Java Script
Study Material

JavaScript

Introduction

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

What You Should Already Know

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

- HTML / XHTML

If you want to study these subjects first, find the tutorials on our Home page.

What is JavaScript?

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

Are Java and JavaScript the same?

NO!

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

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

What can 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.

ECMAScript is developed and maintained by the ECMA organization.

ECMA-262 is the official JavaScript standard.

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.

Put a JavaScript into an HTML page

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

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

The example below shows how to add HTML tags to the JavaScript:

```
<html>
<body>
<script type="text/javascript">
document.write("<h1>Hello World!</h1>");
</script>
</body>
</html>
```

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 document.write command 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. Try it yourself

---

How 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 should 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, like this:

```
<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 can be put in the body and in the head sections of an HTML page.

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, or at a later event, such as when a user clicks a button. When this is the case we put the script inside a function, you will learn about functions in a later chapter.

Scripts in <head>

Scripts to be executed when they are called, or when an event is triggered, are placed in functions.

Put your functions in the head section, this way they are all in one place, and they do not interfere with page content.

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

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

Scripts in <body>

If you don't want your script to be placed inside a function, or if your script should write page content, it should be placed in the body section.

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

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

</html>
```

Scripts in <head> and <body>

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

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

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

</html>
```

Using an External JavaScript

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

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

Note: The external script cannot contain the <script></script> tags!

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

```
<html>
<head>
<script type="text/javascript" src="xxx.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 is Case Sensitive

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

JavaScript Statements

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

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

```
document.write("Hello Dolly");
```

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

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

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

JavaScript Code

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

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

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

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

JavaScript Blocks

JavaScript statements can be grouped together in blocks.

Blocks start with a left curly bracket {, and 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 heading and two paragraphs to a web page:

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

The example above is not very useful. It just demonstrates the use of a block. Normally a block is used to group statements

together in a function or in a condition (where a group of statements should be executed if a condition is met).

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

JavaScript Comments

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

JavaScript Comments

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

Single line comments start with //.

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

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

JavaScript Multi-Line Comments

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

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

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

Using Comments to Prevent Execution

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

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

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

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

Using Comments at the End of a Line

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

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

JavaScript Variables

Variables are "containers" for storing information.

Do You Remember Algebra From School?

Do you remember algebra from school? x=5, y=6, z=x+y

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

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

JavaScript Variables

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

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

Rules for JavaScript variable names:

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

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

Example

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

This example will show you how

Declaring (Creating) JavaScript Variables

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

You can declare JavaScript variables with the var keyword:

```
var x;
varcarname;
```

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

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

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

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

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

Assigning Values to Undeclared JavaScript Variables

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

These statements:

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

have the same effect as:

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

Redeclaring JavaScript Variables

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

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

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

JavaScript Arithmetic

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

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

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

JavaScript Operators

= is used to assign values.

+ 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

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

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

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

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

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

JavaScript Comparison and Logical Operators

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

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x=5, the table below explains the comparison operators:

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

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age<18) document.write("Too young");
```

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

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that x=6 and y=3, the table below explains the logical operators:

Operator	Description	Example
&&	And	(x < 10 && y > 1) is true
||	Or	(x==5 || y==5) is false
!	Not	!(x==y) is true

Conditional Operator

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

Syntax

```
variablename=(condition)?value1:value2
```

Example

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

If Statement

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

Syntax

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

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

```
<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10

var d=new Date();
var time=d.getHours();

if (time<10)
  {
  document.write("<b>Good morning</b>");
  }
</script>
```

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

If...else Statement

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

Syntax

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

```
<script type="text/javascript">
//If the time is less than 10, you will get a "Good morning"
greeting.
//Otherwise you will get a "Good day" greeting.

var d = new Date();
var time = d.getHours();

if (time < 10)
  {
  document.write("Good morning!");
  }
else
{
  document.write("Good day!");
  }
</script>
```

If...else if...else Statement

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

Syntax

```
if (condition1)
  {
  code to be executed if condition1 is true
  }
else if (condition2)
  {
  code to be executed if condition2 is true
  }
else
  {
  code to be executed if condition1 and condition2 are not true
  }
```

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

JavaScript Switch Statement

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

The JavaScript Switch Statement

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

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
```

```
  code to be executed if n is different from case 1 and 2
}
```

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

```
<script type="text/javascript">
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.

var d=new Date();
theDay=d.getDay();
switch (theDay)
{
case 5:
  document.write("Finally Friday");
  break;
case 6:
  document.write("Super Saturday");
  break;
case 0:
  document.write("Sleepy Sunday");
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
```

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
alert("sometext");
```

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
alert("I am an alert box!");
}
</script>
</head>
```

```
<body>

<input type="button" onclick="show_alert()" value="Show alert box" />

</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

```
confirm("sometext");
```

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
  {
  alert("You pressed OK!");
  }
else
  {
  alert("You pressed Cancel!");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_confirm()" value="Show confirm box" />

</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

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

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
  {
  document.write("Hello " + name + "! How are you today?");
  }
}
</script>
</head>
<body>

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

</body>
</html>
```

Alert box with line breaks

JavaScript Functions

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

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 the function.

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

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

How to Define a Function

Syntax

```
functionfunctionname(var1,var2,...,varX)
{
```

```
some code
}
```

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

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

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

JavaScript Function Example

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!");
}
</script>
</head>

<body>
<form>
<input type="button" value="Click me!"
onclick="displaymessage()" />
</form>
</body>
</html>
```

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

You will learn more about JavaScript events in the JS Events chapter.

The return Statement

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

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

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

```
<html>
<head>
<script type="text/javascript">
function product(a,b)
{
return a*b;
```

```
}
</script>
</head>

<body>
<script type="text/javascript">
document.write(product(4,3));
</script>

</body>
</html>
```

The Lifetime of JavaScript Variables

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

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

JavaScript For Loop

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

JavaScript Loops

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

In JavaScript, there are two different kind of loops:

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

---

The for Loop

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

Syntax

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

Example

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

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

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

The while loop

The while loop will be explained in the next chapter.

Looping through HTML headings
Loop through the six different HTML headings.

JavaScript While Loop

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

The while Loop

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

Syntax

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

Note: The <= could be any comparing operator.

Example

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

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

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

The do...while Loop

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

Syntax

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

Example

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

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

JavaScript Break and Continue Statements

The break Statement

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

```
<html>
<body>
<script type="text/javascript">
vari=0;
for (i=0;i<=10;i++)
  {
  if (i==3)
    {
```

```
  break;
  }
 document.write("The number is " + i);
 document.write("<br />");
 }
</script>
</body>
</html>
```

The continue Statement

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

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

JavaScript For...In Statement

JavaScript For...In Statement

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

Syntax

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

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

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

Example

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

```
<html>
<body>

<script type="text/javascript">
var x;
varmycars = 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 a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

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

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

For a complete reference of the events recognized by JavaScript, go to our complete JavaScript reference.

---

onLoad and onUnload

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

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

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

---

onFocus, onBlur and onChange

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

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

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

---

onSubmit

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

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

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

---

onMouseOver and onMouseOut

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

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

```
<a href="http://www.w3schools.com" onmouseover="alert('An onMouseOver event');return false"><imgsrc="w3s.gif" alt="W3Schools" /></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 catch and handle JavaScript error messages, so you don't lose your audience.

The 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!

Examples

The example below is supposed to alert "Welcome guest!" when the button is clicked. However, there's a typo in the message() function. alert() is misspelled as adddlert(). A JavaScript error occurs. 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>
```

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 does 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.w3schools.com/";
    }
  }
}
</script>
</head>

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

</html>
```

The throw Statement

The throw statement can be used together with the try...catch statement, to create an exception for the error. Learn about the throw statement in the next chapter.

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

The example below determines the value of a variable called x. If the value of x is higher than 10, lower than 0, or not a

number, 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";
    }
  else if(isNaN(x))
    {
    throw "Err3";
    }
  }
catch(er)
  {
  if(er=="Err1")
    {
    alert("Error! The value is too high");
    }
  if(er=="Err2")
    {
    alert("Error! The value is too low");
    }
  if(er=="Err3")
    {
    alert("Error! The value is not a number");
    }
  }
</script>
</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);

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

Here is another example:

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

The example above will produce the following output:

You & I are singing!

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

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

JavaScript 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="Hege";
name = "Hege";

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">
varstr="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.

Return the length of a string
How to return the length of a string.

Style strings
How to style strings.

The toLowerCase() and toUpperCase() methods
How to convert a string to lowercase or uppercase letters.

The match() method
How to search for a specified value within a string.

Replace characters in a string - replace()
How to replace a specified value with another value in a string.

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

Complete String Object Reference

For a complete reference of all the properties and methods that can be used with the String object, go to our complete String object reference.

The reference contains a brief description and examples of use for each property and method!

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.

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

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

setFullYear()
How to use setFullYear() to set a specific date.

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

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

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

Complete Date Object Reference

For a complete reference of all the properties and methods that can be used with the Date object, go to our complete Date object reference.

The reference contains a brief description and examples of use for each property and method!

Create a Date Object

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

Date objects are created with the Date() constructor.

There are four ways of instantiating a date:

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

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

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

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

Some examples of instantiating a date:

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

Set Dates

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

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

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

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

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

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

Compare Two Dates

The Date object is also used to compare two dates.

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

```
varmyDate=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 multiple values in a single variable.

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

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

Complete Array Object Reference

For a complete reference of all the properties and methods that can be used with the Array object, go to our complete Array object reference.

The reference contains a brief description and examples of use for each property and method!

What is an Array?

An array is a special variable, which can hold more than one value, at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

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

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array!

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

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

Create an Array

An array can be defined in three ways.

The following code creates an Array object called myCars:

1:

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

2:

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

3:

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

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

Access an Array

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

The following code line:

document.write(myCars[0]);

will result in the following output:

Saab

Modify Values in an Array

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

myCars[0]="Opel";

Now, the following code line:

document.write(myCars[0]);

will result in the following output:

Opel

Join two arrays - concat()

Join three arrays - concat()

Join all elements of an array into a string - join()

Remove the last element of an array - pop()

Add new elements to the end of an array - push()

Reverse the order of the elements in an array - reverse()

Remove the first element of an array - shift()

Select elements from an array - slice()

Sort an array (alphabetically and ascending) - sort()

Sort numbers (numerically and ascending) - sort()

Sort numbers (numerically and descending) - sort()

Add an element to position 2 in an array - splice()

Convert an array to a string - toString()

Add new elements to the beginning of an array - unshift()

JavaScript Boolean Object

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

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

Complete Boolean Object Reference

For a complete reference of all the properties and methods that can be used with the Boolean object, go to our complete Boolean object reference.

The reference contains a brief description and examples of use for each property and method!

Create a Boolean Object

The Boolean object represents two values: "true" or "false".

The following code creates a Boolean object called myBoolean:

varmyBoolean=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:

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

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

```
varmyBoolean=new Boolean(1);
varmyBoolean=new Boolean(true);
varmyBoolean=new Boolean("true");
varmyBoolean=new Boolean("false");
varmyBoolean=new Boolean("Richard");
```

JavaScript Math Object

The Math object allows you to perform mathematical tasks.

---

round()
How to use round().

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

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

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

Complete Math Object Reference

For a complete reference of all the properties and methods that can be used with the Math object, go to our complete Math object reference.

The reference contains a brief description and examples of use for each property and method!

Math Object

The Math object allows you to perform mathematical tasks.

The Math object includes several mathematical constants and methods.

Syntax for using properties/methods of Math:

```
varpi_value=Math.PI;
varsqrt_value=Math.sqrt(16);
```

Note: Math is not a constructor. All properties and methods of Math can be called by using Math as an object without creating it.

Mathematical Constants

JavaScript provides eight mathematical 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 constants 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 constants that can be accessed from the Math object there are also several methods available.

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:

0.13531459769275444

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:

3

JavaScript RegExp Object

RegExp, is short for regular expression.

Complete RegExp Object Reference

For a complete reference of all the properties and methods that can be used with the RegExp object, go to our complete RegExp object reference.

The reference contains a brief description and examples of use for each property and method!

What is RegExp?

A regular expression is an object that describes a pattern of characters.

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

A simple pattern can be one single character.

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

Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

Syntax

var txt=new RegExp(pattern,modifiers);

or more simply:

var txt=/pattern/modifiers;

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

RegExp Modifiers

Modifiers are used to perform case-insensitive and global searches.

The i modifier is used to perform case-insensitive matching.

The g modifier is used to perform a global match (find all matches rather than stopping after the first match).

Do a case-insensitive search for "w3schools" in a string:

```
varstr="Visit W3Schools";
var patt1=/w3schools/i;
```

The marked text below shows where the expression gets a match:

```
Visit W3Schools
```

Do a global search for "is":

```
varstr="Is this all there is?";
var patt1=/is/g;
```

The marked text below shows where the expression gets a match:

```
Is this all there is?
```

Do a global, case-insensitive search for "is":

```
varstr="Is this all there is?";
var patt1=/is/gi;
```

The marked text below shows where the expression gets a match:

```
Is this all there is?
```

test()

The test() method searches a string for a specified value, and returns true or false, depending on the result.

The following example searches a string for the character "e":

```
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, and returns the text of the found value. If no match is found, it returns null.

The following example searches a string for the character "e":

```
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
```

JavaScript Browser Detection

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

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 - especially on older browsers.

Sometimes it can be useful to detect the visitor's browser, and then serve 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.

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

Note: There is no public standard that applies to the navigator object, but all major browsers support it.

The Navigator Object

The Navigator object contains all information about the visitor's browser:

```
<html>
<body>

<script type="text/javascript">
document.write("Browser CodeName: " +
navigator.appCodeName);
document.write("<br /><br />");
document.write("Browser Name: " + navigator.appName);
document.write("<br /><br />");
document.write("Browser Version: " + navigator.appVersion);
document.write("<br /><br />");
document.write("Cookies Enabled: " +
navigator.cookieEnabled);
document.write("<br /><br />");
document.write("Platform: " + navigator.platform);
document.write("<br /><br />");
document.write("User-agent header: " + navigator.userAgent);
</script>

</body>
</html>
```

JavaScript Cookies

A cookie is often used to identify a user.

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 John Doe!" 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)
{
varexdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toUTCString());
}
```

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)
{
varexdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toUTCString());
}

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 Form Validation

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

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

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

Required Fields

The function below checks if a 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="submit.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="submit.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.

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.w3schools.com" target="_blank">
<img border="0" alt="Visit W3Schools!" src="b_pink.gif"
id="b1"
onmouseOver="mouseOver()" onmouseOut="mouseOut()"
/></a>
```

Note that we have given the image an id, to make it possible for a 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.getElementById("b1").src ="b_blue.gif";
}
function mouseOut()
{
document.getElementById("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.getElementById("b1").src ="b_blue.gif";
}
function mouseOut()
{
document.getElementById("b1").src ="b_pink.gif";
}
</script>
</head>

<body>
<a href="http://www.w3schools.com" target="_blank">
<img border="0" alt="Visit W3Schools!" src="b_pink.gif"
id="b1"
onmouseover="mouseOver()" onmouseout="mouseOut()"
/></a>
</body>
</html>
```

JavaScript Image Maps

An image-map is an image with clickable regions.

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. Look at our simple HTML image-map.

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 HTML image-map example, with some JavaScript added:

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

```
function writeText(txt)
{
document.getElementById("desc").innerHTML=txt;
}
</script>
</head>

<body>
<imgsrc="planets.gif" width="145" height="126"
alt="Planets" usemap="#planetmap" />

<map 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

JavaScript can be executed in time-intervals.

This is called timing events.

JavaScript Timing Events

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

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

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

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

The setTimeout() Method

Syntax

```
var t=setTimeout("javascriptstatement",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 a button is clicked, the input field will start to count (for ever), starting at 0.

Notice that we also have a function that checks if the timer is already running, to avoid creating additional timers, if the button is pressed more than once:

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

function timedCount()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("timedCount()",1000);
}

function doTimer()
{
if (!timer_is_on)
  {
  timer_is_on=1;
  timedCount();
  }
}
</script>
</head>

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

The clearTimeout() Method

Syntax

```
clearTimeout(setTimeout_variable)
```

Example

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

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

function timedCount()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("timedCount()",1000);
}

function doTimer()
{
```

```
if (!timer_is_on)
  {
  timer_is_on=1;
  timedCount();
  }
}

function stopCount()
{
clearTimeout(t);
timer_is_on=0;
}
</script>
</head>

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

Another simple timing

A clock created with a timing event

JavaScript Create Your Own Objects

Objects are useful to organize information.

Create a direct instance of an object

Create a template for an object

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="John";
personObj.lastname="Doe";
personObj.age=30;
personObj.eyecolor="blue";

document.write(personObj.firstname);
```

The code above will generate the following output:

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="John";
personObj.lastname="Doe";
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("John","Doe",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:

```
functionnewlastname(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("Doe").

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.

For more information on JavaScript, please look at our JavaScript examples and our JavaScript reference.

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.

If you want to learn more about the DOM, please visit our HTML DOM tutorial.

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."

If you want to learn more about DHTML, please visit our DHTML tutorial.

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.

If you want to learn more about ASP, please visit our ASP tutorial.

CSS - "Cascading Style Sheet"

- CSS is used to control the style of a web document in a simple and easy way.
- CSS is the acronym for "Cascading Style Sheet".
- CSS handles the look and feel part of a web page.
- Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, variations in display for different devices and screen sizes as well as a variety of other effects.

Advantages of CSS

- CSS saves time − You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
- Pages load faster − If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
- Easy maintenance − To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
- Superior styles to HTML − CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- Multiple Device Compatibility − Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
- Global web standards − Now HTML attributes are being deprecated and it is being recommended to use CSS. So its a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.
- Offline Browsing − CSS can store web applications locally with the help of an offline cache. Using of this, we can view offline websites. The cache also ensures faster loading and better overall performance of the website.
- Platform Independence − The Script offer consistent platform independence and can support latest browsers as well.

CSS - Syntax

```
selector
        {
                property: value;
        }
```

A style rule is made of three parts −

- Selector − A selector is an HTML tag at which a style will be applied. This could be any tag like <h1> or <table> etc.
- Property - A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be color, border etc.
- Value - Values are assigned to properties.

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External style sheet

- Internal style sheet
- Inline style

External Style Sheet

- With an external style sheet, you can change the look of an entire website by changing just one file!
- Each page must include a reference to the external style sheet file inside the <link> element.
- The <link> element goes inside the <head> section:
- body
- {
  background-color: lightblue;
  }

  h1
- {
  color: navy;
  margin-left: 20px;
  }

External CSS

- <html>
- <head>
- <link rel="stylesheet" type="text/css" href="mystyle.css">
- </head>
- <body>
- <h1>This is a heading</h1>
- This is a paragraph.
- </body>
- </html>

- An internal style sheet may be used if one single page has a unique style.
- Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

Internal Style Sheet

- An internal style sheet may be used if one single page has a unique style.
- Internal styles are defined within the <style> element, inside the <head> section of an HTML page:
- <html>
- <head>
- <style>
- body {
- background-color: linen;
- }

- h1 {
- color: maroon;
- margin-left: 40px;
- }
- </style>
- </head>
- <body>
- <h1>This is a heading</h1>
- This is a paragraph.
- </body>
- </html>

Inline Styles

- An inline style may be used to apply a unique style for a single element.
- To use inline styles, add the style attribute to the relevant element.
- The style attribute can contain any CSS property.
- The example below shows how to change the color and the left margin of a <h1> element:
- Example
- <h1 style="color:blue;margin-left:30px;">This is a heading</h1>

Positioning of elements

1. Absolute
2. Relative
3. Fixed
4. Static

Absolute: The surrounding elements would occupy the previous element positions.

<html>

<head>

<style type="text/css">

#sec

{

position: absolute;

top:100px;

left:120px;

}

#third

{

position: absolute;

top:200px;

left:140px;

}

</style>

</head>

<body bgcolor=yellow>

```
<p> FIRST PARAGRAPH</p>
<p id="sec"> SECOND PARAGRAPH</p>
<p id="third"> THIRD PARAGRAPH </p>
<p> FOURTH PARAGRAPH </p>
</body>
</html>
```

Relative: The space is preserved and occupied by the surrounding elements.

```
<html>
    <head>
    <style type="text/css">
    #sec
    {
    position: relative;
    top:100px;
    left:120px;
    }
    #third
    {
    position: relative;
    top:200px;
    left:140px;
    }
    </style>
    </head>
    <body bgcolor=yellow>
    <p> FIRST PARAGRAPH</p>
    <p id="sec"> SECOND PARAGRAPH</p>
    <p id="third"> THIRD PARAGRAPH </p>
    <p> FOURTH PARAGRAPH </p>
    </body>
    </html>
```

Stacking of elements : Displaying the elements in the order in which we type.

```
<html>
    <head>
    <style type="text/css">
    #fir
    {
```

```
background-color:red;
position:absolute;
top:20px;
left:20px;
width:100px;
height:100px;
border-style:solid;
}
#sec
{
background-color:green;
position:absolute;
top:30px;
left:60px;
width:100px;
height:100px;
border-style:solid;
}
#third
{
background-color:yellow;
position:absolute;
top:40px;
left:100px;
width:100px;
height:100px;
border-style:solid;
}
</style>
</head>
<body>
<p id="fir"> FIRST</p>
<p id="sec">SECOND</p>
<p id="third">THIRD</p>
</body>
</html>
```

Z- Index: To specify the order of the elements to display.

```
<html
>
    <head>
    <style
    type="text/css">
    #fir
    {
    background-
    color:red;
    position:absolut
    e;
    top:20px;
    left:20px;
    width:100px;
```

```
height:100px;
border-
style:solid;
z-index:10;
}
#sec
{
background-
color:green;
position:absolut
e;
top:30px;
left:60px;
width:100px;
height:100px;
border-
style:solid;
z-index:5;
}
#third
{
background-
color:yellow;
position:absolut
e;
top:40px;
left:100px;
width:100px;
height:100px;
border-
style:solid;
z-index:0;
}
</style>
</head>
<body>
<p id="fir">
FIRST</p>
<p
id="sec">SECO
ND</p>
<p
id="third">THI
RD</p>
</body>
</html>
```

Visibility of elements:

```
<html>
<head>
<style type="text/css">
#imgid
{
```

```
position:absolute;
top:60px;
left:100px;
visibility:visible;
}
</style>
<script
language="javascript">
function check()
{
k=document.getElementById
("imgid").style;
if(k.visibility=="visible")
k.visibility="hidden";
else
k.visibility="visible";
}
</script>
</head>
<body>
<form>
<imgsrc="ex.jpg" alt="Some
Image" id="imgid">
<input type=submit
onclick="check()">
</form>
</body>
</html>
```

Changing of colors:

```
<html
>
    <body bgcolor=yellow>
    <p id="p1">This is an example paragraph1.</p>
    <p id="p2">This is also an example paragraph2.</p>
    <button type="button" onclick="myFunction()">
    Set text color</button>
    <script>
    function myFunction()
    {
    document.getElementById("p1").style.color = "red";
    document.getElementById("p2").style.color = "green";
    }
    </script>
    </body>
    </html>
```

XHTML

History & Versions of HTML

- HTML 1.0: Few elements; does not support bgcolors, tables, frames.

- HTML 2.0: forms & tables are available.
- HTML 3.2: included CSS but not frames
- HTML 4.0 : frames introduced, presentation & content separation
- XML 1.0: use of own tags
- CSS: presentation styles
- HTML 4.1: increased elements & attributes
- XHTML : XML + HTML
- HTML 5: included audio, video, graphics and data storage.

What is SGML?

- This is a language for describing markup languages, particularly those used in electronic document exchange, document management, and document publishing.
- HTML is an example of a language defined in SGML.

What is HTML?

- This is Standard Generalized Markup Language (SGML) application.
- HTML is widely regarded as the standard publishing language of the World Wide Web.

What is XML?

- XML stands for Extensible Markup Language.
- XML is a markup language much like HTML and it was designed to describe data.
- XML tags are not predefined. You must define your own tags according to your needs.

What is XHTML?

- XHTML stands for Extensible Hyper Text Markup Language
- XHTML is almost identical to HTML
- XHTML is stricter than HTML
- XHTML is HTML defined as an XML application
- XHTML is supported by all major browsers
- XHTML stands for Extensible Hyper Text Markup Language.
- XHTML is almost identical to HTML 4.01 with only few differences.
- This is a cleaner and stricter version of HTML 4.01.
- XHTML was developed by World Wide Web Consortium (W3C) to help web developers make the transition from HTML to XML.
- XHTML gives you a more consistent, well-structured format so that your webpages can be easily processed by present and future web browsers.

- By combining the strengths of HTML and XML, XHTML was developed.

The Most Important Differences from HTML:

XHTML Elements

- XHTML elements must be properly nested
- XHTML elements must always be closed
- XHTML elements must be in lowercase
- XHTML documents must have one root element

XHTML Attributes

- Attribute names must be in lower case
- Attribute values must be quoted

Document Structure

- DOCTYPE
- The xmlns attribute
- <head> and <body>

Syntax

- <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

  <html xmlns="http://www.w3.org/1999/xhtml">

  <head>
  <title>Title of document</title>
  </head>

  <body>
  some content
  </body>

  </html>

Example

- <!DOCTYPE html>

<html>
<head>
<title>Title of the document</title>
</head>

<body>
The content of the document.....
</body>

</html>

- The <!DOCTYPE> is an instruction to the web browser about what version of HTML the page is written in.
- In HTML 4.01, the <!DOCTYPE> declaration refers to a DTD.
- The DTD specifies the rules for the markup language.

Common DOCTYPE Declarations

Strict

- This DTD contains all HTML elements and attributes, but does NOT INCLUDE presentational elements (like font).
- <!DOCTYPE ”strict.dtd”>

Transitional

- This DTD contains all HTML elements and attributes, INCLUDING presentational and deprecated elements (like font).
- <!DOCTYPE ”Transitional.dtd">

Frameset

- This DTD is equal to HTML 4.01 Transitional, but allows the use of frameset content.
- <!DOCTYPE ”frameset.dtd">