Introduction to JavaScript

JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.

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

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML / XHTML

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
- A JavaScript consists of lines of executable computer code
- A 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 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 Real Name is ECMAScript

JavaScript's official name is "ECMAScript". The standard is developed and maintained by the ECMA organization (http://www.ecma-international.org/publications/index.html).

ECMA-262 is the official JavaScript standard. The standard is based on JavaScript (Netscape) and JScript (Microsoft).

The language was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996.

The development of ECMA-262 started in 1996, and the first edition of was adopted by the ECMA General Assembly in June 1997.

The standard was approved as an international ISO (ISO/IEC 16262) standard in 1998.

The development of the standard is still in progress.

JavaScript How To ...

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

How to Put a JavaScript Into an HTML Page

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

The code above will produce this output on an HTML page:

```
Hello World!
```

Example Explained

To insert a JavaScript into an HTML page, we use the <script> tag. Inside the <script> tag we use the "type=" attribute to define the scripting language.

So, the <script type="text/javascript"> and </script> tells where the JavaScript starts and ends:

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

The word **document.write** is a standard JavaScript command for writing output to a page.

By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write Hello World! to the page:

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

Note: If we had not entered the <script> tag, the browser would have treated the document.write("Hello World!") command as pure text, and just write the entire line on the page.

HTML Comments to Handle Simple Browsers

Browsers that do not support JavaScript will display JavaScript as page content.

To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag can be used to "hide" the JavaScript. Just add an HTML comment tag <!-- before the first JavaScript statement, and a --> (end of comment) after the last JavaScript statement.

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

The two forward slashes at the end of comment line (//) is the JavaScript comment symbol. This prevents JavaScript from executing the --> tag.

JavaScript Where To ...

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 the head section: Scripts to be executed when they are called, or when an event is triggered, go in the head section. When you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

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

Scripts in the body section: Scripts to be executed when the page loads go in the body section. When you place a script in the body section it generates the content of the page.

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

Scripts in both the body and the head section: 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

Sometimes you might want to run the same JavaScript on several pages, without having to write the same script on every page.

To simplify this, 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:

```
<html>
<head>
<script src="md.js"></script>
</head>
<body>
</body>
</html>
```

Note: Remember to place the script exactly where you normally would write the script!

JavaScript Statements

JavaScript is a sequence of statements to be executed by the browser.

JavaScript Statements

A JavaScript statements is a command to the 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 header and two paragraphs to a web page:

```
<script type="text/javascript">
document.write("<h1>This is a header</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 ends with a right curly bracket }.

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

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

```
<script type="text/javascript">
{
document.write("<h1>This is a header</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.

JavaScript Comments

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

Single line comments start with //.

This example uses single line comments to explain the code:

```
<script type="text/javascript">
// This will write a header:
document.write("<h1>This is a header</h1>");
// This will 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 */.

This example uses a multi line comment to explain the code:

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

Using Comments to Prevent Execution

In this example the comment is used to prevent the execution of a single code line:

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

In this example the comments is used to prevent the execution of multiple code lines:

```
<script type="text/javascript">
/*
document.write("<h1>This is a header</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 this example the comment is placed at the end of a line:

```
<script type="text/javascript">
document.write("Hello"); // This will write "Hello"
document.write("Dolly"); // This will write "Dolly"
</script>
```

JavaScript Variables

Variables are "containers" for storing information:

x=5; length=66.10;

Do You Remember Algebra From School?

Hopefully you remember algebra like this 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 calculate the value of z to be 11?

Sure you do!

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 describing name like **length**.

A JavaScript variable can also hold a text value like in **carname="Volvo"**.

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.

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 has no values, but you can assign values to the variables while you declare them:

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

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

Assigning Values to JavaScript Variables

You assign values to JavaScript variables with **assignment statements**:

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

The variable name is on the left side of the = sign, and the value you want to assign to the variable is on the right.

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

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that has 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 loose 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 Arithmetic

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

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

You will learn more about the operators that can be used between JavaScript variables in the next chapter of this tutorial.

JavaScript Operators

The operator = is used to assign values.

The operator + is used to add values.

The assignment operator **=** is used to assign values to JavaScript variables.

The arithmetic operator + is used to add values together.

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

The value of x, after the execution of the statements above is 7.

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
-	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

Look at these examples:

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

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

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

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

If you add a number and a string, the result will be a string.

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");
```

You will learn more about the use of conditional statements in the next chapter of this tutorial.

Logical Operators

Logical operators are used in 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".

JavaScript If...Else Statements

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

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

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

---

If Statement

You should use the if statement if you want 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 1

```
<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>
```

Example 2

```
<script type="text/javascript">
//Write "Lunch-time!" if the time is 11
var d=new Date();
var time=d.getHours();

if (time==11)
{
document.write("<b>Lunch-time!</b>");
}
</script>
```

Note: When **comparing** variables you must always use two equals signs next to each other (==)!

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

If...else Statement

If you want to execute some code if a condition is true and another code if the condition is not true, use the if....else statement.

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

You should use the if....else if...else statement if you want to select one of many sets of lines to execute.

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 in JavaScript are used to perform different actions based on different conditions.

The JavaScript Switch Statement

You should use the switch statement if you want 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

In JavaScript we can create three kinds 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");
```

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");
```

JavaScript Functions

A function is a reusable code-block that will be executed by an event, or when the function is called.

JavaScript Functions

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 that function.

You may call a function from anywhere within the 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 the function is read/loaded by the browser before it is called, it could be wise to put it in the <head> section.

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 the user hits the button. We have added an onClick event to the button that will execute the function displaymessage() when the button is clicked.

How to Define a Function

The syntax for creating a function is:

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

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:

```
function functionname()
{
some code
}
```

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.

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.

Example

The function below should return the product of two numbers (a and b):

```
function prod(a,b)
{
x=a*b;
return x;
}
```

When you call the function above, you must pass along two parameters:

```
product=prod(2,3);
```

The returned value from the prod() function is 6, and it will be stored in the variable called product.

The Lifetime of JavaScript Variables

When 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 For Loop

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

JavaScript Loops

Very 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

Explanation: 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 10. **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.

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

Result

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

JavaScript While Loop

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

The while loop

The while loop is used when you want the loop to execute and continue executing while the specified condition is true.

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

Note: The <= could be any comparing statement.

Example

Explanation: 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 10. **i** will increase by 1 each time the loop runs.

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=10)
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
</script>
</body>
</html>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true. This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

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

Example

```
<html>
```

```
<body>
<script type="text/javascript">
var i=0;
do
{
document.write("The number is " + i);
document.write("<br />");
i=i+1;
}
while (i<0);
</script>
</body>
</html>
```

Result

```
The number is 0
```

JavaScript Break and Continue

There are two special statements that can be used inside loops: break and continue.

Break

The break command 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>
```

Result

```
The number is 0
The number is 1
The number is 2
```

Continue

The continue command 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>
```

Result

```
The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
The number is 10
```

JavaScript For...In Statement

The for...in statement is used to loop (iterate) through the elements of an array or through the properties of an object.

JavaScript For...In Statement

The for...in statement is used to loop (iterate) through the elements of an array or through the properties of an object.

The code in the body of the for ... in loop is executed once for each element/property.

Syntax

```
for (variable in object)
```

```
{
    code to be executed
}
```

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

Example

Using for...in to loop through an array:

```
<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.

Events

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 JavaScript functions. 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 box 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 Madhavendra Dutt!".

onFocus, 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="md.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.madhavendra.com"
onmouseover="alert('An onMouseOver event');return false">
<img src="madhav.gif" width="100" height="30">
</a>
```

JavaScript Try...Catch Statement

The try...catch statement allows you to test a block of code for errors.

JavaScript - Catching Errors

When browsing Web pages on the internet, we all have seen a JavaScript alert box telling us there is a runtime error and asking "Do you wish to debug?". Error message like this may be useful for developers but not for users. When users see errors, they often leave the Web page.

This chapter will teach you how to trap and handle JavaScript error messages, so you don't lose your audience.

There are two ways of catching errors in a Web page:

- By using the **try...catch** statement (available in IE5+, Mozilla 1.0, and Netscape 6)
- By using the **onerror** event. This is the old standard solution to catch errors (available since Netscape 3)

Try...Catch Statement

The try...catch statement allows you to test a block of code for errors. The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.

Syntax

```
Try
{
//Run some code here
}
catch(err)
{
//Handle errors here
}
```

Note that try...catch is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Example 1

The example below contains a script that is supposed to display the message "Welcome guest!" when you click on a button. However, there's a typo in the message() function. alert() is misspelled as adddlert(). A JavaScript error occurs:

```
<html>
<head>
<script type="text/javascript">
function message()
{
adddlert("Welcome guest!");
}
```

```
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()" />
</body>

</html>
```

To take more appropriate action when an error occurs, you can add a try...catch statement.

The example below contains the "Welcome guest!" example rewritten to use the try...catch statement. Since alert() is misspelled, a JavaScript error occurs. However, this time, the catch block catches the error and executes a custom code to handle it. The code displays a custom error message informing the user what happened:

```
<html>
<head>
<script type="text/javascript">
var txt=""
function message()
{
try
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.description + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
  }
}
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()" />
</body>

</html>
```

Example 2

The next example uses a confirm box to display a custom message telling users they can click OK to continue viewing the page or click Cancel to go to the homepage. If the confirm method returns false, the user clicked Cancel, and the code redirects the user. If the confirm method returns true, the code Dutts nothing:

```
<html>
<head>
<script type="text/javascript">
var txt=""
function message()
{
try
```

```
  {
  adddlert("Welcome guest!");
  }
catch(err)
  {
  txt="There was an error on this page.\n\n";
  txt+="Click OK to continue viewing this page,\n";
  txt+="or Cancel to return to the home page.\n\n";
  if(!confirm(txt))
    {
    document.location.href="http://www.madhavendra.com/";
    }
  }
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()" />
</body>
</html>
```

The onerror Event

The onerror event will be explained soon, but first you will learn how to use the throw statement to create an exception. The throw statement can be used together with the try...catch statement.

JavaScript Throw Statement

The throw statement allows you to create an exception.

The Throw Statement

The throw statement allows you to create an exception. If you use this statement together with the try...catch statement, you can control program flow and generate accurate error messages.

Syntax

```
throw(exception)
```

The exception can be a string, integer, Boolean or an object.

Note that throw is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Example 1

The example below determines the value of a variable called x. If the value of x is higher than 10 or lower than 0 we are going to throw an error. The error is then caught by the catch argument and the proper error message is displayed:

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try
{
if(x>10)
throw "Err1";
else if(x<0)
throw "Err2";
}
catch(er)
{
if(er=="Err1")
alert("Error! The value is too high");
if(er == "Err2")
alert("Error! The value is too low");
}
</script>
</body>
</html>
```

JavaScript The onerror Event

Using the onerror event is the old standard solution to catch errors in a web page.

The onerror Event

We have just explained how to use the try...catch statement to catch errors in a web page. Now we are going to explain how to use the onerror event for the same purpose.

The onerror event is fired whenever there is a script error in the page.

To use the onerror event, you must create a function to handle the errors. Then you call the function with the onerror event handler. The event handler is called with three arguments: msg (error message), url (the url of the page that caused the error) and line (the line where the error occurred).

Syntax

```
onerror=handleErr
function handleErr(msg,url,l)
{
//Handle the error here
return true or false
}
```

The value returned by onerror determines whether the browser displays a standard error message. If you return false, the browser displays the standard error message in the JavaScript console. If you return true, the browser Dutts not display the standard error message.

Example

The following example shows how to catch the error with the onerror event:

```
<html>
<head>
<script type="text/javascript">
onerror=handleErr;
var txt="";
function handleErr(msg,url,l)
{
txt="There was an error on this page.\n\n";
txt+="Error: " + msg + "\n";
txt+="URL: " + url + "\n";
txt+="Line: " + l + "\n\n";
txt+="Click OK to continue.\n\n";
alert(txt);
return true;
}
function message()
{
adddlert("Welcome guest!");
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()" />
</body>
</html>
```

JavaScript Special Characters

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

Insert Special Characters

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);
```

JavasScript 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 Guidelines

Some other important things to know when scripting with JavaScript.

JavaScript is Case Sensitive

A function named "myfunction" is not the same as "myFunction" and a variable named "myVar" is not the same as "myvar".

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

White Space

JavaScript ignores extra spaces. You can add white space to your script to make it more readable. The following lines are equivalent:

```
name="Madhav";
name = "Madhav";
```

Break up a Code Line

You can break up a code line **within a text string** with a backslash. The example below will be displayed properly:

```
document.write("Hello \
World!");
```

However, you cannot break up a code line like this:

```
document.write \
("Hello World!");
```

JavaScript Objects Introduction

JavaScript is an Object Oriented Programming (OOP) language.

An OOP language allows you to define your own objects and make your own variable types.

Object Oriented Programming

JavaScript is an Object Oriented Programming (OOP) language. An OOP language 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!
```

JavaScript String Object

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

Examples

Return the length of a string
Use the length property to find the length of a string.

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

Style strings
Style strings.

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

var txt="Hello World!";

document.write("<p>Big: " + txt.big() + "</p>");
document.write("<p>Small: " + txt.small() + "</p>");

document.write("<p>Bold: " + txt.bold() + "</p>");
document.write("<p>Italic: " + txt.italics() + "</p>");

document.write("<p>Blink: " + txt.blink() + " (Dutts not work in IE)</p>");
document.write("<p>Fixed: " + txt.fixed() + "</p>");
document.write("<p>Strike: " + txt.strike() + "</p>");

document.write("<p>Fontcolor: " + txt.fontcolor("Red") + "</p>");
document.write("<p>Fontsize: " + txt.fontsize(16) + "</p>");

document.write("<p>Lowercase: " + txt.toLowerCase() + "</p>");
document.write("<p>Uppercase: " + txt.toUpperCase() + "</p>");

document.write("<p>Subscript: " + txt.sub() + "</p>");
document.write("<p>Superscript: " + txt.sup() + "</p>");

document.write("<p>Link: " + txt.link("http://www.MadhavendraDutt.com") + "</p>");
</script>
```

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

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

```
<html>
<body>
<script type="text/javascript">
var str="Hello world!";
document.write(str.indexOf("Hello") + "<br />");
document.write(str.indexOf("World") + "<br />");
document.write(str.indexOf("world"));
</script>
</body>
</html>
```

The match() method
How to use the match() method to search for a specified string value within a string and return the string value if found

```
<html>
<body>
<script type="text/javascript">
var str="Hello world!";
document.write(str.match("world") + "<br />");
document.write(str.match("World") + "<br />");
document.write(str.match("worlld") + "<br />");
document.write(str.match("world!"));
</script>
</body>
</html>
```

Replace characters in a string - replace()
How to use the replace() method to replace some characters with some other characters in a string.

```
<html>
<body>
<script type="text/javascript">
var str="Visit Microsoft!";
document.write(str.replace(/Microsoft/,"Madhavendra Dutt"));
</script>
</body>
</html>
```

JavaScript String Object Reference

String Object Methods

FF: Firefox, **N**: Netscape, **IE**: Internet Explorer

Method	Description	FF	N	IE
anchor()	Creates an HTML anchor	1	2	3
big()	Displays a string in a big font	1	2	3

blink()	Displays a blinking string	1	2	
bold()	Displays a string in bold	1	2	3
charAt()	Returns the character at a specified position	1	2	3
charCodeAt()	Returns the Unicode of the character at a specified position	1	4	4
concat()	Joins two or more strings	1	4	4
fixed()	Displays a string as teletype text	1	2	3
fontcolor()	Displays a string in a specified color	1	2	3
fontsize()	Displays a string in a specified size	1	2	3
fromCharCode()	Takes the specified Unicode values and returns a string	1	4	4
indexOf()	Returns the position of the first occurrence of a specified string value in a string	1	2	3
italics()	Displays a string in italic	1	2	3
lastIndexOf()	Returns the position of the last occurrence of a specified string value, searching backwards from the specified position in a string	1	2	3
link()	Displays a string as a hyperlink	1	2	3
match()	Searches for a specified value in a string	1	4	4
replace()	Replaces some characters with some other characters in a string	1	4	4
search()	Searches a string for a specified value	1	4	4
slice()	Extracts a part of a string and returns the extracted part in a new string	1	4	4
small()	Displays a string in a small font	1	2	3
split()	Splits a string into an array of strings	1	4	4
strike()	Displays a string with a strikethrough	1	2	3
sub()	Displays a string as subscript	1	2	3
substr()	Extracts a specified number of characters in a string, from a start index	1	4	4
substring()	Extracts the characters in a string between two specified indices	1	2	3
sup()	Displays a string as superscript	1	2	3
toLowerCase()	Displays a string in lowercase letters	1	2	3
toUpperCase()	Displays a string in uppercase letters	1	2	3
toSource()	Represents the source code of an object	1	4	-
valueOf()	Returns the primitive value of a String object	1	2	4

String Object Properties

Property	Description	FF	N	IE
constructor	A reference to the function that created the object	1	4	4
length	Returns the number of characters in a string	1	2	3
prototype	Allows you to add properties and methods to the object	1	2	4

String object

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 Date Object

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

Examples

Return today's date and time
Use the Date() method to get today's date.

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

getTime()
Use getTime() to calculate the years since 1970.

```
<html>
<body>
<script type="text/javascript">
var minutes = 1000*60;
var hours = minutes*60;
var days = hours*24;
var years = days*365;
var d = new Date();
var t = d.getTime();
var y = t/years;
document.write("It's been: " + y + " years since 1970/01/01!");
</script>
```

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

setFullYear()
Use setFullYear() to set a specific date.

```
<html>
<body>
<script type="text/javascript">
var d = new Date();
d.setFullYear(1992,10,3);
document.write(d);
</script>
</body>
</html>
```

toUTCString()
Use toUTCString() to convert today's date (according to UTC) to a string.

```
<html>
<body>
<script type="text/javascript">
var d = new Date();
document.write (d.toUTCString());
</script>
</body>
</html>
```

getDay()
Use getDay() and an array to write a weekday, and not just a number.

```
<html>
<body>
<script type="text/javascript">
var d=new Date();
var weekday=new Array(7);
weekday[0]="Sunday";
weekday[1]="Monday";
weekday[2]="Tuesday";
weekday[3]="Wednesday";
weekday[4]="Thursday";
weekday[5]="Friday";
weekday[6]="Saturday";
document.write("Today it is " + weekday[d.getDay()]);
</script>
</body>
</html>
```

Display a clock
How to display a clock on your web page.

```
<html>
<head>
<script type="text/javascript">
function startTime()
{
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
// add a zero in front of numbers<10
m=checkTime(m);
s=checkTime(s);
document.getElementById('txt').innerHTML=h+":"+m+":"+s;
t=setTimeout('startTime()',500);
}
function checkTime(i)
{
if (i<10)
  {
  i="0" + i;
  }
return i;
}
</script>
</head>
<body onload="startTime()">
<div id="txt"></div>
</body>
</html>
```

JavaScript Date Object Reference

Date Object Methods

FF: Firefox, **N**: Netscape, **IE**: Internet Explorer

Method	**Description**	**FF**	**N**	**IE**
Date()	Returns today's date and time	1	2	3
getDate()	Returns the day of the month from a Date object (from 1-31)	1	2	3
getDay()	Returns the day of the week from a Date object (from 0-6)	1	2	3
getMonth()	Returns the month from a Date object (from 0-11)	1	2	3
getFullYear()	Returns the year, as a four-digit number, from a Date object	1	4	4
getYear()	Returns the year, as a two-digit or a four-digit number, from a Date object. Use getFullYear() instead !!	1	2	3
getHours()	Returns the hour of a Date object (from 0-23)	1	2	3
getMinutes()	Returns the minutes of a Date object (from 0-59)	1	2	3
getSeconds()	Returns the seconds of a Date object (from 0-59)	1	2	3
getMilliseconds()	Returns the milliseconds of a Date object (from 0-999)	1	4	4
getTime()	Returns the number of milliseconds since midnight Jan 1, 1970	1	2	3
getTimezoneOffset()	Returns the difference in minutes between local time and Greenwich Mean Time (GMT)	1	2	3
getUTCDate()	Returns the day of the month from a Date object according to universal time (from 1-31)	1	4	4
getUTCDay()	Returns the day of the week from a Date object according to universal time (from 0-6)	1	4	4
getUTCMonth()	Returns the month from a Date object according to universal time (from 0-11)	1	4	4
getUTCFullYear()	Returns the four-digit year from a Date object according to universal time	1	4	4
getUTCHours()	Returns the hour of a Date object according to universal time (from 0-23)	1	4	4

getUTCMinutes()	Returns the minutes of a Date object according to universal time (from 0-59)	1	4	4
getUTCSeconds()	Returns the seconds of a Date object according to universal time (from 0-59)	1	4	4
getUTCMilliseconds()	Returns the milliseconds of a Date object according to universal time (from 0-999)	1	4	4
parse()	Takes a date string and returns the number of milliseconds since midnight of January 1, 1970	1	2	3
setDate()	Sets the day of the month in a Date object (from 1-31)	1	2	3
setMonth()	Sets the month in a Date object (from 0-11)	1	2	3
setFullYear()	Sets the year in a Date object (four digits)	1	4	4
setYear()	Sets the year in the Date object (two or four digits). Use setFullYear() instead !!	1	2	3
setHours()	Sets the hour in a Date object (from 0-23)	1	2	3
setMinutes()	Set the minutes in a Date object (from 0-59)	1	2	3
setSeconds()	Sets the seconds in a Date object (from 0-59)	1	2	3
setMilliseconds()	Sets the milliseconds in a Date object (from 0-999)	1	4	4
setTime()	Calculates a date and time by adding or subtracting a specified number of milliseconds to/from midnight January 1, 1970	1	2	3
setUTCDate()	Sets the day of the month in a Date object according to universal time (from 1-31)	1	4	4
setUTCMonth()	Sets the month in a Date object according to universal time (from 0-11)	1	4	4
setUTCFullYear()	Sets the year in a Date object according to universal time (four digits)	1	4	4
setUTCHours()	Sets the hour in a Date object according to universal time (from 0-23)	1	4	4
setUTCMinutes()	Set the minutes in a Date object according to universal time (from 0-59)	1	4	4
setUTCSeconds()	Set the seconds in a Date object according to universal time (from 0-59)	1	4	4
setUTCMilliseconds()	Sets the milliseconds in a Date object according to universal time (from 0-999)	1	4	4
toSource()	Represents the source code of an object	1	4	-
toString()	Converts a Date object to a string	1	2	4
toGMTString()	Converts a Date object, according to Greenwich time, to a string. Use toUTCString() instead !!	1	2	3
toUTCString()	Converts a Date object, according to universal time, to a string	1	4	4
toLocaleString()	Converts a Date object, according to local time, to a string	1	2	3
UTC()	Takes a date and returns the number of milliseconds since midnight of January 1, 1970 according to universal time	1	2	3
valueOf()	Returns the primitive value of a Date object	1	2	4

Date Object Properties

Property	**Description**	**FF**	**N**	**IE**
constructor	A reference to the function that created the object	1	4	4
prototype	Allows you to add properties and methods to the object	1	3	4

Defining Dates

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

We define a Date object with the new keyword. The following code line defines a Date object called myDate:

```
var myDate=new Date()
```

Note: The Date object will automatically hold the current date and time as its initial value!

Manipulate 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!

Comparing Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2010:

```
var myDate=new Date();
myDate.setFullYear(2010,0,14);
var today = new Date();
if (myDate>today)
{
alert("Today is before 14th January 2010");
}
else
{
alert("Today is after 14th January 2010");
}
```

JavaScript Array Object

The Array object is used to store a set of values in a single variable name.

Examples

Create an array
Create an array, assign values to it, and write the values to the output.

```
<html>
<body>
<script type="text/javascript">
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";
for (i=0;i<mycars.length;i++)
{
document.write(mycars[i] + "<br />");
}
</script>
</body>
</html>
```

For...In Statement
Use a for...in statement to loop through the elements of an array.

```
<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>
```

Join two arrays - concat()
Use the concat() method to join two arrays.

```
<html>
<body>
<script type="text/javascript">
var arr = new Array(3);
arr[0] = "Jani";
arr[1] = "Tove";
arr[2] = "Hege";
var arr2 = new Array(3);
arr2[0] = "Madhavendra";
arr2[1] = "Andy";
arr2[2] = "Wendy";
document.write(arr.concat(arr2));
</script>
</body>
```

```
</html>
```

Put array elements into a string - join()
How to use the join() method to put all the elements of an array into a string.

```
<html>
<body>
<script type="text/javascript">
var arr = new Array(3);
arr[0] = "Jani";
arr[1] = "Hege";
arr[2] = "Stale";
document.write(arr.join() + "<br />");
document.write(arr.join("."));
</script>
</body>
</html>
```

Literal array - sort()
Use the sort() method to sort a literal array.

```
<html>
<body>
<script type="text/javascript">
var arr = new Array(6);
arr[0] = "Jani";
arr[1] = "Hege";
arr[2] = "Stale";
arr[3] = "Kai Jim";
arr[4] = "Borge";
arr[5] = "Tove";
document.write(arr + "<br />");
document.write(arr.sort());
</script>
</body>
</html>
```

Numeric array - sort()
Use the sort() method to sort a numeric array.

```
<html>
<body>
<script type="text/javascript">
function sortNumber(a, b)
{
return a - b;
}
var arr = new Array(6);
arr[0] = "10";
arr[1] = "5";
arr[2] = "40";
arr[3] = "25";
arr[4] = "1000";
arr[5] = "1";
document.write(arr + "<br />");
document.write(arr.sort(sortNumber));
</script>
</body>
</html>
```

JavaScript Array Object Reference

Array Object Methods

FF: Firefox, **N**: Netscape, **IE**: Internet Explorer

Method	Description	FF	N	IE
concat()	Joins two or more arrays and returns the result	1	4	4
join()	Puts all the elements of an array into a string. The elements are separated by a specified delimiter	1	3	4
pop()	Removes and returns the last element of an array	1	4	5.5
push()	Adds one or more elements to the end of an array and returns the new length	1	4	5.5
reverse()	Reverses the order of the elements in an array	1	3	4
shift()	Removes and returns the first element of an array	1	4	5.5
slice()	Returns selected elements from an existing array	1	4	4
sort()	Sorts the elements of an array	1	3	4
splice()	Removes and adds new elements to an array	1	4	5.5
toSource()	Represents the source code of an object	1	4	-
toString()	Converts an array to a string and returns the result	1	3	4
unshift()	Adds one or more elements to the beginning of an array and returns the new length	1	4	6
valueOf()	Returns the primitive value of an Array object	1	2	4

Array Object Properties

Property	Description	FF	N	IE
constructor	A reference to the function that created the object	1	2	4
index		1	3	4
input		1	3	4
length	Sets or returns the number of elements in an array	1	2	4
prototype	Allows you to add properties and methods to the object	1	2	4

Defining Arrays

The Array object is used to store a set of values in a single variable name.

We define an Array object with the new keyword. The following code line defines an Array object called myArray:

```
var myArray=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.

Accessing Arrays

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 Existing Arrays

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 Boolean Object

The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).

Examples

Check Boolean value
Check if a Boolean object is true or false.

```
<html>
<body>
<script type="text/javascript">
var b1=new Boolean( 0);
var b2=new Boolean(1);
var b3=new Boolean("");
var b4=new Boolean(null);
var b5=new Boolean(NaN);
var b6=new Boolean("false");
document.write("0 is boolean "+ b1 +"<br />");
document.write("1 is boolean "+ b2 +"<br />");
document.write("An empty string is boolean "+ b3 + "<br />");
document.write("null is boolean "+ b4+ "<br />");
document.write("NaN is boolean "+ b5 +"<br />");
document.write("The string 'false' is boolean "+ b6 +"<br />");
</script>
</body>
</html>
```

JavaScript Boolean Object Reference

Boolean Object Methods

FF: Firefox, **N**: Netscape, **IE**: Internet Explorer

Method	Description	FF	N	IE
toSource()	Represents the source code of an object	1	4	-
toString()	Converts a Boolean value to a string and returns the result	1	4	4
valueOf()	Returns the primitive value of a Boolean object	1	4	4

Boolean Object Properties

Property	Description	FF	N	IE
constructor	A reference to the function that created the object	1	2	4
prototype	Allows you to add properties and methods to the object	1	2	4

Boolean Object

The Boolean object is an object wrapper for a Boolean value.

The Boolean object is used to convert a non-Boolean value to a Boolean value (true or false).

We define a Boolean object with the new keyword. The following code line defines a Boolean object called myBoolean:

```
var myBoolean=new Boolean();
```

Note: If the Boolean object has no initial value or if it is 0, -0, null, "", false, undefined, or NaN, the object is set to false. Otherwise it is true (even with the string "false")!

All the following lines of code create Boolean objects with an initial value of false:

```
var myBoolean=new Boolean();
var myBoolean=new Boolean(0);
var myBoolean=new Boolean(null);
var myBoolean=new Boolean("");
var myBoolean=new Boolean(false);
var myBoolean=new Boolean(NaN);
```

And all the following lines of code create Boolean objects with an initial value of true:

```
var myBoolean=new Boolean(true);
var myBoolean=new Boolean("true");
var myBoolean=new Boolean("false");
var myBoolean=new Boolean("Richard");
```

JavaScript Math Object

The Math object allows you to perform common mathematical tasks.

Examples

round()
Use round().

```
<html>
<body>
<script type="text/javascript">
document.write(Math.round(0.60) + "<br />");
document.write(Math.round(0.50) + "<br />");
document.write(Math.round(0.49) + "<br />");
document.write(Math.round(-4.40) + "<br />");
document.write(Math.round(-4.60));
</script>
```

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

random()
Use random() to return a random number between 0 and 1.

```
<html>
<body>
<script type="text/javascript">
document.write(Math.random());
</script>
</body>
</html>
```

max()
Use max() to return the number with the highest value of two specified numbers.

```
<html>
<body>
<script type="text/javascript">
document.write(Math.max(5,7) + "<br />");
document.write(Math.max(-3,5) + "<br />");
document.write(Math.max(-3,-5) + "<br />");
document.write(Math.max(7.25,7.30));
</script>
</body>
</html>
```

min()
Use min() to return the number with the lowest value of two specified numbers.

```
<html>
<body>
<script type="text/javascript">
document.write(Math.min(5,7) + "<br />");
document.write(Math.min(-3,5) + "<br />");
document.write(Math.min(-3,-5) + "<br />");
document.write(Math.min(7.25,7.30));
</script>
</body>
</html>
```

JavaScript Math Object Reference

Math Object Methods

FF: Firefox, **N**: Netscape, **IE**: Internet Explorer

Method	Description	FF	N	IE
abs(x)	Returns the absolute value of a number	1	2	3
acos(x)	Returns the arccosine of a number	1	2	3
asin(x)	Returns the arcsine of a number	1	2	3
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians	1	2	3

atan2(y,x)	Returns the angle theta of an (x,y) point as a numeric value between -PI and PI radians	1	2	3
ceil(x)	Returns the value of a number rounded upwards to the nearest integer	1	2	3
cos(x)	Returns the cosine of a number	1	2	3
exp(x)	Returns the value of E^x	1	2	3
floor(x)	Returns the value of a number rounded downwards to the nearest integer	1	2	3
log(x)	Returns the natural logarithm (base E) of a number	1	2	3
max(x,y)	Returns the number with the highest value of x and y	1	2	3
min(x,y)	Returns the number with the lowest value of x and y	1	2	3
pow(x,y)	Returns the value of x to the power of y	1	2	3
random()	Returns a random number between 0 and 1	1	2	3
round(x)	Rounds a number to the nearest integer	1	2	3
sin(x)	Returns the sine of a number	1	2	3
sqrt(x)	Returns the square root of a number	1	2	3
tan(x)	Returns the tangent of an angle	1	2	3
toSource()	Represents the source code of an object	1	4	-
valueOf()	Returns the primitive value of a Math object	1	2	4

Math Object Properties

Property	Description	FF	N	IE
constructor	A reference to the function that created the object	1	2	4
E	Returns Euler's constant (approx. 2.718)	1	2	3
LN2	Returns the natural logarithm of 2 (approx. 0.693)	1	2	3
LN10	Returns the natural logarithm of 10 (approx. 2.302)	1	2	3
LOG2E	Returns the base-2 logarithm of E (approx. 1.414)	1	2	3
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)	1	2	3
PI	Returns PI (approx. 3.14159)	1	2	3
prototype	Allows you to add properties and methods to the object	1	2	4
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)	1	2	3
SQRT2	Returns the square root of 2 (approx. 1.414)	1	2	3

Math Object

The Math object allows you to perform common mathematical tasks.

The Math object includes several mathematical values and functions. You do not need to define the Math object before using it.

Mathematical Values

JavaScript provides eight mathematical values (constants) that can be accessed from the Math object. These are: E, PI, square root of 2, square root of 1/2, natural log of 2, natural log of 10, base-2 log of E, and base-10 log of E.

You may reference these values from your JavaScript like this:

```
Math.E
Math.PI
Math.SQRT2
Math.SQRT1_2
Math.LN2
Math.LN10
Math.LOG2E
Math.LOG10E
```

Mathematical Methods

In addition to the mathematical values that can be accessed from the Math object there are also several functions (methods) available.

Examples of functions (methods):

The following example uses the round() method of the Math object to round a number to the nearest integer:

```
document.write(Math.round(4.7));
```

The code above will result in the following output:

```
5
```

The following example uses the random() method of the Math object to return a random number between 0 and 1:

```
document.write(Math.random());
```

The code above can result in the following output:

```
0.5371330459290262
```

The following example uses the floor() and random() methods of the Math object to return a random number between 0 and 10:

```
document.write(Math.floor(Math.random()*11));
```

The code above can result in the following output:

```
2
```

JavaScript RegExp Object

The RegExp object is used to specify what to search for in a text

What is RegExp

RegExp, is short for regular expression.

When you search in a text, you can use a pattern to describe what you are searching for. **RegExp IS this pattern**.

A simple pattern can be a single character.

A more complicated pattern consists of more characters, and can be used for parsing, format checking, substitution and more.

You can specify where in the string to search, what type of characters to search for, and more.

Defining RegExp

The RegExp object is used to store the search pattern.

We define a RegExp object with the *new* keyword. The following code line defines a RegExp object called patt1 with the pattern "e":

```
var patt1=new RegExp("e");
```

When you use this RegExp object to search in a string, you will find the letter "e".

Methods of the RegExp Object

The RegExp Object has 3 methods: test(), exec(), and compile().

test()

The test() method searches a string for a specified value. Returns true or false

Example:

```
var patt1=new RegExp("e");
document.write(patt1.test("The best things in life are free"));
```

Since there is an "e" in the string, the output of the code above will be:

```
true
```

exec()

The exec() method searches a string for a specified value. Returns the text of the found value. If no match is found, it returns *null*

Example 1:

```
var patt1=new RegExp("e");
document.write(patt1.exec("The best things in life are free"));
```

Since there is an "e" in the string, the output of the code above will be:

```
e
```

Example 2:

You can add a second parameter to the RegExp object, to specify your search. For example; if you want to find all occurrences of a character, you can use the "g" parameter ("global").

When using the "g" parameter, the exec() method works like this:

- Finds the first occurence of "e", and stores its position
- If you run exec() again, it starts at the stored position, and finds the next occurence of "e", and stores its position

```
var patt1=new RegExp("e","g");
do
{
result=patt1.exec("The best things in life are free");
document.write(result);
}
while (result!=null)
```

Since there is six "e" letters in the string, the output of the code above will be:

```
eeeeeenull
```

---

compile()

The compile() method is used to change the RegExp.

compile() can change both the search pattern, and add or remove the second parameter.

Example:

```
var patt1=new RegExp("e");
document.write(patt1.test("The best things in life are free"));
patt1.compile("d");
document.write(patt1.test("The best things in life are free"));
```

Since there is an "e" in the string, but not a "d", the output of the code above will be:

```
truefalse
```

JavaScript RegExp Object Reference

RegExp Object Methods

FF: Firefox, **N**: Netscape, **IE**: Internet Explorer

Method	**Description**	**FF**	**N**	**IE**
compile	Change the regular expression (what to search for)	1	4	4
exec	Search a string for a specified value. Returns the found value and remembers the position	1	4	4
test	Search a string for a specified value. Returns true or false	1	4	4

String Object Methods that supports Regular Expressions

Method	**Description**	**FF**	**N**	**IE**
search	Search a string for a specified value. Returns the position of the value	1	4	4
match	Search a string for a specified value. Returns an array of the found value(s)	1	4	4
replace	Replace characters with other characters	1	4	4
split	Split a string into an array of strings	1	4	4

RegExp Modifiers

Property	**Description**	**FF**	**N**	**IE**
	Pattern flags			
i	Ignore the case of characters	1	4	4
G	Global search. Search all occurrences of the regular expression in a string	1	4	4
Gi	Global search, ignore case	1	4	4
M	Multiline mode. Matches occurrences over multiple lines	1	4	4
	Position Matching			
^	Get a match at the beginning of a string	1	4	4
$	Get a match at the end of a string	1	4	4
\b	Word boundary. Get a match at the beginning or end of a word in the string	1	4	4
\B	Non-word boundary. Get a match when it is not at the beginning or end of a word in the string	1	4	4
?=	A positive look ahead. Get a match if a string is followed by a specific string	1	4	4
?!	A negative look ahead. Get a match if a string is not followed by a specific string	1	4	4
	Literals			
\0	Find a NULL character	1	4	4
\n	Find a new line character	1	4	4
\f	Find a form feed character	1	4	4
\r	Find a carriage return character	1	4	4
\t	Find a tab character	1	4	4
\v	Find a vertical tab character	1	4	4
\xxx	Find the ASCII character expressed by the octal number xxx	1	4	4

\xdd	Find the ASCII character expressed by the hex number dd	1	4	4
\uxxxx	Find the ASCII character expressed by the UNICODE xxxx	1	4	4
Character Classes				
[xyz]	Find any character in the specified character set	1	4	4
[^xyz]	Find any character not in the specified character set	1	4	4
. (dot)	Find any character except newline or line terminator	1	4	4
\w	Find any alphanumeric character including the underscore	1	4	4
\W	Find any non-word character	1	4	4
\d	Find any single digit	1	4	4
\D	Find any non-digit	1	4	4
\s	Find any single space character	1	4	4
\S	Find any single non-space character	1	4	4
Repetition				
{x}	Finds the exact (x) number of the regular expression grouped together	1	4	4
{x,}	Finds the exact (x) or more number of the regular expression grouped together	1	4	4
{x,y}	Finds between x and y number of the regular expression grouped together	1	4	4
?	Finds zero or one occurrence of the regular expression	1	4	4
*	Finds zero or more occurrences of the regular expression	1	4	4
+	Finds one or more occurrences of the regular expression	1	4	4
Grouping				
( )	Finds the group of characters inside the parentheses and stores the matched string	1	4	4
(?:)	Finds the group of characters inside the parentheses but Dutts not store the matched string	1	4	4
|	Combines clauses into one regular expression and then matches any of the individual clauses. Similar to "OR" statement	1	4	4
Back references				
( )\n	Back reference. Uses the stored matched string. i.e. from the ( ) modifier	1	4	4

RegExp Object Properties

Property	Description	FF	N	IE
Global	Specifies if the "g" modifier is set	1	4	4
ignoreCase	Specifies if the "i" modifier is set	1	4	4
Input	The string on which the pattern match is performed	1	4	4
lastIndex	An integer specifying the index at which to start the next match	1	4	4
lastMatch	The last matched characters	1	4	4
lastParen	The last matched parenthesized substring	1	4	4
leftContext	The substring in front of the characters most recently matched	1	4	4
Multiline	Specifies if the "m" modifier is set	1	4	4
Prototype	Allows you to add properties and methods to the object	1	4	4
rightContext	The substring after the characters most recently matched	1	4	4
Source	The text used for pattern matching	1	4	4

JavaScript HTML DOM Objects

In addition to the built-in JavaScript objects, you can also access and manipulate all of the HTML DOM objects with JavaScript.

More JavaScript Objects

Follow the links to learn more about the objects and their collections, properties, methods and events.

Object	Description
Window	The top level object in the JavaScript hierarchy. The Window object represents a browser window. A Window object is created automatically with every instance of a <body> or <frameset> tag
Navigator	Contains information about the client's browser
Screen	Contains information about the client's display screen
History	Contains the visited URLs in the browser window
Location	Contains information about the current URL

The HTML DOM

The HTML DOM is a W3C standard and it is an abbreviation for the Document Object Model for HTML.

The HTML DOM defines a standard set of objects for HTML, and a standard way to access and manipulate HTML documents.

All HTML elements, along with their containing text and attributes, can be accessed through the DOM. The contents can be modified or deleted, and new elements can be created.

The HTML DOM is platform and language independent. It can be used by any programming language like Java, JavaScript, and VBScript.

Object	Description
Document	Represents the entire HTML document and can be used to access all elements in a page
Anchor	Represents an <a> element
Area	Represents an <area> element inside an image-map
Base	Represents a <base> element
Body	Represents the <body> element
Button	Represents a <button> element
Event	Represents the state of an event
Form	Represents a <form> element
Frame	Represents a <frame> element
Frameset	Represents a <frameset> element
Iframe	Represents an <iframe> element
Image	Represents an <img> element
Input button	Represents a button in an HTML form
Input checkbox	Represents a checkbox in an HTML form
Input file	Represents a fileupload in an HTML form

Input hidden	Represents a hidden field in an HTML form
Input password	Represents a password field in an HTML form
Input radio	Represents a radio button in an HTML form
Input reset	Represents a reset button in an HTML form
Input submit	Represents a submit button in an HTML form
Input text	Represents a text-input field in an HTML form
Link	Represents a <link> element
Meta	Represents a <meta> element
Option	Represents an <option> element
Select	Represents a selection list in an HTML form
Style	Represents an individual style statement
Table	Represents a <table> element
TableData	Represents a <td> element
TableRow	Represents a <tr> element
Textarea	Represents a <textarea> element

JavaScript Browser Detection

The JavaScript Navigator object contains information about the visitor's browser.

Examples

Detect the visitor's browser and browser version

```
<html>
<body>
<script type="text/javascript">
var browser=navigator.appName;
var b_version=navigator.appVersion;
var version=parseFloat(b_version);
document.write("Browser name: "+ browser);
document.write("<br />");
document.write("Browser version: "+ version);
</script>
</body>
</html>
```

More details about the visitor's browser

```
<html>
<body>
<script type="text/javascript">
document.write("<p>Browser: ");
document.write(navigator.appName + "</p>");

document.write("<p>Browserversion: ");
document.write(navigator.appVersion + "</p>");

document.write("<p>Code: ");
document.write(navigator.appCodeName + "</p>");

document.write("<p>Platform: ");
document.write(navigator.platform + "</p>");

document.write("<p>Cookies enabled: ");
document.write(navigator.cookieEnabled + "</p>");

document.write("<p>Browser's user agent header: ");
document.write(navigator.userAgent + "</p>");
</script>
</body>
</html>
```

All details about the visitor's browser

```
<html>
<body>
<script type="text/javascript">
var x = navigator;
document.write("CodeName=" + x.appCodeName);
document.write("<br />");
document.write("MinorVersion=" + x.appMinorVersion);
document.write("<br />");
document.write("Name=" + x.appName);
document.write("<br />");
document.write("Version=" + x.appVersion);
document.write("<br />");
document.write("CookieEnabled=" + x.cookieEnabled);
document.write("<br />");
document.write("CPUClass=" + x.cpuClass);
document.write("<br />");
document.write("OnLine=" + x.onLine);
document.write("<br />");
document.write("Platform=" + x.platform);
document.write("<br />");
document.write("UA=" + x.userAgent);
document.write("<br />");
document.write("BrowserLanguage=" + x.browserLanguage);
document.write("<br />");
document.write("SystemLanguage=" + x.systemLanguage);
document.write("<br />");
document.write("UserLanguage=" + x.userLanguage);
</script>
</body>
</html>
```

Alert user, depending on browser

```
<html>
<head>
<script type="text/javascript">
function detectBrowser()
{
var browser=navigator.appName;
var b_version=navigator.appVersion;
var version=parseFloat(b_version);
if ((browser=="Netscape"||browser=="Microsoft Internet Explorer") && (version>=4))
  {
  alert("Your browser is good enough!");
  }
else
  {
  alert("It's time to upgrade your browser!");
  }
}
</script>
</head>
<body onload="detectBrowser()">
</body>
</html>
```

Browser Detection

Almost everything in this tutorial works on all JavaScript-enabled browsers. However, there are some things that just don't work on certain browsers - specially on older browsers.

So, sometimes it can be very useful to detect the visitor's browser type and version, and then serve up the appropriate information.

The best way to do this is to make your web pages smart enough to look one way to some browsers and another way to other browsers.

JavaScript includes an object called the Navigator object, that can be used for this purpose.

The Navigator object contains information about the visitor's browser name, browser version, and more.

The Navigator Object

The JavaScript Navigator object contains all information about the visitor's browser. We are going to look at two properties of the Navigator object:

- appName - holds the name of the browser
- appVersion - holds, among other things, the version of the browser

Example

```
<html>
<body>
<script type="text/javascript">
var browser=navigator.appName;
var b_version=navigator.appVersion;
var version=parseFloat(b_version);
document.write("Browser name: "+ browser);
document.write("<br />");
document.write("Browser version: "+ version);
</script>
</body>
</html>
```

The variable browser in the example above holds the name of the browser, i.e. "Netscape" or "Microsoft Internet Explorer".

The appVersion property in the example above returns a string that contains much more information than just the version number, but for now we are only interested in the version number. To pull the version number out of the string we are using a function called parseFloat(), which pulls the first thing that looks like a decimal number out of a string and returns it.

IMPORTANT! The version number is WRONG in IE 5.0 or later! Microsoft starts the appVersion string with the number 4.0. in IE 5.0 and IE 6.0!!! Why did they do that??? However, JavaScript is the same in IE6, IE5 and IE4, so for most scripts it is ok.

Example

The script below displays a different alert, depending on the visitor's browser:

```
<html>
<head>
<script type="text/javascript">
function detectBrowser()
{
var browser=navigator.appName;
var b_version=navigator.appVersion;
var version=parseFloat(b_version);
if ((browser=="Netscape"||browser=="Microsoft Internet Explorer")
&& (version>=4))
{
alert("Your browser is good enough!");
}
else
{
alert("It's time to upgrade your browser!");
}
}
</script>
</head>
<body onload="detectBrowser()">
</body>
</html>
```

JavaScript Cookies

A cookie is often used to identify a user.

Examples

Create a welcome cookie

```
<html>
<head>
<script type="text/javascript">
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=");
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1 ;
    c_end=document.cookie.indexOf(";",c_start);
    if (c_end==-1) c_end=document.cookie.length
    return unescape(document.cookie.substring(c_start,c_end));
    }
  }
return ""
}

function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+((expiredays==null) ? "" : ";
expires="+exdate.toGMTString());
}

function checkCookie()
```

```
{
username=getCookie('username');
if (username!=null && username!="")
  {
  alert('Welcome again '+username+'!');
  }
else
  {
  username=prompt('Please enter your name:',"");
  if (username!=null && username!="")
    {
    setCookie('username',username,365);
    }
  }
}
</script>
</head>
<body onLoad="checkCookie()">
</body>
</html>
```

What is a Cookie?

A cookie is a variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values.

Examples of cookies:

- Name cookie - The first time a visitor arrives to your web page, he or she must fill in her/his name. The name is then stored in a cookie. Next time the visitor arrives at your page, he or she could get a welcome message like "Welcome Madhavendra Dutt!" The name is retrieved from the stored cookie
- Password cookie - The first time a visitor arrives to your web page, he or she must fill in a password. The password is then stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie
- Date cookie - The first time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he or she could get a message like "Your last visit was on Tuesday August 11, 2005!" The date is retrieved from the stored cookie

---

Create and Store a Cookie

In this example we will create a cookie that stores the name of a visitor. The first time a visitor arrives to the web page, he or she will be asked to fill in her/his name. The name is then stored in a cookie. The next time the visitor arrives at the same page, he or she will get welcome message.

First, we create a function that stores the name of the visitor in a cookie variable:

```
function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toGMTString());
}
```

The parameters of the function above hold the name of the cookie, the value of the cookie, and the number of days until the cookie expires.

In the function above we first convert the number of days to a valid date, then we add the number of days until the cookie should expire. After that we store the cookie name, cookie value and the expiration date in the document.cookie object.

Then, we create another function that checks if the cookie has been set:

```
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=");
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1;
    c_end=document.cookie.indexOf(";",c_start);
    if (c_end==-1) c_end=document.cookie.length;
    return unescape(document.cookie.substring(c_start,c_end));
    }
  }
return "";
}
```

The function above first checks if a cookie is stored at all in the document.cookie object. If the document.cookie object holds some cookies, then check to see if our specific cookie is stored. If our cookie is found, then return the value, if not - return an empty string.

Last, we create the function that displays a welcome message if the cookie is set, and if the cookie is not set it will display a prompt box, asking for the name of the user:

```
function checkCookie()
{
username=getCookie('username');
if (username!=null && username!="")
{
alert('Welcome again '+username+'!');
}
else
{
username=prompt('Please enter your name:',"");
if (username!=null && username!="")
{
setCookie('username',username,365);
}
}
}
```

All together now:

```
<html>
<head>
<script type="text/javascript">
function getCookie(c_name)
{
if (document.cookie.length>0)
```

```
  {
  c_start=document.cookie.indexOf(c_name + "=");
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1;
    c_end=document.cookie.indexOf(";",c_start);
    if (c_end==-1) c_end=document.cookie.length;
    return unescape(document.cookie.substring(c_start,c_end));
    }
  }
return "";
}
function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toGMTString());
}
function checkCookie()
{
username=getCookie('username');
if (username!=null && username!="")
  {
  alert('Welcome again '+username+'!');
  }
  else
  {
  username=prompt('Please enter your name:',"");
  if (username!=null && username!="")
    {
    setCookie('username',username,365);
    }
  }
}
</script>
</head>
<body onLoad="checkCookie()">
</body>
</html>
```

The example above runs the checkCookie() function when the page loads.

JavaScript Form Validation

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

JavaScript Form Validation

JavaScript can be used to validate input 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 required field has been left empty. If the required field is blank, an alert box alerts a message and the function returns false. If a value is entered, the function returns true (means that data is OK):

```
function validate_required(field,alerttxt)
{
with (field)
{
  if (value==null||value=="")
  {
  alert(alerttxt);return false;
  }
  else
  {
  return true;
  }
}
}
```

The entire script, with the HTML form could look something like this:

```
<html>
<head>
<script type="text/javascript">
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value=="")
  {alert(alerttxt);return false;}
else {return true}
}
}
function validate_form(thisform)
{
with (thisform)
{
if (validate_required(email,"Email must be filled out!")==false)
  {email.focus();return false;}
}
}
</script>
</head>
```

```
<body>
<form action="submitpage.htm"
onsubmit="return validate_form(this)"
method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

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 at least an @ sign and a dot (.). Also, the @ must not be the first character of the email address, and the last dot must at least be one character after the @ sign:

```
function validate_email(field,alerttxt)
{
with (field)
{
apos=value.indexOf("@");
dotpos=value.lastIndexOf(".");
if (apos<1||dotpos-apos<2)
  {alert(alerttxt);return false;}
else {return true;}
}
}
```

The entire script, with the HTML form could look something like this:

```
<html>
<head>
<script type="text/javascript">
function validate_email(field,alerttxt)
{
with (field)
{
apos=value.indexOf("@");
dotpos=value.lastIndexOf(".");
if (apos<1||dotpos-apos<2)
  {alert(alerttxt);return false;}
else {return true;}
}
}
function validate_form(thisform)
{
with (thisform)
{
if (validate_email(email,"Not a valid e-mail address!")==false)
  {email.focus();return false;}
}
}
</script>
</head>
```

```
<body>
<form action="submitpage.htm"
onsubmit="return validate_form(this);"
method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

JavaScript Animation

With JavaScript we can create animated images.

Examples

Button animation

```
<html>
<head>
<script type="text/javascript">
function mouseOver()
{
document.b1.src ="b_blue.gif";
}
function mouseOut()
{
document.b1.src ="b_pink.gif";
}
</script>
</head>
<body>
<a href="http://www.MadhavendraDutt.com" target="_blank">
<img border="0" alt="Visit MadhavendraDutt!" src="b_pink.gif" name="b1" width="26"
height="26" onmouseover="mouseOver()" onmouseout="mouseOut()" /></a>
</body>
</html>
```

JavaScript Animation

It is possible to use JavaScript to create animated images.

The trick is to let a JavaScript change between different images on different events.

In the following example we will add an image that should act as a link button on a web page. We will then add an onMouseOver event and an onMouseOut event that will run two JavaScript functions that will change between the images.

The HTML Code

The HTML code looks like this:

```
<a href="http://www.MadhavendraDutt.com" target="_blank">
<img border="0" alt="Visit MadhavendraDutt!"
src="b_pink.gif" name="b1"
onmouseOver="mouseOver()"
onmouseOut="mouseOut()" />
</a>
```

Note that we have given the image a name to make it possible for JavaScript to address it later.

The onMouseOver event tells the browser that once a mouse is rolled over the image, the browser should execute a function that will replace the image with another image.

The onMouseOut event tells the browser that once a mouse is rolled away from the image, another JavaScript function should be executed. This function will insert the original image again.

The JavaScript Code

The changing between the images is done with the following JavaScript:

```
<script type="text/javascript">
function mouseOver()
{
document.b1.src ="b_blue.gif";
}
function mouseOut()
{
document.b1.src ="b_pink.gif";
}
</script>
```

The function mouseOver() causes the image to shift to "b_blue.gif".

The function mouseOut() causes the image to shift to "b_pink.gif".

The Entire Code

```
<html>
<head>
<script type="text/javascript">
function mouseOver()
{
document.b1.src ="b_blue.gif";
}
function mouseOut()
{
document.b1.src ="b_pink.gif";
}
</script>
```

```
</head>
<body>
<a href="http://www.MadhavendraDutt.com" target="_blank">
<img border="0" alt="Visit MadhavendraDutt!"
src="b_pink.gif" name="b1"
onmouseOver="mouseOver()"
onmouseOut="mouseOut()" />
</a>
</body>
</html>
```

JavaScript Image Maps

An image-map is an image with clickable regions.

Examples

Simple HTML Image map

```
<html>
<body>
<img src ="planets.gif" width ="145" height ="126" alt="Planets" usemap="#planetmap" />
<map id ="planetmap" name="planetmap">
<area shape ="rect" coords ="0,0,82,126"
href ="sun.htm" target ="_blank" alt="Sun" />

<area shape ="circle" coords ="90,58,3"
href ="mercur.htm" target ="_blank" alt="Mercury" />

<area shape ="circle" coords ="124,58,8"
href ="venus.htm" target ="_blank" alt="Venus" />
</map>
</body>
</html>
```

Image map with added JavaScript

```
<html>
<head>
<script type="text/javascript">
function writeText(txt)
{
document.getElementById("desc").innerHTML=txt;
}
</script>
</head>
<body>
<img src ="planets.gif" width ="145" height ="126" alt="Planets" usemap="#planetmap" />

<map id ="planetmap" name="planetmap">
<area shape ="rect" coords ="0,0,82,126"
```

```
onMouseOver="writeText('The Sun and the gas giant planets like Jupiter are by far the largest objects in our Solar System.')"
href ="sun.htm" target ="_blank" alt="Sun" />

<area shape ="circle" coords ="90,58,3"
onMouseOver="writeText('The planet Mercury is very difficult to study from the Earth because it is always so close to the Sun.')"
href ="mercur.htm" target ="_blank" alt="Mercury" />

<area shape ="circle" coords ="124,58,8"
onMouseOver="writeText('Until the 1960s, Venus was often considered a  twin sister to the Earth because Venus is the nearest planet to us, and because the two planets seem to share many characteristics.')"
href ="venus.htm" target ="_blank" alt="Venus" />
</map>
<p id="desc"></p>
</body>
</html>
```

HTML Image Maps

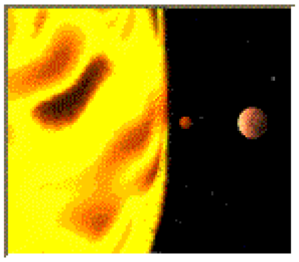
From our HTML tutorial we have learned that an image-map is an image with clickable regions. Normally, each region has an associated hyperlink. Clicking on one of the regions takes you to the associated link.

Example

The example below demonstrates how to create an HTML image map, with clickable regions. Each of the regions is a hyperlink:

```
<img src ="planets.gif"
width ="145" height ="126"
alt="Planets"
usemap ="#planetmap" />
<map id ="planetmap"
name="planetmap">
<area shape ="rect" coords ="0,0,82,126"
  href ="sun.htm" target ="_blank"
  alt="Sun" />
<area shape ="circle" coords ="90,58,3"
  href ="mercur.htm" target ="_blank"
  alt="Mercury" />
<area shape ="circle" coords ="124,58,8"
  href ="venus.htm" target ="_blank"
  alt="Venus" />
</map>
```

Result



Adding some JavaScript

We can add events (that can call a JavaScript) to the <area> tags inside the image map. The <area> tag supports the onClick, onDblClick, onMouseDown, onMouseUp, onMouseOver, onMouseMove, onMouseOut, onKeyPress, onKeyDown, onKeyUp, onFocus, and onBlur events.

Here's the above example, with some JavaScript added:

```
<html>
<head>
<script type="text/javascript">
function writeText(txt)
{
document.getElementById("desc").innerHTML=txt;
}
</script>
</head>
<body>
<img src="planets.gif" width="145" height="126"
alt="Planets" usemap="#planetmap" />

<map id ="planetmap" name="planetmap">
<area shape ="rect" coords ="0,0,82,126"
onMouseOver="writeText('The Sun and the gas giant
planets like Jupiter are by far the largest objects
in our Solar System.')"
href ="sun.htm" target ="_blank" alt="Sun" />

<area shape ="circle" coords ="90,58,3"
onMouseOver="writeText('The planet Mercury is very
difficult to study from the Earth because it is
always so close to the Sun.')"
href ="mercur.htm" target ="_blank" alt="Mercury" />

<area shape ="circle" coords ="124,58,8"
onMouseOver="writeText('Until the 1960s, Venus was
often considered a twin sister to the Earth because
Venus is the nearest planet to us, and because the
two planets seem to share many characteristics.')"
href ="venus.htm" target ="_blank" alt="Venus" />
</map>

<p id="desc"></p>

</body>
</html>
```

JavaScript Timing Events

With JavaScript, it is possible to execute some code NOT immediately after a function is called, but after a specified time interval. This is called timing events.

Examples

Simple timing

```
<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>
<p>Click on the button above. An alert box will be displayed after 5 seconds.</p>
</body>
</html>
```

Another simple timing

```
<html>
<head>
<script type="text/javascript">
function timedText()
{
var t1=setTimeout("document.getElementById('txt').value='2 seconds!'",2000);
var t2=setTimeout("document.getElementById('txt').value='4 seconds!'",4000);
var t3=setTimeout("document.getElementById('txt').value='6 seconds!'",6000);
}
</script>
</head>
<body>
<form>
<input type="button" value="Display timed text!" onClick="timedText()">
<input type="text" id="txt">
</form>
<p>Click on the button above. The input field will tell you when two, four, and six seconds have
passed.</p>
</body>
</html>
```

Timing event in an infinite loop

```
<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>
<p>Click on the button above. The input field will count for ever, starting at 0.</p>
</body>
</html>
```

Timing event in an infinite loop - with a Stop button

```
<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>
<p>
Click on the "Start count!" button above to start the timer. The input field will count forever,
starting at 0. Click on the "Stop count!" button to stop the counting.
</p>
</body>
</html>
```

A clock created with a timing event

```
<html>
<head>
<script type="text/javascript">
function startTime()
{
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
// add a zero in front of numbers<10
m=checkTime(m);
s=checkTime(s);
document.getElementById('txt').innerHTML=h+":"+m+":"+s;
t=setTimeout('startTime()',500);
}
```

```
function checkTime(i)
{
if (i<10)
  {
  i="0" + i;
  }
return i;
}
</script>
</head>
<body onload="startTime()">
<div id="txt"></div>
</body>
</html>
```

JavaScript Timing Events

With JavaScript, it is possible to execute some code NOT immediately after a function is called, but 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.

setTimeout()

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.

```
<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:

```
<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>
```

---

clearTimeout()

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:

```
<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 Create Your Own Objects

Objects are useful to organize information.

Examples

Create a direct instance of an object

```
<html>
<body>
<script type="text/javascript">
personObj=new Object();
personObj.firstname="Madhavendra";
personObj.lastname="Dutt";
personObj.age=50;
personObj.eyecolor="blue";
document.write(personObj.firstname + " is " + personObj.age + " years old.");
</script>
</body>
</html>
```

Create a template for an object

```
<html>
<body>
<script type="text/javascript">
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
}
myFather=new person("Madhavendra","Dutt",50,"blue");
document.write(myFather.firstname + " is " + myFather.age + " years old.");
</script>
</body>
</html>
```

JavaScript Objects

Earlier in this tutorial we have seen that JavaScript has several built-in objects, like String, Date, Array, and more. In addition to these built-in objects, you can also create your own.

An object is just a special kind of data, with a collection of properties and methods.

Let's illustrate with an example: A person is an object. Properties are the values associated with the object. The persons' properties include name, height, weight, age, skin tone, eye color, etc. All persons have these properties, but the values of those properties will differ from person to person. Objects also have methods. Methods are the actions that can be performed on objects. The persons' methods could be eat(), sleep(), work(), play(), etc.

Properties

The syntax for accessing a property of an object is:

```
objName.propName
```

You can add properties to an object by simply giving it a value. Assume that the personObj already exists - you can give it properties named firstname, lastname, age, and eyecolor as follows:

```
personObj.firstname="Madhavendra";
personObj.lastname="Dutt";
personObj.age=30;
personObj.eyecolor="blue";
document.write(personObj.firstname);
```

The code above will generate the following output:

```
Madhavendra
```

Methods

An object can also contain methods.

You can call a method with the following syntax:

```
objName.methodName()
```

Note: Parameters required for the method can be passed between the parentheses.

To call a method called sleep() for the personObj:

```
personObj.sleep();
```

Creating Your Own Objects

There are different ways to create a new object:

1. Create a direct instance of an object

The following code creates an instance of an object and adds four properties to it:

```
personObj=new Object();
personObj.firstname="Madhavendra";
personObj.lastname="Dutt";
personObj.age=50;
personObj.eyecolor="blue";
```

Adding a method to the personObj is also simple. The following code adds a method called eat() to the personObj:

```
personObj.eat=eat;
```

2. Create a template of an object

The template defines the structure of an object:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
}
```

Notice that the template is just a function. Inside the function you need to assign things to this.propertyName. The reason for all the "this" stuff is that you're going to have more than one person at a time (which person you're dealing with must be clear). That's what "this" is: the instance of the object at hand.

Once you have the template, you can create new instances of the object, like this:

```
myFather=new person("Madhavendra","Dutt",50,"blue");
myMother=new person("Sally","Rally",48,"green");
```

You can also add some methods to the person object. This is also done inside the template:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
this.newlastname=newlastname;
}
```

Note that methods are just functions attached to objects. Then we will have to write the newlastname() function:

```
function newlastname(new_lastname)
{
this.lastname=new_lastname;
}
```

The newlastname() function defines the person's new last name and assigns that to the person. JavaScript knows which person you're talking about by using "this.". So, now you can write: myMother.newlastname("Dutt").

You Have Learned JavaScript, Now What?

JavaScript Summary

This tutorial has taught you how to add JavaScript to your HTML pages, to make your web site more dynamic and interactive.

You have learned how to create responses to events, validate forms and how to make different scripts run in response to different scenarios.

You have also learned how to create and use objects, and how to use JavaScript's built-in objects.

Now You Know JavaScript, What's Next?

The next step is to learn about the HTML DOM and DHTML.

If you want to learn about server-side scripting, the next step is to learn ASP.

HTML DOM

The HTML DOM defines a standard way for accessing and manipulating HTML documents.

The HTML DOM is platform and language independent and can be used by any programming language like Java, JavaScript, and VBScript.

DHTML

DHTML is a combination of HTML, CSS, and JavaScript. DHTML is used to create dynamic and interactive Web sites.

W3C once said: "Dynamic HTML is a term used by some vendors to describe the combination of HTML, style sheets and scripts that allows documents to be animated."

ASP

While scripts in an HTML file are executed on the client (in the browser), scripts in an ASP file are executed on the server.

With ASP you can dynamically edit, change or add any content of a Web page, respond to data submitted from HTML forms, access any data or databases and return the results to a browser, customize a Web page to make it more useful for individual users.

Since ASP files are returned as plain HTML, they can be viewed in any browser.