

JAVASCRIPT

INDEX

<u>Topics</u>	<u>Page No</u>
1. What is Javascript?	4
2. What can a JavaScript do?	4
3. Where to Implement JavaScript	5
4. Comments in Javascript	7
5. Javascript Variables	8
6. Javascript Operators	
6.1. Arithmetic	10
6.2. Assignment	10
6.3. Comparison	11
6.4. Logical	12
6.5. Conditional	12
7. Conditional Statement	12
7.1. If statement	13
7.2. IfElse statement	13
7.3. IfElse if Else statement	14
7.4. Switch Case statement	15
8. Pop Up Box	
8.1. Alert Box	16
8.2. Confirm Box	16
8.3. Prompt Box	17
9. Javascript Functions	18
10. Javascript Loops	19
10.1. For Loop	19
10.2. While Loop	20
10.3. do ... while Loop	21
10.4. For ... In Loop	22
11. Javascript Events	
11.1. onLoad and onUnload	24
11.2. onFocus, onBlur, onClick and onChange	24
11.3. OnSubmit	24

11.4. onmouseover and onmouseout	24
12. Special Characters	25
13. Javascript Strings	27
14. Javascript Arrays	27
15. Javascript Timeout Functions	29
16. Javascript Validations	30

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++.

JavaScript was created by Netscape to extend the functions a browser can perform.

What can a 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 put dynamic text into an HTML page** - A JavaScript statement like this: `document.write("<h1>" + name + "</h1>")` can write a variable text into an HTML page
 - **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

The HTML <script> tag is used to insert a JavaScript into an HTML page.

The example below shows how to use JavaScript to write text on a web page:

Example

```
<html>
<body>
<script type="text/javascript">
document.write("Hello World!");
</script>
</body>
</html>
```

Where To Implement JavaScript

JavaScripts in the body section will be executed WHILE the page loads.

JavaScripts in the head section will be executed when CALLED.

Where to Put the JavaScript

JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.

Scripts in <head>

Scripts to be executed when they are called, or when an event is triggered, go in the head section.

If you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

Example

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>
```

```
<body onload="message()" ">
</body>
</html>
```

Scripts in <body>

Scripts to be executed when the page loads go in the body section.

If you place a script in the body section, it generates the content of a page.

Example

```
<html>
<head>
</head>

<body>
<script type="text/javascript">
document.write("This message is written by JavaScript");
</script>
</body>

</html>
```

Scripts in <head> and <body>

You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>
<head>
<script type="text/javascript">
....
</script>
</head>
<body>
<script type="text/javascript">
....
</script>
</body>
```

Using an External JavaScript

If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file.

Save the external JavaScript file with a .js file extension.

Note: The external script cannot contain the <script> tag!

To use the external script, point to the .js file in the "src" attribute of the <script> tag:

Example

```
<html>
<head>
<script type="text/javascript" src="xxx.js"></script>
</head>
<body>
</body>
</html>
```

Comments In Javascript

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!"
</script>
```

JavaScript Variables

Variables are "containers" for storing information.

Do you remember algebra from school?

`x=5, y=6, z=x+y`

Do you remember that a letter (like `x`) could be used to hold a value (like 5), and that you could use the information above to calculate the value of `z` to be 11?

These letters are called **variables**, and variables can be used to hold values (`x=5`) or expressions (`z=x+y`).

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 or the underscore character

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

Example

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 can declare JavaScript variables with the **var statement**:

```
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.

Assigning Values to Undeclared JavaScript Variables

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

These statements:

```
x=5;
carname="Volvo";
```

have the same effect as:

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

Redeclaring JavaScript Variables

If you redeclare a JavaScript variable, it will not lose its original value.

```
var x=5;
var x;
```

After the execution of the statements above, the variable **x** will still have the value of 5. The value of **x** is not reset (or cleared) when you redeclare it.

JavaScript Operators

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
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1
++	Increment	x=++y	x=6

--	Decrement	x=--y	x=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);
```

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	Example
==	is equal to	x==8 is false
===	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");
```

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

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

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".

Conditional Statements

Conditional statements are used to perform different actions based on different conditions.

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 condition1 and condition2 are not 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>
```

JavaScript Switch Statement

Conditional statements are used to perform different actions based on different conditions.

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();
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 Popup Boxes

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");
```

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("Hello " + name + "! How are you today?");
}
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()" value="Show prompt box" />

</body>
</html>
```

JavaScript 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 line 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 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 Loop

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 (var=startvalue;var<=endvalue;var=var+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>
```

JavaScript While Loop

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

The while Loop

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

Syntax

```
while (var<=endvalue)
{
  code to be executed
}
```

Note: The <= could be any comparing statement.

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 (var<=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 elements of an array or 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 element/property.

Note: The variable argument can be a named variable, an array element, or a property of an object.

Example

Use the for...in statement to loop through an array:

Example

```
<html>
<body>

<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";

for (x in mycars)
{
    document.write(mycars[x] + "<br />");
}
</script>

</body>
</html>
```

JavaScript Events

Events are actions that can be detected by JavaScript.

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.

Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

- A mouse click
 - A web page or an image loading
 - Mousing over a hot spot on the web page
 - Selecting an input field in an HTML form
 - Submitting an HTML form
 - A keystroke
-

Note: Events are normally used in combination with functions, and the function will not be executed before the event occurs!

onLoad and onUnload

The onLoad and onUnload events are triggered when the user enters or leaves the page.

The onLoad event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

Both the onLoad and onUnload events are also often used to deal with cookies that should be set when a user enters or leaves a page. For example, you could have a popup asking for the user's name upon his first arrival to your page. The name is then stored in a cookie. Next time the visitor arrives at your page, you could have another popup saying something like: "Welcome John Doe!".

onFocus, onClick onBlur and onChange

The onFocus, onBlur and onChange events are often used in combination with validation of form fields.

Below is an example of how to use the onChange event. The checkEmail() function will be called whenever the user changes the content of the field:

```
<input type="text" size="30" id="email" onchange="checkEmail()">
```

onSubmit

The onSubmit event is used to validate ALL form fields before submitting it.

Below is an example of how to use the onSubmit event. The checkForm() function will be called when the user clicks the submit button in the form. If the field values are not accepted, the submit should be cancelled. The function checkForm() returns either true or false. If it returns true the form will be submitted, otherwise the submit will be cancelled:

```
<form method="post" action="xxx.htm" onsubmit="return checkForm()">
```

onMouseOver and onMouseOut

onMouseOver and onMouseOut are often used to create "animated" buttons.

Below is an example of an onMouseOver event. An alert box appears when an onMouseOver event is detected:

```
<a href="http://www.w3schools.com" onmouseover="alert('An onMouseOver event');return false"></a>
```

JavaScript Special Characters

In JavaScript you can add special characters to a text string by using the backslash sign.

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.

Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called

To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript will now output the proper text string: We are the so-called "Vikings" from the north.

Here is another example:

```
document.write ("You \& I are singing!");
```

The example above will produce the following output:

```
You & I are singing!
```

The table below lists other special characters that can be added to a text string with the backslash sign:

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

JavaScript String Object

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

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

Examples of use:

The following example uses the length property of the String object to find the length of a string:

```
var txt="Hello world!";  
document.write(txt.length);
```

The code above will result in the following output:

```
12
```

The following example uses the toUpperCase() method of the String object to convert a string to uppercase letters:

```
var txt="Hello world!";  
document.write(txt.toUpperCase());
```

The code above will result in the following output:

```
HELLO WORLD!
```

JavaScript Array Object

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:

```
$cars1="Saab";  
$cars2="Volvo";  
$cars3="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

The following code creates an Array object called myCars:

```
var myCars=new Array();
```

There are two ways of adding values to an array (you can add as many values as you need to define as many variables you require).

1:

```
var myCars=new Array();  
myCars[0]="Saab";  
myCars[1]="Volvo";  
myCars[2]="BMW";
```

You could also pass an integer argument to control the array's size:

```
var myCars=new Array(3);  
myCars[0]="Saab";  
myCars[1]="Volvo";  
myCars[2]="BMW";
```

2:

```
var myCars=new Array("Saab","Volvo","BMW");
```

Note: If you specify numbers or true/false values inside the array then the type of variables will be numeric 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

JavaScript Timing Events

With JavaScript, it is possible to execute some code after a specified time-interval. This is called timing events.

It's very easy to time events in JavaScript. The two key methods that are used are:

- `setTimeout()` - executes a code some time in the future
- `clearTimeout()` - cancels the `setTimeout()`

Note: The `setTimeout()` and `clearTimeout()` are both methods of the HTML DOM Window object.

The `setTimeout()` Method

Syntax

```
var t=setTimeout("javascript statement",milliseconds);
```

The `setTimeout()` method returns a value - In the statement above, the value is stored in a variable called `t`. If you want to cancel this `setTimeout()`, you can refer to it using the variable name.

The first parameter of `setTimeout()` is a string that contains a JavaScript statement. This statement could be a statement like `"alert('5 seconds!')"` or a call to a function, like `"alertMsg()"`.

The second parameter indicates how many milliseconds from now you want to execute the first parameter.

Note: There are 1000 milliseconds in one second.

Example

When the button is clicked in the example below, an alert box will be displayed after 5 seconds.

Example

```
<html>
<head>
<script type="text/javascript">
function timedMsg()
{
var t=setTimeout("alert('5 seconds!')",5000);
}
</script>
</head>

<body>
<form>
<input type="button" value="Display timed alertbox!"
onClick="timedMsg()" />
</form>
```

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

Example - Infinite Loop

To get a timer to work in an infinite loop, we must write a function that calls itself. In the example below, when the button is clicked, the input field will start to count (for ever), starting at 0:

Example

```
<html>
<head>
<script type="text/javascript">
var c=0
var t
function timedCount()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("timedCount()",1000);
}
</script>
</head>

<body>
<form>
<input type="button" value="Start count!" onClick="timedCount()" />
<input type="text" id="txt" />
</form>
</body>

</html>
```

The clearTimeout() Method

Syntax

```
clearTimeout(setTimeout_variable)
```

Example

The example below is the same as the "Infinite Loop" example above. The only difference is that we have now added a "Stop Count!" button that stops the timer:

Example

```
<html>
<head>
<script type="text/javascript">
var c=0
var t

function timedCount()
{
document.getElementById('txt').value=c;
```

```
c=c+1;
t=setTimeout("timedCount()",1000);
}

function stopCount()
{
clearTimeout(t);
}
</script>
</head>

<body>
<form>
<input type="button" value="Start count!" onClick="timedCount()" />
<input type="text" id="txt" />
<input type="button" value="Stop count!" onClick="stopCount()" />
</form>
</body>

</html>
```

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?

To check a empty field

```
var field_val=document.getElementById('fieldid').value;

if(field_val=="")
{
    alert("The Field is Empty");
}
else
{
    alert("The Field is not Empty");
}
```