Number Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [toExponential(x)](http://www.w3schools.com/jsref/jsref_toexponential.asp) | Converts a number into an exponential notation |
| [toFixed(x)](http://www.w3schools.com/jsref/jsref_tofixed.asp) | Formats a number with x numbers of digits after the decimal point |
| [toPrecision(x)](http://www.w3schools.com/jsref/jsref_toprecision.asp) | Formats a number to x length |
| [toString()](http://www.w3schools.com/jsref/jsref_tostring_number.asp) | Converts a number to a string |
| [valueOf()](http://www.w3schools.com/jsref/jsref_valueof_number.asp) | Returns the primitive value of a number |

**JavaScript Operators Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_obj_number.asp)

[Next Chapter »](http://www.w3schools.com/jsref/jsref_statements.asp)

JavaScript operators are used to assign values, compare values, perform arithmetic operations, and more.

**JavaScript Arithmetic Operators**

Arithmetic operators are used to perform arithmetic between variables and/or values.

Given that **y = 5**, the table below explains the arithmetic operators:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Result in y** | **Result in x** | **Try it** |
| + | Addition | x = y + 2 | y = 5 | x = 7 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_add) |
| - | Subtraction | x = y - 2 | y = 5 | x = 3 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_sub) |
| \* | Multiplication | x = y \* 2 | y = 5 | x = 10 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_mult) |
| / | Division | x = y / 2 | y = 5 | x = 2.5 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_div) |
| % | Modulus (division remainder) | x = y % 2 | y = 5 | x = 1 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_mod) |
| ++ | Increment | x = ++y | y = 6 | x = 6 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_incr) |
| x = y++ | y = 6 | x = 5 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_incr2) |
| -- | Decrement | x = --y | y = 4 | x = 4 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_decr) |
| x = y-- | y = 4 | x = 5 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_decr2) |

For a tutorial about arithmetic operators, read our [JavaScript Arithmetic Tutorial](http://www.w3schools.com/js/js_arithmetic.asp).

**JavaScript Assignment Operators**

Assignment operators are used to assign values to JavaScript variables.

Given that **x = 10** and **y = 5**, the table below explains the assignment operators:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Result in x** | **Try it** |
| = | x = y | x = y | x = 5 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_equal) |
| += | x += y | x = x + y | x = 15 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_plusequal) |
| -= | x -= y | x = x - y | x = 5 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_minequal) |
| \*= | x \*= y | x = x \* y | x = 50 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_multequal) |
| /= | x /= y | x = x / y | x = 2 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_divequal) |
| %= | x %= y | x = x % y | x = 0 | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_modequal) |

For a tutorial about assignment operators, read our [JavaScript Assignment Tutorial](http://www.w3schools.com/js/js_assignment.asp).

**JavaScript String Operators**

The + operator, and the += operator can also be used to concatenate (add) strings.

Given that **text1 = "Good "**, **text2 = "Morning"**, **and text3 = ""**, the table below explains the operators:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Example** | **text1** | **text2** | **text3** | **Try it** |
| + | text3 = text1 + text2 | "Good " | "Morning" | "Good Morning" | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_string1) |
| += | text1 += text2 | "Good " | "Morning" | "" | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_string2) |

**Comparison Operators**

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x = 5**, the table below explains the comparison operators:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Comparing** | **Returns** | **Try it** |
| == | equal to | x == 8 | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison1) |
| x == 5 | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison2) |
| === | equal value and equal type | x === "5" | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison3) |
| x === 5 | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison4) |
| != | not equal | x != 8 | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison5) |
| !== | not equal value or not equal type | x !== "5" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison6) |
| x !== 5 | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison7) |
| > | greater than | x > 8 | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison8) |
| < | less than | x < 8 | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison9) |
| >= | greater than or equal to | x >= 8 | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison10) |
| <= | less than or equal to | x <= 8 | *true* | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison11) |

For a tutorial about comparison operators, read our [JavaScript Comparisons Tutorial](http://www.w3schools.com/js/js_comparisons.asp).

**Conditional (Ternary) Operator**

The conditional operator assigns a value to a variable based on a condition.

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Example** | **Try it** |
| *variablename* = (*condition*) ? *value1*:*value2* | voteable = (age < 18) ? "Too young":"Old enough"; | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_comparison) |

**Example explained:** If the variable "age" is a value below 18, the value of the variable "voteable" will be "Too young", otherwise the value of voteable will be "Old enough".

**Logical Operators**

Logical operators are used to determine the logic between variables or values.

Given that **x = 6 and y = 3**, the table below explains the logical operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| && | and | (x < 10 && y > 1) is true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_and) |
| || | or | (x === 5 || y === 5) is false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_or) |
| ! | not | !(x === y) is true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_not) |

**JavaScript Bitwise Operators**

Bit operators work on 32 bits numbers. Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Operator** | | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | | AND | x = 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | | OR | x = 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | | NOT | x = ~ 5 | ~0101 | 1010 | 10 |
| ^ | | XOR | x = 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | | Left shift | x = 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | | Right shift | x = 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| **Note** | The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.  Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.  ~00000000000000000000000000000101 will return 11111111111111111111111111111010 | | | | | | |

**The typeof Operator**

The **typeof** operator returns the type of a variable, object, function or expression:

**Example**

typeof "John"                 // Returns string   
typeof 3.14                   // Returns number  
typeof NaN                    // Returns number  
typeof false                  // Returns boolean  
typeof [1, 2, 3, 4]           // Returns object  
typeof {name:'John', age:34}  // Returns object  
typeof new Date()             // Returns object  
typeof function () {}         // Returns function  
typeof myCar                  // Returns undefined (if myCar is not declared)  
typeof null                   // Returns object

[Try it Yourself »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_typeof)

Please observe:

* The data type of NaN is number
* The data type of an array is object
* The data type of a date is object
* The data type of null is object
* The data type of an undefined variable is undefined

|  |  |
| --- | --- |
| **Note** | You cannot use **typeof** to define if a JavaScript object is an array (or a date). |

**The delete Operator**

The **delete** operator deletes a property from an object:

**Example**

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
delete person.age;   // or delete person["age"];

[Try it Yourself »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_delete)

The delete operator deletes both the value of the property and the property itself.

After deletion, the property cannot be used before it is added back again.

The delete operator is designed to be used on object properties. It has no effect on variables or functions.

**Note:** The delete operator should not be used on predefined JavaScript object properties. It can crash your application.

**The in Operator**

The **in** operator returns true if the specified property is in the specified object, otherwise false:

**Example**

// Arrays  
var cars = ["Saab", "Volvo", "BMW"];  
"Saab" in cars          // Returns false (specify the index number instead of value)  
0 in cars               // Returns true  
1 in cars               // Returns true  
4 in cars               // Returns false (does not exist)  
"length" in cars        // Returns true  (length is an Array property)  
  
// Objects  
var person = {firstName:"John", lastName:"Doe", age:50};  
"firstName" in person   // Returns true  
"age" in person         // Returns true  
  
// Predefined objects  
"PI" in Math            // Returns true  
"NaN" in Number         // Returns true  
"length" in String      // Returns true

[Try it Yourself »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_in)

**The instanceof Operator**

The **instanceof** operator returns true if the specified object is an instance of the specified object:

**Example**

var cars = ["Saab", "Volvo", "BMW"];  
  
cars instanceof Array;          // Returns true  
cars instanceof Object;         // Returns true  
cars instanceof String;         // Returns false  
cars instanceof Number;         // Returns false

[Try it Yourself »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_instanceof)

**The void Operator**

The **void** operator evaluates an expression and returns **undefined**. This operator is often used to obtain the undefined primitive value, using "void(0)" (useful when evaluating an expression without using the return value).

**Example**

<a href="javascript:void(0);">  
  Useless link  
</a>  
  
<a href="javascript:void(document.body.style.backgroundColor='red');">  
  Click me to change the background color of body to red  
</a>

[Try it Yourself »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_oper_void)

**JavaScript Statements Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_operators.asp)

[Next Reference »](http://www.w3schools.com/jsref/jsref_obj_math.asp)

**JavaScript Statements**

In HTML, JavaScript statements are "instructions" to be "executed" by the web browser.

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

**Example**

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

[Try it Yourself »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_statement)

For a tutorial about Statements, read our [JavaScript Statements Tutorial](http://www.w3schools.com/js/js_statements.asp).

**JavaScript Statement Identifiers**

JavaScript statements often start with a **statement identifier** to identify the JavaScript action to be performed.

Statement identifiers are reserved words and cannot be used as variable names (or any other things).

The following table lists all JavaScript statements:

|  |  |
| --- | --- |
| **Statement** | **Description** |
| [break](http://www.w3schools.com/jsref/jsref_break.asp) | Exits a switch or a loop |
| [continue](http://www.w3schools.com/jsref/jsref_continue.asp) | Breaks one iteration (in the loop) if a specified condition occurs, and continues with the next iteration in the loop |
| [debugger](http://www.w3schools.com/jsref/jsref_debugger.asp) | Stops the execution of JavaScript, and calls (if available) the debugging function |
| [do ... while](http://www.w3schools.com/jsref/jsref_dowhile.asp) | Executes a block of statements and repeats the block while a condition is true |
| [for](http://www.w3schools.com/jsref/jsref_for.asp) | Marks a block of statements to be executed as long as a condition is true |
| [for ... in](http://www.w3schools.com/jsref/jsref_forin.asp) | Marks a block of statements to be executed for each element of an object (or array) |
| [function](http://www.w3schools.com/jsref/jsref_function.asp) | Declares a function |
| [if ... else ... else if](http://www.w3schools.com/jsref/jsref_if.asp) | Marks a block of statements to be executed depending on a condition |
| [return](http://www.w3schools.com/jsref/jsref_return.asp) | Stops the execution of a function and returns a value from that function |
| [switch](http://www.w3schools.com/jsref/jsref_switch.asp) | Marks a block of statements to be executed depending on different cases |
| [throw](http://www.w3schools.com/jsref/jsref_throw.asp) | Throws (generates) an error |
| [try ... catch ... finally](http://www.w3schools.com/jsref/jsref_try_catch.asp) | Marks the block of statements to be executed when an error occurs in a try block, and implements error handling |
| [var](http://www.w3schools.com/jsref/jsref_var.asp) | Declares a variable |
| [while](http://www.w3schools.com/jsref/jsref_while.asp) | Marks a block of statements to be executed while a condition is true |

**JavaScript Math Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_statements.asp)

[Next Reference »](http://www.w3schools.com/jsref/jsref_obj_date.asp)

**Math Object**

The Math object allows you to perform mathematical tasks.

Math is not a constructor. All properties/methods of Math can be called by using Math as an object, without creating it.

**Syntax**

var x = Math.PI;            // Returns PI  
var y = Math.sqrt(16);      // Returns the square root of 16

For a tutorial about the Math object, read our [JavaScript Math Tutorial](http://www.w3schools.com/js/js_math.asp).

**Math Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [E](http://www.w3schools.com/jsref/jsref_e.asp) | Returns Euler's number (approx. 2.718) |
| [LN2](http://www.w3schools.com/jsref/jsref_ln2.asp) | Returns the natural logarithm of 2 (approx. 0.693) |
| [LN10](http://www.w3schools.com/jsref/jsref_ln10.asp) | Returns the natural logarithm of 10 (approx. 2.302) |
| [LOG2E](http://www.w3schools.com/jsref/jsref_log2e.asp) | Returns the base-2 logarithm of E (approx. 1.442) |
| [LOG10E](http://www.w3schools.com/jsref/jsref_log10e.asp) | Returns the base-10 logarithm of E (approx. 0.434) |
| [PI](http://www.w3schools.com/jsref/jsref_pi.asp) | Returns PI (approx. 3.14) |
| [SQRT1\_2](http://www.w3schools.com/jsref/jsref_sqrt1_2.asp) | Returns the square root of 1/2 (approx. 0.707) |
| [SQRT2](http://www.w3schools.com/jsref/jsref_sqrt2.asp) | Returns the square root of 2 (approx. 1.414) |

**Math Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [abs(x)](http://www.w3schools.com/jsref/jsref_abs.asp) | Returns the absolute value of x |
| [acos(x)](http://www.w3schools.com/jsref/jsref_acos.asp) | Returns the arccosine of x, in radians |
| [asin(x)](http://www.w3schools.com/jsref/jsref_asin.asp) | Returns the arcsine of x, in radians |
| [atan(x)](http://www.w3schools.com/jsref/jsref_atan.asp) | Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians |
| [atan2(y,x)](http://www.w3schools.com/jsref/jsref_atan2.asp) | Returns the arctangent of the quotient of its arguments |
| [ceil(x)](http://www.w3schools.com/jsref/jsref_ceil.asp) | Returns x, rounded upwards to the nearest integer |
| [cos(x)](http://www.w3schools.com/jsref/jsref_cos.asp) | Returns the cosine of x (x is in radians) |
| [exp(x)](http://www.w3schools.com/jsref/jsref_exp.asp) | Returns the value of Ex |
| [floor(x)](http://www.w3schools.com/jsref/jsref_floor.asp) | Returns x, rounded downwards to the nearest integer |
| [log(x)](http://www.w3schools.com/jsref/jsref_log.asp) | Returns the natural logarithm (base E) of x |
| [max(x,y,z,...,n)](http://www.w3schools.com/jsref/jsref_max.asp) | Returns the number with the highest value |
| [min(x,y,z,...,n)](http://www.w3schools.com/jsref/jsref_min.asp) | Returns the number with the lowest value |
| [pow(x,y)](http://www.w3schools.com/jsref/jsref_pow.asp) | Returns the value of x to the power of y |
| [random()](http://www.w3schools.com/jsref/jsref_random.asp) | Returns a random number between 0 and 1 |
| [round(x)](http://www.w3schools.com/jsref/jsref_round.asp) | Rounds x to the nearest integer |
| [sin(x)](http://www.w3schools.com/jsref/jsref_sin.asp) | Returns the sine of x (x is in radians) |
| [sqrt(x)](http://www.w3schools.com/jsref/jsref_sqrt.asp) | Returns the square root of x |
| [tan(x)](http://www.w3schools.com/jsref/jsref_tan.asp) | Returns the tangent of an angle |

**JavaScript Date Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_obj_math.asp)

[Next Reference »](http://www.w3schools.com/jsref/jsref_obj_array.asp)

**Date Object**

The Date object is used to work with dates and times.

Date objects are created with new Date().

There are four ways of instantiating a date:

var d = new Date();  
var d = new Date(*milliseconds*);  
var d = new Date(*dateString*);  
var d = new Date(*year*, *month*, *day*, *hours*, *minutes*, *seconds*, *milliseconds*);

For a tutorial about date and times, read our [JavaScript Date Tutorial](http://www.w3schools.com/js/js_dates.asp).

**Date Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](http://www.w3schools.com/jsref/jsref_constructor_date.asp) | Returns the function that created the Date object's prototype |
| [prototype](http://www.w3schools.com/jsref/jsref_prototype_date.asp) | Allows you to add properties and methods to an object |

**Date Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [getDate()](http://www.w3schools.com/jsref/jsref_getdate.asp) | Returns the day of the month (from 1-31) |
| [getDay()](http://www.w3schools.com/jsref/jsref_getday.asp) | Returns the day of the week (from 0-6) |
| [getFullYear()](http://www.w3schools.com/jsref/jsref_getfullyear.asp) | Returns the year (four digits) |
| [getHours()](http://www.w3schools.com/jsref/jsref_gethours.asp) | Returns the hour (from 0-23) |
| [getMilliseconds()](http://www.w3schools.com/jsref/jsref_getmilliseconds.asp) | Returns the milliseconds (from 0-999) |
| [getMinutes()](http://www.w3schools.com/jsref/jsref_getminutes.asp) | Returns the minutes (from 0-59) |
| [getMonth()](http://www.w3schools.com/jsref/jsref_getmonth.asp) | Returns the month (from 0-11) |
| [getSeconds()](http://www.w3schools.com/jsref/jsref_getseconds.asp) | Returns the seconds (from 0-59) |
| [getTime()](http://www.w3schools.com/jsref/jsref_gettime.asp) | Returns the number of milliseconds since midnight Jan 1, 1970 |
| [getTimezoneOffset()](http://www.w3schools.com/jsref/jsref_gettimezoneoffset.asp) | Returns the time difference between UTC time and local time, in minutes |
| [getUTCDate()](http://www.w3schools.com/jsref/jsref_getutcdate.asp) | Returns the day of the month, according to universal time (from 1-31) |
| [getUTCDay()](http://www.w3schools.com/jsref/jsref_getutcday.asp) | Returns the day of the week, according to universal time (from 0-6) |
| [getUTCFullYear()](http://www.w3schools.com/jsref/jsref_getutcfullyear.asp) | Returns the year, according to universal time (four digits) |
| [getUTCHours()](http://www.w3schools.com/jsref/jsref_getutchours.asp) | Returns the hour, according to universal time (from 0-23) |
| [getUTCMilliseconds()](http://www.w3schools.com/jsref/jsref_getutcmilliseconds.asp) | Returns the milliseconds, according to universal time (from 0-999) |
| [getUTCMinutes()](http://www.w3schools.com/jsref/jsref_getutcminutes.asp) | Returns the minutes, according to universal time (from 0-59) |
| [getUTCMonth()](http://www.w3schools.com/jsref/jsref_getutcmonth.asp) | Returns the month, according to universal time (from 0-11) |
| [getUTCSeconds()](http://www.w3schools.com/jsref/jsref_getutcseconds.asp) | Returns the seconds, according to universal time (from 0-59) |
| getYear() | Deprecated. Use the [getFullYear()](http://www.w3schools.com/jsref/jsref_getfullyear.asp) method instead |
| [parse()](http://www.w3schools.com/jsref/jsref_parse.asp) | Parses a date string and returns the number of milliseconds since January 1, 1970 |
| [setDate()](http://www.w3schools.com/jsref/jsref_setdate.asp) | Sets the day of the month of a date object |
| [setFullYear()](http://www.w3schools.com/jsref/jsref_setfullyear.asp) | Sets the year (four digits) of a date object |
| [setHours()](http://www.w3schools.com/jsref/jsref_sethours.asp) | Sets the hour of a date object |
| [setMilliseconds()](http://www.w3schools.com/jsref/jsref_setmilliseconds.asp) | Sets the milliseconds of a date object |
| [setMinutes()](http://www.w3schools.com/jsref/jsref_setminutes.asp) | Set the minutes of a date object |
| [setMonth()](http://www.w3schools.com/jsref/jsref_setmonth.asp) | Sets the month of a date object |
| [setSeconds()](http://www.w3schools.com/jsref/jsref_setseconds.asp) | Sets the seconds of a date object |
| [setTime()](http://www.w3schools.com/jsref/jsref_settime.asp) | Sets a date to a specified number of milliseconds after/before January 1, 1970 |
| [setUTCDate()](http://www.w3schools.com/jsref/jsref_setutcdate.asp) | Sets the day of the month of a date object, according to universal time |
| [setUTCFullYear()](http://www.w3schools.com/jsref/jsref_setutcfullyear.asp) | Sets the year of a date object, according to universal time (four digits) |
| [setUTCHours()](http://www.w3schools.com/jsref/jsref_setutchours.asp) | Sets the hour of a date object, according to universal time |
| [setUTCMilliseconds()](http://www.w3schools.com/jsref/jsref_setutcmilliseconds.asp) | Sets the milliseconds of a date object, according to universal time |
| [setUTCMinutes()](http://www.w3schools.com/jsref/jsref_setutcminutes.asp) | Set the minutes of a date object, according to universal time |
| [setUTCMonth()](http://www.w3schools.com/jsref/jsref_setutcmonth.asp) | Sets the month of a date object, according to universal time |
| [setUTCSeconds()](http://www.w3schools.com/jsref/jsref_setutcseconds.asp) | Set the seconds of a date object, according to universal time |
| setYear() | Deprecated. Use the [setFullYear()](http://www.w3schools.com/jsref/jsref_setfullyear.asp) method instead |
| [toDateString()](http://www.w3schools.com/jsref/jsref_todatestring.asp) | Converts the date portion of a Date object into a readable string |
| toGMTString() | Deprecated. Use the [toUTCString()](http://www.w3schools.com/jsref/jsref_toutcstring.asp) method instead |
| [toISOString()](http://www.w3schools.com/jsref/jsref_toisostring.asp) | Returns the date as a string, using the ISO standard |
| [toJSON()](http://www.w3schools.com/jsref/jsref_tojson.asp) | Returns the date as a string, formatted as a JSON date |
| [toLocaleDateString()](http://www.w3schools.com/jsref/jsref_tolocaledatestring.asp) | Returns the date portion of a Date object as a string, using locale conventions |
| [toLocaleTimeString()](http://www.w3schools.com/jsref/jsref_tolocaletimestring.asp) | Returns the time portion of a Date object as a string, using locale conventions |
| [toLocaleString()](http://www.w3schools.com/jsref/jsref_tolocalestring.asp) | Converts a Date object to a string, using locale conventions |
| [toString()](http://www.w3schools.com/jsref/jsref_tostring_date.asp) | Converts a Date object to a string |
| [toTimeString()](http://www.w3schools.com/jsref/jsref_totimestring.asp) | Converts the time portion of a Date object to a string |
| [toUTCString()](http://www.w3schools.com/jsref/jsref_toutcstring.asp) | Converts a Date object to a string, according to universal time |
| [UTC()](http://www.w3schools.com/jsref/jsref_utc.asp) | Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time |
| [valueOf()](http://www.w3schools.com/jsref/jsref_valueof_date.asp) | Returns the primitive value of a Date object |

**JavaScript Array Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_obj_date.asp)

[Next Reference »](http://www.w3schools.com/jsref/jsref_obj_boolean.asp)

**Array Object**

The Array object is used to store multiple values in a single variable:

var cars = ["Saab", "Volvo", "BMW"];

Array indexes are zero-based: The first element in the array is 0, the second is 1, and so on.

For a tutorial about Arrays, read our [JavaScript Array Tutorial](http://www.w3schools.com/js/js_arrays.asp).

**Array Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](http://www.w3schools.com/jsref/jsref_constructor_array.asp) | Returns the function that created the Array object's prototype |
| [length](http://www.w3schools.com/jsref/jsref_length_array.asp) | Sets or returns the number of elements in an array |
| [prototype](http://www.w3schools.com/jsref/jsref_prototype_array.asp) | Allows you to add properties and methods to an Array object |

**Array Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [concat()](http://www.w3schools.com/jsref/jsref_concat_array.asp) | Joins two or more arrays, and returns a copy of the joined arrays |
| [indexOf()](http://www.w3schools.com/jsref/jsref_indexof_array.asp) | Search the array for an element and returns its position |
| [join()](http://www.w3schools.com/jsref/jsref_join.asp) | Joins all elements of an array into a string |
| [lastIndexOf()](http://www.w3schools.com/jsref/jsref_lastindexof_array.asp) | Search the array for an element, starting at the end, and returns its position |
| [pop()](http://www.w3schools.com/jsref/jsref_pop.asp) | Removes the last element of an array, and returns that element |
| [push()](http://www.w3schools.com/jsref/jsref_push.asp) | Adds new elements to the end of an array, and returns the new length |
| [reverse()](http://www.w3schools.com/jsref/jsref_reverse.asp) | Reverses the order of the elements in an array |
| [shift()](http://www.w3schools.com/jsref/jsref_shift.asp) | Removes the first element of an array, and returns that element |
| [slice()](http://www.w3schools.com/jsref/jsref_slice_array.asp) | Selects a part of an array, and returns the new array |
| [sort()](http://www.w3schools.com/jsref/jsref_sort.asp) | Sorts the elements of an array |
| [splice()](http://www.w3schools.com/jsref/jsref_splice.asp) | Adds/Removes elements from an array |
| [toString()](http://www.w3schools.com/jsref/jsref_tostring_array.asp) | Converts an array to a string, and returns the result |
| [unshift()](http://www.w3schools.com/jsref/jsref_unshift.asp) | Adds new elements to the beginning of an array, and returns the new length |
| [valueOf()](http://www.w3schools.com/jsref/jsref_valueof_array.asp) | Returns the primitive value of an array |

**JavaScript Boolean Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_obj_array.asp)

[Next Reference »](http://www.w3schools.com/jsref/jsref_obj_regexp.asp)

**JavaScript Booleans**

JavaScript booleans can have one of two values: **true** or **false**.

**The Boolean() Function**

You can use the Boolean() function to find out if an expression is true:

**Example**

Boolean(10 > 9)       // returns true

[Try it Yourself »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_boolean_expression1)

Or even easier:

**Example**

(10 > 9)              // also returns true  
10 > 9                // also returns true

[Try it Yourself »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_boolean_expression2)

For a tutorial about booleans, read our [JavaScript Boolean Tutorial](http://www.w3schools.com/js/js_booleans.asp).

**Boolean Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](http://www.w3schools.com/jsref/jsref_constructor_boolean.asp) | Returns the function that created JavaScript's Boolean prototype |
| [prototype](http://www.w3schools.com/jsref/jsref_prototype_boolean.asp) | Allows you to add properties and methods to the Boolean prototype |

**Boolean Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [toString()](http://www.w3schools.com/jsref/jsref_tostring_boolean.asp) | Converts a boolean value to a string, and returns the result |
| [valueOf()](http://www.w3schools.com/jsref/jsref_valueof_boolean.asp) | Returns the primitive value of a boolean |

**JavaScript RegExp Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_obj_boolean.asp)

[Next Reference »](http://www.w3schools.com/jsref/jsref_obj_global.asp)

**RegExp Object**

A regular expression is an object that describes a pattern of characters.

Regular expressions are used to perform pattern-matching and "search-and-replace" functions on text.

**Syntax**

/*pattern*/*modifiers*;

**Example**

var patt = /w3schools/i

Example explained:

* **/w3schools/i**  is a regular expression.
* **w3schools**  is a pattern (to be used in a search).
* **i**  is a modifier (modifies the search to be case-insensitive).

For a tutorial about Regular Expressions, read our [JavaScript RegExp Tutorial](http://www.w3schools.com/js/js_regexp.asp).

**Modifiers**

Modifiers are used to perform case-insensitive and global searches:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| [i](http://www.w3schools.com/jsref/jsref_regexp_i.asp) | Perform case-insensitive matching |
| [g](http://www.w3schools.com/jsref/jsref_regexp_g.asp) | Perform a global match (find all matches rather than stopping after the first match) |
| [m](http://www.w3schools.com/jsref/jsref_regexp_m.asp) | Perform multiline matching |

**Brackets**

Brackets are used to find a range of characters:

|  |  |
| --- | --- |
| **Expression** | **Description** |
| [[abc]](http://www.w3schools.com/jsref/jsref_regexp_charset.asp) | Find any character between the brackets |
| [[^abc]](http://www.w3schools.com/jsref/jsref_regexp_charset_not.asp) | Find any character NOT between the brackets |
| [[0-9]](http://www.w3schools.com/jsref/jsref_regexp_0-9.asp) | Find any digit between the brackets |
| [[^0-9]](http://www.w3schools.com/jsref/jsref_regexp_not_0-9.asp) | Find any digit NOT between the brackets |
| [(x|y)](http://www.w3schools.com/jsref/jsref_regexp_xy.asp) | Find any of the alternatives specified |

**Metacharacters**

Metacharacters are characters with a special meaning:

|  |  |
| --- | --- |
| **Metacharacter** | **Description** |
| [.](http://www.w3schools.com/jsref/jsref_regexp_dot.asp) | Find a single character, except newline or line terminator |
| [\w](http://www.w3schools.com/jsref/jsref_regexp_wordchar.asp) | Find a word character |
| [\W](http://www.w3schools.com/jsref/jsref_regexp_wordchar_non.asp) | Find a non-word character |
| [\d](http://www.w3schools.com/jsref/jsref_regexp_digit.asp) | Find a digit |
| [\D](http://www.w3schools.com/jsref/jsref_regexp_digit_non.asp) | Find a non-digit character |
| [\s](http://www.w3schools.com/jsref/jsref_regexp_whitespace.asp) | Find a whitespace character |
| [\S](http://www.w3schools.com/jsref/jsref_regexp_whitespace_non.asp) | Find a non-whitespace character |
| [\b](http://www.w3schools.com/jsref/jsref_regexp_begin.asp) | Find a match at the beginning/end of a word |
| [\B](http://www.w3schools.com/jsref/jsref_regexp_begin_not.asp) | Find a match not at the beginning/end of a word |
| [\0](http://www.w3schools.com/jsref/jsref_regexp_nul.asp) | Find a NUL character |
| [\n](http://www.w3schools.com/jsref/jsref_regexp_newline.asp) | Find a new line character |
| [\f](http://www.w3schools.com/jsref/jsref_regexp_formfeed.asp) | Find a form feed character |
| [\r](http://www.w3schools.com/jsref/jsref_regexp_carriagereturn.asp) | Find a carriage return character |
| [\t](http://www.w3schools.com/jsref/jsref_regexp_tab.asp) | Find a tab character |
| [\v](http://www.w3schools.com/jsref/jsref_regexp_vtab.asp) | Find a vertical tab character |
| [\xxx](http://www.w3schools.com/jsref/jsref_regexp_octal.asp) | Find the character specified by an octal number xxx |
| [\xdd](http://www.w3schools.com/jsref/jsref_regexp_hex.asp) | Find the character specified by a hexadecimal number dd |
| [\uxxxx](http://www.w3schools.com/jsref/jsref_regexp_unicode_hex.asp) | Find the Unicode character specified by a hexadecimal number xxxx |

**Quantifiers**

|  |  |
| --- | --- |
| **Quantifier** | **Description** |
| [n+](http://www.w3schools.com/jsref/jsref_regexp_onemore.asp) | Matches any string that contains at least one *n* |
| [n\*](http://www.w3schools.com/jsref/jsref_regexp_zeromore.asp) | Matches any string that contains zero or more occurrences of *n* |
| [n?](http://www.w3schools.com/jsref/jsref_regexp_zeroone.asp) | Matches any string that contains zero or one occurrences of *n* |
| [n{X}](http://www.w3schools.com/jsref/jsref_regexp_nx.asp) | Matches any string that contains a sequence of *X* *n*'s |
| [n{X,Y}](http://www.w3schools.com/jsref/jsref_regexp_nxy.asp) | Matches any string that contains a sequence of X to Y *n*'s |
| [n{X,}](http://www.w3schools.com/jsref/jsref_regexp_nxcomma.asp) | Matches any string that contains a sequence of at least X *n*'s |
| [n$](http://www.w3schools.com/jsref/jsref_regexp_ndollar.asp) | Matches any string with *n* at the end of it |
| [^n](http://www.w3schools.com/jsref/jsref_regexp_ncaret.asp) | Matches any string with *n* at the beginning of it |
| [?=n](http://www.w3schools.com/jsref/jsref_regexp_nfollow.asp) | Matches any string that is followed by a specific string *n* |
| [?!n](http://www.w3schools.com/jsref/jsref_regexp_nfollow_not.asp) | Matches any string that is not followed by a specific string *n* |

**RegExp Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](http://www.w3schools.com/jsref/jsref_regexp_constructor.asp) | Returns the function that created the RegExp object's prototype |
| [global](http://www.w3schools.com/jsref/jsref_regexp_global.asp) | Checks whether the "g" modifier is set |
| [ignoreCase](http://www.w3schools.com/jsref/jsref_regexp_ignorecase.asp) | Checks whether the "i" modifier is set |
| [lastIndex](http://www.w3schools.com/jsref/jsref_regexp_lastindex.asp) | Specifies the index at which to start the next match |
| [multiline](http://www.w3schools.com/jsref/jsref_regexp_multiline.asp) | Checks whether the "m" modifier is set |
| [source](http://www.w3schools.com/jsref/jsref_regexp_source.asp) | Returns the text of the RegExp pattern |

**RegExp Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [compile()](http://www.w3schools.com/jsref/jsref_regexp_compile.asp) | Deprecated in version 1.5. Compiles a regular expression |
| [exec()](http://www.w3schools.com/jsref/jsref_regexp_exec.asp) | Tests for a match in a string. Returns the first match |
| [test()](http://www.w3schools.com/jsref/jsref_regexp_test.asp) | Tests for a match in a string. Returns true or false |
| [toString()](http://www.w3schools.com/jsref/jsref_regexp_tostring.asp) | Returns the string value of the regular expression |

**JavaScript Global Reference**

[« Previous](http://www.w3schools.com/jsref/jsref_obj_regexp.asp)

[Next Reference »](http://www.w3schools.com/jsref/jsref_type_conversion.asp)

The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

**JavaScript Global Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [Infinity](http://www.w3schools.com/jsref/jsref_infinity.asp) | A numeric value that represents positive/negative infinity |
| [NaN](http://www.w3schools.com/jsref/jsref_nan.asp) | "Not-a-Number" value |
| [undefined](http://www.w3schools.com/jsref/jsref_undefined.asp) | Indicates that a variable has not been assigned a value |

**JavaScript Global Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| [decodeURI()](http://www.w3schools.com/jsref/jsref_decodeuri.asp) | Decodes a URI |
| [decodeURIComponent()](http://www.w3schools.com/jsref/jsref_decodeuricomponent.asp) | Decodes a URI component |
| [encodeURI()](http://www.w3schools.com/jsref/jsref_encodeuri.asp) | Encodes a URI |
| [encodeURIComponent()](http://www.w3schools.com/jsref/jsref_encodeuricomponent.asp) | Encodes a URI component |
| [escape()](http://www.w3schools.com/jsref/jsref_escape.asp) | Deprecated in version 1.5. Use [encodeURI()](http://www.w3schools.com/jsref/jsref_encodeuri.asp) or [encodeURIComponent()](http://www.w3schools.com/jsref/jsref_encodeuricomponent.asp) instead |
| [eval()](http://www.w3schools.com/jsref/jsref_eval.asp) | Evaluates a string and executes it as if it was script code |
| [isFinite()](http://www.w3schools.com/jsref/jsref_isfinite.asp) | Determines whether a value is a finite, legal number |
| [isNaN()](http://www.w3schools.com/jsref/jsref_isnan.asp) | Determines whether a value is an illegal number |
| [Number()](http://www.w3schools.com/jsref/jsref_number.asp) | Converts an object's value to a number |
| [parseFloat()](http://www.w3schools.com/jsref/jsref_parsefloat.asp) | Parses a string and returns a floating point number |
| [parseInt()](http://www.w3schools.com/jsref/jsref_parseint.asp) | Parses a string and returns an integer |
| [String()](http://www.w3schools.com/jsref/jsref_string.asp) | Converts an object's value to a string |
| [unescape()](http://www.w3schools.com/jsref/jsref_unescape.asp) | Deprecated in version 1.5. Use [decodeURI()](http://www.w3schools.com/jsref/jsref_decodeuri.asp) or [decodeURIComponent()](http://www.w3schools.com/jsref/jsref_decodeuricomponent.asp) instead |

**JavaScript Type Conversion**

[« Previous](http://www.w3schools.com/jsref/jsref_obj_global.asp)

[Next Reference »](http://www.w3schools.com/jsref/obj_window.asp)

**JavaScript Type Conversion Table**

The table below shows the result of converting different JavaScript values to Number, String, and Boolean:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Original Value** | **Converted to Number** | **Converted to String** | **Converted to Boolean** | **Try it** |
| false | 0 | "false" | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_false) |
| true | 1 | "true" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_true) |
| 0 | 0 | "0" | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_number_0) |
| 1 | 1 | "1" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_number_1) |
| "0" | 0 | "0" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_string_0) |
| "1" | 1 | "1" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_string_1) |
| NaN | NaN | "NaN" | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_nan) |
| Infinity | Infinity | "Infinity" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_infinity) |
| -Infinity | -Infinity | "-Infinity" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_infinity_minus) |
| "" | 0 | "" | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_string_empty) |
| "20" | 20 | "20" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_string_number) |
| "twenty" | NaN | "twenty" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_string_text) |
| [ ] | 0 | "" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_array_empty) |
| [20] | 20 | "20" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_array_one_number) |
| [10,20] | NaN | "10,20" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_array_two_numbers) |
| ["twenty"] | NaN | "twenty" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_array_one_string) |
| ["ten","twenty"] | NaN | "ten,twenty" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_array_two_strings) |
| function(){} | NaN | "function(){}" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_function) |
| { } | NaN | "[object Object]" | true | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_object) |
| null | 0 | "null" | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_null) |
| undefined | NaN | "undefined" | false | [Try it »](http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_type_convert_undefined) |

**Note:** Values in quotes ("") indicate string values. Values in red indicate values (some) programmers might not expect.

For a tutorial about JavaScript Type Conversion, read our [JavaScript Type Conversion Tutorial](http://www.w3schools.com/js/js_type_conversion.asp).

**Window Object**

The window object represents an open window in a browser.

If a document contain frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

**Note:** There is no public standard that applies to the Window object, but all major browsers support it.

**Window Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [closed](http://www.w3schools.com/jsref/prop_win_closed.asp) | Returns a Boolean value indicating whether a window has been closed or not |
| [defaultStatus](http://www.w3schools.com/jsref/prop_win_defaultstatus.asp) | Sets or returns the default text in the statusbar of a window |
| [document](http://www.w3schools.com/jsref/dom_obj_document.asp) | Returns the Document object for the window ([See Document object](http://www.w3schools.com/jsref/dom_obj_document.asp)) |
| [frameElement](http://www.w3schools.com/jsref/prop_win_frameElement.asp) | Returns the <iframe> element in which the current window is inserted |
| [frames](http://www.w3schools.com/jsref/prop_win_frames.asp) | Returns all <iframe> elements in the current window |
| [history](http://www.w3schools.com/jsref/obj_history.asp) | Returns the History object for the window ([See History object](http://www.w3schools.com/jsref/obj_history.asp)) |
| [innerHeight](http://www.w3schools.com/jsref/prop_win_innerheight.asp) | Returns the inner height of a window's content area |
| [innerWidth](http://www.w3schools.com/jsref/prop_win_innerheight.asp) | Returns the inner width of a window's content area |
| [length](http://www.w3schools.com/jsref/prop_win_length.asp) | Returns the number of <iframe> elements in the current window |
| localStorage | Returns a reference to the local storage object used to store data. Stores data with no expiration date |
| [location](http://www.w3schools.com/jsref/obj_location.asp) | Returns the Location object for the window ([See Location object](http://www.w3schools.com/jsref/obj_location.asp)) |
| [name](http://www.w3schools.com/jsref/prop_win_name.asp) | Sets or returns the name of a window |
| [navigator](http://www.w3schools.com/jsref/obj_navigator.asp) | Returns the Navigator object for the window ([See Navigator object](http://www.w3schools.com/jsref/obj_navigator.asp)) |
| [opener](http://www.w3schools.com/jsref/prop_win_opener.asp) | Returns a reference to the window that created the window |
| [outerHeight](http://www.w3schools.com/jsref/prop_win_outerheight.asp) | Returns the outer height of a window, including toolbars/scrollbars |
| [outerWidth](http://www.w3schools.com/jsref/prop_win_outerheight.asp) | Returns the outer width of a window, including toolbars/scrollbars |
| [pageXOffset](http://www.w3schools.com/jsref/prop_win_pagexoffset.asp) | Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window |
| [pageYOffset](http://www.w3schools.com/jsref/prop_win_pagexoffset.asp) | Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window |
| [parent](http://www.w3schools.com/jsref/prop_win_parent.asp) | Returns the parent window of the current window |
| [screen](http://www.w3schools.com/jsref/obj_screen.asp) | Returns the Screen object for the window [(See Screen object)](http://www.w3schools.com/jsref/obj_screen.asp) |
| [screenLeft](http://www.w3schools.com/jsref/prop_win_screenleft.asp) | Returns the horizontal coordinate of the window relative to the screen |
| [screenTop](http://www.w3schools.com/jsref/prop_win_screenleft.asp) | Returns the vertical coordinate of the window relative to the screen |
| [screenX](http://www.w3schools.com/jsref/prop_win_screenx.asp) | Returns the horizontal coordinate of the window relative to the screen |
| [screenY](http://www.w3schools.com/jsref/prop_win_screenx.asp) | Returns the vertical coordinate of the window relative to the screen |
| sessionStorage | Returns a reference to the local storage object used to store data. Stores data for one session (lost when the browser tab is closed) |
| scrollX | An alias of [pageXOffset](http://www.w3schools.com/jsref/prop_win_pagexoffset.asp) |
| scrollY | An alias of [pageYOffset](http://www.w3schools.com/jsref/prop_win_pagexoffset.asp) |
| [self](http://www.w3schools.com/jsref/prop_win_self.asp) | Returns the current window |
| [status](http://www.w3schools.com/jsref/prop_win_status.asp) | Sets or returns the text in the statusbar of a window |
| [top](http://www.w3schools.com/jsref/prop_win_top.asp) | Returns the topmost browser window |

**Window Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [alert()](http://www.w3schools.com/jsref/met_win_alert.asp) | Displays an alert box with a message and an OK button |
| [atob()](http://www.w3schools.com/jsref/met_win_atob.asp) | Decodes a base-64 encoded string |
| [blur()](http://www.w3schools.com/jsref/met_win_blur.asp) | Removes focus from the current window |
| [btoa()](http://www.w3schools.com/jsref/met_win_btoa.asp) | Encodes a string in base-64 |
| [clearInterval()](http://www.w3schools.com/jsref/met_win_clearinterval.asp) | Clears a timer set with setInterval() |
| [clearTimeout()](http://www.w3schools.com/jsref/met_win_cleartimeout.asp) | Clears a timer set with setTimeout() |
| [close()](http://www.w3schools.com/jsref/met_win_close.asp) | Closes the current window |
| [confirm()](http://www.w3schools.com/jsref/met_win_confirm.asp) | Displays a dialog box with a message and an OK and a Cancel button |
| [createPopup()](http://www.w3schools.com/jsref/met_win_createpopup.asp) | Creates a pop-up window |
| [focus()](http://www.w3schools.com/jsref/met_win_focus.asp) | Sets focus to the current window |
| getComputedStyle() | Gets the current computed CSS styles applied to an element |
| getSelection() | Returns a Selection object representing the range of text selected by the user |
| matchMedia() | Returns a MediaQueryList object representing the specified CSS media query string |
| [moveBy()](http://www.w3schools.com/jsref/met_win_moveby.asp) | Moves a window relative to its current position |
| [moveTo()](http://www.w3schools.com/jsref/met_win_moveto.asp) | Moves a window to the specified position |
| [open()](http://www.w3schools.com/jsref/met_win_open.asp) | Opens a new browser window |
| [print()](http://www.w3schools.com/jsref/met_win_print.asp) | Prints the content of the current window |
| [prompt()](http://www.w3schools.com/jsref/met_win_prompt.asp) | Displays a dialog box that prompts the visitor for input |
| [resizeBy()](http://www.w3schools.com/jsref/met_win_resizeby.asp) | Resizes the window by the specified pixels |
| [resizeTo()](http://www.w3schools.com/jsref/met_win_resizeto.asp) | Resizes the window to the specified width and height |
| scroll() | Deprecated. This method has been replaced by the [scrollTo()](http://www.w3schools.com/jsref/met_win_scrollto.asp) method. |
| [scrollBy()](http://www.w3schools.com/jsref/met_win_scrollby.asp) | Scrolls the document by the specified number of pixels |
| [scrollTo()](http://www.w3schools.com/jsref/met_win_scrollto.asp) | Scrolls the document to the specified coordinates |
| [setInterval()](http://www.w3schools.com/jsref/met_win_setinterval.asp) | Calls a function or evaluates an expression at specified intervals (in milliseconds) |
| [setTimeout()](http://www.w3schools.com/jsref/met_win_settimeout.asp) | Calls a function or evaluates an expression after a specified number of milliseconds |
| [stop()](http://www.w3schools.com/jsref/met_win_stop.asp) | Stops the window from loading |

**The Navigator Object**

[« Previous](http://www.w3schools.com/jsref/obj_window.asp)

[Next Reference »](http://www.w3schools.com/jsref/obj_screen.asp)

**Navigator Object**

The navigator object contains information about the browser.

**Note:** There is no public standard that applies to the navigator object, but all major browsers support it.

**Navigator Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [appCodeName](http://www.w3schools.com/jsref/prop_nav_appcodename.asp) | Returns the code name of the browser |
| [appName](http://www.w3schools.com/jsref/prop_nav_appname.asp) | Returns the name of the browser |
| [appVersion](http://www.w3schools.com/jsref/prop_nav_appversion.asp) | Returns the version information of the browser |
| [cookieEnabled](http://www.w3schools.com/jsref/prop_nav_cookieenabled.asp) | Determines whether cookies are enabled in the browser |
| [geolocation](http://www.w3schools.com/jsref/prop_nav_geolocation.asp) | Returns a Geolocation object that can be used to locate the user's position |
| [language](http://www.w3schools.com/jsref/prop_nav_language.asp) | Returns the language of the browser |
| [onLine](http://www.w3schools.com/jsref/prop_nav_online.asp) | Determines whether the browser is online |
| [platform](http://www.w3schools.com/jsref/prop_nav_platform.asp) | Returns for which platform the browser is compiled |
| [product](http://www.w3schools.com/jsref/prop_nav_product.asp) | Returns the engine name of the browser |
| [userAgent](http://www.w3schools.com/jsref/prop_nav_useragent.asp) | Returns the user-agent header sent by the browser to the server |

**Navigator Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [javaEnabled()](http://www.w3schools.com/jsref/met_nav_javaenabled.asp) | Specifies whether or not the browser has Java enabled |
| [taintEnabled()](http://www.w3schools.com/jsref/met_nav_taintenabled.asp) | Removed in JavaScript version 1.2. Specifies whether the browser has data tainting enabled |

**The Screen Object**

[« Previous](http://www.w3schools.com/jsref/obj_navigator.asp)

[Next Reference »](http://www.w3schools.com/jsref/obj_history.asp)

**Screen Object**

The screen object contains information about the visitor's screen.

**Note:** There is no public standard that applies to the screen object, but all major browsers support it.

**Screen Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [availHeight](http://www.w3schools.com/jsref/prop_screen_availheight.asp) | Returns the height of the screen (excluding the Windows Taskbar) |
| [availWidth](http://www.w3schools.com/jsref/prop_screen_availwidth.asp) | Returns the width of the screen (excluding the Windows Taskbar) |
| [colorDepth](http://www.w3schools.com/jsref/prop_screen_colordepth.asp) | Returns the bit depth of the color palette for displaying images |
| [height](http://www.w3schools.com/jsref/prop_screen_height.asp) | Returns the total height of the screen |
| [pixelDepth](http://www.w3schools.com/jsref/prop_screen_pixeldepth.asp) | Returns the color resolution (in bits per pixel) of the screen |
| [width](http://www.w3schools.com/jsref/prop_screen_width.asp) | Returns the total width of the screen |

**The History Object**

[« Previous](http://www.w3schools.com/jsref/obj_screen.asp)

[Next Reference »](http://www.w3schools.com/jsref/obj_location.asp)

**History Object**

The history object contains the URLs visited by the user (within a browser window).

The history object is part of the window object and is accessed through the window.history property.

**Note:** There is no public standard that applies to the history object, but all major browsers support it.

**History Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [length](http://www.w3schools.com/jsref/prop_his_length.asp) | Returns the number of URLs in the history list |

**History Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [back()](http://www.w3schools.com/jsref/met_his_back.asp) | Loads the previous URL in the history list |
| [forward()](http://www.w3schools.com/jsref/met_his_forward.asp) | Loads the next URL in the history list |
| [go()](http://www.w3schools.com/jsref/met_his_go.asp) | Loads a specific URL from the history list |

**The Location Object**

[« Previous](http://www.w3schools.com/jsref/obj_history.asp)

[Next Reference »](http://www.w3schools.com/jsref/dom_obj_document.asp)

**Location Object**

The location object contains information about the current URL.

The location object is part of the window object and is accessed through the window.location property.

**Note:** There is no public standard that applies to the location object, but all major browsers support it.

**Location Object Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [hash](http://www.w3schools.com/jsref/prop_loc_hash.asp) | Sets or returns the anchor part (#) of a URL |
| [host](http://www.w3schools.com/jsref/prop_loc_host.asp) | Sets or returns the hostname and port number of a URL |
| [hostname](http://www.w3schools.com/jsref/prop_loc_hostname.asp) | Sets or returns the hostname of a URL |
| [href](http://www.w3schools.com/jsref/prop_loc_href.asp) | Sets or returns the entire URL |
| [origin](http://www.w3schools.com/jsref/prop_loc_origin.asp) | Returns the protocol, hostname and port number of a URL |
| [pathname](http://www.w3schools.com/jsref/prop_loc_pathname.asp) | Sets or returns the path name of a URL |
| [port](http://www.w3schools.com/jsref/prop_loc_port.asp) | Sets or returns the port number of a URL |
| [protocol](http://www.w3schools.com/jsref/prop_loc_protocol.asp) | Sets or returns the protocol of a URL |
| [search](http://www.w3schools.com/jsref/prop_loc_search.asp) | Sets or returns the querystring part of a URL |

**Location Object Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [assign()](http://www.w3schools.com/jsref/met_loc_assign.asp) | Loads a new document |
| [reload()](http://www.w3schools.com/jsref/met_loc_reload.asp) | Reloads the current document |
| [replace()](http://www.w3schools.com/jsref/met_loc_replace.asp) | Replaces the current document with a new one |

**The HTML DOM Document Object**

[« Previous](http://www.w3schools.com/jsref/obj_location.asp)

[Next Reference »](http://www.w3schools.com/jsref/dom_obj_all.asp)

**HTML DOM Nodes**

In the HTML DOM (Document Object Model), everything is a **node**:

* The document itself is a document node
* All HTML elements are element nodes
* All HTML attributes are attribute nodes
* Text inside HTML elements are text nodes
* Comments are comment nodes

**The Document Object**

When an HTML document is loaded into a web browser, it becomes a **document object**.

The document object is the root node of the HTML document and the "owner" of all other nodes:  
(element nodes, text nodes, attribute nodes, and comment nodes).

The document object provides properties and methods to access all node objects, from within JavaScript.

**Tip:** The document is a part of the Window object and can be accessed as window.document.

**Browser Support**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Object** |  |  |  |  |  |
| Document | Yes | Yes | Yes | Yes | Yes |

The Document Object is supported in all major browsers.

**Document Object Properties and Methods**

The following properties and methods can be used on HTML documents:

|  |  |
| --- | --- |
| **Property / Method** | **Description** |
| [document.activeElement](http://www.w3schools.com/jsref/prop_document_activeelement.asp) | Returns the currently focused element in the document |
| [document.addEventListener()](http://www.w3schools.com/jsref/met_document_addeventlistener.asp) | Attaches an event handler to the document |
| [document.adoptNode()](http://www.w3schools.com/jsref/met_document_adoptnode.asp) | Adopts a node from another document |
| [document.anchors](http://www.w3schools.com/jsref/coll_doc_anchors.asp) | Returns a collection of all <a> elements in the document that have a name attribute |
| [document.applets](http://www.w3schools.com/jsref/coll_doc_applets.asp) | Returns a collection of all <applet> elements in the document |
| [document.baseURI](http://www.w3schools.com/jsref/prop_doc_baseuri.asp) | Returns the absolute base URI of a document |
| [document.body](http://www.w3schools.com/jsref/prop_doc_body.asp) | Sets or returns the document's body (the <body> element) |
| [document.close()](http://www.w3schools.com/jsref/met_doc_close.asp) | Closes the output stream previously opened with document.open() |
| [document.cookie](http://www.w3schools.com/jsref/prop_doc_cookie.asp) | Returns all name/value pairs of cookies in the document |
| [document.createAttribute()](http://www.w3schools.com/jsref/met_document_createattribute.asp) | Creates an attribute node |
| [document.createComment()](http://www.w3schools.com/jsref/met_document_createcomment.asp) | Creates a Comment node with the specified text |
| [document.createDocumentFragment()](http://www.w3schools.com/jsref/met_document_createdocumentfragment.asp) | Creates an empty DocumentFragment node |
| [document.createElement()](http://www.w3schools.com/jsref/met_document_createelement.asp) | Creates an Element node |
| [document.createTextNode()](http://www.w3schools.com/jsref/met_document_createtextnode.asp) | Creates a Text node |
| [document.doctype](http://www.w3schools.com/jsref/prop_document_doctype.asp) | Returns the Document Type Declaration associated with the document |
| [document.documentElement](http://www.w3schools.com/jsref/prop_document_documentelement.asp) | Returns the Document Element of the document (the <html> element) |
| [document.documentMode](http://www.w3schools.com/jsref/prop_doc_documentmode.asp) | Returns the mode used by the browser to render the document |
| [document.documentURI](http://www.w3schools.com/jsref/prop_document_documenturi.asp) | Sets or returns the location of the document |
| [document.domain](http://www.w3schools.com/jsref/prop_doc_domain.asp) | Returns the domain name of the server that loaded the document |
| document.domConfig | Obsolete. Returns the DOM configuration of the document |
| [document.embeds](http://www.w3schools.com/jsref/coll_doc_embeds.asp) | Returns a collection of all <embed> elements the document |
| [document.forms](http://www.w3schools.com/jsref/coll_doc_forms.asp) | Returns a collection of all <form> elements in the document |
| [document.getElementById()](http://www.w3schools.com/jsref/met_document_getelementbyid.asp) | Returns the element that has the ID attribute with the specified value |
| [document.getElementsByClassName()](http://www.w3schools.com/jsref/met_document_getelementsbyclassname.asp) | Returns a NodeList containing all elements with the specified class name |
| [document.getElementsByName()](http://www.w3schools.com/jsref/met_doc_getelementsbyname.asp) | Returns a NodeList containing all elements with a specified name |
| [document.getElementsByTagName()](http://www.w3schools.com/jsref/met_document_getelementsbytagname.asp) | Returns a NodeList containing all elements with the specified tag name |
| [document.hasFocus()](http://www.w3schools.com/jsref/met_document_hasfocus.asp) | Returns a Boolean value indicating whether the document has focus |
| [document.head](http://www.w3schools.com/jsref/prop_doc_head.asp) | Returns the <head> element of the document |
| [document.images](http://www.w3schools.com/jsref/coll_doc_images.asp) | Returns a collection of all <img> elements in the document |
| [document.implementation](http://www.w3schools.com/jsref/prop_document_implementation.asp) | Returns the DOMImplementation object that handles this document |
| [document.importNode()](http://www.w3schools.com/jsref/met_document_importnode.asp) | Imports a node from another document |
| [document.inputEncoding](http://www.w3schools.com/jsref/prop_document_inputencoding.asp) | Returns the encoding, character set, used for the document |
| [document.lastModified](http://www.w3schools.com/jsref/prop_doc_lastmodified.asp) | Returns the date and time the document was last modified |
| [document.links](http://www.w3schools.com/jsref/coll_doc_links.asp) | Returns a collection of all <a> and <area> elements in the document that have a href attribute |
| [document.normalize()](http://www.w3schools.com/jsref/met_document_normalize.asp) | Removes empty Text nodes, and joins adjacent nodes |
| [document.normalizeDocument()](http://www.w3schools.com/jsref/met_document_normalizedocument.asp) | Removes empty Text nodes, and joins adjacent nodes |
| [document.open()](http://www.w3schools.com/jsref/met_doc_open.asp) | Opens an HTML output stream to collect output from document.write() |
| [document.querySelector()](http://www.w3schools.com/jsref/met_document_queryselector.asp) | Returns the first element that matches a specified CSS selector(s) in the document |
| [document.querySelectorAll()](http://www.w3schools.com/jsref/met_document_queryselectorall.asp) | Returns a static NodeList containing all elements that matches a specified CSS selector(s) in the document |
| [document.readyState](http://www.w3schools.com/jsref/prop_doc_readystate.asp) | Returns the (loading) status of the document |
| [document.referrer](http://www.w3schools.com/jsref/prop_doc_referrer.asp) | Returns the URL of the document that loaded the current document |
| [document.removeEventListener()](http://www.w3schools.com/jsref/met_document_removeeventlistener.asp) | Removes an event handler from the document (that has been attached with the [addEventListener()](http://www.w3schools.com/jsref/met_document_addeventlistener.asp) method) |
| [document.renameNode()](http://www.w3schools.com/jsref/met_document_renamenode.asp) | Renames the specified node |
| [document.scripts](http://www.w3schools.com/jsref/coll_doc_scripts.asp) | Returns a collection of <script> elements in the document |
| [document.strictErrorChecking](http://www.w3schools.com/jsref/prop_document_stricterrorchecking.asp) | Sets or returns whether error-checking is enforced or not |
| [document.title](http://www.w3schools.com/jsref/prop_doc_title.asp) | Sets or returns the title of the document |
| [document.URL](http://www.w3schools.com/jsref/prop_doc_url.asp) | Returns the full URL of the HTML document |
| [document.write()](http://www.w3schools.com/jsref/met_doc_write.asp) | Writes HTML expressions or JavaScript code to a document |
| [document.writeln()](http://www.w3schools.com/jsref/met_doc_writeln.asp) | Same as write(), but adds a newline character after each statement |

**Warning !!!**

In the W3C DOM Core, the Document object inherits all properties and methods from the Node object.

Many of these properties and methods make no sense used on documents.

**Avoid using these node object properties and methods on HTML document objects:**

|  |  |
| --- | --- |
| **Property / Method** | **Reason for avoiding** |
| document.attributes | Documents don't have attributes |
| document.hasAttributes() | Documents don't have attributes |
| document.nextSibling | Documents don't have siblings |
| document.nodeName | This is always #document |
| document.nodeType | This is always 9 (DOCUMENT\_NODE) |
| document.nodeValue | Documents don't have an node value |
| document.ownerDocument | Documents don't have an owner document |
| document.ownerElement | Documents don't have an owner element |
| document.parentNode | Documents don't have a parent node |
| document.previousSibling | Documents don't have siblings |
| document.textContent | Documents don't have a text content |

**The HTML DOM Element Object**

[« Previous](http://www.w3schools.com/jsref/dom_obj_document.asp)

[Next Reference »](http://www.w3schools.com/jsref/dom_obj_attributes.asp)

**HTML DOM Nodes**

In the HTML DOM (Document Object Model), everything is a **node**:

* The document itself is a document node
* All HTML elements are element nodes
* All HTML attributes are attribute nodes
* Text inside HTML elements are text nodes
* Comments are comment nodes

**The Element Object**

In the HTML DOM, the **Element object** represents an HTML element.

Element objects can have **child nodes** of type element nodes, text nodes, or comment nodes.

A **NodeList object** represents a list of nodes, like an HTML element's collection of child nodes.

Elements can also have attributes. Attributes are attribute nodes (See next chapter).

**Browser Support**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Object** |  |  |  |  |  |
| Element | Yes | Yes | Yes | Yes | Yes |
| NodeList | Yes | Yes | Yes | Yes | Yes |

The Element Object and the NodeList Object is supported in all major browsers.

**Properties and Methods**

The following properties and methods can be used on all HTML elements:

|  |  |
| --- | --- |
| **Property / Method** | **Description** |
| [*element*.accessKey](http://www.w3schools.com/jsref/prop_html_accesskey.asp) | Sets or returns the accesskey attribute of an element |
| [*element*.addEventListener()](http://www.w3schools.com/jsref/met_element_addeventlistener.asp) | Attaches an event handler to the specified element |
| [*element*.appendChild()](http://www.w3schools.com/jsref/met_node_appendchild.asp) | Adds a new child node, to an element, as the last child node |
| [*element*.attributes](http://www.w3schools.com/jsref/prop_node_attributes.asp) | Returns a NamedNodeMap of an element's attributes |
| [*element*.blur()](http://www.w3schools.com/jsref/met_html_blur.asp) | Removes focus from an element |
| [*element*.childElementCount](http://www.w3schools.com/jsref/prop_element_childelementcount.asp) | Returns the number of child elements an element has |
| [*element*.childNodes](http://www.w3schools.com/jsref/prop_node_childnodes.asp) | Returns a collection of an element's child nodes (including text and comment nodes) |
| [*element*.children](http://www.w3schools.com/jsref/prop_element_children.asp) | Returns a collection of an element's child element (excluding text and comment nodes) |
| [*element*.classList](http://www.w3schools.com/jsref/prop_element_classlist.asp) | Returns the class name(s) of an element |
| [*element*.className](http://www.w3schools.com/jsref/prop_html_classname.asp) | Sets or returns the value of the class attribute of an element |
| [*element*.click()](http://www.w3schools.com/jsref/met_html_click.asp) | Simulates a mouse-click on an element |
| [*element*.clientHeight](http://www.w3schools.com/jsref/prop_element_clientheight.asp) | Returns the height of an element, including padding |
| [*element*.clientLeft](http://www.w3schools.com/jsref/prop_element_clientleft.asp) | Returns the width of the left border of an element |
| [*element*.clientTop](http://www.w3schools.com/jsref/prop_element_clienttop.asp) | Returns the width of the top border of an element |
| [*element*.clientWidth](http://www.w3schools.com/jsref/prop_element_clientwidth.asp) | Returns the width of an element, including padding |
| [*element*.cloneNode()](http://www.w3schools.com/jsref/met_node_clonenode.asp) | Clones an element |
| [*element*.compareDocumentPosition()](http://www.w3schools.com/jsref/met_node_comparedocumentposition.asp) | Compares the document position of two elements |
| [*element*.contains()](http://www.w3schools.com/jsref/met_node_contains.asp) | Returns true if a node is a descendant of a node, otherwise false |
| [*element*.contentEditable](http://www.w3schools.com/jsref/prop_html_contenteditable.asp) | Sets or returns whether the content of an element is editable or not |
| [*element*.dir](http://www.w3schools.com/jsref/prop_html_dir.asp) | Sets or returns the value of the dir attribute of an element |
| [*element*.firstChild](http://www.w3schools.com/jsref/prop_node_firstchild.asp) | Returns the first child node of an element |
| [*element*.firstElementChild](http://www.w3schools.com/jsref/prop_element_firstelementchild.asp) | Returns the first child element of an element |
| [*element*.focus()](http://www.w3schools.com/jsref/met_html_focus.asp) | Gives focus to an element |
| [*element*.getAttribute()](http://www.w3schools.com/jsref/met_element_getattribute.asp) | Returns the specified attribute value of an element node |
| [*element*.getAttributeNode()](http://www.w3schools.com/jsref/met_element_getattributenode.asp) | Returns the specified attribute node |
| [*element*.getElementsByClassName()](http://www.w3schools.com/jsref/met_element_getelementsbyclassname.asp) | Returns a collection of all child elements with the specified class name |
| [*element*.getElementsByTagName()](http://www.w3schools.com/jsref/met_element_getelementsbytagname.asp) | Returns a collection of all child elements with the specified tag name |
| *element*.getFeature() | Returns an object which implements the APIs of a specified feature |
| [*element*.hasAttribute()](http://www.w3schools.com/jsref/met_element_hasattribute.asp) | Returns true if an element has the specified attribute, otherwise false |
| [*element*.hasAttributes()](http://www.w3schools.com/jsref/met_node_hasattributes.asp) | Returns true if an element has any attributes, otherwise false |
| [*element*.hasChildNodes()](http://www.w3schools.com/jsref/met_node_haschildnodes.asp) | Returns true if an element has any child nodes, otherwise false |
| [*element*.id](http://www.w3schools.com/jsref/prop_html_id.asp) | Sets or returns the value of the id attribute of an element |
| [*element*.innerHTML](http://www.w3schools.com/jsref/prop_html_innerhtml.asp) | Sets or returns the content of an element |
| [*element*.insertBefore()](http://www.w3schools.com/jsref/met_node_insertbefore.asp) | Inserts a new child node before a specified, existing, child node |
| [*element*.isContentEditable](http://www.w3schools.com/jsref/prop_html_iscontenteditable.asp) | Returns true if the content of an element is editable, otherwise false |
| [*element*.isDefaultNamespace()](http://www.w3schools.com/jsref/met_node_isdefaultnamespace.asp) | Returns true if a specified namespaceURI is the default, otherwise false |
| [*element*.isEqualNode()](http://www.w3schools.com/jsref/met_node_isequalnode.asp) | Checks if two elements are equal |
| [*element*.isSameNode()](http://www.w3schools.com/jsref/met_node_issamenode.asp) | Checks if two elements are the same node |
| [*element*.isSupported()](http://www.w3schools.com/jsref/met_node_issupported.asp) | Returns true if a specified feature is supported on the element |
| [*element*.lang](http://www.w3schools.com/jsref/prop_html_lang.asp) | Sets or returns the value of the lang attribute of an element |
| [*element*.lastChild](http://www.w3schools.com/jsref/prop_node_lastchild.asp) | Returns the last child node of an element |
| [*element*.lastElementChild](http://www.w3schools.com/jsref/prop_element_lastelementchild.asp) | Returns the last child element of an element |
| [*element*.namespaceURI](http://www.w3schools.com/jsref/prop_node_namespaceuri.asp) | Returns the namespace URI of an element |
| [*element*.nextSibling](http://www.w3schools.com/jsref/prop_node_nextsibling.asp) | Returns the next node at the same node tree level |
| [*element*.nextElementSibling](http://www.w3schools.com/jsref/prop_element_nextelementsibling.asp) | Returns the next element at the same node tree level |
| [*element*.nodeName](http://www.w3schools.com/jsref/prop_node_nodename.asp) | Returns the name of a node |
| [*element*.nodeType](http://www.w3schools.com/jsref/prop_node_nodetype.asp) | Returns the node type of a node |
| [*element*.nodeValue](http://www.w3schools.com/jsref/prop_node_nodevalue.asp) | Sets or returns the value of a node |
| [*element*.normalize()](http://www.w3schools.com/jsref/met_node_normalize.asp) | Joins adjacent text nodes and removes empty text nodes in an element |
| [*element*.offsetHeight](http://www.w3schools.com/jsref/prop_element_offsetheight.asp) | Returns the height of an element, including padding, border and scrollbar |
| [*element*.offsetWidth](http://www.w3schools.com/jsref/prop_element_offsetwidth.asp) | Returns the width of an element, including padding, border and scrollbar |
| *element*.offsetLeft | Returns the horizontal offset position of an element |
| *element*.offsetParent | Returns the offset container of an element |
| *element*.offsetTop | Returns the vertical offset position of an element |
| [*element*.ownerDocument](http://www.w3schools.com/jsref/prop_node_ownerdocument.asp) | Returns the root element (document object) for an element |
| [*element*.parentNode](http://www.w3schools.com/jsref/prop_node_parentnode.asp) | Returns the parent node of an element |
| [*element*.parentElement](http://www.w3schools.com/jsref/prop_node_parentelement.asp) | Returns the parent element node of an element |
| [*element*.previousSibling](http://www.w3schools.com/jsref/prop_node_previoussibling.asp) | Returns the previous node at the same node tree level |
| [*element*.previousElementSibling](http://www.w3schools.com/jsref/prop_element_previouselementsibling.asp) | Returns the previous element at the same node tree level |
| [*element*.querySelector()](http://www.w3schools.com/jsref/met_element_queryselector.asp) | Returns the first child element that matches a specified CSS selector(s) of an element |
| [*element*.querySelectorAll()](http://www.w3schools.com/jsref/met_element_queryselectorall.asp) | Returns all child elements that matches a specified CSS selector(s) of an element |
| [*element*.removeAttribute()](http://www.w3schools.com/jsref/met_element_removeattribute.asp) | Removes a specified attribute from an element |
| [*element*.removeAttributeNode()](http://www.w3schools.com/jsref/met_element_removeattributenode.asp) | Removes a specified attribute node, and returns the removed node |
| [*element*.removeChild()](http://www.w3schools.com/jsref/met_node_removechild.asp) | Removes a child node from an element |
| [*element*.replaceChild()](http://www.w3schools.com/jsref/met_node_replacechild.asp) | Replaces a child node in an element |
| [*element*.removeEventListener()](http://www.w3schools.com/jsref/met_element_removeeventlistener.asp) | Removes an event handler that has been attached with the addEventListener() method |
| [*element*.scrollHeight](http://www.w3schools.com/jsref/prop_element_scrollheight.asp) | Returns the entire height of an element, including padding |
| [*element*.scrollLeft](http://www.w3schools.com/jsref/prop_element_scrollleft.asp) | Sets or returns the number of pixels an element's content is scrolled horizontally |
| [*element*.scrollTop](http://www.w3schools.com/jsref/prop_element_scrolltop.asp) | Sets or returns the number of pixels an element's content is scrolled vertically |
| [*element*.scrollWidth](http://www.w3schools.com/jsref/prop_element_scrollwidth.asp) | Returns the entire width of an element, including padding |
| [*element*.setAttribute()](http://www.w3schools.com/jsref/met_element_setattribute.asp) | Sets or changes the specified attribute, to the specified value |
| [*element*.setAttributeNode()](http://www.w3schools.com/jsref/met_element_setattributenode.asp) | Sets or changes the specified attribute node |
| [*element*.style](http://www.w3schools.com/jsref/prop_html_style.asp) | Sets or returns the value of the style attribute of an element |
| [*element*.tabIndex](http://www.w3schools.com/jsref/prop_html_tabindex.asp) | Sets or returns the value of the tabindex attribute of an element |
| [*element*.tagName](http://www.w3schools.com/jsref/prop_element_tagname.asp) | Returns the tag name of an element |
| [*element*.textContent](http://www.w3schools.com/jsref/prop_node_textcontent.asp) | Sets or returns the textual content of a node and its descendants |
| [*element*.title](http://www.w3schools.com/jsref/prop_html_title.asp) | Sets or returns the value of the title attribute of an element |
| *element*.toString() | Converts an element to a string |
|  |  |
| [*nodelist*.item()](http://www.w3schools.com/jsref/met_nodelist_item.asp) | Returns the node at the specified index in a NodeList |
| [*nodelist*.length](http://www.w3schools.com/jsref/prop_nodelist_length.asp) | Returns the number of nodes in a NodeList |

**The HTML DOM Attribute Object**

[« Previous](http://www.w3schools.com/jsref/dom_obj_all.asp)

[Next Reference »](http://www.w3schools.com/jsref/dom_obj_event.asp)

**HTML DOM Nodes**

In the HTML DOM (Document Object Model), everything is a **node**:

* The document itself is a document node
* All HTML elements are element nodes
* All HTML attributes are attribute nodes
* Text inside HTML elements are text nodes
* Comments are comment nodes

**The Attr Object**

In the HTML DOM, the **Attr object** represents an HTML attribute.

An HTML attribute always belongs to an HTML element.

**The NamedNodeMap Object**

In the HTML DOM, the **NamedNodeMap object** represents an unordered collection of an elements attribute nodes.

Nodes in a NamedNodeMap can be accessed by name or by index (number).

**Browser Support**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Object** |  |  |  |  |  |
| Attr | Yes | Yes | Yes | Yes | Yes |
| NamedNodeMap | Yes | Yes | Yes | Yes | Yes |

The Attr Object and the NamedNodeMap Object is supported in all major browsers.

**Properties and Methods**

|  |  |
| --- | --- |
| **Property / Method** | **Description** |
| [*attr*.isId](http://www.w3schools.com/jsref/prop_attr_isid.asp) | Returns true if the attribute is of type Id, otherwise it returns false |
| [*attr*.name](http://www.w3schools.com/jsref/prop_attr_name.asp) | Returns the name of an attribute |
| [*attr*.value](http://www.w3schools.com/jsref/prop_attr_value.asp) | Sets or returns the value of the attribute |
| [*attr*.specified](http://www.w3schools.com/jsref/prop_attr_specified.asp) | Returns true if the attribute has been specified, otherwise it returns false |
|  |  |
| [*nodemap*.getNamedItem()](http://www.w3schools.com/jsref/met_namednodemap_getnameditem.asp) | Returns a specified attribute node from a NamedNodeMap |
| [*nodemap*.item()](http://www.w3schools.com/jsref/met_namednodemap_item.asp) | Returns the attribute node at a specified index in a NamedNodeMap |
| [*nodemap*.length](http://www.w3schools.com/jsref/prop_namednodemap_length.asp) | Returns the number of attribute nodes in a NamedNodeMap |
| [*nodemap*.removeNamedItem()](http://www.w3schools.com/jsref/met_namednodemap_removenameditem.asp) | Removes a specified attribute node |
| [*nodemap*.setNamedItem()](http://www.w3schools.com/jsref/met_namednodemap_setnameditem.asp) | Sets the specified attribute node (by name) |

**NoteDOM 4 Warning !!!**

In the W3C DOM Core, the Attr (attribute) object inherits all properties and methods from the Node object.

In DOM 4, the Attr object no longer inherits from Node.

**For future code quality, you should avoid using node object properties and methods on attribute objects:**

|  |  |
| --- | --- |
| **Property / Method** | **Reason for avoiding** |
| *attr*.appendChild() | Attributes don't have child nodes |
| *attr*.attributes | Attributes don't have attributes |
| *attr*.baseURI | use document.baseURI instead |
| *attr*.childNodes | Attributes don't have child nodes |
| *attr*.cloneNode() | Get or set the *attr*.value instead |
| *attr*.firstChild | Attributes don't have child nodes |
| *attr*.hasAttributes() | Attributes don't have attributes |
| *attr*.hasChildNodes | Attributes don't have child nodes |
| *attr*.insertBefore() | Attributes don't have child nodes |
| *attr*.isEqualNode() | Makes no sense |
| *attr*.isSameNode() | Makes no sense |
| *attr*.isSupported() | Is always true |
| *attr*.lastChild | Attributes don't have child nodes |
| *attr*.nextSibling | Attributes don't have siblings |
| *attr*.nodeName | Use *attr*.name instead |
| *attr*.nodeType | This is always 2 (ATTRIBUTE\_NODE) |
| *attr*.nodeValue | Use *attr*.value instead |
| *attr*.normalize() | Attributes cannot be normalized |
| *attr*.ownerDocument | This is always your HTML document |
| *attr*.ownerElement | This is the HTML element you used to access the attribute |
| *attr*.parentNode | This is the HTML element you used to access the attribute |
| *attr*.previousSibling | Attributes don't have siblings |
| *attr*.removeChild | Attributes don't have child nodes |
| *attr*.replaceChild | Attributes don't have child nodes |
| *attr*.textContent | Use *attr*.value instead |

**HTML DOM Events**

[« Previous](http://www.w3schools.com/jsref/dom_obj_attributes.asp)

[Next Reference »](http://www.w3schools.com/jsref/dom_obj_anchor.asp)

**HTML DOM Events**

HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document.

Events are normally used in combination with functions, and the function will not be executed before the event occurs (such as when a user clicks a button).

**Tip:** The event model was standardized by the W3C in DOM Level 2.

**HTML DOM Events**

**DOM:** Indicates in which DOM Level the property was introduced.

**Mouse Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [onclick](http://www.w3schools.com/jsref/event_onclick.asp) | The event occurs when the user clicks on an element | 2 |
| [oncontextmenu](http://www.w3schools.com/jsref/event_oncontextmenu.asp) | The event occurs when the user right-clicks on an element to open a context menu | 3 |
| [ondblclick](http://www.w3schools.com/jsref/event_ondblclick.asp) | The event occurs when the user double-clicks on an element | 2 |
| [onmousedown](http://www.w3schools.com/jsref/event_onmousedown.asp) | The event occurs when the user presses a mouse button over an element | 2 |
| [onmouseenter](http://www.w3schools.com/jsref/event_onmouseenter.asp) | The event occurs when the pointer is moved onto an element | 2 |
| [onmouseleave](http://www.w3schools.com/jsref/event_onmouseleave.asp) | The event occurs when the pointer is moved out of an element | 2 |
| [onmousemove](http://www.w3schools.com/jsref/event_onmousemove.asp) | The event occurs when the pointer is moving while it is over an element | 2 |
| [onmouseover](http://www.w3schools.com/jsref/event_onmouseover.asp) | The event occurs when the pointer is moved onto an element, or onto one of its children | 2 |
| [onmouseout](http://www.w3schools.com/jsref/event_onmouseout.asp) | The event occurs when a user moves the mouse pointer out of an element, or out of one of its children | 2 |
| [onmouseup](http://www.w3schools.com/jsref/event_onmouseup.asp) | The event occurs when a user releases a mouse button over an element | 2 |

**Keyboard Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [onkeydown](http://www.w3schools.com/jsref/event_onkeydown.asp) | The event occurs when the user is pressing a key | 2 |
| [onkeypress](http://www.w3schools.com/jsref/event_onkeypress.asp) | The event occurs when the user presses a key | 2 |
| [onkeyup](http://www.w3schools.com/jsref/event_onkeyup.asp) | The event occurs when the user releases a key | 2 |

**Frame/Object Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [onabort](http://www.w3schools.com/jsref/event_onabort.asp) | The event occurs when the loading of a resource has been aborted | 2 |
| [onbeforeunload](http://www.w3schools.com/jsref/event_onbeforeunload.asp) | The event occurs before the document is about to be unloaded | 2 |
| [onerror](http://www.w3schools.com/jsref/event_onerror.asp) | The event occurs when an error occurs while loading an external file | 2 |
| [onhashchange](http://www.w3schools.com/jsref/event_onhashchange.asp) | The event occurs when there has been changes to the anchor part of a URL | 3 |
| [onload](http://www.w3schools.com/jsref/event_onload.asp) | The event occurs when an object has loaded | 2 |
| [onpageshow](http://www.w3schools.com/jsref/event_onpageshow.asp) | The event occurs when the user navigates to a webpage | 3 |
| [onpagehide](http://www.w3schools.com/jsref/event_onpagehide.asp) | The event occurs when the user navigates away from a webpage | 3 |
| [onresize](http://www.w3schools.com/jsref/event_onresize.asp) | The event occurs when the document view is resized | 2 |
| [onscroll](http://www.w3schools.com/jsref/event_onscroll.asp) | The event occurs when an element's scrollbar is being scrolled | 2 |
| [onunload](http://www.w3schools.com/jsref/event_onunload.asp) | The event occurs once a page has unloaded (for <body>) | 2 |

**Form Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [onblur](http://www.w3schools.com/jsref/event_onblur.asp) | The event occurs when an element loses focus | 2 |
| [onchange](http://www.w3schools.com/jsref/event_onchange.asp) | The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <keygen>, <select>, and <textarea>) | 2 |
| [onfocus](http://www.w3schools.com/jsref/event_onfocus.asp) | The event occurs when an element gets focus | 2 |
| [onfocusin](http://www.w3schools.com/jsref/event_onfocusin.asp) | The event occurs when an element is about to get focus | 2 |
| [onfocusout](http://www.w3schools.com/jsref/event_onfocusout.asp) | The event occurs when an element is about to lose focus | 2 |
| [oninput](http://www.w3schools.com/jsref/event_oninput.asp) | The event occurs when an element gets user input | 3 |
| [oninvalid](http://www.w3schools.com/jsref/event_oninvalid.asp) | The event occurs when an element is invalid | 3 |
| [onreset](http://www.w3schools.com/jsref/event_onreset.asp) | The event occurs when a form is reset | 2 |
| [onsearch](http://www.w3schools.com/jsref/event_onsearch.asp) | The event occurs when the user writes something in a search field (for <input="search">) | 3 |
| [onselect](http://www.w3schools.com/jsref/event_onselect.asp) | The event occurs after the user selects some text (for <input> and <textarea>) | 2 |
| [onsubmit](http://www.w3schools.com/jsref/event_onsubmit.asp) | The event occurs when a form is submitted | 2 |

**Drag Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [ondrag](http://www.w3schools.com/jsref/event_ondrag.asp) | The event occurs when an element is being dragged | 3 |
| [ondragend](http://www.w3schools.com/jsref/event_ondragend.asp) | The event occurs when the user has finished dragging an element | 3 |
| [ondragenter](http://www.w3schools.com/jsref/event_ondragenter.asp) | The event occurs when the dragged element enters the drop target | 3 |
| [ondragleave](http://www.w3schools.com/jsref/event_ondragleave.asp) | The event occurs when the dragged element leaves the drop target | 3 |
| [ondragover](http://www.w3schools.com/jsref/event_ondragover.asp) | The event occurs when the dragged element is over the drop target | 3 |
| [ondragstart](http://www.w3schools.com/jsref/event_ondragstart.asp) | The event occurs when the user starts to drag an element | 3 |
| [ondrop](http://www.w3schools.com/jsref/event_ondrop.asp) | The event occurs when the dragged element is dropped on the drop target | 3 |

**Clipboard Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [oncopy](http://www.w3schools.com/jsref/event_oncopy.asp) | The event occurs when the user copies the content of an element |  |
| [oncut](http://www.w3schools.com/jsref/event_oncut.asp) | The event occurs when the user cuts the content of an element |  |
| [onpaste](http://www.w3schools.com/jsref/event_onpaste.asp) | The event occurs when the user pastes some content in an element |  |

**Print Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [onafterprint](http://www.w3schools.com/jsref/event_onafterprint.asp) | The event occurs when a page has started printing, or if the print dialogue box has been closed | 3 |
| [onbeforeprint](http://www.w3schools.com/jsref/event_onbeforeprint.asp) | The event occurs when a page is about to be printed | 3 |

**Media Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [onabort](http://www.w3schools.com/jsref/event_onabort_media.asp) | The event occurs when the loading of a media is aborted | 3 |
| [oncanplay](http://www.w3schools.com/jsref/event_oncanplay.asp) | The event occurs when the browser can start playing the media (when it has buffered enough to begin) | 3 |
| [oncanplaythrough](http://www.w3schools.com/jsref/event_oncanplaythrough.asp) | The event occurs when the browser can play through the media without stopping for buffering | 3 |
| [ondurationchange](http://www.w3schools.com/jsref/event_ondurationchange.asp) | The event occurs when the duration of the media is changed | 3 |
| onemptied | The event occurs when something bad happens and the media file is suddenly unavailable (like unexpectedly disconnects) | 3 |
| [onended](http://www.w3schools.com/jsref/event_onended.asp) | The event occurs when the media has reach the end (useful for messages like "thanks for listening") | 3 |
| [onerror](http://www.w3schools.com/jsref/event_onerror_media.asp) | The event occurs when an error occurred during the loading of a media file | 3 |
| [onloadeddata](http://www.w3schools.com/jsref/event_onloadeddata.asp) | The event occurs when media data is loaded | 3 |
| [onloadedmetadata](http://www.w3schools.com/jsref/event_onloadedmetadata.asp) | The event occurs when meta data (like dimensions and duration) are loaded | 3 |
| [onloadstart](http://www.w3schools.com/jsref/event_onloadstart.asp) | The event occurs when the browser starts looking for the specified media | 3 |
| [onpause](http://www.w3schools.com/jsref/event_onpause.asp) | The event occurs when the media is paused either by the user or programmatically | 3 |
| [onplay](http://www.w3schools.com/jsref/event_onplay.asp) | The event occurs when the media has been started or is no longer paused | 3 |
| [onplaying](http://www.w3schools.com/jsref/event_onplaying.asp) | The event occurs when the media is playing after having been paused or stopped for buffering | 3 |
| [onprogress](http://www.w3schools.com/jsref/event_onprogress.asp) | The event occurs when the browser is in the process of getting the media data (downloading the media) | 3 |
| [onratechange](http://www.w3schools.com/jsref/event_onratechange.asp) | The event occurs when the playing speed of the media is changed | 3 |
| [onseeked](http://www.w3schools.com/jsref/event_onseeked.asp) | The event occurs when the user is finished moving/skipping to a new position in the media | 3 |
| [onseeking](http://www.w3schools.com/jsref/event_onseeking.asp) | The event occurs when the user starts moving/skipping to a new position in the media | 3 |
| [onstalled](http://www.w3schools.com/jsref/event_onstalled.asp) | The event occurs when the browser is trying to get media data, but data is not available | 3 |
| [onsuspend](http://www.w3schools.com/jsref/event_onsuspend.asp) | The event occurs when the browser is intentionally not getting media data | 3 |
| [ontimeupdate](http://www.w3schools.com/jsref/event_ontimeupdate.asp) | The event occurs when the playing position has changed (like when the user fast forwards to a different point in the media) | 3 |
| [onvolumechange](http://www.w3schools.com/jsref/event_onvolumechange.asp) | The event occurs when the volume of the media has changed (includes setting the volume to "mute") | 3 |
| [onwaiting](http://www.w3schools.com/jsref/event_onwaiting.asp) | The event occurs when the media has paused but is expected to resume (like when the media pauses to buffer more data) | 3 |

**Animation Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [animationend](http://www.w3schools.com/jsref/event_animationend.asp) | The event occurs when a CSS animation has completed | 3 |
| [animationiteration](http://www.w3schools.com/jsref/event_animationiteration.asp) | The event occurs when a CSS animation is repeated | 3 |
| [animationstart](http://www.w3schools.com/jsref/event_animationstart.asp) | The event occurs when a CSS animation has started | 3 |

**Transition Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [transitionend](http://www.w3schools.com/jsref/event_transitionend.asp) | The event occurs when a CSS transition has completed | 3 |

**Server-Sent Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| [onerror](http://www.w3schools.com/jsref/event_onerror_sse.asp) | The event occurs when an error occurs with the event source |  |
| [onmessage](http://www.w3schools.com/jsref/event_onmessage_sse.asp) | The event occurs when a message is received through the event source |  |
| [onopen](http://www.w3schools.com/jsref/event_onopen_sse.asp) | The event occurs when a connection with the event source is opened |  |

**Misc Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| onmessage | The event occurs when a message is received through or from an object (WebSocket, Web Worker, Event Source or a child frame or a parent window) | 3 |
| onmousewheel | Deprecated. Use the [onwheel](http://www.w3schools.com/jsref/event_onwheel.asp) event instead |  |
| [ononline](http://www.w3schools.com/jsref/event_ononline.asp) | The event occurs when the browser starts to work online | 3 |
| [onoffline](http://www.w3schools.com/jsref/event_onoffline.asp) | The event occurs when the browser starts to work offline | 3 |
| onpopstate | The event occurs when the window's history changes | 3 |
| [onshow](http://www.w3schools.com/jsref/event_onshow.asp) | The event occurs when a <menu> element is shown as a context menu | 3 |
| onstorage | The event occurs when a Web Storage area is updated | 3 |
| [ontoggle](http://www.w3schools.com/jsref/event_ontoggle.asp) | The event occurs when the user opens or closes the <details> element | 3 |
| [onwheel](http://www.w3schools.com/jsref/event_onwheel.asp) | The event occurs when the mouse wheel rolls up or down over an element | 3 |

**Touch Events**

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **DOM** |
| ontouchcancel | The event occurs when the touch is interrupted |  |
| ontouchend | The event occurs when a finger is removed from a touch screen |  |
| ontouchmove | The event occurs when a finger is dragged across the screen |  |
| ontouchstart | The event occurs when a finger is placed on a touch screen |  |

**Event Object**

**Constants**

|  |  |  |
| --- | --- | --- |
| **Constant** | **Description** | **DOM** |
| CAPTURING\_PHASE | The current event phase is the capture phase (1) | 1 |
| AT\_TARGET | The current event is in the target phase, i.e. it is being evaluated at the event target (2) | 2 |
| BUBBLING\_PHASE | The current event phase is the bubbling phase (3) | 3 |

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| [bubbles](http://www.w3schools.com/jsref/event_bubbles.asp) | Returns whether or not a specific event is a bubbling event | 2 |
| [cancelable](http://www.w3schools.com/jsref/event_cancelable.asp) | Returns whether or not an event can have its default action prevented | 2 |
| [currentTarget](http://www.w3schools.com/jsref/event_currenttarget.asp) | Returns the element whose event listeners triggered the event | 2 |
| [defaultPrevented](http://www.w3schools.com/jsref/event_defaultprevented.asp) | Returns whether or not the preventDefault() method was called for the event | 3 |
| [eventPhase](http://www.w3schools.com/jsref/event_eventphase.asp) | Returns which phase of the event flow is currently being evaluated | 2 |
| [isTrusted](http://www.w3schools.com/jsref/event_istrusted.asp) | Returns whether or not an event is trusted | 3 |
| [target](http://www.w3schools.com/jsref/event_target.asp) | Returns the element that triggered the event | 2 |
| [timeStamp](http://www.w3schools.com/jsref/event_timestamp.asp) | Returns the time (in milliseconds relative to the epoch) at which the event was created | 2 |
| [type](http://www.w3schools.com/jsref/event_type.asp) | Returns the name of the event | 2 |
| [view](http://www.w3schools.com/jsref/event_view.asp) | Returns a reference to the Window object where the event occured | 2 |

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **DOM** |
| [preventDefault()](http://www.w3schools.com/jsref/event_preventdefault.asp) | Cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur | 2 |
| [stopImmediatePropagation()](http://www.w3schools.com/jsref/event_stopimmediatepropagation.asp) | Prevents other listeners of the same event from being called | 3 |
| stopPropagation() | Prevents further propagation of an event during event flow | 2 |

**MouseEvent Object**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| [altKey](http://www.w3schools.com/jsref/event_altkey.asp) | Returns whether the "ALT" key was pressed when the mouse event was triggered | 2 |
| [button](http://www.w3schools.com/jsref/event_button.asp) | Returns which mouse button was pressed when the mouse event was triggered | 2 |
| [buttons](http://www.w3schools.com/jsref/event_buttons.asp) | Returns which mouse buttons were pressed when the mouse event was triggered | 3 |
| [clientX](http://www.w3schools.com/jsref/event_clientx.asp) | Returns the horizontal coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered | 2 |
| [clientY](http://www.w3schools.com/jsref/event_clienty.asp) | Returns the vertical coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered | 2 |
| [ctrlKey](http://www.w3schools.com/jsref/event_ctrlkey.asp) | Returns whether the "CTRL" key was pressed when the mouse event was triggered | 2 |
| [detail](http://www.w3schools.com/jsref/event_detail.asp) | Returns a number that indicates how many times the mouse was clicked | 2 |
| [metaKey](http://www.w3schools.com/jsref/event_metakey.asp) | Returns whether the "META" key was pressed when an event was triggered | 2 |
| [relatedTarget](http://www.w3schools.com/jsref/event_relatedtarget.asp) | Returns the element related to the element that triggered the mouse event | 2 |
| [screenX](http://www.w3schools.com/jsref/event_screenx.asp) | Returns the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered | 2 |
| [screenY](http://www.w3schools.com/jsref/event_screeny.asp) | Returns the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered | 2 |
| [shiftKey](http://www.w3schools.com/jsref/event_shiftkey.asp) | Returns whether the "SHIFT" key was pressed when an event was triggered | 2 |
| [which](http://www.w3schools.com/jsref/event_which.asp) | Returns which mouse button was pressed when the mouse event was triggered | 2 |

**KeyboardEvent Object**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| [altKey](http://www.w3schools.com/jsref/event_key_altkey.asp) | Returns whether the "ALT" key was pressed when the key event was triggered | 2 |
| [ctrlKey](http://www.w3schools.com/jsref/event_key_ctrlkey.asp) | Returns whether the "CTRL" key was pressed when the key event was triggered | 2 |
| [charCode](http://www.w3schools.com/jsref/event_key_charcode.asp) | Returns the Unicode character code of the key that triggered the onkeypress event | 2 |
| [key](http://www.w3schools.com/jsref/event_key_key.asp) | Returns the key value of the key represented by the event | 3 |
| [keyCode](http://www.w3schools.com/jsref/event_key_keycode.asp) | Returns the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event | 2 |
| [location](http://www.w3schools.com/jsref/event_key_location.asp) | Returns the location of a key on the keyboard or device | 3 |
| [metaKey](http://www.w3schools.com/jsref/event_key_metakey.asp) | Returns whether the "meta" key was pressed when the key event was triggered | 2 |
| [shiftKey](http://www.w3schools.com/jsref/event_key_shiftkey.asp) | Returns whether the "SHIFT" key was pressed when the key event was triggered | 2 |
| [which](http://www.w3schools.com/jsref/event_key_which.asp) | Returns the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event | 2 |

**HashChangeEvent Object**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| [newURL](http://www.w3schools.com/jsref/event_hashchange_newurl.asp) | Returns the URL of the document, after the hash has been changed |  |
| [oldURL](http://www.w3schools.com/jsref/event_hashchange_oldurl.asp) | Returns the URL of the document, before the hash was changed |  |

**PageTransitionEvent Object**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| [persisted](http://www.w3schools.com/jsref/event_pagetransition_persisted.asp) | Returns whether the webpage was cached by the browser |  |

**FocusEvent Object**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| [relatedTarget](http://www.w3schools.com/jsref/event_focus_relatedtarget.asp) | Returns the element related to the element that triggered the event | 3 |

**AnimationEvent Object**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| [animationName](http://www.w3schools.com/jsref/event_animation_animationName.asp) | Returns the name of the animation |  |
| [elapsedTime](http://www.w3schools.com/jsref/event_animation_elapsedtime.asp) | Returns the number of seconds an animation has been running |  |

**TransitionEvent Object**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| [propertyName](http://www.w3schools.com/jsref/event_transition_propertyName.asp) | Returns the name of the CSS property associated with the transition |  |
| [elapsedTime](http://www.w3schools.com/jsref/event_transition_elapsedtime.asp) | Returns the number of seconds a transition has been running |  |

**WheelEvent Object**

|  |  |  |
| --- | --- | --- |
| **Property** | **Description** | **DOM** |
| deltaX | Returns the horizontal scroll amount of a mouse wheel (x-axis) | 3 |
| deltaY | Returns the vertical scroll amount of a mouse wheel (y-axis) | 3 |
| deltaZ | Returns the scroll amount of a mouse wheel for the z-axis | 3 |
| deltaMode | Returns a number that represents the unit of measurements for delta values (pixels, lines or pages) |  |