JavaScript is *THE* scripting language of the Web.

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

What is JavaScript?

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

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

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.

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

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

```
<html>
<body>

<script type="text/javascript">
document.write("Hello World!");  // document.write("<h1>Hello World!</h1>"); (to add HTML tags)
</script>

</body>
</html>
```

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.

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.
Java script command tag:

```
<!--
document.write("Hello World!");
//-->
```

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

JavaScripts in the head section will be executed when CALLED.

Sometimes we want to execute a script when a page loads, other times when a user triggers an event.

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

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

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

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

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

Using an External JavaScript

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

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

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

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

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

JavaScript is Case Sensitive

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.

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.

JavaScript Comments

```
// Write a heading
document.write("<h1>This is a heading</h1>");

/*
The code below will write
one heading and two paragraphs
*/
```

Variables:
letters are called **variables**, and variables can be used to hold values

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

```
<script type="text/javascript">
var firstname;           "declaring" variables
firstname="Hege";        assign values, assign a text value to a variable, use quotes around the value.
document.write(firstname);
document.write("<br />");
firstname="Tove";
document.write(firstname);
</script>
</body>
</html>
```

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

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 Operators
+,-,*,%,++,--,/,+=,-=,*=,%=,/=

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:

or insert a space into the expression:

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

```
<script type="text/javascript">
x=5+5;
```

```
document.write(x);  10
document.write("<br />");
x="5"+"5";         55
document.write(x);
document.write("<br />");
x=5+"5";        55
document.write(x);
document.write("<br />");
x="5"+5;  55
document.write(x);
document.write("<br />");
</script>
```

Comparison: if x=5

==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x==="5" is false

<,>,<=,>=!=

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

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

Logical op: &&, ||, !

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 ";
```

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

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

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

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

The JavaScript Switch Statement

```
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!");
}
```

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

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

Syntax

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

Example

```
<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" />
```

Confirm Box

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

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

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

Syntax

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

Example

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

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

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

Prompt Box

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

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

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

Syntax

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

Example

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

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

</body>
</html>
```

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

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

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

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

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

JavaScript Function Example

Example

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

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

Try it yourself »

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

The return Statement

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

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

```
document.write(product(4,3));
</script>
```

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 Loops

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

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

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

The do...while Loop

```
<script type="text/javascript">
var i=0;
do
```

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

The break Statement

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

```
<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>
```

The continue Statement

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

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:

```
<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>
```

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!

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"><img src="w3s.gif" alt="W3Schools" /></a>
```

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:

Example

```
<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>
```

Try it yourself »

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:

Example

```
<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>
```

Try it yourself »

The throw Statement

The throw statement can be used together with the try...catch statement, to create an exception for the error.

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:

Example

```
<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>
```

Insert Special Characters

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

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

document.write ("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

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.

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>
```

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());  //toLowerCase()
</script>
```

STRING:
Style a string:

```
<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>Fixed: " + txt.fixed() + "</p>");
document.write("<p>Strike: " + txt.strike() + "</p>");

document.write("<p>Fontcolor: " + txt.fontcolor("green") + "</p>");
document.write("<p>Fontsize: " + txt.fontsize(6) + "</p>");

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

document.write("<p>Link: " + txt.link("http://www.w3schools.com") + "</p>");

document.write("<p>Blink: " + txt.blink() + " (does not work in IE, Chrome, or Safari)</p>");

</script>
```

Match:

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

Replace char in a string:

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

var str="Visit Microsoft!";
document.write(str.replace("Microsoft","W3Schools"));     //Visit W3Schools!

</script>
```

The indexOf() method:

the position of the first found occurrence of a specified value in a string.

```
var str="Hello world!";
document.write(str.indexOf("Hello") + "<br />");  0
document.write(str.indexOf("WORLD") + "<br />"); -1
document.write(str.indexOf("world")); 6
```

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):

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

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

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

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

Compare Two Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 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");
  }
```

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

var d=new Date();
document.write(d);  // Tue Jan 12 23:54:29 UTC+0530 2010

</script>
```

Use getTime() to calculate the years since 1970.

```
<script type="text/javascript">
var d=new Date();
document.write(d.getTime() + " milliseconds since 1970/01/01");
</script>
```

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

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

</script>
```

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

```
<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 is " + weekday[d.getDay()]);
</script>
```

How to display a clock on your web page.

```
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>
```

Create an Array

An array can be defined in three ways.

The following code creates an Array object called myCars:

1:

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

2:

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

3:

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

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

Access an Array

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

The following code line:

```
document.write(myCars[0]);
```

will result in the following output:

```
Saab
```

Modify Values in an Array

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

```
myCars[0]="Opel";
```

Now, the following code line:

```
document.write(myCars[0]);
```

will result in the following output:

```
Opel
```

Assignment:
Concat(),join(),pop(),push(),reverse(),shift(),slice(),sort(),splice(),toString(),unShift() etc

Join two arrays - concat()var parents = ["Jani", "Tove"];

var children = ["Cecilie", "Lone"];

var family = parents.concat(children);

Join three arrays - concat() var family = parents.concat(brothers, children);

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

var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.write(fruits.join() + "<br />");

document.write(fruits.join("+") + "<br />");

document.write(fruits.join(" and "));

o/p: Banana,Orange,Apple,Mango
Banana+Orange+Apple+Mango
Banana and Orange and Apple and Mango

Remove the last element of an array - pop()

var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.write(fruits.pop() + "<br />");

```
document.write(fruits + "<br />");

document.write(fruits.pop() + "<br />");

document.write(fruits);
```

```
Mango
Banana,Orange,Apple
Apple
Banana,Orange
```

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

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.write(fruits.push("Kiwi") + "<br />");

document.write(fruits.push("Lemon","Pineapple") + "<br />");

document.write(fruits);
```

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

```
document.write(fruits.reverse());
```

[Remove the first element of an array - shift()]

[Select elements from an array - slice()]

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.write(fruits.slice(0,1) + "<br />");

document.write(fruits.slice(1) + "<br />");

document.write(fruits.slice(-2) + "<br />");

document.write(fruits);
```

o/p:

```
Banana
Orange,Apple,Mango
Apple,Mango
Banana,Orange,Apple,Mango
```

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

```
document.write(fruits.sort());
```

[Sort numbers (numerically and ascending) - sort()]

```
function sortNumber(a, b)

{

return a - b;

}

var n = ["10", "5", "40", "25", "100", "1"];

document.write(n.sort(sortNumber));
```

Sort numbers (numerically and descending) - sort()

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

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.write("Removed: " + fruits.splice(2,0,"Lemon") + "<br />");

document.write(fruits);
```

Convert an array to a string - toString()

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

document.write("Removed: " + fruits.splice(2,0,"Lemon") + "<br />");

document.write(fruits);
```

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

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.write(fruits.unshift("Kiwi") + "<br />");
document.write(fruits.unshift("Lemon","Pineapple") + "<br />");
document.write(fruits);
```

Boolean :

```
<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>
```

o/p:

```
0 is boolean false
1 is boolean true
```

An empty string is boolean false
null is boolean false
NaN is boolean false
The string 'false' is boolean true

Create a Boolean Object

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

The following code creates 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");
```

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:

```
var pi_value=Math.PI;
var sqrt_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));  /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());/0.4938101275747413
```

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));/4
```

Round() //document.write(Math.round(0.60) + "<br />");
,random()/document.write(Math.random() + "
");,max()/document.write(Math.max(0,150,30,20,38) + "
");,min()/document.write(Math.min(1.5,2.5));

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).

Example 1

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

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

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

```
Visit W3Schools
```

Try it yourself »

Example 2

Do a global search for "is":

```
var str="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?
```

Try it yourself »

Example 3

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

```
var str="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?
```

Try it yourself »

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":

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
```

Try it yourself »

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":

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
```