JavaScript Basics

JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

## What is JavaScript?

* JavaScript was designed to add interactivity to HTML pages
* JavaScript is a scripting language
* A scripting language is a lightweight programming language
* JavaScript is usually embedded directly into HTML pages
* JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
* Everyone can use JavaScript without purchasing a license

## Are Java and JavaScript the same?

NO!

Java and JavaScript are two completely different languages in both concept and design!

Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

## What Can JavaScript do?

* **JavaScript gives HTML designers a programming tool -**HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
* **JavaScript can react to events -**A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
* **JavaScript can read and write HTML elements -**A JavaScript can read and change the content of an HTML element
* **JavaScript can be used to validate data -**A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
* **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
* **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

**JavaScript has a number of big benefits to anyone who wants to make their Web site dy­namic:**

* it is widely supported in Web browsers;
* it gives easy access to the document objects and can manipulate most of them;
* JavaScript can give interesting animations without the long download times associ­ated with many multimedia data types;
* Web surfers don't need a special plug-in to use your scripts;
* JavaScript is relatively secure - JavaScript can neither read from your local hard drive nor write to it, and you can't get a virus infection directly from JavaScript.

JavaScript is Case Sensitive

Unlike HTML, JavaScript is case sensitive - therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions.

JavaScript Statements

A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do.

This JavaScript statement tells the browser to write "Hello Dolly" to the web page:

document.write("Hello Dolly");

It is normal to add a semicolon at the end of each executable statement. Most people think this is a good programming practice, and most often you will see this in JavaScript examples on the web.

The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. Because of this you will often see examples without the semicolon at the end.

**Note:** Using semicolons makes it possible to write multiple statements on one line.

JavaScript Code

JavaScript code (or just JavaScript) is a sequence of JavaScript statements.

Each statement is executed by the browser in the sequence they are written.

This example will write a heading and two paragraphs to a web page:

Example

<script type="text/javascript">  
document.write("<h1>This is a heading</h1>");  
document.write("<p>This is a paragraph.</p>");  
document.write("<p>This is another paragraph.</p>");  
</script>

JavaScript Blocks

JavaScript statements can be grouped together in blocks.

Blocks start with a left curly bracket {, and end with a right curly bracket }.

The purpose of a block is to make the sequence of statements execute together.

This example will write a heading and two paragraphs to a web page:

Example

<script type="text/javascript">  
{  
document.write("<h1>This is a heading</h1>");  
document.write("<p>This is a paragraph.</p>");  
document.write("<p>This is another paragraph.</p>");  
}  
</script>

The example above is not very useful. It just demonstrates the use of a block. Normally a block is used to group statements together in a function or in a condition (where a group of statements should be executed if a condition is met).

You will learn more about functions and conditions in later chapters.

JavaScript Comments

JavaScript comments can be used to make the code more readable.

Comments can be added to explain the JavaScript, or to make the code more readable.

Single line comments start with //.

The following example uses single line comments to explain the code:

Example

<script type="text/javascript">  
// Write a heading  
document.write("<h1>This is a heading</h1>");  
// Write two paragraphs:  
document.write("<p>This is a paragraph.</p>");  
document.write("<p>This is another paragraph.</p>");  
</script>

JavaScript Multi-Line Comments

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

The following example uses a multi line comment to explain the code:

Example

<script type="text/javascript">  
/\*  
The code below will write  
one heading and two paragraphs  
\*/  
document.write("<h1>This is a heading</h1>");  
document.write("<p>This is a paragraph.</p>");  
document.write("<p>This is another paragraph.</p>");  
</script>

Using Comments to Prevent Execution

In the following example the comment is used to prevent the execution of a single code line (can be suitable for debugging):

Example

<script type="text/javascript">  
//document.write("<h1>This is a heading</h1>");  
document.write("<p>This is a paragraph.</p>");  
document.write("<p>This is another paragraph.</p>");  
</script>

In the following example the comment is used to prevent the execution of a code block (can be suitable for debugging):

Example

<script type="text/javascript">  
/\*  
document.write("<h1>This is a heading</h1>");  
document.write("<p>This is a paragraph.</p>");  
document.write("<p>This is another paragraph.</p>");  
\*/  
</script>

Using Comments at the End of a Line

In the following example the comment is placed at the end of a code line:

Example

## <script type="text/javascript"> document.write("Hello"); // Write "Hello" document.write(" Dolly!"); // Write " Dolly!"

# Where and how to write Javascript code.

First of all, there are two ways to write your Javascript code. The first is by embedding the code in your HTML document. We do this by using the opening and closing script tags, defining the script type & typing our code.

The second method is to write our code in an external file. This benefit of this method is that we can run the same code on multiple pages if necessary.

How to reference an external Javascript file:

**<script type="text/javascript" src="javascript.js" </script>**

Please note, when writing code in an external file you do not need to use the opening and closing script tags. Also, the file must be saved with a **.js** file extension.

## Where to write your code

You can place your code in either the head section, or the body section of your HTML page. A couple of things to consider:

Upon loading a page, a web browser will automatically execute all javascript code. Sometimes, this is not what we require.

For example, lets say we want a message to appear when a user clicks a certain button.

Well in a scenario like this we would need to place our code inside a function. An example follows:

<head>

<script type="text/javascript">

function showMessage()

{

alert("You have just clicked the button!")

}

</script>

<body>

<input type="button" onclick="showMessage()" value="Click me!"/>

</body>

Scripts such as the above, that is, scripts which are to be executed when called should always be placed in the **head section.**

Other scripts, which are to be executed when the page loads should be placed in the body section.

JavaScript Variables

Variables are "containers" for storing information.

JavaScript Variables

As with algebra, JavaScript variables are used to hold values or expressions.

A variable can have a short name, like x, or a more descriptive name, like carname.

Rules for JavaScript variable names:

* Variable names are case sensitive (y and Y are two different variables)
* Variable names must begin with a letter, the $ character, or the underscore character

**Note:** Because JavaScript is case-sensitive, variable names are case-sensitive.

A variable's value can change during the execution of a script. You can refer to a variable by its name to display or change its value.

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables.

You declare JavaScript variables with the **var** keyword:

var x;  
var carname;

After the declaration shown above, the variables are empty (they have no values yet).

However, you can also assign values to the variables when you declare them:

var x=5;  
var carname="Volvo";

After the execution of the statements above, the variable **x** will hold the value **5**, and **carname** will hold the value **Volvo**.

**Note:** When you assign a text value to a variable, use quotes around the value.

**Note:** If you redeclare a JavaScript variable, it will not lose its value.

Local JavaScript Variables

A variable declared within a JavaScript function becomes **LOCAL** and can only be accessed within that function. (the variable has local scope).

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

Local variables are destroyed when you exit the function.

You will learn more about functions in a later chapter of this tutorial.

Global JavaScript Variables

Variables declared outside a function become **GLOBAL**, and all scripts and functions on the web page can access it.

Global variables are destroyed when you close the page.

If you declare a variable, without using "var", the variable always becomes **GLOBAL**.

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared as global variables.

These statements:

x=5;  
carname="Volvo";

will declare the variables x and carname as global variables (if they don't already exist).

As with algebra, you can do arithmetic operations with JavaScript variables:

y=x-5;  
z=y+5;

JavaScript Operators

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** | |
| + | Addition | x=y+2 | x=7 | y=5 |
| - | Subtraction | x=y-2 | x=3 | y=5 |
| \* | Multiplication | x=y\*2 | x=10 | y=5 |
| / | Division | x=y/2 | x=2.5 | y=5 |
| % | Modulus (division remainder) | x=y%2 | x=1 | y=5 |
| ++ | Increment | x=++y | x=6 | y=6 |
| x=y++ | x=5 | y=6 |
| -- | Decrement | x=--y | x=4 | y=4 |
| x=y-- | x=5 | y=4 |

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** |
| = | x=y |  | x=5 |
| += | x+=y | x=x+y | x=15 |
| -= | x-=y | x=x-y | x=5 |
| \*= | x\*=y | x=x\*y | x=50 |
| /= | x/=y | x=x/y | x=2 |
| %= | x%=y | x=x%y | x=0 |

The + Operator Used on Strings

The + operator can also be used to add string variables or text values together.

To add two or more string variables together, use the + operator.

txt1="What a very";  
txt2="nice day";  
txt3=txt1+txt2;

After the execution of the statements above, the variable txt3 contains "What a verynice day".

To add a space between the two strings, insert a space into one of the strings:

txt1="What a very ";  
txt2="nice day";  
txt3=txt1+txt2;

or insert a space into the expression:

txt1="What a very";  
txt2="nice day";  
txt3=txt1+" "+txt2;

After the execution of the statements above, the variable txt3 contains:

"What a very nice day"

Adding Strings and Numbers

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

Example

x=5+5;  
document.write(x);  
  
x="5"+"5";  
document.write(x);  
  
x=5+"5";  
document.write(x);  
  
x="5"+5;  
document.write(x);

## Comparison Operators

Comparison and Logical operators are used to test for true or false.

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** | **Example** |
| == | is equal to | x==8 is false x==5 is true |
| === | is exactly equal to (value and type) | x===5 is true x==="5" is false |
| != | is not equal | x!=8 is true |
| > | is greater than | x>8 is false |
| < | is less than | x<8 is true |
| >= | is greater than or equal to | x>=8 is false |
| <= | is less than or equal to | x<=8 is true |

## 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) document.write("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** |
| && | and | (x < 10 && y > 1) is true |
| || | or | (x==5 || y==5) is false |
| ! | not | !(x==y) is true |

## Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

### Syntax

variablename=(condition)?value1:value2

### Example

If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value "Dear President " else it will be assigned "Dear":

<script type="text/javascript">  
  
var visitor="PRES";  
var greeting=(visitor=="PRES")?"Dear President ":"Dear ";  
document.write(greeting);  
  
</script>

# DataTypes

* Numbers - are values that can be processed and calculated. You don't enclose them in quotation marks. The numbers can be either positive or negative.
* Strings - are a series of letters and numbers enclosed in quotation marks. JavaScript uses the string literally; it doesn't process it. You'll use strings for text you want displayed or values you want passed along.
* Boolean (**true**/**false**) - lets you evaluate whether a condition meets or does not meet specified criteria.
* Null - is an empty value. **null** is not the same as 0 **.**  zero is a real, calculable number, whereas **null** is the **absence of any value**.

**Data Types**

|  |  |
| --- | --- |
| **TYPE** | **EXAMPLE** |
| **Numbers** | Any number, such as 17, 21, or 54e7 |
| **Strings** | "Greetings!" or "Fun" |
| **Boolean** | Either true or false |
| **Null** | A special keyword for exactly that – the null value (that is, nothing) |

# JavaScript Popup Boxes (Messages and Confirmation)

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

alert("*sometext*");

## Example

<html>  
<head>  
<script type="text/javascript">  
function show\_alert()  
{  
alert("I am an alert box!");  
}  
</script>  
</head>  
<body>  
  
<input type="button" onclick="show\_alert()" value="Show alert box" />  
  
</body>  
</html>

## 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

confirm("*sometext*");

## Example

<html>  
<head>  
<script type="text/javascript">  
function show\_confirm()  
{  
var r=confirm("Press a button");  
if (r==true)  
  {  
  alert("You pressed OK!");  
  }  
else  
  {  
  alert("You pressed Cancel!");  
  }  
}  
</script>  
</head>  
<body>  
  
<input type="button" onclick="show\_confirm()" value="Show confirm box" />  
  
</body>  
</html>

## 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

prompt("*sometext*","*defaultvalue*");

## Example

<html>  
<head>  
<script type="text/javascript">  
function show\_prompt()  
{  
var name=prompt("Please enter your name","Harry Potter");  
if (name!=null && name!="")  
  {  
  document.write("<p>Hello " + name + "! How are you today?</p>");  
  }  
}  
</script>  
</head>  
<body>  
  
<input type="button" onclick="show\_prompt()" value="Show prompt box" />  
  
</body>  
</html>

JavaScript Statements

## 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:

* **if statement** - use this statement to execute some code only if a specified condition is true
* **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false
* **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
* **switch statement** - use this statement to select one of many blocks of code to be executed

## If Statement

Use the if statement to execute some code only if a specified condition is true.

### Syntax

if (*condition*)  
  {  
*code to be executed if condition is true*  
  }

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

## Example

<script type="text/javascript">  
//Write a "Good morning" greeting if  
//the time is less than 10  
  
var d=new Date();  
var time=d.getHours();  
  
if (time<10)  
  {  
  document.write("<b>Good morning</b>");  
  }  
</script>

Notice that there is no ..else.. in this syntax. You tell the browser to execute some code **only if the specified condition is true**.

## If...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

### Syntax

if (*condition*)  
  {  
*code to be executed if condition is true*  
  }  
else  
  {  
*code to be executed if condition is not true*  
  }

## Example

<script type="text/javascript">  
//If the time is less than 10, you will get a "Good morning" greeting.  
//Otherwise you will get a "Good day" greeting.  
  
var d = new Date();  
var time = d.getHours();  
  
if (time < 10)  
  {  
  document.write("Good morning!");  
  }  
else  
  {  
  document.write("Good day!");  
  }  
</script>

## If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

### Syntax

if (*condition1*)  
  {  
*code to be executed if condition1 is true*  
  }  
else if (*condition2*)  
  {  
*code to be executed if condition2 is true*  
  }  
else  
  {  
*code to be executed if neither condition1 nor condition2 is true*  
  }

## Example

<script type="text/javascript">  
var d = new Date()  
var time = d.getHours()  
if (time<10)  
  {  
  document.write("<b>Good morning</b>");  
  }  
else if (time>=10 && time<16)  
  {  
  document.write("<b>Good day</b>");  
  }  
else  
  {  
  document.write("<b>Hello World!</b>");  
  }  
</script>

## The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

### Syntax

switch(n)  
{  
case 1:  
*execute code block 1*  
  break;  
case 2:  
*execute code block 2*  
  break;  
default:  
*code to be executed if n is different from case 1 and 2*  
}

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically.

## Example

<script type="text/javascript">  
//You will receive a different greeting based  
//on what day it is. Note that Sunday=0,  
//Monday=1, Tuesday=2, etc.  
  
var d=new Date();  
var theDay=d.getDay();  
switch (theDay)  
{  
case 5:  
  document.write("Finally Friday");  
  break;  
case 6:  
  document.write("Super Saturday");  
  break;  
case 0:  
  document.write("Sleepy Sunday");  
  break;  
default:  
  document.write("I'm looking forward to this weekend!");  
}  
</script>

## JavaScript Loops

Loops execute a block of code a specified number of times, or while a specified condition is true.

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript, there are two different kind of loops:

* **for**- loops through a block of code a specified number of times
* **while**- loops through a block of code while a specified condition is true

## The for Loop

The for loop is used when you know in advance how many times the script should run.

### Syntax

for (*variable*=*startvalue*;*variable*<=*endvalue*;*variable*=*variable*+*increment*)  
{  
*code to be executed*  
}

### Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs.

**Note:** The increment parameter could also be negative, and the <= could be any comparing statement.

## Example

<html>  
<body>  
<script type="text/javascript">  
var i=0;  
for (i=0;i<=5;i++)  
{  
document.write("The number is " + i);  
document.write("<br />");  
}  
</script>  
</body>  
</html>

## The while Loop

The while loop loops through a block of code while a specified condition is true.

### Syntax

while (*variable*<=*endvalue*)  
  {  
*code to be executed*  
  }

**Note:** The <= could be any comparing operator.

### Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs:

## Example

<html>  
<body>  
<script type="text/javascript">  
var i=0;  
while (i<=5)  
  {  
  document.write("The number is " + i);  
  document.write("<br />");  
  i++;  
  }  
</script>  
</body>  
</html>

**The do...while Loop**

The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

### Syntax

do  
  {  
*code to be executed*}  
while (*variable*<=*endvalue*);

### Example

The example below uses a do...while loop. The do...while loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested:

## Example

<html>  
<body>  
<script type="text/javascript">  
var i=0;  
do  
  {  
  document.write("The number is " + i);  
  document.write("<br />");  
  i++;  
  }  
while (i<=5);  
</script>  
</body>  
</html>

## The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

## Example

<html>  
<body>  
<script type="text/javascript">  
var i=0;  
for (i=0;i<=10;i++)  
  {  
  if (i==3)  
    {  
    break;  
    }  
  document.write("The number is " + i);  
  document.write("<br />");  
  }  
</script>  
</body>  
</html>

**The continue Statement**

The continue statement will break the current loop and continue with the next value.

## Example

<html>  
<body>  
<script type="text/javascript">  
var i=0  
for (i=0;i<=10;i++)  
  {  
  if (i==3)  
    {  
    continue;  
    }  
  document.write("The number is " + i);  
  document.write("<br />");  
  }  
</script>  
</body>  
</html>

## JavaScript For...In Statement

The for...in statement loops through the properties of an object.

### Syntax

for (*variable* in *object*)  
  {  
*code to be executed*  
  }

**Note:**The code in the body of the for...in loop is executed once for each property.

### Example

Looping through the properties of an object:

## Example

var person={fname:"John",lname:"Doe",age:25};   
var x;  
  
for (x in person)  
{  
document.write(person[x] + " ");  
}

## Functions

A function will be executed by an event or by a call to the function.

To keep the browser from executing a script when the page loads, you can put your script into a function.

A function contains code that will be executed by an event or by a call to the function.

You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

## How to Define a Function

### Syntax

function *functionname*(*var1,var2,...,varX*)  
{  
*some code*  
}

The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function.

**Note:** A function with no parameters must include the parentheses () after the function name.

**Note:** Do not forget about the importance of capitals in JavaScript! The word *function* must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

## JavaScript Function Example

## Example

<html>  
<head>  
<script type="text/javascript">  
function displaymessage()  
{  
alert("Hello World!");  
}  
</script>  
</head>  
  
<body>  
<form>  
<input type="button" value="Click me!" onclick="displaymessage()" />  
</form>  
</body>  
</html>

If the line: alert("Hello world!!") in the example above had not been put within a function, it would have been executed as soon as the page was loaded. Now, the script is not executed before a user hits the input button. The function displaymessage() will be executed if the input button is clicked.

## The return Statement

The return statement is used to specify the value that is returned from the function.

So, functions that are going to return a value must use the return statement.

The example below returns the product of two numbers (a and b):

## Example

<html>  
<head>  
<script type="text/javascript">  
function product(a,b)  
{  
return a\*b;  
}  
</script>  
</head>  
  
<body>  
<script type="text/javascript">  
document.write(product(4,3));  
</script>  
  
</body>  
</html>

## The Lifetime of JavaScript Variables

If you declare a variable, using "var", within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

# JavaScript Array Object

The Array object is used to store multiple values in a single variable.

## 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 best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own ID so that it can be easily accessed.

## Create an Array

An array can be defined in three ways.

The following code creates an Array object called myCars:

1:

var myCars=new Array(); // regular array (add an optional integer  
myCars[0]="Saab";       // argument to control array's size)  
myCars[1]="Volvo";  
myCars[2]="BMW";

2:

var myCars=new Array("Saab","Volvo","BMW"); // condensed array

3:

var myCars=["Saab","Volvo","BMW"]; // literal array

**Note:** If you specify numbers or true/false values inside the array then the variable type will be Number or Boolean, instead of String.

## Access an Array

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0.

The following code line:

document.write(myCars[0]);

will result in the following output:

Saab

## Modify Values in an Array

To modify a value in an existing array, just add a new value to the array with a specified index number:

myCars[0]="Opel";

Now, the following code line:

document.write(myCars[0]);

will result in the following output:

Opel

## Array Object

The Array object is used to store multiple values in a single variable.

## Array Object 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 object |

## Array Object 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() |  |
| [join()](http://www.w3schools.com/jsref/jsref_join.asp) | Joins all elements of an array into a string |
| [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 String Manipulation

## String Object

The String object is used to manipulate a stored piece of text.

String objects are created with new String().

## Syntax

var txt = new String("*string*");

or more simply:

var txt = "*string*";

## String Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](http://www.w3schools.com/jsref/jsref_constructor_string.asp) | Returns the function that created the String object's prototype |
| [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 Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [charAt()](http://www.w3schools.com/jsref/jsref_charat.asp) | Returns the character at the specified index |
| [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 copy of the 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 |
| [match()](http://www.w3schools.com/jsref/jsref_match.asp) | Searches for a match between a regular expression and a string, and returns the matches |
| [replace()](http://www.w3schools.com/jsref/jsref_replace.asp) | Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring |
| [search()](http://www.w3schools.com/jsref/jsref_search.asp) | Searches for a match between a regular expression and a string, 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 |
| [toLowerCase()](http://www.w3schools.com/jsref/jsref_tolowercase.asp) | Converts a string to lowercase letters |
| [toUpperCase()](http://www.w3schools.com/jsref/jsref_touppercase.asp) | Converts a string to uppercase letters |
| [valueOf()](http://www.w3schools.com/jsref/jsref_valueof_string.asp) | Returns the primitive value of a String object |

# JavaScript charAt() Method

## Definition and Usage

The charAt() method returns the character at the specified index in a string.

The index of the first character is 0, and the index of the last character in a string called "txt", is txt.length-1.

## Syntax

*string*.charAt(index)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| index | Required. An integer between 0 and *string*.length-1 |

## Example

Return the first and last character of a string:

<script type="text/javascript">  
  
var str = "Hello world!";  
document.write("First character: " + str.charAt(0) + "<br />");  
document.write("Last character: " + str.charAt(str.length-1));  
  
</script>

The output of the code above will be:

First character: H  
Last character: !

# JavaScript concat() Method

## Definition and Usage

The concat() method is used to join two or more strings.

This method does not change the existing strings, it only returns a copy of the joined strings.

## Syntax

*string*.concat(string2, string3, ..., stringX)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| string2, string3, ..., stringX | Required. The strings to be joined |

## Example

Join two strings:

<script type="text/javascript">  
  
var str1="Hello ";  
var str2="world!";  
document.write(str1.concat(str2));  
  
</script>

The output of the code above will be:

Hello world!

# JavaScript indexOf() Method

## Definition and Usage

The indexOf() method returns the position of the first occurrence of a specified value in a string.

This method returns -1 if the value to search for never occurs.

## Syntax

*string*.indexOf(searchstring, start)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| searchstring | Required. The string to search for |
| start | Optional. The start position in the string to start the search. If omitted, the search starts from position 0 |

**Note:** The indexOf() method is case sensitive!

## Example

Do different searches within a string:

<script type="text/javascript">  
  
var str="Hello world!";  
document.write(str.indexOf("d") + "<br />");  
document.write(str.indexOf("WORLD") + "<br />");  
document.write(str.indexOf("world"));  
  
</script>

The output of the code above will be:

10  
-1  
6

# JavaScript lastIndexOf() Method

## Definition and Usage

The lastIndexOf() method returns the position of the last found occurrence of a specified value in a string.

**Note:** The string is searched backward, but the index returned is the character position from left to right (starting at 0).

This method returns -1 if the value to search for never occurs.

## Syntax

*string*.lastIndexOf(searchstring, start)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| searchstring | Required. The string to search for |
| start | Optional. The position where to start the search. If omitted, the default value is the length of the string |

## Example

Do different searches within a string:

<script type="text/javascript">  
  
var str="Hello world!";  
document.write(str.lastIndexOf("d") + "<br />");  
document.write(str.lastIndexOf("WORLD") + "<br />");  
document.write(str.lastIndexOf("world"));  
  
</script>

The output of the code above will be:

10  
-1  
6

# JavaScript length Property

## Definition and Usage

The length property returns the length of a string (in characters).

## Syntax

*string*.length

## Example

Return the number of characters in a string:

<script type="text/javascript">  
  
var txt = "Hello World!";  
document.write(txt.length);  
  
</script>

The output of the code above will be:

12

# JavaScript split() Method

## Definition and Usage

The split() method is used to split a string into an array of substrings, and returns the new array.

## Syntax

*string*.split(separator, limit)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| separator | Optional. Specifies the character to use for splitting the string. If omitted, the entire string will be returned |
| limit | Optional. An integer that specifies the number of splits |

**Tip:** If an empty string ("") is used as the separator, the string is split between each character.

## Example

Split a string in different ways:

<script type="text/javascript">  
  
var str="How are you doing today?";  
  
document.write(str.split() + "<br />");  
document.write(str.split(" ") + "<br />");  
document.write(str.split("") + "<br />");  
document.write(str.split(" ",3));  
  
</script>

The output of the code above will be:

How are you doing today?  
How,are,you,doing,today?  
H,o,w, ,a,r,e, ,y,o,u, ,d,o,i,n,g, ,t,o,d,a,y,?  
How,are,you

# JavaScript substr() Method

## Definition and Usage

The substr() method extracts the characters from a string, beginning at "start" and through the specified number of character, and returns the new sub string.

## Syntax

*string*.substr(start,length)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| start | Required. The index where to start the extraction. First character is at index 0 |
| length | Optional. The number of characters to extract. If omitted, it extracts the rest of the string |

**Tip:** To extract characters from the end of the string, use a negative start number

## Example

Extract characters from a string:

<script type="text/javascript">  
  
var str="Hello world!";  
document.write(str.substr(3)+"<br />");  
document.write(str.substr(3,4));  
  
</script>

The output of the code above will be:

lo world!  
lo w

# JavaScript substring() Method

## Definition and Usage

The substring() method extracts the characters from a string, between two specified indices, and returns the new sub string.

This method extracts the characters in a string between "from" and "to", not including "to" itself.

## Syntax

*string*.substring(from, to)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| from | Required. The index where to start the extraction. First character is at index 0 |
| to | Optional. The index where to stop the extraction. If omitted, it extracts the rest of the string |

## Example

Extract characters from a string:

<script type="text/javascript">  
  
var str="Hello world!";  
document.write(str.substring(3)+"<br />");  
document.write(str.substring(3,7));  
  
</script>

The output of the code above will be:

lo world!  
lo w

# JavaScript toLowerCase() Method

## Definition and Usage

The toLowerCase() method converts a string to lowercase letters.

## Syntax

*string*.toLowerCase()

## Example

Convert a string to lowercase letters:

<script type="text/javascript">  
  
var str="Hello World!";  
document.write(str.toLowerCase());  
  
</script>

# JavaScript toUpperCase() Method

## Definition and Usage

The toUpperCase() method converts a string to uppercase letters.

## Syntax

*string*.toUpperCase()

## Example

Convert a string to uppercase letters:

<script type="text/javascript">  
  
var str="Hello world!";  
document.write(str.toUpperCase());  
  
</script>

**Javascript mathematical functions**

## 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

## Math Object Properties (Numerical Constants)

|  |  |
| --- | --- |
| **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. 3.14159) |
| [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_atan.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 Objects

JavaScript is an Object Based Programming language.

An Object Based Programming language allows you to define your own objects and make your own variable types.

## Object Based Programming

JavaScript is an Object Based Programming language, and allows you to define your own objects and make your own variable types.

However, creating your own objects will be explained later, in the Advanced JavaScript section. We will start by looking at the built-in JavaScript objects, and how they are used. The next pages will explain each built-in JavaScript object in detail.

Note that an object is just a special kind of data. An object has properties and methods.

## Properties

Properties are the values associated with an object.

In the following example we are using the length property of the String object to return the number of characters in a string:

<script type="text/javascript">  
var txt="Hello World!";  
document.write(txt.length);  
</script>

The output of the code above will be:

12

## Methods

Methods are the actions that can be performed on objects.

In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

<script type="text/javascript">  
var str="Hello world!";  
document.write(str.toUpperCase());  
</script>

The output of the code above will be:

HELLO WORLD!

#### Creating objects using new Object()

New

There are several ways to create objects in JavaScript, and all of them have their place. The simplest way is to use the new operator, specifically, new Object():

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

<!--

person = new Object()

person.name = "Tim Scarfe"

person.height = "6Ft"

person.run = function() {

this.state = "running"

this.speed = "4ms^-1"

}

//-->

</script>

We define a custom object "person," then add to it its own properties and method afterwards. In this case, the custom method merely initializes two more properties.

This

To differentiate between global variables and those which are part of an object but *may* have the same name, JavaScript uses this. Whenever you refer to a variable which is part of an object you *must* precede the variable name by this. Separate the variable name from this with a dot.

DOT

When referring to a property of an object, whether a method or a variable, a dot is placed between the object name and the property.

# JavaScript Regular expressions

## 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

var patt=new RegExp(pattern,modifiers);  
  
or more simply:  
  
var patt=/pattern/modifiers;

* pattern specifies the pattern of an expression
* modifiers specify if a search should be global, case-sensitive, etc.

## Modifiers or flags

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 | 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] | Find any digit from 0 to 9 |
| [A-Z] | Find any character from uppercase A to uppercase Z |
| [a-z] | Find any character from lowercase a to lowercase z |
| [A-z] | Find any character from uppercase A to lowercase z |
| [adgk] | Find any character in the given set |
| [^adgk] | Find any character outside the given set |
| (red|blue|green) | Find any of the alternatives specified |

## Metacharacters

Metacharacters are characters with a special meaning:

|  |  |
| --- | --- |
| **Metacharacter(tokens)** | **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 | Find a NUL character |
| [\n](http://www.w3schools.com/jsref/jsref_regexp_newline.asp) | Find a new line character |
| \f | Find a form feed character |
| \r | Find a carriage return character |
| \t | Find a tab character |
| \v | 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 (tokens)** | **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 |

Regular expressions are manipulated using functions which belong to either the **RegExp** or **S t r i ng** classes.

String *Functions*

**match (pattern**)

Searches for a matching pattern. Returns an array holding the results, or null if no match is found.

**Replace(patter1,pattern2)**

Searches for pattern1 If the search is successful pattern1 is replaced with pattern2.

**search(pattern)**

Searches for a pattern in the string. If the match is successful, the index, offset, of the start of the match is returned. If the search fails, the function returns -1.

**split(pattern)**

Splits the string into parts based upon the pattern, or regular expression, which is sup­plied as a parameter.

## RegExp Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [compile()](http://www.w3schools.com/jsref/jsref_regexp_compile.asp) | 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 |

Built in objects in javascript

The window object represents an open window in a browser.

If a document contain frames (<frame> or <iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.

## 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 |
| [blur()](http://www.w3schools.com/jsref/met_win_blur.asp) | Removes focus from the current window |
| [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  window.close() |
| [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 |
| [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  window.open(*URL,name,specs,replace*) |
| [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() |  |
| [scrollBy()](http://www.w3schools.com/jsref/met_win_scrollby.asp) | Scrolls the content by the specified number of pixels  scrollBy(*xnum,ynum*)   |  |  | | --- | --- | | **Parameter** | **Description** | | xnum | How many pixels to scroll by, along the x-axis (horizontal) | | ynum | How many pixels to scroll by, along the y-axis (vertical) | |
| [scrollTo()](http://www.w3schools.com/jsref/met_win_scrollto.asp) | Scrolls the content to the specified coordinates  scrollTo(*xpos,ypos*)   |  |  | | --- | --- | | **Parameter** | **Description** | | xpos | The coordinate to scroll to, along the x-axis | | ypos | The coordinate to scroll to, along the y-axis | |
| [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 |

## Document Object

Each HTML document loaded into a browser window becomes a Document object.

The Document object provides access to all HTML elements in a page, from within a script.

**Tip:** The Document object is also part of the Window object, and can be accessed through the window.document property.

## Document Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [anchors](http://www.w3schools.com/jsref/coll_doc_anchors.asp) | Returns a collection of all the anchors in the document  document.anchors[].*property* |
| applets | Returns a collection of all the applets in the document |
| body | Returns the body element of the document |
| [cookie](http://www.w3schools.com/jsref/prop_doc_cookie.asp) | Returns all name/value pairs of cookies in the document |
| [documentMode](http://www.w3schools.com/jsref/prop_doc_documentmode.asp) | Returns the mode used by the browser to render the document |
| [domain](http://www.w3schools.com/jsref/prop_doc_domain.asp) | Returns the domain name of the server that loaded the document |
| [forms](http://www.w3schools.com/jsref/coll_doc_forms.asp) | Returns a collection of all the forms in the document |
| [images](http://www.w3schools.com/jsref/coll_doc_images.asp) | Returns a collection of all the images in the document |
| [lastModified](http://www.w3schools.com/jsref/prop_doc_lastmodified.asp) | Returns the date and time the document was last modified |
| [links](http://www.w3schools.com/jsref/coll_doc_links.asp) | Returns a collection of all the links in the document |
| [readyState](http://www.w3schools.com/jsref/prop_doc_readystate.asp) | Returns the (loading) status of the document |
| [referrer](http://www.w3schools.com/jsref/prop_doc_referrer.asp) | Returns the URL of the document that loaded the current document |
| [title](http://www.w3schools.com/jsref/prop_doc_title.asp) | Sets or returns the title of the document |
| [URL](http://www.w3schools.com/jsref/prop_doc_url.asp) | Returns the full URL of the document |

## Document Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [close()](http://www.w3schools.com/jsref/met_doc_close.asp) | Closes the output stream previously opened with document.open() |
| [getElementsByName()](http://www.w3schools.com/jsref/met_doc_getelementsbyname.asp) | Accesses all elements with a specified name |
| [open()](http://www.w3schools.com/jsref/met_doc_open.asp) | Opens an output stream to collect the output from document.write() or document.writeln() |
| [write()](http://www.w3schools.com/jsref/met_doc_write.asp) | Writes HTML expressions or JavaScript code to a document |
| [writeln()](http://www.w3schools.com/jsref/met_doc_writeln.asp) | Same as write(), but adds a newline character after each statement  document.writeln(*exp1,exp2,exp3,...*) |

## Form Object

The Form object represents an HTML form.

For each <form> tag in an HTML document, a Form object is created.

Forms are used to collect user input, and contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more. A form can also contain select menus, textarea, fieldset, legend, and label elements.

|  |  |
| --- | --- |
| **Collection** | **Description** |
| [elements[]](http://www.w3schools.com/jsref/coll_form_elements.asp) | Returns an array of all elements in a form |

## Form Object Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [acceptCharset](http://www.w3schools.com/jsref/prop_form_acceptcharset.asp) | Sets or returns the value of the accept-charset attribute in a form |
| [action](http://www.w3schools.com/jsref/prop_form_action.asp) | Sets or returns the value of the action attribute in a form |
| [enctype](http://www.w3schools.com/jsref/prop_form_enctype.asp) | Sets or returns the value of the enctype attribute in a form |
| [length](http://www.w3schools.com/jsref/prop_form_length.asp) | Returns the number of elements in a form |
| [method](http://www.w3schools.com/jsref/prop_form_method.asp) | Sets or returns the value of the method attribute in a form |
| [name](http://www.w3schools.com/jsref/prop_form_name.asp) | Sets or returns the value of the name attribute in a form |
| [target](http://www.w3schools.com/jsref/prop_form_target.asp) | Sets or returns the value of the target attribute in a form |

## Form Object Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [reset()](http://www.w3schools.com/jsref/met_form_reset.asp) | Resets a form |
| [submit()](http://www.w3schools.com/jsref/met_form_submit.asp) | Submits a form |

## Form Object Events

|  |  |
| --- | --- |
| **Event** | **The event occurs when...** |
| [onreset](http://www.w3schools.com/jsref/event_form_onreset.asp) | The reset button is clicked |
| [onsubmit](http://www.w3schools.com/jsref/event_form_onsubmit.asp) | The submit button is clicked |

**The browser object**

No two browser model will process your JavaScript in the same way. It's important that you try to find out which browser is being used to view your page. You can then make a choice for your visitors:

* exclude browsers that are unable to use your code;
* redirect them to a non-scripted version of your site;
* present scripts that are tailored to suit each browser. You'll be glad to know that this can be done from within your code and doesn't involve rewriting the entire site.

The browser is a JavaScript object and can be queried from within your code. For his­torical reasons the browser object is actually called the navigator object. The following properties are just some that can be gathered:

Navigator.appCodeName

The internal name for the browser. For both major products this is *Mozilla,* which was the name of the original Netscape code source.

Navigator.appName

This is the public name of the browser - navigator or Internet explorer for the big two.

navigator.appVersion

The version number, platform on which the browser is running, and (for Internet Ex­plorer) the version of Navigator with which it is compatible.

navigator.userAgent

The strings appCodeName and appVersion concatenated together.

navigator.plugins

An array containing details of all installed plug-ins.

navigator.mimeTypes

An array of all supported MIME types - useful if you need to make sure that the browser can handle your data.

The Navigator object contains information about the visitor's browser name, version, and more. JavaScript Browser Sniffer is a browser identifier written in JavaScript. It will tell which browser, version and operating system you (the visitor) are using.

# Date Object

The Date object is used to work with dates and times.

## Create a Date Object

The Date object is used to work with dates and times.

Date objects are created with the Date() constructor.

There are four ways of instantiating a date:

new Date() // current date and time  
new Date(milliseconds) //milliseconds since 1970/01/01  
new Date(dateString)  
new Date(year, month, day, hours, minutes, seconds, milliseconds)

Most parameters above are optional. Not specifying, causes 0 to be passed in.

Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object, using either local time or UTC (universal, or GMT) time.

All dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC) with a day containing 86,400,000 milliseconds.

Some examples of instantiating a date:

var today = new Date()  
var d1 = new Date("October 13, 1975 11:13:00")  
var d2 = new Date(79,5,24)  
var d3 = new Date(79,5,24,11,33,0)

## Set Dates

We can easily manipulate the date by using the methods available for the Date object.

In the example below we set a Date object to a specific date (14th January 2010):

var myDate=new Date();  
myDate.setFullYear(2010,0,14);

And in the following example we set a Date object to be 5 days into the future:

var myDate=new Date();  
myDate.setDate(myDate.getDate()+5);

**Note:** If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

## Compare Two Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2100:

var x=new Date();  
x.setFullYear(2100,0,14);  
var today = new Date();  
  
if (x>today)  
  {  
  alert("Today is before 14th January 2100");  
  }  
else  
  {  
  alert("Today is after 14th January 2100");  
  }

## 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 GMT 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() method instead |
| [parse()](http://www.w3schools.com/jsref/jsref_parse.asp) | Parses a date string and returns the number of milliseconds since midnight of January 1, 1970 |
| [setDate()](http://www.w3schools.com/jsref/jsref_setdate.asp) | Sets the day of the month (from 1-31) |
| [setFullYear()](http://www.w3schools.com/jsref/jsref_setfullyear.asp) | Sets the year (four digits) |
| [setHours()](http://www.w3schools.com/jsref/jsref_sethours.asp) | Sets the hour (from 0-23) |
| [setMilliseconds()](http://www.w3schools.com/jsref/jsref_setmilliseconds.asp) | Sets the milliseconds (from 0-999) |
| [setMinutes()](http://www.w3schools.com/jsref/jsref_setminutes.asp) | Set the minutes (from 0-59) |
| [setMonth()](http://www.w3schools.com/jsref/jsref_setmonth.asp) | Sets the month (from 0-11) |
| [setSeconds()](http://www.w3schools.com/jsref/jsref_setseconds.asp) | Sets the seconds (from 0-59) |
| [setTime()](http://www.w3schools.com/jsref/jsref_settime.asp) | Sets a date and time by adding or subtracting a specified number of milliseconds to/from midnight January 1, 1970 |
| [setUTCDate()](http://www.w3schools.com/jsref/jsref_setutcdate.asp) | Sets the day of the month, according to universal time (from 1-31) |
| [setUTCFullYear()](http://www.w3schools.com/jsref/jsref_setutcfullyear.asp) | Sets the year, according to universal time (four digits) |
| [setUTCHours()](http://www.w3schools.com/jsref/jsref_setutchours.asp) | Sets the hour, according to universal time (from 0-23) |
| [setUTCMilliseconds()](http://www.w3schools.com/jsref/jsref_setutcmilliseconds.asp) | Sets the milliseconds, according to universal time (from 0-999) |
| [setUTCMinutes()](http://www.w3schools.com/jsref/jsref_setutcminutes.asp) | Set the minutes, according to universal time (from 0-59) |
| [setUTCMonth()](http://www.w3schools.com/jsref/jsref_setutcmonth.asp) | Sets the month, according to universal time (from 0-11) |
| [setUTCSeconds()](http://www.w3schools.com/jsref/jsref_setutcseconds.asp) | Set the seconds, according to universal time (from 0-59) |
| setYear() | Deprecated. Use the setFullYear() 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() method instead |
| [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 string since midnight of January 1, 1970, according to universal time |
| [valueOf()](http://www.w3schools.com/jsref/jsref_valueof_date.asp) | Returns the primitive value of a Date object |

# JavaScript Validation

## JavaScript Form Validation

JavaScript can be used to validate data in HTML forms before sending off the content to a server.

Form data that typically are checked by a JavaScript could be:

* has the user left required fields empty?
* has the user entered a valid e-mail address?
* has the user entered a valid date?
* has the user entered text in a numeric field?

## Required Fields

The function below checks if a field has been left empty. If the field is blank, an alert box alerts a message, the function returns false, and the form will not be submitted:

function validateForm()  
{  
var x=document.forms["myForm"]["fname"].value;  
if (x==null || x=="")  
  {  
  alert("First name must be filled out");  
  return false;  
  }  
}

The function above could be called when a form is submitted:

## Example

<form name="myForm" action="demo\_form.asp" onsubmit="return validateForm()" method="post">  
First name: <input type="text" name="fname">  
<input type="submit" value="Submit">  
</form>

**E-mail Validation**

The function below checks if the content has the general syntax of an email.

This means that the input data must contain an @ sign and at least one dot (.). Also, the @ must not be the first character of the email address, and the last dot must be present after the @ sign, and minimum 2 characters before the end:

function validateForm()  
{  
var x=document.forms["myForm"]["email"].value;  
var atpos=x.indexOf("@");  
var dotpos=x.lastIndexOf(".");  
if (atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)  
  {  
  alert("Not a valid e-mail address");  
  return false;  
  }  
}

The function above could be called when a form is submitted:

## Example

<form name="myForm" action="demo\_form.asp" onsubmit="return validateForm();" method="post">  
Email: <input type="text" name="email">  
<input type="submit" value="Submit">  
</form>

## Frame Object

The Frame object represents an HTML frame.

The <frame> tag defines one particular window (frame) within a frameset.

For each <frame> tag in an HTML document, a Frame object is created.

## Frameset Object

The Frameset object represents an HTML frameset.

The HTML frameset element holds two or more frame elements. Each frame element holds a separate document.

The HTML frameset element states only how many columns or rows there will be in the frameset.

# Frameset cols Property

## Definition and Usage

The cols property sets or returns the value of the cols attribute in a frameset.

The cols attribute specifies the size of, and the number of frames in a frameset. Each frame’s width is specified in a comma-separated list.

## Syntax

framesetObject.cols=*values*

The cols property can have one or more of the following values:

|  |  |
| --- | --- |
| **Value** | **Description** |
| *pixels* | The column size in pixels (like "100px" or just "100") |
| *%* | The column size in percent of the available space (like "50%") |
| \* | The rest of the available space should be assigned this column |

## Example

Create a frameset with two columns. Each column is set to 50% of the browser window:

<html>  
<frameset id="main" cols="50%,50%">  
  <frame src="frame\_cols.htm" />  
  <frame src="frame\_a.htm" />  
</frameset>  
</html>

The source code of "frame\_cols.htm" is as follows:

## Example

<html>  
<head>  
<script type="text/javascript">  
function changeCols()  
  {  
  parent.document.getElementById("main").cols="30%,70%"  
  }  
function restoreCols()  
  {  
  parent.document.getElementById("main").cols="50%,50%"  
  }  
</script>  
</head>  
<body>  
  
<input type="button" onclick="changeCols()" value="Change column size" />  
<input type="button" onclick="restoreCols()" value="Restore column size" />  
  
</body>  
</html>

# Frameset rows Property

## Definition and Usage

The rows property sets or returns the value of the rows attribute in a frameset.

The rows attribute specifies the size of, and the number of frames in a frameset. Each frame’s height is specified in a comma-separated list.

## Syntax

framesetObject.rows=*values*

The rows property can have one or more of the following values:

|  |  |
| --- | --- |
| **Value** | **Description** |
| *pixels* | The row height in pixels (like "100px" or just "100") |
| *%* | The row height in percent of the available space (like "50%") |
| \* | The rest of the available space should be assigned this row |

## Example

Create a frameset with two rows. Each row is set to 50% of the browser window:

<html>  
<frameset id="main" rows="50%,50%">  
  <frame src="frame\_rows.htm" />  
  <frame src="frame\_a.htm" />  
</frameset>  
</html>

The source code of "frame\_rows.htm" is as follows:

## Example

<html>  
<head>  
<script type="text/javascript">  
function changeRows()  
  {  
  parent.document.getElementById("main").rows="30%,70%"  
  }  
function restoreRows()  
  {  
  parent.document.getElementById("main").rows="50%,50%"  
  }  
</script>  
</head>  
<body>  
  
<form>  
<input type="button" onclick="changeRows()" value="Change row size" />  
<input type="button" onclick="restoreRows()" value="Restore row size" />  
</form>  
  
</body>  
</html>

## Frameset Object Events

|  |  |
| --- | --- |
| **Event** | **Description** |
| [onload](http://www.w3schools.com/jsref/event_frameset_onload.asp) | Script to be run immediately after a page is loaded |

# Frameset onload Event

## Syntax

onload="*JavaScriptCode*"

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| JavaScriptCode | Required. Specifies a JavaScript to be executed when the event occurs |

## Example

Alert "Page is loaded" immediately after a frameset is loaded:

<html>  
<head>  
<script type="text/javascript">  
function load()  
{  
alert("Page is loaded");  
}  
</script>  
</head>  
  
<frameset onload="load()" cols="50%,50%">  
  <frame src="frame\_a.htm" />  
  <frame src="frame\_b.htm" />  
</frameset>  
  
</html>