This book has been prepared exclusively for

VEIS COMPUTER EDUCATION

Java Script & jquery

JavaScript Overview

JavaScript gives you the freedom to make your web pages more attractive & good looking.
You should already be familiar with **HTML** and **CSS**.

JavaScript and Java

Java (developed by Sun Microsystems) is a powerful and much more complex programming language in the same category as C and C++.
JavaScript was created by **Brendan Eich** at **Netscape** and was first introduced in December 1995 under the name of *LiveScript*. However, it was rather quickly renamed JavaScript, although JavaScript's official name is **ECMAScript**, which is developed and maintained by the **ECMA (European Computer Manufacturer's Association)** International organization.
JavaScript is a scripting language, that is, a lightweight programming language and it is **interpreted** by the browser engine when the web page is loaded.

Note: Java Script & Java are not the same.

JavaScript:

JavaScript is a scripting language which is primarily used within HTML webpages. JavaScript is executed directly in the browser and all main browsers contain a compiler or interpreter for JavaScript. JavaScript is case sensitive. JavaScript and Java are completely different programming languages even though they have a similar name.

You can put JavaScript into an external file or directly into the HTML page. JavaScript which is embedded into an HTML page must be surrounded by <script type="text/javascript"> </script> tags. The type is mandatory, even if the browser only supports JavaScript.

JavaScripts in the body will be executed while the page loads. JavaScripts in the header will be executed when other JavaScript functions in the body call them.

Common uses of JavaScript include:

- Alert messages
- Popup windows
- Dynamic dropdown menus
- Form validation
- Displaying date/time

JavaScript usually runs on the *client-side* (the browser's side), as opposed to *server-side* (on the web server). One benefit of doing this is performance. On the client side, JavaScript is loaded into the browser and can run as soon as it is called. Without running on the client side, the page would need to refresh each time you needed a script to run. A JavaScript script cannot access server resources such as databases.

Create JavaScript

You can create JavaScript using the same equipment you use when creating HTML. That is:

- Computer off course.
- Text editor. For example, Notepad (for Windows), Pico (for Linux), or Simpletext (Mac). You could use a HTML editor if you like but it's not needed.
- Web Browser. For example, Internet Explorer or Firefox. You will need to ensure JavaScript is enabled within your browser's settings (this is normally enabled by default).

Enable JavaScript

To view web pages with JavaScript, you need to ensure your browser has JavaScript enabled. Without JavaScript enabled, you can still view the webpage without JavaScript, but you will not be able to take advantage of the JavaScript functionality. You normally do this by checking your browser's *options*. This will depend on the browser you're using. Enabling common browsers are below:

Internet Explorer:

1. Go to *Tools* from the top menu
2. Select *Internet Options*
3. Click on the *Security* tab
4. Click *Custom Level*
5. Scroll down until you see the *Scripting* section
6. Ensure that the *Active Scripting* option is set at *Enable*
7. When the *Warning!* window asks *Are you sure you want to change the settings for this zone?* click *Yes*

Google Chrome:

1. Go to *Chrome* from the top menu
2. Select *Preferences...*
3. Click on *Show advanced settings... Click on Content Settings...*
4. Ensure that the *Allow all sites to run JavaScript (recommended)* option is selected
5. Click *OK*

Mozilla Firefox

- Go to the *Firefox* menu
- Select *Options*
- Select the *Content* tab
- Check the *Enable JavaScript* checkbox
- Click *OK* to close the *Options* window

Opera:

1. Go to *Menu* from the top menu
2. Select *Settings*
3. Select *Quick Preferences*
4. Ensure that the *Enable JavaScript* option is checked

Netscape Navigator:

1. Go to *Edit* from the top menu
2. Select *Preferences*
3. Select *Advanced*
4. Select *Scripts & Plugins*
5. Check the *Enable JavaScript* checkbox
6. Click *OK*

Apple Safari:

1. Go to *Safari* from the top menu
2. Select *Preferences*
3. Select *Security*
4. Ensure that the *Enable JavaScript* option is checked
5. Click *OK*

JavaScript Syntax

JavaScript syntax refers to a set of rules that determine how the language will be written (by the programmer) and interpreted (by the browser).
The JavaScript syntax is loosely based on the Java syntax. Java is a full blown programming environment and JavaScript could be seen as a sub-set of the Java syntax.

Example

```
<script type="text/javascript">
<!--
document.write("JavaScript is not Java");
-->
</script>
```

Explanation of code

- The `<script>` tags tell the browser to expect a script detail in between them. You specify the language using the `type` attribute. The most popular scripting language on the web is JavaScript.
- The bits that look like HTML comments tag (<-- -->) are just that - HTML comment tags. These are optional but recommended. They tell browsers that don't support JavaScript (or with JavaScript disabled) to ignore the code in between. This prevents the code from being written out to your website users
- The part that writes the actual text is only 1 line (`document.write("JavaScript is not Java");`). This is how you write text to a web page in JavaScript. This is an example of using a JavaScript function.

Location of scripts:

You can place your scripts in any of the following locations:

- Between the HTML document's head tags.
- Within the HTML document's body (i.e. between the body tags).
- In an external file (and link to it from your HTML document).

Putting JavaScript Code in the HTML Header

Here's how the code looks in the HTML header of this page:

```
<html>
  <head>
    <title>
      JavaScript Example  -- Hello Jatin
    </title>
<script type="text/javascript">
    <!-- to hide script contents from old browsers
    document.write("Hello Jatin!");
    // end hiding contents from old browsers  -->
</script>
  </head>
  <body>
  ... etc.
```

Putting JavaScript Code In the HTML Body

Here's the HTML and JavaScript that created the paragraph above:

```
    <blockquote>I said, <i>"<script type="text/javascript">
      <!-- to hide script contents from old browsers
      document.write("Hello Jatin!");
      // end hiding contents from old browsers  -->
      </script>"</i>.
    </blockquote>
```

Calling `document.write()`

The `write()` method simply takes a list of one or more expressions (separated by commas) and passes them to the HTML parser. In other words, you can generate plain text, HTML tags, even JavaScript code. In fact one could dynamically generate the entire displayed document.

Your first JavaScript (embedded))

Create the following HTML page (via a standard text editor) and open it with a browser. In this example we will embedded the JavaScript into the HTML page.

```
<!DOCTYPE html>
<html>
<body>
<script type="text/javascript">
<!--This is single line comment -->
/*
This is a multi line commend
The following command write output the text into the HTML code. -->
*/
document.writeln("Hello, Jatin!");
```

</script>
</body>
</html>
Open the HTML page in a browser. The commands within the script tags will run and "Hello, Jatin!" will be written to the webpage. **Congratulations, for your first page.**

Your first JavaScript (external))

You can also put the JavaScript in a separate file and include it into the HTML page. For example create the file "hello2.html" with the following content.
<!DOCTYPE html>
<html>
<body>
<script src="myfile.js"></script>
</body>
</html>
Create the file "myfile.js" in the same directory as the HTML page. As the JavaScript is in a separate file you do not have to use the <script> </script> tag. Simple write the code directly into the file.
document.writeln("Hello World via an external file!");
Open the HTML page in a browser. The script should get executed and you should see the message.

Hello World via an external file!

JavaScript Guidelines

JavaScript ignores whitespace

The two lines below process exactly the same since Javascript ignores white space.
Javascript CODE EXAMPLE
var myName="Jatin";
var myName = "Jatin";

JavaScript Is Case Sensitive

The two variables below are different due to the variation in uppercase and lowercase letters in the variable naming. They are treated as two separate objects because Javascript is case sensitive.
Javascript CODE EXAMPLE
var myname = "Jatin";
var MyName = "Jatin";

Naming your variables and functions

When creating variable and function names do not start them with a number. You can use numbers within the name, but just not as the first character. Another good rule is to use only letters, numbers and underscores when naming things.
JavaScript CODE EXAMPLE
var 3rd_name = "Jatin"; // will cause script to fail
var name_3 = "Jatin"; // will process just fine

JavaScript Reserved Words

There are some reserved words that are part of Javascript syntax that we should try not use as variable/function names when we create and use objects while coding Javascript. In some cases using the reserved words can make your script operate unexpectedly.

abstract	boolean	break	byte
case	catch	char	class
const	continue	debugger	default
delete	double	do	else
enum	export	extends	finally
final	float	for	function
goto	if	implements	import
in	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	super	switch
synchronized	this	throw	throws
transient	try	typeof	var
void	volatile	while	with

JavaScript Popup Boxes

A popup box that displays a message, These are all built into JavaScript and are what I call "JavaScript popup boxes". They can also referred to as "dialog boxes", "JavaScript dialogs", "popup dialog" etc.

Types of Popups

JavaScript has three different types of popup box available for you to use. Here they are:

Alert

Displays a message to the user. Example:

To create a JavaScript alert box, you use the `alert()` method.

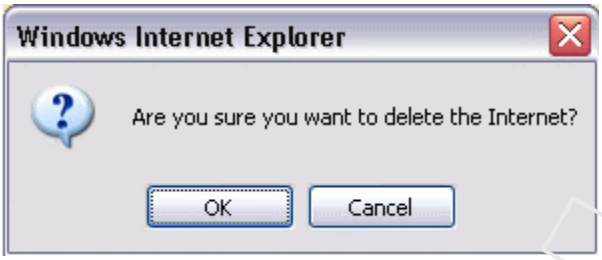
Example:

<input type="button" onclick="alert('*Hey, remember to tell your friends about veinstitution.com*!');" value="Remember" />

Confirm

The user to confirm something. Often, this is in conjunction with another action that the user is attempting to perform.

Example:



To create a JavaScript confirm box, you use the `confirm()` method.

Example:

<input type="button" onclick="confirm('Are you sure you want to delete the Internet?');" value="Example" />

Prompt

Prompts the user for information

Example:

To create a JavaScript prompt, you use the `prompt()` method.

Example:

<input type="button" onclick="prompt('Please enter your favorite website', '*veinstitution.com*');" value="Example" />

Debugging

The easiest way to "debug" JavaScript is to use the alert() method which allows to give the user some feedback. This is the equivalent to "println" in other programming languages. For example the following program give a popup with the currently value to the user.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>

```
<body>

<script type="text/javascript">
/*Declare a text variable and print it to the HTML page*/
var mytext= "Example Text.";
/*Call the function*/
alert(mytext);

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

JavaScript Comments

Comments are handy for leaving yourself notes in large scripts, or for disabling whole sections of code for troubleshooting faulty code. They can be placed anywhere in your JavaScript.

Example:

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

// This is a single line comment

/* This is a single line comment */

/*
This multiline comment will not be processed
by the browser software when the script runs
*/

</script>
```

Programming in JavaScript

Variables and Constants

Variables and **constants** are like containers where you store data and values for processing in JavaScript

The difference between a **variable** and a **constant** is this: once you give a **value** to a **constant the value is meant to be kept unchanged** throughout the script. In all circumstances where you reasonably foresee that the **original value is modified** through the script, use a **variable** as your storage. In fact, **you're going to use variables most of the times**.

For example, you could prompt your website users for their first name. When they enter their first name you could store it in a variable called say, firstName. Now that you have the user's first name assigned to a variable, you could do all sorts of things with it like display a personalised welcome message back to the user for example. By using a variable to store the user's first name, you can write one piece of code for all your users.

First, you need to declare your variables. You do this using the `var` keyword. You can declare one variable at a time or more than one. You can also assign values to the variables at the time you declare them.

The following demonstrates how to declare and use variables in JavaScript.

```
<!DOCTYPE html>
<html>
<body>
<script type="text/javascript">
/*Declare a text variable and print it to the HTML page*/
var mytext= "Hello, hello. Turn the radio on. Is there anybody out there...";
document.writeln(mytext);


/* Declare Variables for doing some math*/
a=5;
var b=1;
document.writeln(a-b);

/* Concatenate two strings*/
var mytext1 ="I like spanish";
var mytext2 =" because the melodie is nice.";
document.writeln(mytext1+mytext2);
</script>
</body>
</html>
```

You may notice that there are two ways of declaring variables.

```
a=5;
var b=1;
```

Variables defined without the keyword "var" are global variables. Variables defined with "var" are scoped according to their declaration, e.g. if you define a "var" in a function this variable is only valid in this function.

Note: JavaScript has four basic types, Object and the primitive types Number, String, and Boolean.

Create variables and constants

You **declare**, that is, create variables and constants in a similar way.
Here's how you **declare a variable**:

```
/*To declare a variable you give it a name preceded

by the JavaScript keyword var*/

var amountDue;
```

In the sample code above you declare a variable named *amountDue*.
The **variable declaration** ends with a (;) semicolon, which makes it a self-contained statement.
Also, because JavaScript is **case sensitive**, every time you refer to the *amountDue variable*, make sure you **keep the letter case exactly the same as the original declaration**
Note: this particular notation is called **camelCase** because it looks like a camel's hunch.

Here's how you **declare a constant**:

```
/*To declare a constant you give it a name preceded

by the JavaScript keyword const

Also take note: const is not supported by Internet Explorer

use var instead: it's safer*/

const taxRate;
```

As seen above, you declare a taxRate constant. This declaration is very similar to the variable declaration, the only difference being the keyword used: **var in the case of a variable** and **const in the case of a constant**.

Assign values to variables

After you create a variable, you must **assign** (give) a value to it (or said in Geeeky talk **"initialize a variable")** . The **(=) equal sign** is called **assignment operator**: you assign the value of what is on the right side of the = sign to whatever is on the left side of the = sign.
Notice: you cannot perform operations with empty variables.
Ready to fire off your text editor? Let's get coding!
Prepare a basic HTML document with the JavaScript code illustrated below:

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 4: Variables and Constants</title>
</head>
<body>
<h1>Lesson 4: Variables and Constants</h1>

<script type="text/javascript">

//Create a variable

var amountDue;

/* Assign a value to it:

you do not know the value yet

so for now it is 0 */

amountDue = 0;

/* Create 2 more vars and assign
```

```
a value to each at the same time */

var productPrice = 5;

var quantity = 2;

/* Assign a value to amountDue by

multiplying productPrice by quantity */

amountDue = productPrice * quantity;

/* Notice how the value of amountDue has

changed from 0 to 10 -

-Alert the result to check it is OK

Notice: you do not use ' ' with var name in alert() */

alert(amountDue);

</script>

</body>
</html>
```

execute above code & see the result in browser.
Notice: When tracing bugs (errors) in your JavaScript code, look out for brackets, semicolons (;), quotes (' '), and letter casing.

Name variables (and constants)

Choose **descriptive names** for your variables (**var amountDue;** rather than **var x;**): this way your code will be more **readable and understandable**.
While this can be called good programming practice, there are also **syntax rules**, when it comes to naming variables, and they must be obeyed, at least if you want your JavaScript script to work.
Keep an eye on the following simple rules when you name variables:

1) The first character must be a letter, an (_) underscore, or a ($) dollar sign:

5total is wrong: var name cannot start with a number

2) Each character after the first character can be a letter, an (_) underscore, a ($) dollar sign, or a number:

3) Spaces and special characters other than (_) and $ are not allowed anywhere:

- to tal is wrong: spaces are not allowed;
- total# is wrong: # character is not allowed;
- total£ is wrong: £ character is not allowed.

Events

When you write a JavaScript function, you will need to determine when it will run. Often, this will be when a user does something like click, submit a form, double clicks on something or hover over something etc.

The most common events you're likely to deal with in your JavaScript are:
onLoad/onUnload;
onClick;
onSubmit (triggered by the user submitting a form);
onFocus / onBlur (triggered by a user as, for example, she clicks in a textbox or clicks away from a textbox respectively);
onMouseOver / onMouseOut (triggered by the user moving the mouse over or away from an HTML element respectively).
There are other events that might eventually be of interest to you as you write sophisticated JavaScript scripts, such as **scroll** events (as users scroll up and down a web page), **onTextChanged** events (as users type in a textbox or textarea), even **touch** events, which are likely to gain interest in today's mobile and tablet-invaded world.

Operators

JavaScript operators are used to perform an operation. There are different types of operators for different uses.
we need a way to **manipulate** data and variables. We do this with **operators**.

- **arithmetic operators**;
- **Assignment operators, the + sign to concatenate text (concatenation operator)**;
- **comparison operators**;
- **logical operators.(boolean Operator)**
- **string operator**

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder of a division)
++	Increment
--	Decrement

Assignment Operators

Operator	Description
=	Assign
+=	Add and assign. For example, x+=y is the same as x=x+y.
-=	Subtract and assign. For example, x-=y is the same as x=x-y.
*=	Multiply and assign. For example, x*=y is the same as x=x*y.
/=	Divide and assign. For example, x/=y is the same as x=x/y.
%=	Modulus and assign. For example, x%=y is the same as x=x%y.

Comparison Operators

Operator	Description
==	Is equal to
===	Is identical (is equal to and is of the same type)
!=	Is not equal to
!==	Is not identical
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Logical/boolean Operators

Operator	Description
&&	and
||	or
!	not

String Operators

In JavaScript, a *string* is simply a piece of text.

Operator	Description
=	Assignment
+	Concatenate (join two strings together)
+=	Concatenate and assign

JavaScript Conditions

Condition statements are used to evaluate things and add logic to your scripts. To create lots of different logic for your conditional evaluations, check out all of the Comparison Operators Javascript comes equipped with. Full understanding of the Comparison Operators is essential for conditional programming.
A conditional statement refers to a piece of code that does one thing based on one condition, and another based on another condition. In fact, you could have as many conditions as you like.

JavaScript If statements are an example of conditional statements. With If statements, you can tell the browser to execute a piece of code only **if** a given condition is true.

Example

```
<script type="text/javascript">
<!--
function analyzeColor1(myColor) {
```

```
        if (myColor == "Blue") {
            alert("Just like the sky!");
        }
}
//-->
</script>
<h3>Favorite Color</h3>
<input type="radio" name="fav_color1" value="Blue" onclick="analyzeColor1(this.value);" /> Blue <br />
<input type="radio" name="fav_color1" value="Red" onclick="analyzeColor1(this.value);" /> Red <br />
<input type="radio" name="fav_color1" value="Green" onclick="analyzeColor1(this.value);" /> Green <br />
```

Explanation:

- We create three radio buttons using the HTML `<input>` element. Each of the radio buttons have a different value. The value is set using the `value` attribute (eg, `value="Blue"`)
- When the user clicks any of the radio buttons, we use the `onclick` event handler to call the `analyzeColor1()` function. When we call that function, we pass in the value of the radio button (using `this.value`). The function then takes that value and performs an `if` statement on it.
- The `if` statement's first line is `if (myColor == "Blue") {`. This means that it will test the value of the `myColor` variable. It will test to see if it's value is equal to `Blue`. Notice the opening curly brace on this line (the closing one is two lines down).
- In between the curly braces, we write whatever we want to occur in the event the test is true (i.e. the color is equal to `Blue`). In this case, we display an alert box with a customized message.

JavaScript If statement

<="" p="">

1. Start with the word "if"
2. Between open and closed brackets, write the actual condition that is being tested (i.e. if something is equal to something else).
3. Between open and closed **curly** brackets (or braces), specify what will happen if the condition is satisfied.

JavaScript If Else statement

The above code is OK if you only want to display something when the condition is true. But what if you want to display something when the condition is **not** true. For example, what if the variable `myColor` was equal to, say, `Red`?

This is where an If Else statement comes in handy. The `else` part is what we're interested in here. The `else` is what tells the browser what to do if the condition is not true.

Example:

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

```
<!--
function analyzeColor2(myColor) {
	if (myColor == "Blue") {
		alert("Just like the sky!");
		}
	else {
		alert("Didn't pick blue huh?");
	}
}
//-->
</script>
<h3>Favorite Color</h3>
<input type="radio" name="fav_color2" value="Blue" onclick="analyzeColor2(this.value);" /> Blue <br
/>
<input type="radio" name="fav_color2" value="Red" onclick="analyzeColor2(this.value);" /> Red <br
/>
<input type="radio" name="fav_color2" value="Green" onclick="analyzeColor2(this.value);" /> Green
<br />
<input type="radio" name="fav_color2" value="None" onclick="analyzeColor2(this.value);" /> None
```

Explanation :

The first part of this code is the same as in the IF Statement. The second part is where we specify what to do if the condition is not true. Therefore, you write `else` followed by what you want to occur, surrounded by curly braces.

1. Write an if statement.
2. Follow this with the word "else"
3. Between open and closed **curly** brackets (or braces), specify what to do next.

JavaScript If Else If statement

The If Else If statement is more powerful than the previous ones. This is because you can specify many different outputs based on many different conditions - all within the one statement. You can also end with an "else" to specify what to do if none of the conditions are true.

Example:

```
<script type="text/javascript">
<!--
function analyzeColor3(myColor) {
	if (myColor == "Blue") {
		alert("Just like the sky!");
		}
	else if (myColor == "Red") {
		alert("Just like shiraz!");
	}
	else {
		alert("Suit yourself then...");
```

```
        }
}
//-->
</script>
<h3>Favorite Color</h3>
<input type="radio" name="fav_color3" value="Blue" onclick="analyzeColor3(this.value);" /> Blue <br />
<input type="radio" name="fav_color3" value="Red" onclick="analyzeColor3(this.value);" /> Red <br />
<input type="radio" name="fav_color3" value="Green" onclick="analyzeColor3(this.value);" /> Green <br />
<input type="radio" name="fav_color3" value="None" onclick="analyzeColor3(this.value);" /> None
```

Explanation :

We started with an `if` and ended with an `else`, however, in the middle, we used an `else if` statement to specify what to do if the `myColor` variable was equal to Red. This statement looks exactly like the `if` statement - just with an `else` pretended to it.

switch statements

In alternative to an *if ... else if ... else* statement you can switch to a . . well a ***switch statement***. Here's its basic structure:

```
//condition to evaluate

switch(condition)

{

//in this case do this

case 1:

execute code block 1

//stop here: no need to keep going

break;

//in this other case do that

case 2:

execute code block 2

//stop here: no need to keep going

break;
```

```
//otherwise fall back on this

default:

code to execute when all previous conditions evaluate to false

}
```

Ternary

The ternary conditional operator(?) is not a statement but it creates conditional logic just the. Itwill return the value on the left of the colon(:) if the expression is true, and return the value on the right of the colon if the expression is false.

Javascript CODE EXAMPLE

```
var a = 5;
var b = 3;

var result = (a > b) ? "That is true" : "That is false";

document.write(result);
```

Result

That is true

JavaScript Loops

Loops are JavaScript statements that allow you to process code in a looping fashion. You can also use them to process any custom code repeatedly, or iterate over arrays and object properties. Loops run lightning fast and are extremely useful for information processing and animations. All a programmer has to watch out for when scripting loops is creating never-ending (or extremely large) loops that will exceed the document's processing entry.

JavaScript scripts can do all sort of calculating discounts, comparing values, making choices and many more. JavaScript loves repetition. Types of Loops are given below.

- the **for loop**;
- the **while loop**;
- the **do ... while loop**.

Loops

At times you will have to repeat some piece of code a number of times:
for instance: a web page has 10 radio buttons and your script needs to find out which radio button was checked by the user.
To do so, your script has to go over each single radio button and verify whether its **checked attribute** is set to true or false. Does this mean having to write this kind of *verification code* 10 times?

You **dump the code in a loop once, and it's going to execute any number of times** you set it to.
Any loop is made of **4 basic parts**:

1. **The start value**

   An initial value is assigned to a variable, usually called **i** (but you can call it anything you like). This variable acts as counter for the loop.

2. **The end value or test condition**

   The loop needs a limit to be set: either a definite number (loop 5 times) or a truth condition (loop until this condition evaluates to true). Failing this, you run the risk of triggering an **infinite loop**. This is very bad: it's a never-ending repetition of the same code that stops users' browsers from responding. Avoid infinite loops at all costs by making sure you set a boundary condition to your loops;

3. **The action or code to be executed**

   You **type a block of code once and it'll be executed the number of times between your start value and end value**;

4. **The increment**

   This is the part that moves the loop forward: the counter you initialize has to move up (or down in case you opt for looping backwards). As long as the loop **does not reach the end value or the test condition is not satisfied**, the counter is incremented (or decremented). This is usually done using mathematical operators.

The for Loop

This kind of loop, well ... loops through a block of code a set number of times. Choose a **for loop** if you know in advance how many times your script should run.
Here's its basic structure:

```
//loop for the number of times between start value

//and end value.  Increment the value each time

//the limit has not been reached.

for (var=startvalue; var<=endvalue; var=var+increment)

{

//code to be executed

}
```

The while Loop

If you don't know the exact number of times your code is supposed to execute, use a **while loop**. With a while loop your code **executes while a given condition is true**; as soon as this condition evaluates to false, the while loop stops.
Here's its basic structure:

```
//loop while initial value is less than or equal to end value

//Take note: if the condition evaluates to false from the start,

//the code will never be executed

while (variable <= endvalue)

{

//code to be executed

}
```

do ... while Loop

This kind of loop is similar to the *while loop*. The difference between the two is this:
In the case of the *while loop*, **if the test condition is false from the start**, the **code in the loop will never be executed**.
In the case of the *do ... while loop*, the test condition is evaluated **after** the loop has performed the first cycle. Therefore, even **if the test condition is false, the code in the loop will execute once**.
Here's the basic structure of a do ... while loop:

```
//the command is given before the test condition is checked

do

{

//code to be executed

}

//condition is evaluated at this point:

//if it's false the loop stops here

while (variable <= endvalue)
```

Add random number of images with a *for loop*

Time to get coding: prepare a fresh HTML document with a div tag and an id of "wrapper" and add the following JavaScript magic:

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 7: Leave Boring Repetitive Stuff to JavaScript with Loops</title>
</head>
<body>
<h1>Lesson 7: for Loop</h1>

<div id="wrapper"></div>

<script type="text/javascript">

//Grab the html div by its id attribute

//and store it in a var named container

var container = document.getElementById("wrapper");

//use Math.random to generate a random number

//between 0 and 1. Store random number in a var

var randomNumber = Math.random();

//use Math.floor() to round off the random number

//to an integer value between 0 and 10 and store it in a var

var numImages = Math.floor(randomNumber * 11);

//just for testing purposes, lets alert the resulting random number

alert(numImages);

//the loop will run for the number of times indicated

//by the resulting random number

for (var i=1; i <= numImages; i++)

{

//code to be executed:

//create a var and store a string made of a new HTML img element
```

```
//and the iteration variable i (use + concatenation operator):
//i contains the iteration number
//Take note:make sure the img src corresponds to a real image file
var newImage = '<img src="loop.gif" alt="#" />' + i;
//use the container var that stores the div element
//and use innerHTML to insert the string stored in the newImage var
//Use shortcut assignment += so previous values do not get wiped off
container.innerHTML += newImage;
}
</script>
</body>
</html>
```

Add images to the page with a *while loop*

```
<script type="text/javascript">
//Grab the html div by its id attribute
//and store it in a var named container
var container = document.getElementById("wrapper");
//create the var containing the counter
//and give it an initial value of 0
var numImages = 0;
//start the loop: while the number of images is less than or
//equal to 10 (loop starts at 0 not 1, so type 9 not 10),
//keep cycling and increase the counter by 1
while (numImages <= 9)
```

```
{
//this is the block of code to be executed:
//build the string to insert in the HTML document
//and store it in the newImage var
//Take note:make sure the img src corresponds to a real image file
var newImage = '<img src="loop.gif" alt="#" />' + numImages;
//use the container var that stores the div element
//and use innerHTML to insert the string stored in the newImage var
//Use shortcut assignment += so previous values do not get wiped off
container.innerHTML += newImage;
//increase the counter by 1 using the increment operator
numImages ++;
}
</script>
```

Make sure the image file is in place, save your work and run the page in the browser.

Add images to the page with a *do ... while loop*

```
<script type="text/javascript">
var container = document.getElementById("wrapper");
var numImages = 0;
do
{
//this block gets executed first:
var newImage = '<img src="loop.gif" alt="#" />' + newImage;
container.innerHTML += newImage;
```

```
numImages ++;

}

//here's the condition to evaluate:

//the first cycle has already executed.

while (numImages <= 9);

</script>
```

JavaScript Functions

Functions are segments of your script that do not run until they are called to execute in your document. They can be executed by HTML events or Javascript events, and they will only run when that event fires off.
A function (also known as a *method*) is a self-contained piece of code that performs a particular "function". You can recognise a function by its format - it's a piece of descriptive text, followed by open and close brackets.
Sometimes there will be text in between the brackets. This text is known as an *argument*. An argument is passed to the function to provide it with further info that it needs to process. This info could be different depending on the context in which the function is being called.
Arguments can be extremely handy, such as allowing your users to provide information (say via a form) that is passed to a function to process. For example, your users could enter their name into a form, and the function would take that name, do some processing, then present them with a personalised message that includes their name.
A function doesn't actually do anything until it is called. Once it is called, it takes any arguments, then performs it's function.

Writing a function in JavaScript

The first thing you need to do is write the function:

```
<script type="text/javascript">
<!--
function displayMessage(firstName) {
   alert("Hello " + firstName + ", hope you like JavaScript functions!")
}
//-->
</script>
```

Call the function

Once you have written your function, you can "call" that function from within your HTML code. Here, when the user clicks the button, it runs the function. In this case, we use the `onclick` event handler to call the function.

```
<form>
First name:
```

```
<input type="input" name="yourName" />
<input
  type="button"
  onclick="displayMessage(form.yourName.value)"
  value="Display Message" />
</form>
```

So both combined should work like this:

```
<script type="text/javascript">
<!--
function displayMessage(firstName) {
   alert("Hello " + firstName + ", hope you like JavaScript functions!")
}
//-->
</script>
<form>
First name:
<input type="input" name="yourName" />
<input
  type="button"
  onclick="displayMessage(form.yourName.value)"
  value="Display Message" />
</form>
```

Explanation of code

Writing the function:

1. We started by using the *function* keyword. This tells the browser that a function is about to be defined
2. Then we gave the function a name, so we made up our own name called "displayMessage". We specified the name of an argument ("firstName") that will be passed in to this function.
3. After the function name came a curly bracket `{`. This opens the function. There is also a closing bracket later, to close the function.
4. In between the curly brackets we write all our code for the function. In this case, we use JavaScript's built in `alert()` function to pop up a message for the user.

Calling the function:

1. We created an HTML form with an input field and submit button
2. We assigned a name ("yourName") to the input field
3. We added the `onclick` *event handler* to the button. This event handler is called when the user clicks on the button (more about event handlers later). This is where we call our JavaScript function from. We pass it the value from the form's input field. We can reference this value by using "form.yourName.value".

Feed information to and retrieve information from a function

Functions manipulate data: the alert() function has messages to display as its data, the write() function has text to write on the web page as its data, etc.
You can also **feed data to a function** for manipulation. The way you do this is through **arguments (or parameters)**. An **argument is placed inside the function's brackets when the function is declared**. You can place one or more arguments separated by commas(,) inside a function.
Here's what the basic structure looks like:

```
function functionName(arg1, arg2, arg3)

{

//body of the function goes here

}
```

The function argument is like a **placeholder for the value that gets fed when the function is called**. This will appear clearer in the example below.
Finally, one useful thing functions can do with data, once it's been manipulated, is to **return it**.
For example, take the **floor(number) function** that you already know from the previous lesson. This function **takes in a number argument** and **returns an integer number**.
If your script does something with the **returned value then it needs to assign the function to a variable** when calling the function.
Let's put this knowledge into practice right away. Create a function and then call it to use its **return value** in your document.

Declare a function with a parameter

Prepare a simple HTML page and type the following JavaScript code within <*script*> ... </*script*> tags:

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 8: Function Arguments and Return Values</title>
</head>
<body>
<h1>Lesson 8: Function Arguments and Return Values</h1>

<script type="text/javascript">

//declare a function with two arguments of type number

function addNumbers(num1, num2)

{

//this is a simple function:

// just return the sum of the two number arguments
```

```
return num1 + num2;

//your function ends here

}

</script>

</body>
</html>
```

Call your function and use its return value

Continue on from the previous code. Outside your function, just **after the } closing curly brace**, type the following:

```
//Call your function: create a var

//and assign the value returned by your function to it

//Also, give a value to the number arguments

//by typing 2 comma-separated numbers within brackets

var total = addNumbers(3, 5);

//Display the returned data on the page

document.write(total);
```

Save your work and run the page in the browser. If all goes well, the sum of the numbers you inserted as arguments of your addNumbers() function should be displayed on the web page.
Function arguments are not limited to a specific data type. You can also use strings and booleans, and even other functions. However, we don't want to be too involved at this stage.

Variables inside and outside functions:

Scope

A variable can have **local or global scope on the basis of whether it's declared inside or outside a function block.**. But, what does this mean exactly?
Simply put, a variable having **local scope means that it's visible, or accessible, only within the function in which it lives**. No other portion of your code can see a local variable.
On the other hand, a variable having **global scope means that it's accessible, that is, it can be used, anywhere in your script.**. Let's see this in practice. Get your text editor ready and type the following snippet between opening and closing <script> tags in an HTML page:

```
//create 2 global variables

var message = "outside global message";

var otherGlobalVariable = "other global variable";

//create a function with a local variable

//having the same name as the first global variable

function getMessage()

{

var message = "inside local variable";

//the function alerts the message variable

alert(message);

//and it also alerts the second global variable

alert(otherGlobalVariable);

//the function ends here

}

//call the function

getMessage();

//alert the message variable

//(which one will it be, local or global?)

alert(message);
```

Save your work and preview the page in your browser. As you can see, the **alert() function outside the variable doesn't have access to the message variable inside the function**. On the other hand, the **function can have access both to its local variable and to any variable in the global space**.
This might seem a bit confusing at first, but it's all a matter of practicing your coding skills as often as you can. The important thing to remember is that, if at times the variables seem not to have the values you expect, it might be a scope-related bug.

Arrays

Arrays are a fundamental part of most programming languages and scripting languages. Arrays are simply an ordered stack of data items with the same data type. Using arrays, you can store multiple

values under a single name. Instead of using a separate variable for each item, you can use one array to hold all of them. Arrays are Objects in JavaScript. You can define them and use the method pop() to remove the first element and push(newValue) to add a new element at the end.
For example, say you have three Frequently Asked Questions that you want to store and write to the screen. You could store these in a simple variable like this:

Arrays are Objects in JavaScript. You can define them and use the method pop() to remove the first element and push(newValue) to add a new element at the end. You can also iterate over it.

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

var myArray = ["Android", true, 7]
alert(myArray[0]);
alert(myArray.length);
myArray.pop();
alert(myArray);
myArray.push("Eclipse")
alert(myArray);

/*Iterate over the array and write it to the console of the browser*/

for(var i = 0; i < myArray.length; i++) { alert(myArray[i]) }

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

Functions

A function in JavaScript encapsulates reusable code and are represented as Objects. Functions can be directly called via other JavaScript code. It is recommended that you put functions in the header of the HTML page.
Functions are declared via the function keyword. You can call a function directly, or use the apply method on the function.

```
<!DOCTYPE html>
<html>
<header>
<script type="text/javascript">
/*Write the text twice to the document*/
function writetext(text)
{
document.writeln(text);
document.writeln(text+ "<p>");
}
</script>

</header>
<body>
```

```
<script type="text/javascript">
/*Declare a text variable and print it to the HTML page*/
var mytext= "Example Text.";
/*Call the function*/
writetext(mytext)
/*Call it via apply*/
writetext(mytext + "apply").apply();
</script>
</body>
</html>
```

Prototype in JavaScript

JavaScript does not support classes and inheritance of classes like object orientated programming languages. JavaScript is a prototype-based language, by this approach you can reuse functions by cloning existing objects.

Assigning functions to HTML buttons

The following gives an example how to assign a function to an HTML button.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<script type="text/javascript">
function writetext(mytext)
{
document.writeln(mytext);
}
function addNumber(a,b){
  var result = a+b;
  return result;
}
</script>

</head>
<body>

<input type="button" value="Write Text after click"
onclick="writetext('Hello function')" >

<input type="button" value="Add number"
onclick="writetext(addNumber(1,3))" >
</body>
</html>
```

HTML Events

JavaScript can react to certain event on the page and / or on certain webpage elements, e.g. buttons. You can register a function to a event in the HTML page.

Table 1. Events in Javascript

Event	Description
Onload	Triggered then the user loads the page. The onload even can for example be used to check the visitors browser type.
onChange	Called whenever a field is changed. Can for example be used to validate an input field of a form.
onSubmit	Called then a user clicks on the submit button of a form.
OnMouseOver and OnMouseOut	Called then the mouse enters a certain element on the page or leaves it.

Introduction to Objects

In JavaScript, objects are data (variables), with properties and methods.
Almost "everything" in JavaScript can be objects. Strings, Dates, Arrays, Functions....
You can also create your own objects.
This example creates an object called "person", and adds four properties to it:

Example:

```
var person = {firstName:"Jatin", lastName:"Bedi", age:18, eyeColor:"black"};
```

Spaces and line breaks are not important. A declaration can span multiple lines:

```
var person = {
   firstName:"Jatin",
   lastName :"Bedi",
   id       :5566
};
```

There are many different ways to create new JavaScript objects.
You can also add new properties and methods to already existing objects.

Accessing Object Properties

Example

```
name = person.lastName;
name = person["lastName"];
```

Accessing Object Methods

You can call a method with the following syntax:
objectName.methodName()
This example uses the fullName() method of a person object, to get the full name:

Example

name = person.fullName();

Object methods are just functions defined as object properties.
a JavaScript script interacts with its object-modelled data by means of the objects' qualities and behavior. These are called **properties and methods**.
Properties are values associated with an object. For example, an **HTML element object has a value property** (like the button object you're familiar with), and an **innerHTML property**, that you used to add new mark-up to the web page.
Methods represent what an object can do, its behavior, and are very much like functions.

Associate an object with a property or a method:

If you wondered what that odd-looking (.) dot notation in *document.write()* or *Math.random()*, and so on, meant in previous lessons, here's the answer.
You use the **object.property and object.method syntax** to interact with JavaScript objects.

```
//use the random() method of the Math object

Math.random();

//use the write() method of the document object

document.write();

//use the length property of the string object

myStringText.length;
```

Properties and methods of the string object:

- **length property**;
- **toLowerCase()/toUpperCase()**;
- **match()**;
- **replace()**;
- **indexOf()**.

The String object

JavaScript interpreter reads any piece of text enclosed in quotes ' ' (or double quotes " ") as an instance of the string object. This puts the magic of all the properties and methods belonging to the string object in our hands.
The **Object Oriented Programming** can use all that JavaScript goodness to achieve our script's goals without needing to have any clue whatsoever of the inner workings of those properties and methods. All we need to know is **what a property or method can do** for us, not **how it does** it.

Example: if we need to know the length of a piece of text, just using **pieceOfText.length** will achieve this. This is accomplished without us knowing anything of the programming virtuosity responsible for the power of the **length property of the string object**.

How to use *length*

The **length property of the string object** contains the number of characters (including spaces) in the text value of a string.
Here's a basic code snippet to demonstrate its use:

```
//create and initialize a string variable

var myString = "Hello JavaScript"

//apply the length property to the string

document.write(myString.length);

//JavaScript will print 16:

//spaces are included in the count

//If the string is empty length returns 0
```

How to use *toUpperCase()*

The **toUpperCase() method of the string object** turns the text value of a string into, well ... uppercase letters. Its companion **toLowerCase()** does the opposite: it turns the text value of a string into lowercase letters.
Here's a basic code snippet to demonstrate its use:

```
//create and initialize a string variable

var myString = "Hello JavaScript"

//apply the toUpperCase() method to the string

document.write(myString.toUpperCase());

//JavaScript will print HELLO JAVASCRIPT

//Try using myString.toLowerCase() on your own
```

How to use *match()*

The **match() method of the string object** is used to search for a specific value inside a string.
Here's a basic code snippet to demonstrate its use:

```
//create and initialize a string variable

var myString = "Hello JavaScript"
```

```
//apply the match() method to the string.

//match(stringToMatch) takes the value to search for as argument

document.write(myString.match("JavaScript"));

//JavaScript will print JavaScript

//If no match is found the method returns null
```

How to use *replace()*

The **replace() method of the string object** is used to replace a value for another value inside a string.
Here's a basic code snippet to demonstrate its use:

```
//create and initialize a string variable

var myString = "Hello JavaScript"

//apply the replace() method to the string.

//replace() takes 2 arguments and returns the new string value:

//replace(valueToReplace, newValue)

document.write(myString.replace("JavaScript", "World"));

//JavaScript will print Hello World
```

How to use *indexOf()*

The **indexOf() method of the string object** is used to know the position of the first found occurrence of a value inside a string.
Here's a basic code snippet to demonstrate its use:

```
//create and initialize a string variable

var myString = "Hello JavaScript"

//apply the indexOf() method to the string.

//indexOf() takes in the value to look for in the string as argument

//and returns the position number (index)

//indexOf(valueToFind)

document.write(myString.indexOf("Java"));
```

```
//JavaScript will print 6

//indexOf() includes spaces and starts counting at 0

//if no value is found the method returns -1
```

Example: guessing game

You're going to build a simple guessing game application using all the new knowledge you've acquired in this lesson. Your application is expected to check a name entered by the user and respond accordingly: Your **JavaScript code will be placed in its own external file** and referenced from the HTML document. **Separation of concerns**, that is, **separation between structure (HTML mark-up), appearance (CSS), and behavior (JavaScript)**, reflects contemporary best practices in web design.
Let's start with the HTML document. This is a simple web page with a textbox, a button, and a div where the JavaScript result will be displayed.

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 10: JavaScript Objects - Strings</title>
<script type="text/javascript" src="lesson10.js"></script>
</head>
<body>
<h1>Lesson 10: JavaScript Objects - Strings</h1>

<h2>What's the name of the wizard boy in J.K. Rowling's novels?</h2>

<p>Your answer: <input type="text" id="txtName" /></p>
<p><input type="button" value="Submit your answer" id="btnAnswer" />

<p id="message"></p>

</body>
</html>
```

Pay attention to the enclosing <head> tags. These contain a **reference to an external JavaScript file** named *lesson10.js* located in the same directory as the HTML file (you're free to place the JavaScript file in its own directory, if you prefer. However, make sure this is reflected in the src value of your <script> tag in the HTML document).
Note: the **elements that are going to play a role in our script all have an *id attribute*. This gives JavaScript a hook on those elements** and the data they will eventually contain.
Now open the JavaScript file *lesson10.js* and type the following code:

```
//This is the function that initializes our program

function init()

{
```

```
//Store a reference to the HTML button element in a variable:
//use the id of the HTML button element to do this
var myButton = document.getElementById("btnAnswer");

//use the onclick event of the button
//and assign the value of the getAnswer() function to it.
//This function performs the main job in your application
//and runs after the user clicks the button on the page.
//Take note: getAnswer is assigned without brackets.
//This is so because otherwise getAnswer() would be called
        //as soon as the page loads (and we don't want that).
myButton.onclick = getAnswer;

//the init function ends here
}

//Assign the init() function to the onload event.
//this event fires when the HTML page is loaded in the browser.
//Take note: init is assigned without brackets
onload = init;

//Now write the getAnswer() function
function getAnswer()
{
//Create all the vars you need to manipulate your data:
//secretName stores the correct answer the user is expected to guess:
var secretName = "Harry Potter";
```

```
//Turn the value of secretName into lower case:
//you do this because you're going to compare this value
//to the value entered by the user of your application.
//Given that users might type the answer either in upper or lower case,
//reducing the relevant text values to the same casing automatically
//ensures that only the content and not also the letter case plays a role in the comparison.
var secretNameLower = secretName.toLowerCase();

//Get the value the user types into the textbox
var myTextBox = document.getElementById("txtName");
var name = myTextBox.value;

//Also turn the value entered by the user to lower case
var nameLower = name.toLowerCase();

//Get a reference to the HTML paragraph that will display your result
//after the script has run by storing its id value in a var
var message = document.getElementById("message");

//These are the test cases your application needs to evaluate
//and respond to: if the user clicks the button but did not
//enter any value in the textbox:
if(nameLower.length <= 0)
{
alert("I didn't quite catch your answer.  Please enter an answer");
}

//If the user gets right the first half but not the latter half of the name:
```

```
//Take note of the use of the logical operator &&
//(go back to lesson 5 if you need to revise logical operators)
else if(nameLower.indexOf("harry") == 0 && nameLower.indexOf("potter") == -1)
{
alert("Almost there: think again");
}

//If the secret name and the name entered by the user match:
else if(nameLower.match(secretNameLower))
{
alert("You got it!");
message.innerHTML = "Congratulations, you win!";
}

//Default case - if the user types in the wrong answer:
else
{
alert("Wrong!");
message.innerHTML = "Sorry.  The correct answer is: ";
message.innerHTML += name.replace(name, "Harry Potter");
}

//the getAnswer() function ends here
}
```

If you **click the button without entering any value** in the textbox, an alert will pop up inviting you to enter an answer.
If you **get right the first half ("Harry") - indexOf() returns 0 ("Harry" is at position 0, which is the start of the index) - but not the latter half ("Potter") - indexOf() returns -1 (it finds no**

corresponding value inside the string), you get an alert letting you know that your answer is almost right.
If you **enter the correct answer**, well you'll get a prize.
Finally, if your **answer is totally wrong**, your application will say so and give you the correct answer.

Date object.

Here's what you will do in this lesson:

- **create a Date object**;
- **set dates**;
- **compare dates**.

The Date object

The **Date object** exposes plenty of methods to manipulate dates and times.

Create a Date object

You create a date object by using one of the following ways:

```
//First way:

//the var today is initialized with a Date object

//containing the current date and time

//on the basis of the user's computer

//use toLocaleDateString() to adjust the date

//to the local time zone and format it automatically

var today = new Date().toLocaleDateString();


//Second way:

//you pass a millisecond argument that starts at 1970/01/01:

//new Date(milliseconds)

var date = new Date(1000).toLocaleDateString();

//On my system this outputs: 01 January 1970


//Third way:
```

```
//You pass a string as argument:
//new Date(dateString)
var date = new Date("10 November, 2011").toLocaleDateString();
//On my system this outputs: 10 November 2011
```

Fourth way:

```
//new Date(year, month, day, hours, minutes, seconds, milliseconds)
//only year and month are required,
//the others are optional arguments and
//where not present 0 is passed by default
//Below you pass the year in a 4-digit format, and the month in number form:
//months are represented by numbers from 0 (January) to 11 (December)
var date = new Date(2011, 10).toLocaleDateString();
//On my system this outputs: 01 November 2011
```

After a Date object is created, you can use its methods to get and set dates and times.

Use getDate() to retrieve a date

```
//Create a Date object containing the current date and time
var myDate = new Date();
//use getDate() to extract the day of the month
document.write(myDate.getDate());
//At the time of writing, this prints 11
//days of the month range from 1 to 31
```

Use getTime() to retrieve a time

```
//Create a Date object containing the current date and time
var myTime = new Date();
//use getTime() to extract the  time
document.write(myTime.getTime());
//At the time of writing, this prints 1321021815555
//This is the millisecond representation of the current
//date object: the number of milliseconds between
//1/1/1970 (GMT) and the current Date object
```

Get Date object components

Once you have a Date object, one interesting thing you can do with it is to get its various components. JavaScript offers some interesting methods to do just that.

Use getFullYear() to extract the year component

```
//Create a Date object containing the current date and time
var myDate = new Date();
//extract the year component and print it on the page
document.write(myDate.getFullYear());
//At the time of writing, this prints 2011
```

Use getMonth() to extract the month component

```
//Create a Date object containing the current date and time
var myDate = new Date();
//extract the month component and print it on the page
document.write(myDate.getMonth());
```

```
//At the time of writing, this prints 10
//which represents the month of November - months are represented
//with numbers starting at 0 (January) and ending with 11 (December)
```

Use getDay() to extract the day component

```
//Create a Date object containing the current date and time
var myDate = new Date();
//extract the day component and print it on the page
document.write(myDate.getDay());
//At the time of writing, this prints 5
//which represents Friday - days are represented
//with numbers starting at 0 (Sunday) and ending with 6 (Saturday)
```

Use getHours() and getMinutes() to extract time components

```
//Create a Date object containing the current date and time
var myDate = new Date();
//extract the hours component
var hours = myDate.getHours();
//extract the minutes component
var minutes = myDate.getMinutes();
//format and display the result on the page:
//check the minutes value is greater than 1 digit
//by being less than 10
if (minutes < 10)
{
```

```
//If it's just a one digit number, then add a 0 in front
minutes = "0" + minutes;
}
var timeString = hours + " : " + minutes;
document.write(timeString);
//Based on my computer clock, this prints:
//14 : 44
```

Set dates

You can also set dates and times as easily as calling the appropriate methods. Here are a few examples.

Use setFullYear() to set a specific date

In this demo, you will set a specific date and then retrieve its day component for display

```
//Create a Date object containing the current date and time
var myDate = new Date();
//set the Date object to a specific date: 31 October, 2011
var mySetDate = myDate.setFullYear(2011, 9, 31);
//Now retrieve the newly set date
var mySetDay = myDate.getDay(mySetDate);
//mySetDay will contain a number between 0 - 6 (Sunday - Saturday).
//Use mySetDay as test case for a switch statement
//to assign the corresponding week day to the number value
switch (mySetDay)
{
case 0:
mySetDay = "Sunday";
```

```
break;
case 1:
mySetDay = "Monday";
break;
case 2:
mySetDay = "Tuesday";
break;
case 3:
mySetDay = "Wednesday";
break;
case 4:
mySetDay = "Thursday";
break;
case 5:
mySetDay = "Friday";
break;
case 6:
mySetDay = "Saturday";
break;
}
//display the result on the page
document.write("The 31st October 2011 is a " + mySetDay);
//If you check the date on a calendar, you'll find that
//the 31st Oct 2011 is indeed a Monday
```

Compare 2 dates

In this demo you will compare 2 Date objects.

```
//Create a Date object

var myDate = new Date();

//set the Date object to a specific date: 31 October, 2011

var mySetDate = myDate.setFullYear(2011, 9, 31);

//Create a second Date object

var now = new Date();

//Make the comparison: if the set date is bigger than today's date

if (mySetDate > now)

{

//the set date is in the future

document.write("The 31st October is in the future");

}

//if the set date is smaller than today's date, it's in the past

else

{

document.write("The 31st October is in the past");

}
```

Math object allows you to write scripts that perform complex mathematical tasks in the blink of an eye.

- **Math.PI** to calculate the circumference of a circle;
- **Math.sqrt()** to calculate a square root value.

In the process, you will learn how to use:

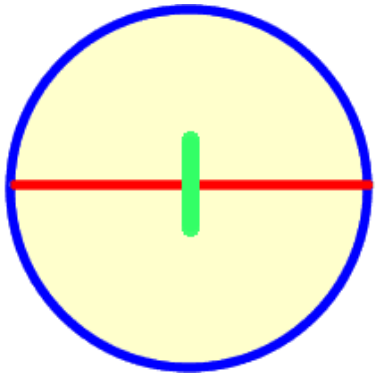
- **parseInt()/parseFloat()** to convert strings to numbers;
- **isNaN()** to check whether a value is or is not a number;

Math.PI

This property stores the value of PI, a mathematical constant that amounts to approximately 3.14159. Here are some examples:

Circle

A circle is a shape with all points the same distance from the center:



- the round border surrounding the shape (the color blue in the graphic above), is the circle **circumference**;
- the line cutting the circle horizontally in half (the color red in the graphic above) is the circle **diameter**;
- each half of the diameter represents a **radius**.

PI is the **ratio of the circumference of a circle to the diameter**.

If you need this value in your JavaScript program, use **Math.PI**. Just try this one out between enclosing <script> tags of an HTML page:

```
//store and display the value of PI:

var pi = Math.PI;

document.write(pi);
```

Example: use Math.PI to find out the circumference of a circle

We're going to build a simple application where the user enters a number representing the radius of a circle, clicks a button, and the program calculates the circumference and displays it on the page.

Keeping in mind that the radius is half the diameter, and that the formula to calculate the circumference is **PI * diameter**, our JavaScript statement to calculate the circumference given the radius will be: **2 * (Math.PI * radius)**.

Prepare an HTML document like this one:

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 12: JavaScript Objects - Math Object</title>
<script type="text/javascript" src="lesson_example.js"></script>
</head>
<body>
<h1>Lesson 12: Circumference calculator</h1>

<p>Enter length of radius: <input type="text" id="txtRadius" /></p>

<p><input type="button" id="btnSubmit" value="Calculate circumference" /></p>

<p id="result"></p>

</body>
</html>
```

The HTML page above references an external JavaScript file, *lesson_ example.js*, and contains an inputbox, a button, and a paragraph where the result of the calculation will be displayed. Make sure these 3 crucial elements have an **id value** so your JavaScript script will have an easy to reach hook on those elements.

Now create a JavaScript document and name it *lesson_example.js*. This file will contain 3 functions:

init() initializes the script by binding the *processAnswer() function* to its appropriate event, that is, **the btnSubmit onclick event**;

processAnswer() checks the answer submitted by the user, calls the function that performs the calculation, and displays the result;

getCircumference(rad) gets passed the radius value as argument, performs the actual calculation, and returns the circumference.

```
//init() function binds the processAnswer() function

//to the onclick event of the button

function init()

{

var myButton = document.getElementById("btnSubmit");

myButton.onclick = processAnswer;

}


/*******************************************/


//Bind the init function to the onload event
```

```
onload = init;

/*****************************************/

//This function checks the user's input,
//calls the function that performs the calculation,
//and displays the result
function processAnswer()
{
//store a reference to the inputbox
var inputBox = document.getElementById("txtRadius");
//get the radius value entered in the inputbox
//make sure to convert the string (text) to a number:
//calculations can only be performed with numbers,
//and the value in the inputbox is processed as string.
//parseInt(string) and parseFloat(string)
//take a string as input and return an integer
//or a floating point number respectively.
var radius = parseInt(inputBox.value);
//get a reference to the paragraph to display result
var result = document.getElementById("result");
//create and initialize a var to store the result
var resultString = "";
//Perform some validation on the user's input:
//if the value entered by the user is not a number
//alert the user.  isNaN(value) takes
//a value as input and returns true if the value is not a number,
```

```
//it returns false if the value is a number.
//the expression below is short for:
//if (isNaN(radius) == true)
if (isNaN(radius))
{
alert("Please, enter a number");
}
else
{
//if the user has correctly entered a number,
//call the function that performs the calculation
//and display the result on the page
var circ = getCircumference(radius);
resultString = "Circumference is: " + circ;
result.innerHTML = resultString;
}
}

/*******************************************/

//This is the function that performs the calculation:
//it takes the radius as input and returns the circumference
function getCircumference(rad)
{
var circumference = 2 * (Math.PI * rad);
return circumference;
```

```
}
```

Save all your files and run the HTML page in the browser. You should see something like the page indicated by following the example link above.
If you click the button without entering any value in the inputbox, or if you enter a letter rather than a number, the program asks you to enter a number value.
As you click the button, the calculation is performed and the result is displayed on the page.

Math.sqrt()

Math.sqrt(number) takes a **number as argument and returns the square root of that number**.
If a negative number is entered as argument, for instance **Math.sqrt(-5)**, the function returns **NaN**, that is, a value that JavaScript does not treat as a number.
Brushing up on our school math, the square root of 25, for example, is that number that multiplied by itself yields 25 as a result, that is, 5.
The formula is: **square of n = n * n**. JavaScript performs this calculation automatically. Here's a quick demo: type the following code between enclosing <script> tags of an HTML page:

```
var squareRoot = Math.sqrt(25);

document.write("The square root of 25 is: " + squareRoot);

//This should print:

//The square root of 25 is: 5
```

Try out: square root calculator

You will build a simple square root calculator. Here are the requirements for this little application:

- the user will be able to enter a number value into an inputbox;
- the user will be able to click a button that calculates the square root of the number entered into the inputbox;
- the result will be displayed on the web page.

Create a fresh HTML page with an inputbox, 1 button, and a paragraph where the result will be displayed. These HTML elements contain **id attributes** that will provide a handy hook for our JavaScript code.

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 12: JavaScript Objects - Math Object</title>
<script type="text/javascript" src="lesson_example2.js"></script>
</head>
<body>
<h1>Lesson 12: Square root calculator</h1>
```

```
<p>Enter a number: <input type="text" id="txtNumber" /></p>

<p><input type="button" id="btnSubmit" value="Calculate square root" /></p>

<p id="result"></p>

</body>
</html>
```

Now create the *lesson12_tryout2.js* file. This file will contain 3 functions:
init() initializes the script by binding the *displaySquare() function* to its appropriate event, that is, **the btnSubmit onclick event**;
displaySquare() checks the answer submitted by the user, calls the function that performs the calculation, and displays the result;
calculateSquare(input) gets passed the number value entered by the user as argument, performs the actual calculation, and returns the the square root.

```
//write the function to initialize the script

function init()

{

//Bind function that processes the result to

//the onclick event of the button

var myButton = document.getElementById("btnSubmit");

myButton.onclick = displaySquare;

}


/*************************************************/


//Bind the init() function to the onload event

onload = init;


/*************************************************/


//Here we call the function that performs
```

```
//the square root calculation, then we format and
//display the result
function displaySquare()
{
//we store the value from the inputbox into a var.
//Notice how we concatenate several methods
//in one statement (be careful with those brackets)
//If you find it a bit confusing, store a reference to the inputbox
//in its own var and then use it to extract its value property
//and finally pass it as argument of parseInt(), like so:
//var inputBox = document.getElementById("txtNumber");
//var inputVal = parseInt(inputBox.value);
var inputVal = parseInt(document.getElementById("txtNumber").value);
//we store a reference to the paragraph
//where the result will be displayed
var result = document.getElementById("result");
//we create the var that stores the result message
var resultMessage = "";
//we ensure the input value is a number:
//if the user did not enter a number, we send an alert
if (isNaN(inputVal))
{
alert("Please, enter a number");
}
else
```

```
{
//If the input is in correct format, we call
//the function that performs the calculation
var squareVal = calculateSquare(inputVal);
//this is a nested if statement: it's inside another
//if statement to perform further checks:
//if squareVal stores a value (if the calculation was successful) -
//- this is short for: if (squareVal == true)
if (squareVal)
{
//we build this message to the user:
resultMessage = "Square root of " + inputVal + " is " +  squareVal;
}
//else, that is, if the calculation was not successful
else
{
//we build a different message to the user.
resultMessage = "Sorry, an error occurred";
}
}
//finally, we display the message using innerHTML
result.innerHTML = resultMessage;
}

/************************************************/
```

```
//This function calculates the square root:

//it takes in the number entered by the user

//and returns the result of the calculation

function calculateSquare(input)

{

var squareVal = Math.sqrt(input);

return squareVal;

}
```

Save all files and preview your work in a browser. You should see something similar to the example indicated by following the link above.
Enter a number into the inputbox and click the button. If you enter anything but a number into the inputbox (or if you leave the inputbox empty), you'll be alerted by a popup box asking you to enter a number. If all goes well, the square root of the number you enter in the inputbox will be displayed on the page.

JavaScript Timing Events

The **global window object** offers a few little functions that are great if you want to create the effect of movement or of passing time on the web.

In this lesson, you will learn about:

- **setTimeout()**;
- **clearTimeout()**;
- **setInterval()**;
- **clearInterval()**.

In the process, you will also learn how to **preload images** with JavaScript and how to **build a simple photo gallery application**.

Use setTimeout()

If your program has a chunk of code that needs to be executed after a specified time, then **setTimeout(actionToExecute, timeInMilliseconds)** is your function.

In this simple example, we will call up an alertbox 2 seconds after page load. Type the code below in enclosing <script> tags on a simple HTML page:

```
//Package the code to insert in the
```

```
//timer in a simple function:

function sayHello()

{

alert("Hello");

}

//Package the timer into its own function

function timeGreeting()

{

//assign the timer to a variable:

//you need to do this so the timer can be

//referenced by another function when you want

//to stop it:

var greeting = setTimeout(sayHello, 2000);

//Notice how the time is set in milliseconds.

}

//Now call the timer function

timeGreeting();
```

Save your work and run it in a browser. After 2 seconds, an alertbox should pop up to greet you.

Use setInterval()

If you need to execute a chunk of code at specified time intervals, then **setInterval(codeToExecute, timeIntervalInMilliseconds)** is your function.

To see it in action in its most annoying manifestation, just replace **setTimeout** with **setInterval** in the previous example and you're done.

Now you should see an alertbox greeting you every 2 seconds! That's enough, just close that window in your browser and let's modify our script so that we can put a stop to that nuisance.

Stop the timer

JavaScript offers 2 handy methods to get rid of timers: **clearTimeout(timerVariable)** if your timer uses setTimeOut(), and **clearInterval(timerVariable)** if your timer uses setInterval(). Let's put *clearInterval()* to good use right away by stopping that annoying alertbox. First of all, add an inputbox to your HTML page, like so:
<input type="button" value="Stop timer" onclick="stopTimer()" />
Next, rewrite the preceding JavaScript code and then add the new function as follows:

```
//Make the variable holding the timer

//global, so that it can be accessed both

//by the function setting the timer and by the function

//stopping the timer:

var greeting;


//sayHello() remains unchanged from the previous example

function sayHello()

{

alert("Hello");

}


//we increase the time interval

//to give more time to the user to

//click the button that stops the alert:

function timeGreeting()

{

greeting = setInterval(sayHello, 5000);

}


timeGreeting();


//package the code that cancels the timer
```

```
//in its own function that will be called when

//the button on the page is clicked:

function stopTimer()

{

//you call clearInterval() and pass the variable

//containing the timer as argument

clearInterval(greeting);

}
```

Save your work and preview it in a browser. Just click the button and the annyoing alertbox disappears: that's much better, great! I leave you to experiment with clearTimeout() on your own.

Try out: a photo gallery application

JavaScript timers are often used to produce animation effects on the web. Photo galleries are one of those widely used applications that often employ JavaScript timers.
It's easy to find a number of sophisticated photo gallery widgets on the web, that you can download and readily plug into your own web page.
However, it's still nice to be able to build a simple prototype, just to really challenge yourself with your newly gained knowledge. In any case, your application can be considered a work in progress, something that evolves and grows as your JavaScript programming skills evolve and grow.
At this stage, your photo gallery will consist of a photo container and 2 buttons: one button to stop the gallery, and one button to restart the gallery. As soon as the page loads, the gallery automatically displays its images one at a time every 2 seconds.
You will also need a few images, possibly all of the same size. For this demo, I prepared four 620px X 378px graphics and saved them in their own *images* folder.
Let's start from the HTML page. This is made of a , an <img> tag, and 2 <input> tags. Also included are: **1) a small style declaration in the <head> of the document** to the effect that the element that contains the gallery will be centered and sized to the same width and height as the gallery graphics; and **2) a reference to an external JavaScript file at the bottom of the <body> element of the HTML page**. This location is the most appropriate one because we need to make sure the HTML page and, most importantly, the image referenced by the <img> tag, are fully loaded before the script kicks in.

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson ab: JavaScript Timing Events</title>
<style type="text/css">
  #gallery
        {
```

```
        width:620px;
        height:378px;
        margin: 0 auto;
        border:2px solid #ccc;
        }
</style>
</head>
<body>
<h1>Lesson ab: My Photo Gallery</h1>

<div id="gallery">

<img src="images/1.jpg" alt="Photo gallery image" id="photo" />

<input type="button" id="btnStart" value="Restart gallery" />

<input type="button" id="btnStop" value="Stop gallery" />

</div>

<script type="text/javascript" src="lesson17.js"></script>

</body>
</html>
```

Now prepare the **lessonab.js** file. Your JavaScript code consists of 3 functions:
init() contains initialization routines: it preloads the photo gallery images so that they will be ready for display as soon as the script calls them. In addition, it binds event handlers to the 2 button elements to stop and restart the gallery;
startGallery() displays each graphic in the gallery every 2 seconds;
stopGallery() stops the gallery so that the photo that comes next with respect to the current photo is not displayed.
Furthermore, the code contains a few global variables at the top that need to be accessed by all the functions in the script. Let's get started.

```
//Global variables: a reference to the

//photo currently displayed on the page,

var curImage = document.getElementById("photo");

//a variable to store the timer,

var galleryStarter;

//a variable to store a true/false value indicating

//to the program whether the gallery is on or off,

var isGalleryOn = true;
```

```
//an array containing 4 strings representing the filepaths
//to the image files in the images folder,
var images = ["images/1.jpg", "images/2.jpg", "images/3.jpg", "images/4.jpg"];
//an empty array that will be filled with 4 preloaded
//image objects: the src property of these image objects will correspond
//to the filepaths contained in the images[] array,
var preloadedImgs = [];
//a variable that works as our counter to
//advance from one image to the next.  It starts at 0.
var counter = 0;

/*************************************/

//Init() starts with the image preloading routine.
//First fill the preloadedImgs[] array with a
//number of image objects corresponding to the length
//of the images[] array:
function init()
{
for (var i = 0; i < images.length; i++)
{
preloadedImgs[i] = new Image();
}
//now assign the value of the strings contained in the
//images[] array to the src property of each image object
```

```
//in the preloadedImgs[] array, using one more loop:
for (var i = 0; i < preloadedImgs.length; i++)
{
preloadedImgs[i].src = images[i];
}
//Now, assign event handlers to the 2 buttons
//to restart and stop the gallery:
var btnStart = document.getElementById("btnStart");
var btnStop = document.getElementById("btnStop");
btnStart.onclick = startGallery;
btnStop.onclick = stopGallery;
//Finally, check the isGalleryOn flag is true.  If it is
//call the function that starts the gallery:
if (isGalleryOn)
{
startGallery();
}
}

/*****************************************/

//Assign the init() function to the onload event
onload = init;

/*****************************************/
```

```
//startGallery() contains the functionality
//to extract the photos from the preloadedImgs[] array
//for display and to set the timer in motion:
function startGallery()
{
//extract the image filepath using the counter
//variable as array index and assign it to the src
//property of the curImage variable:
curImage.src = preloadedImgs[counter].src;
//advance the counter by 1:
counter ++;
//if the counter has reached the length of the
//preloadedImgs[] array, take it back to 0, so the
//photo gallery redisplays the images from the start:
if (counter == preloadedImgs.length)
{
counter = 0;
}
//Set the timer using this same function as one
//of the arguments and 2000 (2 milliseconds) as the other argument.
galleryStarter = setTimeout("startGallery()", 2000);
//Signal that the gallery is on to the rest of the program:
isGalleryOn = true;
}

/***********************************************/
```

```
//stopGallery() uses clearTimeout()

//to stop the gallery

function stopGallery()

{

clearTimeout(galleryStarter);

//signal that the gallery has stopped to the

//rest of the program:

isGalleryOn = false;

}
```

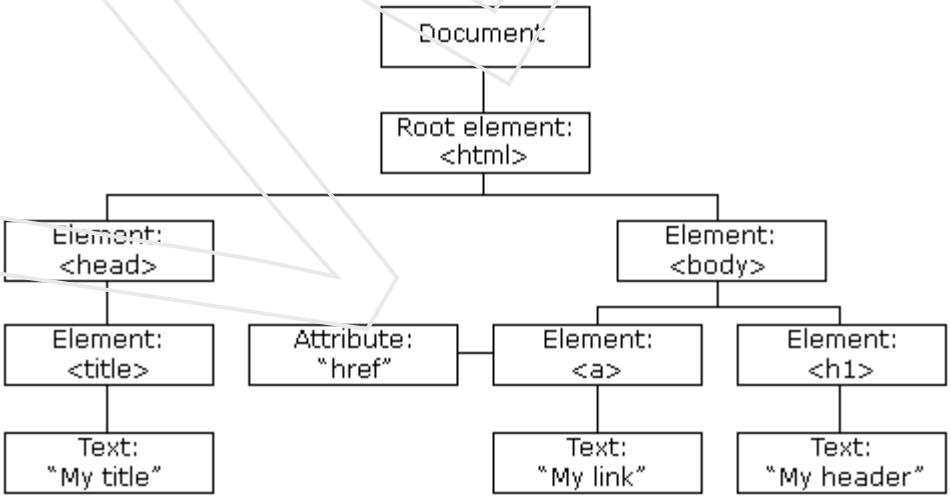
Save your work and preview it in a browser.

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page

- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

DOM

The DOM is a W3C (World Wide Web Consortium) standard
The DOM defines a standard for accessing documents:
"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
The W3C DOM standard is separated into 3 different parts:
Core DOM - standard model for all document types
XML DOM - standard model for XML documents
HTML DOM - standard model for HTML documents

HTML DOM

A HTML webpage is represented via a HTML Document Object Model (DOM). The HTML DOM is defined via a W3C standard This HTML DOM defines a standard way to access and manipulate HTML documents. You can use JavaScript to modify the HTML DOM.

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

Manipulation HTML with JavaScript

You can for example search via JavaScript for certain elements and modify their properties. The following example shows how to find HTML elements with a certain div class and set the display property to hide / show them.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<script type="text/javascript">

function hideshow() {

  var allHTMLTags = new Array();

  //Create Array of All HTML Tags
  var allHTMLTags=document.getElementsByTagName("*");

  //Loop through all tags using a for loop
```

```
  for (i=0; i<allHTMLTags.length; i++) {

  //Get all tags with the specified class name.
    if (allHTMLTags[i].className=="revhistory") {

  // hide or display the elements
      if (allHTMLTags[i].style.display == "none") {
        allHTMLTags[i].style.display = "";
      } else {
        allHTMLTags[i].style.display = "none";
      }
    }
  }
}
</script>

</head>
<body>

<div class="revhistory"> This is the text which will be manipulated via JavaScript</div>

<input type="button" value="Show/ Hide"
onclick="hideshow()">


</body>
</html>
```

Examples

Create a dynamic link using the webpage information

The following shows how an URL for an link can be dynamically created via JavaScript. This examples uses JavaScript to create a link for the website "digg".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>

/* This script will read the current URL and the title of the document
and submit this webpage to the social bookmarking website. */

<script type="text/javascript">
function toDiggUrl(){
  var result = "http://veinstitution.com/submit?phase=2&url=";
  result += encodeURIComponent(location.href);
  result += "&title=";
  result += encodeURIComponent(document.title);
  result += "&topic=programming";
  window.location = result;
}
</script>
</head>

<body>
```

```
<b>Here we write some information to the HTML page.</b>
<BR>
Write the current URL to this HTML document
<script type="text/javascript">
document.writeln(location.href);
</script>

<BR>
<BR>
<b>Encodes the current URL so that special characters will get
encoded</b>
<BR>
<script type="text/javascript">
document.writeln(encodeURIComponent(location.href))
</script>


<BR>
<BR>
<b>More fancy stuff, evalute page information and send this to a
social bookmarking website (veinstitution)</b>
<BR>


<table border="0" cellpadding="10" cellspacing="0" width="100%">
  <tr>
    <td width="320">If you press the hyperlink below the JavaScript
    will run <BR>
    <a href="javascript: toVeinstitutionUrl()">Veinstitution this site</a></td>
  </tr>
</table>


</body>
</html>
```

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).
In the DOM, all HTML elements are defined as **objects**.
The programming interface is the properties and methods of each object.
A **property** is a value that you can get or set (like changing the content of an HTML element).
A **method** is an action you can do (like add or deleting an HTML element).

Example

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

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

In the example above, getElementById is a **method**, while innerHTML is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the id of the element.
In the example above the getElementById method used id="demo" to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the **innerHTML** property.
The innerHTML property is useful for getting or replacing the content of HTML elements.

The HTML DOM Document

In the HTML DOM object model, the document object represents your web page.
The document object is the owner of all other objects in your web page.
If you want to access objects in an HTML page, you always start with accessing the document object.
Below are some examples of how you can use the document object to access and manipulate HTML.
The next chapters demonstrate the methods.

Finding HTML Elements

Method	Description
document.getElementById()	Find an element by element id
document.getElementsByTagName()	Find elements by tag name
document.getElementsByClassName()	Find elements by class name

Changing HTML Elements

Method	Description
element.innerHTML=	Change the inner HTML of an element
element.attribute=	Change the attribute of an HTML element
element.setAttribute*(attribute,value)*	Change the attribute of an HTML element
element.style.*property*=	Change the style of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement()	Create an HTML element
document.removeChild()	Remove an HTML element
document.appendChild()	Add an HTML element
document.replaceChild()	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick=function(){*code*}	Adding event handler code to an onclick event

Finding HTML Objects

The first HTML DOM Level 1 (1998). defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.
Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Method	Description	DOM
document.anchors	Returns all <a> with a value in the name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <image> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements value in href	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1

document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	

HTML DOM Elements

Study how to find and access HTML elements in an HTML page.

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.
To do so, you have to find the elements first. There are a couple of ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by HTML object collections

Finding HTML Elements by Id

The easiest way to find HTML elements in the DOM, is by using the element id.

This example finds the element with id="intro":

Example

```
var x = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in x).

If the element is not found, x will contain null.

Finding HTML Elements by Tag Name

This example finds the element with id="main", and then finds all <p> elements inside "main":

Example

```
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
```

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name. Use this method:
document.getElementsByClassName("intro");
The example above returns a list of all elements with class="intro".
Finding elements by class name does not work in Internet Explorer 5,6,7, and 8.

Finding HTML Elements by HTML Object Collections

This example finds the form element with id="frm1", in the forms collection, and displays all element values:

Example

```
var x = document.getElementById("frm1");
var text = "";
for (var i = 0; i <x.length; i++) {
   text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

The following HTML objects (and object collections) are also accessible:

- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- document.head
- document.images
- document.links
- document.scripts
- document.title

Changing the HTML Output Stream

JavaScript can create dynamic HTML content:
Date: Mon May 19 2014 13:04:40 GMT+0530 (India Standard Time)
In JavaScript, document.write() can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>
```

Never use document.write() after the document is loaded. It will overwrite the document.

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the **innerHTML** property.
To change the content of an HTML element, use this syntax:
document.getElementById(*id*).innerHTML = *new HTML*

This example changes a <p> element:

Example

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

This example changes the content of an <h1> element:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="header">Old Header</h1>

<script>
var element = document.getElementById("header");
element.innerHTML = "New Header";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an <h1> element with id="header"
- We use the HTML DOM to get the element with id="header"
- A JavaScript changes the content (innerHTML) of that element

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:
document.getElementById(*id*).*attribute=new value*

This example changes the value of the src attribute of an <img> element:

Example

```
<!DOCTYPE html>
<html>
<body>

<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

Example explained:

- The HTML document above contains an <img> element with id="myImage"
- We use the HTML DOM to get the element with id="myImage"
- A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg"

HTML DOM - Changing CSS

The HTML DOM allows JavaScript to change the style of HTML elements.

Changing HTML Style

To change the style of an HTML element, use this syntax:

document.getElementById(*id*).style.*property=new style*

The following example changes the style of a <p> element:

Example

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>

<p>The paragraph above was changed by a script.</p>

</body>
</html>
```

Using Events

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

- An element is clicked on
- The page has loaded
- Input fields are changed

This example changes the style of the HTML element with id="id1", when the user clicks a button:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

HTML DOM Style Object Reference

For all HTML DOM style properties, look at our complete HTML DOM Style Object Reference.

JavaScript HTML DOM Events

HTML DOM allows JavaScript to react to HTML events.

Example

Mouse Over Me
Click Me

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

onclick=*JavaScript*

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

In this example, the content of the <h1> element is changed when a user clicks on it:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>
</body>
</html>
```

In this example, a function is called from the event handler:

Example

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id) {
   id.innerHTML="Ooops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text!</h1>
</body>
</html>
```

HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

```
<button onclick="displayDate()">Try it</button>
```

In the example above, a function named *displayDate* will be executed when the button is clicked.

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick=function(){displayDate()};
</script>
```

In the example above, a function named *displayDate* is assigned to an HTML element with the id="myBtn".
The function will be executed when the button is clicked.

The onload and onunload Events

The onload and onunload events are triggered when the user enters or leaves the page.
The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
The onload and onunload events can be used to deal with cookies.

Example

```
<body onload="checkCookies()">
```

The onchange Event

The onchange event are often used in combination with validation of input fields.
Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

Example

```
<input type="text" id="fname" onchange="upperCase()">
```

The onmouseover and onmouseout Events

The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.

Example

A simple onmouseover-onmouseout example:

Mouse Over Me

The onmousedown, onmouseup and onclick Events

The onmousedown, onmouseup, and onclick events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

Example

A simple onmousedown-onmouseup example:

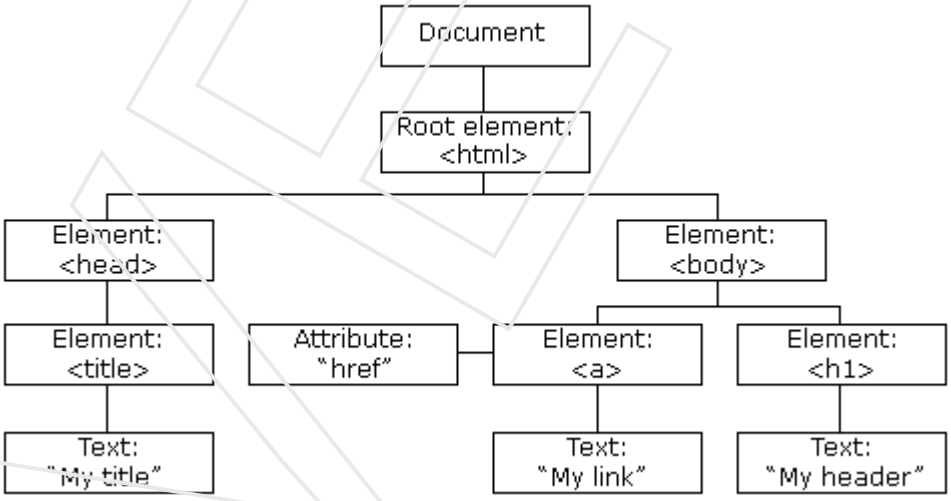
Click Me

With the HTML DOM, you can navigate the node tree using node relationships.

DOM Nodes

According to the W3C HTML DOM standard, everything in an HTML document is a node:

- The entire document is a document node
- Every HTML element is an element node
- The text inside HTML elements are text nodes
- Every HTML attribute is an attribute node
- All comments are comment nodes



With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.
New nodes can be created, and all nodes can be modified or deleted.

Node Relationships

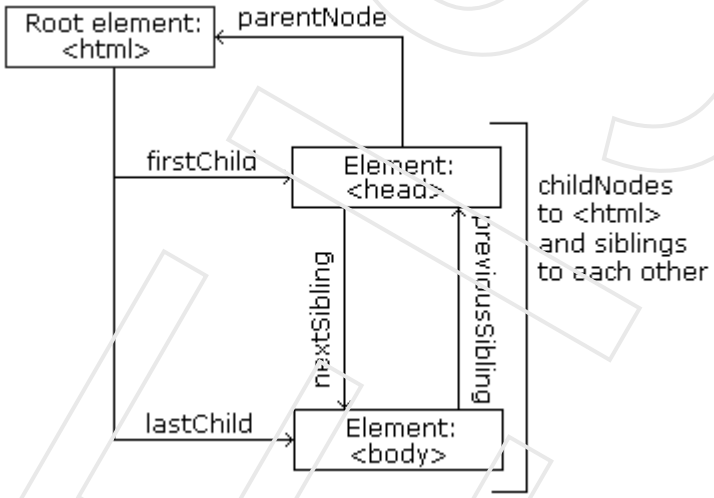
The nodes in the node tree have a hierarchical relationship to each other.
The terms parent, child, and sibling are used to describe the relationships.
In a node tree, the top node is called the root (or root node)
Every node has exactly one parent, except the root (which has no parent)
A node can have a number of children
Siblings (brothers or sisters) are nodes with the same parent

```
<html>

  <head>
    <title>DOM Tutorial</title>
  </head>


  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>

</html>
```



From the HTML above you can read:

- <html> is the root node
- <html> has no parents
- <html> is the parent of <head> and <body>
- <head> is the first child of <html>
- <body> is the last child of <html>

and:

- <head> has one child: <title>
- <title> has one child (a text node): "DOM Tutorial"
- <body> has two children: <h1> and <p>
- <h1> has one child: "DOM Lesson one"
- <p> has one child: "Hello world!"
- <h1> and <p> are siblings

Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

- parentNode
- childNodes[*nodenumber*]
- firstChild
- lastChild
- nextSibling
- previousSibling

Note:

A common error in DOM processing is to expect an element node to contain text.
In this example: **<title>DOM Tutorial</title>**, the element node <title> does not contain text. It contains a **text node** with the value "DOM Tutorial".
The value of the text node can be accessed by the node's **innerHTML** property, or the **nodeValue**.

Child Nodes and Node Values

In addition to the innerHTML property, you can also use the childNodes and nodeValue properties to get the content of an element.

The following example collects the node value of an <h1> element and copies it into a <p> element:

Example

```
<html>
<body>

<h1 id="intro">My First Page</h1>

<p id="demo">Hello!</p>

<script>
var myText = document.getElementById("intro").childNodes[0].nodeValue;
document.getElementById("demo").innerHTML = myText;
</script>

</body>
</html>
```

In the example above, getElementById is a method, while childNodes and nodeValue are properties.

Using the firstChild property is the same as using childNodes[0]:

Example

```
<html>
<body>

<h1 id="intro">My First Page</h1>

<p id="demo">Hello World!</p>

<script>
myText = document.getElementById("intro").firstChild.nodeValue;
document.getElementById("demo").innerHTML = myText;
</script>

</body>
</html>
```

DOM Root Nodes

There are two special properties that allow access to the full document:

- document.body - The body of the document
- document.documentElement - The full document

Example

```
<html>
<body>

<p>Hello World!</p>
<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the <b>document.body</b> property.</p>
</div>

<script>
alert(document.body.innerHTML);
</script>

</body>
</html>
```

Example

```
<html>
<body>

<p>Hello World!</p>
<div>
<p>The DOM is very useful!</p>
<p>This example demonstrates the <b>document.documentElement</b> property.</p>
</div>
```

```
<script>
alert(document.documentElement.innerHTML);
</script>

</body>
</html>
```

The nodeName Property: The nodeName property specifies the name of a node.

- nodeName is read-only
- nodeName of an element node is the same as the tag name
- nodeName of an attribute node is the attribute name
- nodeName of a text node is always #text
- nodeName of the document node is always #document

Note: node Name always contains the uppercase tag name of an HTML element.

The nodeValue Property : The nodeValue property specifies the value of a node.

- nodeValue for element nodes is undefined
- nodeValue for text nodes is the text itself
- nodeValue for attribute nodes is the attribute value

The nodeType Property

The nodeType property returns the type of node. nodeType is read only.

The most important node types are:

Element type	NodeType
Element	1
Attribute	2
Text	3
Comment	8
Document	9

Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
```

```
<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var element = document.getElementById("div1");
element.appendChild(para);
</script>
```

Example Explained

This code creates a new <p> element:

```
var para = document.createElement("p");
```

To add text to the <p> element, you must create a text node first. This code creates a text node:

```
var node = document.createTextNode("This is a new paragraph.");
```

Then you must append the text node to the <p> element:

```
para.appendChild(node);
```

Finally you must append the new element to an existing element.

This code finds an existing element:

```
var element = document.getElementById("div1");
```

This code appends the new element to the existing element.

```
element.appendChild(para),
```

Creating new HTML Elements - insertBefore()

The appendChild() method in the previous example, appended the new element as the last child of the parent.
If you don't want that you can use the insertBefore() method:

Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
```

```
para.appendChild(node);

var element = document.getElementById("div1");
var child = document.getElementById("p1");
element.insertBefore(para,child);
</script>
```

Removing Existing HTML Elements

To remove an HTML element, you must know the parent of the element:

Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>
```

Example Explained

This HTML document contains a element with two child nodes (two <p> elements):

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>
```

Find the element with id="div1":

```
var parent = document.getElementById("div1");
```

Find the <p> element with id="p1":

```
var child = document.getElementById("p1");
```

Remove the child from the parent:

```
parent.removeChild(child);
```

It would be nice to be able to remove an element without referring to the parent.
But sorry. The DOM needs to know both the element you want to remove, and its parent.
Here is a common workaround: Find the child you want to remove, and use its parentNode property to find the parent:

```
var child = document.getElementById("p1");
child.parentNode.removeChild(child);
```

Replacing HTML Elements

To replace an element to the HTML DOM, use the replaceChild() method:

Example

```
<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);

var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.replaceChild(para,child);
</script>
```

HTML DOM Node List

The getElementsByTagName() method returns a **node list**. A node list is an array of nodes.

The following code selects all <p> nodes in a document:

Example

```
var x = document.getElementsByTagName("p");
```

The nodes can be accessed by index number. To access the second <p> you can write:

```
y = x[1];
```

Note: The index starts at 0.

HTML DOM Node List Length

The length property defines the number of nodes in a node-list:

Example

```
<p>How many paragraphs in this document?</p>
```

```
myNodelist = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML = myNodelist.length;
```

Example explained:

1. Get all <p> elements into a node list (array)
2. Display the length of the node list

Reading META Tags via JavaScript

You can use JavaScript to read existing meta tags from the webpage. The following reads the content of the meta tag description from the webpage.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>

<head>
<meta name="description" content="Tutorials from www.vogella.com!">

<script type="text/javascript">
function getMetaDescription () {
  var metaElements = document.all ?
    document.all.tags('META') :
    document.getElementsByTagName ?
    document.getElementsByTagName ('META') : new Array();
  var metaKeywords = new Array();
  var i = 0;
  for (var m = 0; m < metaElements.length; m++)
    if (metaElements[m].name == 'description')
      metaKeywords[i++] = metaElements[m].content;
  return metaKeywords;
}
</script>
</head>

<h2>
This will write the meta tag description to this page
</h2>

<body>

<script type="text/javascript">
  document.write(getMetaDescription());
</script>


</body>
</html>
```

Making AJAX Calls

One core feature of 21st century web applications is their fantastic responsiveness. A pioneer in the business of making the web more and more similar to a desktop application has been *Google*: as soon as you start typing a character in a Google box, all kinds of suggestions and search results keep popping up to speed up your search; and again, just type something using Google maps, and you're instantly catapulted to almost any street on the planet.
Behind this amazing feat of human cleverness is the coming together of a few technologies collectively known as **AJAX**.

AJAX

The acronym AJAX stands for **Asynchronous JavaScript And XML**.

In actual fact, it's a combination of internet standards made up of:

- **standards-based presentation** using **HTML and CSS**;
- **dynamic display** using the **DOM**;
- **data interchange** and manipulation using **XML**;
- **asynchronous** data retrieval using the **XMLHttpRequest object**;
- **JavaScript** magic to orchestrate the whole process.

In non-AJAX web applications, the interaction between servers and clients can be a tedious business:

1. a user action from the client sends a request to the web server via HyperText Transfer Protocol (HTTP);
2. the web server receives the request and processes it by invoking server-side scripts, database data, etc., and sends a response back to the client via HTTP;
3. the client browser receives the response and loads the entire updated page.

Having to go from browser to server and back again each time a user requests some piece of data from the server, in addition to undergoing an entire page refresh at each update, can be quite stressful on servers and on users alike.

AJAX helps in at least 2 respects:

piecemeal page updates and **asynchronous communication** between server and client.

What this means in a few words, is that every time the user interacts with the web page, only those bits that need updating are refreshed, not the entire page. Furthermore, the fact that AJAX operations are performed asynchronously, means that during the time that it takes for the server to respond, the page is not locked and the user can still interact with the website.

Make an AJAX request

An **AJAX request** is made using the **XMLHttpRequest object** and its **open() and send() methods**. This is supported by all major browsers. However, older browsers, namely older varsions of Microsoft Internet Explorer (vesions 5 and 6), support an **ActiveXObject**. This little hurdle is easily overcome by testing for feature support in the script.

The **open(retrievalMethod, url, bool)** method has 3 arguments:

1. **retrievalMethod**: this can either be a **GET** (used to fetch data from the server), or a **POST** (used to send data to the server);
2. **url**: this is the location where the data is made available. It can be a text file, an XML document, or a server-side script that processes data coming from a database;
3. **bool**: this is a true/false value. If it's false the request is made synchronously, if it's true the request is made asynchronously, which is what we usually want.

The XMLHttpRequest object has an **onreadystatechange** property that deals with the response from the server. This proceeds over the following 5 stages:

- 0) the **request is uninitialized** because **open() has not been called**;
- 1) the **request is specified**, but **send() has not been called yet**;
- 2) the **request is being sent**, because now **send() has been called**;
- 3) the **response is being received, but not yet completed**;
- 4) the **response is complete and data is available for manipulation and display**.

Upon completion (stage 4), the XMLHttpRequest object's **status property** gets assigned an **HTTP status code** that describes the result of the request as follows:

- **200**: success!
- **401**: unauthorized - authentication is required and was not provided;
- **403**: forbidden - the server refuses to respond;
- **404**: not found - the requested resource cannot be found.

This is a simple code snippet that translates what's just been said into practice:

```
//Global variable to store the XMLHttpRequest object

var myRequest;

//Package the request into its own function

function getData()

{

//Feature-testing technique to make sure the browser

//supports the XMLHttpRequest object

if (window.XMLHttpRequest)

{

//create a new instance of the object

myRequest = new XMLHttpRequest();

}
```

```
//else - the XMLHttpRequest object is not supported:
//create an instance of ActiveXObject
else
{
myRequest = new ActiveXObject("Microsoft.XMLHTTP");
}
//Use the open(retrievalMethod, "myDataFile.txt", bool) method
myRequest.open("GET", url, true);
//Use send() with a null argument - we have nothing to send:
myRequest.send(null);
//attach a function to handle onreadystatechange,
//that is, the response from the server:
myRequest.onreadystatechange = getData;
}
```

Handle the AJAX response

If the client browser requests text data, the server response is automatically stored in the **responseText property**.

If the client browser requests data in XML format, the server response is stored in the **responseXML property**.

Here's a code snippet that illustrates this in practice.

```
//Package the response-handling code in a function
function getData()
{
//Make sure the response is at the completion stage (4):
if (myRequest.readyState ===4)
```

```
{

//if it is, make sure the status code is 200 (success):

if (myRequest.status === 200)

{

//if all is well, handle the response:

var text = myRequest.responseText;

alert(text);

}

}

}
```

Let's tackle two examples that include both request and response using AJAX. The first example will be dealing with a response in plain text format, the second example with a response in XML format.

AJAX request - response example: plain text response

To follow along, you need a server and 1 simple text file uploaded to the same server that hosts your JavaScript code.
The page you're going to build allows the user to click a button to display some text stored in your text file using AJAX.
Here's a simple HTML page with a <header> tag and a button element:

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 18: Making AJAX Calls</title>

</head>
<body>
<h1>Lesson 18: Making AJAX Calls - Plain Text Response</h1>

<div>

<h2 id="myHeader">Click the button to call your data</h2>

<input type="button" value="Click Me!" onclick="getText('lesson18_test.txt')" />

</div>

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

```
//JavaScript AJAX code here

</script>

</body>
</html>
```

Here's the JavaScript code that goes in enclosing <script> tags in the HTML page above:

```
//Prepare the global variable for the request

var myRequest;

//Write the getText(url) function

function getText(url)

{

//check support for the XMLHttpRequest object

if (window.XMLHttpRequest)

{

myRequest = new XMLHttpRequest();

}

//else, create an ActiveXObject for IE6

else

{

myRequest = new ActiveXObject("Microsoft.XMLHTTP");

}

//Call the open() method to make the request

myRequest.open("GET", url, true);

//Send the request

myRequest.send(null);

//Assign the getData() function to the
```

```
//onreadystatechange property to handle server response
myRequest.onreadystatechange = getData;
}

/*****************************************/

//This function handles the server response
function getData()
{
//Get a reference to the header element where
//the returned result will be displayed
var myHeader = document.getElementById("myHeader");
//Check the response is complete
if (myRequest.readyState ===4)
{
//Check the status code of the response is successful
if (myRequest.status === 200)
{
//Store the response
var text = myRequest.responseText;
//Assing the returned text to the nodeValue
//property of the header element (you can also use
//innerHTML here if you feel it simplifies your task)
myHeader.firstChild.nodeValue = text;
}
}
```

```
}
```

Save your work and preview it in a browser. You should see something like the page indicated by following the example link above.
Notice how the result is returned from the server with no page flashing, no change of URL, no whole page refresh: that's the AJAX magic.

AJAX request - response example: XML format response

The page you're going to build in this second example allows the user to click a button to display a series of programming languages coming from an XML file using AJAX.
To follow along in this exercise, prepare a simple XML document and an HTML page.
Here's what my XML document looks like:

```
<?xml version="1.0" ?>

<languages>
 <language>PHP</language>
 <language>Ruby On Rails</language>
 <language>C#</language>
 <language>JavaScript</language>
</languages>
```

Save the file above as **lessonbc.xml** in the same location as your HTML page.
Here's the HTML page:

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson bc: Making AJAX Calls</title>

</head>
<body>
<h1>Lesson bc: Making AJAX Calls - XML Response</h1>

<div id="test">

<h2>Click the button to get a list of programming languages</h2>

<input type="button" value="Click Me!" onclick="loadXml('lessonbc_test.xml')"
/>

</div>

<script type="text/javascript">

//JavaScript AJAX code here
```

```
</script>

</body>
</html>
```

Now proceed with the JavaScript code as follows:

```
var myRequest;

//The greatest part of the loadXML() function

//is similar to the previous example:

function loadXML(url)

{

if (window.XMLHttpRequest)

{

myRequest = new XMLHttpRequest();

}

else

{

myRequest = new ActiveXObject("Microsoft.XMLHTTP");

}

myRequest.open("GET", url, true);

myRequest.send(null);

myRequest.onreadystatechange = getData;

}

/*****************************************/

function getData()

{

//Get a reference to the div element
```

```
//where the returned data will be displayed:
var myDiv = document.getElementById("test");
//The part of the code that checks the response
//is the same as the previous example:
if (myRequest.readyState ===4)
{
if (myRequest.status === 200)
{
//Use the responseXML property to catch
//the returned data, select the xml tag
//called language, and store returned
//language items in an array variable:
var languages = myRequest.responseXML.getElementsByTagName("language");
//Loop over each array item containing the data
//and create a <p> element to contain each returned
//piece of data.  Notice the use of firstChild.data
//to retrieve the value of the XML <language> tag:
for (var i = 0; i < languages.length; i++)
{
var paragraph = document.createElement("p");
myDiv.appendChild(paragraph);
paragraph.appendChild(document.createTextNode(languages[i].firstChild.data));
//You can also assign the returned result to myDiv.innerHTML
//if you feel it would be simpler.
}
}
```

```
}

}
```

Save your work and preview it in a browser. You should see something like the page indicated by following the example link above.

Introduction to jQuery

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto: **Write less, do more**.

jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code. Here is the list of important core features supported by jQuery:

- **DOM manipulation:** The jQuery made it easy to select DOM elements, traverse them and modifying their content by using cross-browser open source selector engine called **Sizzle**.
- **Event handling:** The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support:** The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.
- **Animations:** The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight:** The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).
- **Cross Browser Support:** The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology:** The jQuery supports CSS3 selectors and basic XPath syntax.

Uses of jQuery

Here are some good reasons why using jQuery in your JavaScript projects is a great idea.

jQuery is:

- extremely light-weight (just 31 kb in its latest minified and zipped production-ready version);
- it's free and very easy to include in your projects: just download its latest version from the jQuery website, or use an online Content Delivery Network;
- it's continually upgraded, maintained and documented by a dedicated community of great developers. This ensures high quality and support on the internet;
- it helps overcoming inconsistencies in the way some JavaScript features are implemented across different browsers;
- and last but not least, it offers a wealth of ready-made animation effects that are a joy to use.

Include jQuery in my website?

Including jQuery in your project is fast and easy. You have the choice of downloading a copy of the library and save it locally, or you can use a **Content Delivery Network (CDN)** to serve jQuery to your website on the Internet.

Go to the jQuery website download page

Include a local copy of jQuery

If you prefer having a local copy of jQuery:

1. go to the *Download jQuery* section of the download page and click on the **minified version of the latest release** of jQuery. At the time of writing this is **version 1.7**;
2. copy and paste the code in a text file;
3. save the file in your web project directory (better if you save it in its own subfolder);
4. in your html page include the appropriate <script> tags in the following order:

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 19: Introduction to jQuery</title>

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

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

</head>
<body>
<h1>Lesson 19: Introduction to jQuery</h1>

</body>
</html>
```

The reference to jQuery comes before your own JavaScript code file. In fact, your code needs to find jQuery already fully loaded to be able to use it.

Include a hosted copy of jQuery

If you decide to opt for a CDN to serve up a copy of jQuery to your website, then follow these easy steps:
go to the *CDN Hosted jQuery* section of the download page and pick one of the different CDN services available;
copy the URL of your CDN of choice and paste it as the value of the **<script> tag's src attribute**, like so:
<script type="text/javascript" src"yourCDNUrlAddressjQueryFile.js"></script>
place a <script> tag with a reference to your own JavaScript file underneath the jQuery <script> tag, like you did in the previous example.

Make sure jQuery is working

Now the jQuery library can be used in your project. To make sure things work as expected, create a text file and call it *jquery_test.js*. Enter the following code:

```
//Most of your code goes inside

//the jQuery ready() function:

$(document).ready(function()

{

alert("jQuery is working!");

});

//Make sure you keep track of braces, brackets, and semi-colons ( ; )
```

Save all your files and run the HTML page in a browser. If you typed the code exactly like the example above, you should be greeted by ... well, the good old alert box.
The **$ sign** is an alias for jQuery. You could replace it with the keyword *jQuery* and your jQuery-powered JavaScript code would work just fine. However, less typing is jQuery's (and all coders') motto, so using *$* is the accepted convention.
.ready(function() { //JavaScript code here ... });
is where most of your JavaScript code will be placed. This is jQuery's clever way of making sure your code runs when the document is ready for it: that is, when the html is fully loaded.

Access DOM elements using jQuery

jQuery appeals so much to web designers also because it enables us to select DOM elements in code just like we do with CSS. Let's see it in action.

Select html elements by id

Remember our old document.getElementById()? Well, jQuery makes selection by id much shorter. Here's how it's done:

```
$(document).ready(function()

{

var myDiv = $("#myDiv");

});
```

The *$ sign* in front of *("#myDiv")* can be translated like: "Hey, browser, get me the element with the id of myDiv!"
As you can see, the id attribute is grabbed by using the # sign, the same as CSS.

Select html elements by tag name

You also came across document.getElementsByTagName(). This method allows us to grab all html elements by their tag.
Look how jQuery makes it a lot snappier:

```
$(document).ready(function)

{

//store all images on the page in an array

var images = $("img");

});
```

As you can see, the magic $ does most of the talking.

Select html elements by class name

A quick way to target html elements that share the same class is as follows:

```
$(document).ready(function()

{

var redElements = $(".classRed");

});
```

Now *redElements* contains an array of all the html elements on your page with a class of *classRed*.
Once again, notice how jQuery facilitates selection by using the same notation you use in your **CSS declarations to target classes: .className**.

Select html elements using jQuery filters

jQuery enables you to be as precise as a surgeon when targeting html elements and attributes on the page. jQuery offers a **filtering syntax to query the DOM** that is both simple and efficient.
Here's a list of the most common filters:

Filter	Description
:first	The first item in a matched set `$('p:first')` returns the first paragraph.
:last	The last item in a matched set `$('p:last')` returns the last paragraph.
:odd	The odd-indexed items in a matched set

	`$('tr:odd')` returns table rows indexed at 1, 3, etc.
:even	The even-indexed items in a matched set `$('tr:even')` returns table rows indexed at 0, 2, etc.
:has	finds elements having the child element specified `$('p:has(span)')` returns all paragraphs containing a span element.
:eq	returns the element with the matched index `$('p:eq(1)')` returns the second paragraph starting to count at 0.
:contains(x)	Returns all elements containing the text x `$('div:contains(foo)')` targets all divs with the text 'foo'.

A complete list of all selection methods and filters is available on the jQuery website. Have a look and experiment with the available code samples to gain more familiarity with the new approach.

Assign event handlers using jQuery

Event handlers are code blocks, usually functions, that specify what your application should do when a specified event like, for example, a button click, occurs.
So far, you've used 2 approaches:

- **hard-wiring a function to the html element itself**: `<input type="button" onclick="doSomething()" />`
- **assigning a function to the DOM object's appropriate property**: `myElement.onclick = doSomething`;

Both approaches are fine. However, the first one sins against the **separation of concerns principle**. In fact, it mixes in JavaScript code with HTML code.
The second approach complies with the separation of concerns principles, but comes short in case we want to attach more than a function to the same element.
There's actually a third approach that overcomes all shortcomings but one: it's implemented differently in Internet Explorer browsers with respect to all other major browsers.
The good news is: jQuery is here to make event handling quick and easy. Here are a few jQuery approaches to choose from.

The bind() method

bind(eventType, handler) works as follows.

Suppose you have an html element with an id of *myElement*, and a function called *sayHello()*. You want sayHello() to kick in when the user clicks on the html element. This is how you achieve this using bind():

```
$(document).ready(function()

{

//create the sayHello function

function sayHello()

{

alert("Hello jQuery");
```

```
}

//Attach the handler using .bind():

myElement.bind('click', sayHello);

});
```

You can find more details on the bind() method on http://api.jquery.com/bind/.

The ready-made approach

jQuery offers some ready-made handlers corresponding to JavaScript events. Here's a list of the most widely used ones:

Handler	Description
click()	Binds a handler to an **onclick** event.
hover()	Binds a handler to the **onmouseover** and **onmouseout** events.
change()	Binds a handler to the **onchange** event (when the content of a field changes).
select()	Binds a handler to the **onselect** event (when text is selected).
submit()	Binds a handler to the **onsubmit** event of a form element.
focus()	Binds a handler to the **onfocus** event (when an element gets focus).
keypress()	Binds a handler to the **onkeypress** event (when a key on the computer keyboard is pressed).

Assuming you have an html element with an id attribute of *myElement*, and a function called *sayHello()* - as in the previous example - you can bind sayHello() to the onclick event of myElement inside jQuery's ready() function like so:

```
myElement.click(sayHello);
```

The on() method

With this latest version of jQuery (v.1.7), a brand new method to handle events has been introduced. The **on()** method is highly recommended by the jQuery team if you start a new project.
It can be used to attach handlers both to elements already present in the HTML page from the start and on dynamically added elements (unlike bind()). Here's how it's used.
Using the same *sayHello() function*, inside jQuery's .ready() function simply write:

```
myElement.on('click', sayHello);
```

More details on the **on() method** can be found on http://api.jquery.com/on/.

Anonymous functions

Code samples using jQuery often employ **anonymous functions**. These are functions without a name, and can be quite handy, although I find names more helpful in terms of code readability. If we replace the *sayHello()* function with an anonymous function, we have:

```
//Using bind():

myElement.bind('click', function()

{

alert("Hello jQuery");

});


/*******************************/


//Using click(), we have:

myElement.click(function()

{

alert("Hello jQuery");

});


/*************************************/


//Using on(), we have:

myElement.on('click', function()

{

alert("Hello jQuery");

});
```

Add and remove DOM elements

Once grabbed by your JavaScript code, DOM elements can easily be manipulated with jQuery.
You've had a taste of standards-compliant DOM manipulation techniques in this tutorial. Therefore, you will readily appreciate the jQuery way of performing the same tasks.

Add new DOM elements

You can both retrieve html elements and add new html elements to the page using jQuery's **.html()** method. When you don't pass any arguments, html() retrieves a DOM element, if you pass a string argument, html() adds that string to the page.
If you just need to add new text inside an html element, you can use jQuery's **.text()** method and pass a string argument representing the text you intend to add. Like .html(), you can use .text() without passing any arguments to retrieve text content from the HTML page.
To test .html() prepare an HTML page containing a tag with an id of *myElement*, a <p> tag and some text. First we retrieve the existing paragraph element, then we create a new paragraph.

```
$(document).ready(function()

{

//Use html() to retrieve content:

var pageParagraph = $('p').html();

//the alert should display <p>'s content

alert(pageParagraph);

});


/*************************************************/


//Now let's change the div's content

//by adding a new paragraph.

var newParagraph = $('#myElement').html('<p>Hello new Paragraph!</p>');

//close off all your brackets:

});
```

Save all your files and preview the result in a browser. The alert should display the original paragraph's content, and the page should subsequently display the new paragraph's content.

The way you implement .text() is the same as .html(). I leave you to experiment with it on your own. More details on .html() can be found on http://api.jquery.com/html/, and on .text() can be found on http://api.jquery.com/text/.

Remove DOM elements

A simple way of doing this is with jQuery's **remove()** method. Let's put it to the test.
Use the same HTML page from the previous example and add a button element with an id of *btnDelete*.
In a fresh JavaScript file, add the following code:

```
$(document).ready(function()

{

//use a click handler and an anonymous function

//to do the job

$('#btnDelete').click(function()

{

//delete all <p> elements

$('p').remove();

});

});
```

Save all your files and preview the HTML page in a browser. Click the button and see the paragraphs on the page disappear.

Cool Animation Effects with jQuery

jQuery is so rich with ready-made functions that adding fancy effects on your web page is a breeze. It's hard not to get carried away with it.

Here, we are going to explore the most commonly used jQuery effects. Once again, my recommendation is that of integrating this lesson with a visit to the jQuery website for more code samples and detailed explanations.

In this lesson you will learn how to:

- add **effects by dynamically manipulating styles**;
- use **jQuery's show()/hide()/toggle()** methods;
- use **jQuery's fadeIn()/fadeOut()/fadeToggle()** methods;
- use **jQuery's slideUp()/slideDown()/slideToggle()** methods.

In the process, you will build a simple **sliding menu** and get more practice with the new jQuery approach.

Add and Remove CSS classes with jQuery

jQuery enables you to manipulate CSS styles in code with **$.css(styleName, styleValue)**. It also enables you to add and remove CSS classes in relation to DOM elements.

Because $.css() would sprinkle your HTML page with inline CSS style rules, we will focus on applying and removing CSS classes with **$.addClass(className)**, **$.removeClass(className)**, and **$.toggleClass(className)**. This approach uses the CSS class declarations stored either in the <head> of your HTML page or in a separate file, therefore it's better suited to current best practices in web design.

If you're curious about $.css(), by all means visit http://api.jquery.com/css/ for more details.

Suppose you want to replace the CSS class *bigDiv* with the CSS class *smallDiv* when the user clicks inside the div element. This is how you could do the task with jQuery:

```
$(document).ready(function()
{
//Store a reference to the div in a variable using its id:
var myDiv = $('#myDiv');
//Attach a click handler to manipulate the CSS classes:
myDiv.click(function()
{
//Remove the existing CSS class and replace it with the new one.
//Because the element is the same, chain both methods:
myDiv.removeClass('bigDiv').addClass('smallDiv');
});
});
```

What about toggling between the CSS bigDiv and smallDiv classes by clicking inside the div?

jQuery has this little handy method for you - **$.toggleClass()**. Assuming the CSS class *bigDiv* is hard-coded in the class attribute of the div element, you can toggle the CSS class *smallDiv* using jQuery, like so:

```
$(document).ready(function()

{

var myDiv = $('#myDiv');

myDiv.click(function()

{

myDiv.toggleClass('smallDiv');

});

});
```

Now, keep clicking inside the div element, and you'll see it getting smaller and bigger at each click of the mouse.

Use jQuery show()/hide()

If you want to show/hide HTML elements on a page, you can do so easily with jQuery's **show()/hide()**. You can also modulate the way elements are revealed or hidden by adding the **'slow'** or **'fast'** arguments, or even the **number of milliseconds** you'd like the transition to last.

Let's say you have a paragraph element inside a div with an id of *myDiv*. You want the paragraph to slowly disappear when the user clicks inside the div element. Here's how you could accomplish this with jQuery **hide()**.

```
$(document).ready(function()

{

var myDiv = $('#myDiv');

myDiv.click(function()

{

//Replace 'slow' with 2000 for a slower transition.

$('p').hide('slow');

});

});
```

Clicking inside the div element results in the paragraph slowly disappearing from view. jQuery **show()** works the same way. I leave you to experiment with it on your own.

What if you wanted to toggle between show/hide transitions as the user clicks inside the div element? No problem, jQuery offers the **$.toggle()** method. To see it in action, simply replace *hide()* with *toggle()* in the previous example, like so.

```
$(document).ready(function()
{
var myDiv = $('#myDiv');
myDiv.click(function()
{
$('p').toggle('slow');
});
});
```

Make DOM elements fade in and out of the page

For a more sophisticated fading effect, use **$.fadeIn()/$.fadeOut()/$.fadeToggle()**. You use these methods the same way as you used $.show()/$.hide()/$.toggle() in the previous examples. Here's the paragraph from the previous code sample toggling between visible and hidden using $.fadeToggle().

```
$(document).ready(function()
{
var myDiv = $('#myDiv');
myDiv.click(function()
{
$('p').fadeToggle('slow');
});
});
```

The effect is very similar to that of $.toggle('slow'). Just try it out.

Make DOM elements slide up and down with jQuery

You can easily make DOM elements appear and disappear with a sliding effect using **$.slideUp()/$.slideDown()/$.slideToggle()**. You implement these methods similarly to the previous ones.

For example, let's say you want to slide the paragraph element from the previous example up and down as the user clicks inside the div element. Here's how you would accomplish this with **$.slideToggle()**.

```
$(document).ready(function()

{

var myDiv = $('#myDiv');

myDiv.click(function()

{

$('p').slideToggle('slow');

});

});
```

Now keep clicking inside the div element and see the paragraph inside the div sliding up and down.
I recommend you experiment with the code samples above as much as you can before moving on with the next challenge.

Try out: sliding menu

Here we are at our jQuery try out exercise. You will build a simple sliding menu similar to what you might have already seen on many websites.
The user moves the mouse over a main menu item and a list of sub-menu items slides down. As the user moves the mouse away from the menu item, its sub-menu items slide back up away from view.
Let's start from the HTML page:

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 20: jQuery Sliding Menu</title>

<script type="text/javascript" src="http://code.jquery.com/jquery-
1.7.min.js"></script>

<script type="text/javascript" src="lesson20_ex.js"></script>

</head>
<body>
<h1>Lesson 20: jQuery Sliding Menu</h1>

<div id="myDiv">

<ul class="outerList">
<li><a href="#">Menu Item 1</a></li>
```

```
<li><a href="#">Menu Item 2 ↓</a>

<div>
<ul class="innerList">
<li><a href="#">Sub-Menu Item 1</a></li>
<li><a href="#">Sub-Menu Item 2</a></li>
<li><a href="#">Sub-Menu Item 3</a></li>
</ul>
</div>

</li>
<li><a href="#">Menu Item 3</a></li>
</ul>

</div>

</body>
</html>
```

The HTML page above has a reference to a CDN-served copy of jQuery and to an external JavaScript file in the <head> section.
The <body> section has a div element with an id of *myDiv* that contains a list with 3 main menu items and 3 sub-menu items nested inside the second main menu item. The sub-menu items are contained inside a div element.
Style your menu so that the main menu items are horizontally lined up and the nested sub-menu items are vertically stacked under their parent li element.
We want the div element that wraps the sub-menu items to slide down when the user moves the mouse over its parent li element.
Prepare your *lessonex.js* file and write the following code.

```
$(document).ready(function()

{

//retrieve the menu subitems (div element child of the list element)

//with the powerful jQuery selectors

//and store them in a variable

var subItems = $('ul li div');

//retrieve the main menu items that

//have the retrieved subitems as children.

//Notice the handy .has() method:

var mainItems = $('ul li').has(subItems);

//Hide all subitems on page load
```

```
subItems.hide();

//Attach the .hover() function to the main

//menu items:

$(mainItems).hover(function()

{

//Apply .slideToggle() to the sub-menu

subItems.slideToggle('fast');

});

});
```

Save your work and preview it in a browser. You should see something like the page indicated by following the example link above, only reflecting your own CSS styles.
Move your mouse over the second menu item and if all goes as expected you'll see the sub-menu sliding down. It's taken us just a few lines of code to accomplish an animation effect that in raw JavaScript would have made us sweat quite a bit.

Easy AJAX Calls with jQuery

jQuery provides a rich set of handy methods you can use to Ajaxify your web pages.
Back in lesson 18 you used AJAX by dealing directly with the XMLHttpRequest object. You also performed some feature testing for Internet Explorer browsers that didn't support the XMLHttpRequest object.
jQuery provides **wrapper methods** that shield you from the inner mechanisms of an AJAX request.

How do I load HTML content with jQuery AJAX?

jQuery offers a very simple approach to loading HTML content on your web page asynchronously. Just use the **$.load()** function and you're done.

If you use $.load(htmlPageUrl) passing only the URL of the HTML document as argument, the entire content of the HTML document is loaded into the calling page.

Instead, I like to use $.load(htmlPageUrl fragmentIdentifier), which exactly targets the bit of content I intend to retrieve.

Prepare an HTML page containing a div element with an id of *content* and a paragraph element with some dummy content. Save it as *content.html* (or anything you like), and upload it to your server. This document provides the content to retrieve using AJAX.

Prepare a second HTML page containing a link element, a div element with an id of *result*, a reference to the jQuery library in the <head> section, and enclosing <script> tags in the <body> section for your own jQuery-powered JavaScript code.

```
<!DOCTYPE html>
<html>
<head>
<title>Lesson 21: Easy AJAX Calls with jQuery</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-
1.7.min.js"></script>
</head>
<body>
<h1>Lesson 21: Easy AJAX Calls with jQuery load()</h1>

<p><a href="#">Click here to fetch HTML content</a></p>

<div id="result">

</div>

<script type="text/javascript">

//JavaScript AJAX code here

</script>

</body>
</html>
```

For these simple demos, we're going to embed our JavaScript code inside the HTML page. However, keep in mind that, if you write code for a website, it's highly recommended that you use external JavaScript files.
When the user clicks on the link on your HTML page, an AJAX request will be made to *content.html* targeting the text inside the div element with an id of *content*. This text is dynamically inserted in the div with an id of *result* in the calling page.

To achieve this, enter the following code snippet inside the enclosing <script> tags:

```
$(document).ready(function()

{

//Attach a handler to the click event

//of the link on the page:

$('a').click(function()
```

```
{

//Target the div with id of result

//and load the content from the specified url

//and the specified div element into it:

$('#result').load('content.html #content');

});

});
```

Save your work and preview it in a browser. You should see something like the page indicated by following the example link above.
Click the link, and if all goes well, the dummy text from *content.html* is loaded inside the div element on the page. The operation takes place asynchronously without a full page refresh.

Use $.get()?

As nice and simple as $.load() is, it can't perform all types of content requests. A more flexible approach is offered by the **$.get() method**.
You can use $.get() to load data from the server with a **GET HTTP request** - that is, via a **query string** $.get(url, {dataKey1 : 'dataValue1', dataKey2 : 'dataValue2'}, optionalSuccessFunction) takes in 3 arguments:
the**URL where the data you want to retrieve is stored**;
optionally, **some data**, if you want to send data to the server with the request. This is done using either a string or what is called **object literal or map**, that is, comma-separated key:value pairs inside curly braces. For instance: **{'First Name' : 'John', 'Last Name' : 'Smith'}** sends the First Name and Last Name values to the server together with the AJAX GET request;
and a **function that deals with the returned data if you want to display or process the successful response in any way.**
Let's see how to use jQuery .get() to retrieve the XML document you used as u read before.
The HTML page that makes the AJAX request remains unchanged from the previous example. Rewrite the JavaScript code as follows:

```
$(document).ready(function()

{

//Store the URL value in a variable

var url = "content.xml";


/************************************/


//Package the result-handling code
```

```
//in its own function: it's more readable
function processData(data)
{
//This variable will hold the result
//converted into a string for display
var resultStr = "";
//use jQuery .find() to extract the language
//element from the returned data
//and store it in an array
var items = $(data).find('language');
//loop over each language item with
//jQuery .each() function
$(items).each(function(i)
{
//extract the text of each language item and
//add it to the resultStr variable with a line break.
//Notice the use of $(this)
//to refer to the item currently being
//inspected by the loop
resultStr += $(this).text() + '<br />';
//add the final string result to div element
//with the id of result using .html()
$('#result').html(resultStr);
});
}
```

```
/**************************************/

//Attach a click handler to the link element:

//when the user clicks on the link, the AJAX

//request is sent to the server:

$('a').click(function()

{

//use $.get() passing the url variable and

//the name of the result-handling function

//as arguments:

$.get(url, processData),

});

});
```

Save all your files on the server and click on the link. If all goes well, you should see a list of programming languages being displayed on the page without a full page refresh.

In the example above, you came across **jQuery .find()** and **jQuery .each()**.
$.find() is used to find a DOM element's descendants or children. In the example above, you used it to find all children called *language* of the root XML element contained in the variable called *data*. Further details on $.find() can be accessed on http://api.jquery.com/find/.

$.each(index) is a for ... loop done the jQuery way. It's extremely concise and efficient. Notice the use of **$(this)** inside the $.each() function block. This is a snappy way of referring to the item the loop is currently processing: in our example above $(this) refers to one of the *language items* in the *items array*. More details on $.each() can be found on http://api.jquery.com/each/.

How do I use $.post()?

If you want to use POST instead of GET in your AJAX calls, you can use **$.post()**.
Unlike GET requests, POST requests don't use a query string to send data. If you intend to send more than a few bits of data to the sever, or if you intend to send sensitive data, it's recommended you use an HTTP POST request.
The way you implement $.post() is very similar to the way you implemented $.get() in the previous example. I invite you to experiment with it on your own and to visit http://api.jquery.com/jQuery.post/ for more code samples and useful details.

How do I use $.ajax()?

- If you need greater flexibility, the full-blown **$.ajax()** function offers a great number of settings.
- For instance, let's say you want to retrieve a list of programming languages from the XML document you used in the previous example. You might want to specify the following options:
- the request must be an **HTTP GET**;
- the page from which the result is returned **must not be in the browser's cache**;
- the response returned by the server **is of data-type XML**;
- the request **is made in html**;
- there must be a **function that handles the returned result if all goes well**;
- and, finally, there must be a **function that handles errors** in case the request is not successful.
- Use the HTML page and the XML document from the previous example. Also, keep the url variable and the processData() function from the previous exercise - you will use both as the url and the success arguments respectively inside the $.ajax() function. Delete everything else inside the document.ready() function. Just below the processData() function, write the following code:

```
//Package the code that handles

//error message in case the request

//is not successful:

function errorAlert(e, jqxhr)

{

alert("Your request was not successful: " + jqxhr);

}

/*********************************/

//Attach a click handler to the

//link element on the page

$('a').click(function()

{

//Prepare the AJAX request that

//will be sent when the user clicks the link:

$.ajax(

{

type: "GET",
```

```
cache: false,

url: url,

dataType: "xml",

contentType: "text/html",

success: processData,

error: errorAlert

}); //end of $.ajax

}); //end of click handler

});  //end of $.ready function
```

Save your work and preview it in a browser.

The result should be similar to the previous example. If an error occurs, you'll be presented with an alert box. The errorAlert() function has an **e** argument that represents the **type of error**, and an **jqxhr** argument that represents the **request as a jQuery object**.
In case an error occurs, details about the error are automatically contained in the arguments provided and will be displayed in the alert box.

Do you want to test the error catching function? Simply replace `dataType: "xml"` in the $.ajax() function with `dataType: "text/xml"`. Save all your files and run the HTML page. Now, when you click the link, an alert box should pop up displaying a parser error message.

jQuery - Utilities

jQuery provides a set of miscellaneous methods which can be used for various reasons explained in this tutorial:

There are two important methods provided by latest version of jQuery to detect the browser and its associated features.

Getting Browser Version:

The **jQuery.browser** method contains flags for the useragent, read from navigator.userAgent.

Following example would show how to get browser name and its version:

```
<html>
<head>
<title>the title</title>
  <script type="text/javascript"
  src="/jquery/jquery-1.3.2.min.js"></script>
  <script type="text/javascript" language="javascript">

  $(document).ready(function() {

   jQuery.each(jQuery.browser, function(i, val) {
     $("<div>" + i + " : <span>" + val + "</span>")
            .appendTo(document.body);
   });

  });

  </script>
  <style>
  p { color:green; font-weight:bolder; margin:3px 0 0 10px; }
  div { color:blue; margin-left:20px; font-size:14px; }
  span { color:red; }
 </style>
</head>
<body>

 <p>Browser info:</p>

</body>
</html>
```

For my PC it would display following result:

Browser info:

version : 7.0
safari : false
opera : false
msie : true
mozilla : false

Getting Browser Properties:

The **jQuery.support** method contains a collection of properties that represent the presence of different browser features or bugs.

You can test the following properties using jQuery.support :

Property	Description
boxModel	Is equal to true if the page and browser are rendering according to the W3C CSS Box Model. This is currently false in IE 6 and 7 when they are in Quirks Mode.
cssFloat	Is equal to true if style.cssFloat is used to access the current CSS float value. This is currently false in IE, it uses styleFloat instead
hrefNormalized	Is equal to true if the browser leaves intact the results from getAttribute. This is currently false in IE, the URLs are normalized.
htmlSerialize	Is equal to true if the browser properly serializes link elements when innerHTML is used. This is currently false in IE.
leadingWhitespace	Is equal to true if the browser preserves leading whitespace when innerHTML is used. This is currently false in IE 6-8.
noCloneEvent	Is equal to true if the browser does not clone event handlers when elements are cloned. This is currently false in IE.
objectAll	Is equal to true if doing getElementsByTagName("*") on an object element returns all descendant elements. This is currently false in IE 7 and IE 8.
opacity	Is equal to true if a browser can properly interpret the opacity style property. This is currently false in IE, it uses alpha filters instead.
scriptEval	Is equal to true if using appendChild/createTextNode to inject inline scripts executes them. This is currently false in IE, it uses .text to insert executable scripts.
style	Is equal to true if getAttribute("style") is able to return the inline style specified by an element. This is currently false in IE - it uses cssText instead.
tbody	Is equal to true if the browser allows table elements without tbody elements. This is currently false in IE, which automatically inserts tbody if it is not present in a string assigned to innerHTML.

Following example would show how to get box model property of a document:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {

    $("p").html("This document uses the W3C box model: <span>" +
                jQuery.support.boxModel + "</span>");

   });

   </script>
   <style>
   p { color:blue; margin:20px; }
   span { color:red; }
  </style>
</head>
```

```
<body>

  <p></p>

</body>
</html>
```

For my PC it would display following result:

```
This frame uses the W3C box model: true
```

jQuery - Plugins

A plug-in is piece of code written in a standard JavaScript file. These files provide useful jQuery methods which can be used along with jQuery library methods.

How to use Plugins:

To make a plug-in's methods available to us, we include plug-in file very similar to jQuery library file in the <head> of the document.
We must ensure that it appears after the main jQuery source file, and before our custom JavaScript code.
Following example shows how to include **jquery.plug-in.js** plugin:

```
<html>
<head>
<title>the title</title>
  <script type="text/javascript"
  src="/jquery/jquery-1.3.2.min.js"></script>

  <script src="jquery.plug-in.js" type="text/javascript">
  </script>

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

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

  $(document).ready(function() {

   .....your custom code.....

  });

  </script>
</head>
<body>

  .....

</body>
</html>
```

How to develop a Plug-in

This is very simple to write your own plug-in. Following is the syntax to create a a method:

```
jQuery.fn.methodName = methodDefinition;
```

Here *methodNameM* is the name of new method and *methodDefinition* is actual method definition.
The guideline recommended by the jQuery team is as follows:
Any methods or functions you attach must have a semicolon (;) at the end.
Your method must return the jQuery object, unless explicity noted otherwise.
You should use this.each to iterate over the current set of matched elements - it produces clean and compatible code that way.
Prefix the filename with jquery, follow that with the name of the plugin and conclude with .js.
Always attach the plugin to jQuery directly instead of $, so users can use a custom alias via noConflict() method.
For example, if we write a plugin that we want to name *debug*, our JavaScript filename for this plugin is:

```
jquery.debug.js
```

The use of the **jquery.** prefix eliminates any possible name collisions with files intended for use with other libraries.

Example:

Following is a small plug-in to have warning method for debugging purpose. Keep this code in *jquery.debug.js* file:

```
jQuery.fn.warning = function() {
   return this.each(function() {
      alert('Tag Name:"' + $(this).attr("tagName") + '".');
   });
};
```

Here is the example showing usage of warning() method. Assuming we put *jquery.debug.js* file in */jquery* subdirectory:

```
<html>
<head>
<title>the title</title>
  <script type="text/javascript"
  src="/jquery/jquery-1.3.2.min.js"></script>
  <script src="/jquery/jquery.debug.js" type="text/javascript">
  </script>

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

  $(document).ready(function() {

     $("div").warning();
     $("p").warning();
```

```
   });

   </script>
</head>
<body>

  <p>This is paragraph</p>
  <div>This is division</div>

</body>
</html>
```

This would produce following result:

```
Tag Name:"DIV"
Tag Name:"P"
```

HTML forms and JavaScript

The following is an example on checking input data of HTML forms and setting hidden parameters via Javascript, in this case the URL is automatically determined and send to the service once submit is pressed.

```
function updateFormAction(){

  document.forms["myform"].elements["webpage"].value = encodeURIComponent(location.href);
  var value = document.forms["myform"].elements["user"].value;
  if (value==null||value=="")
    {
      var Textknoten = document.createTextNode("Please fill out all fields");
      document.getElementById("replace").replaceChild(Textknoten,
document.getElementById("replace").firstChild);

       return false;
     }
  var value = document.forms["myform"].elements["email"].value;
  if (value==null||value=="")
    {
     return false;
     }
   var value = document.forms["myform"].elements["summary"].value;
  if (value==null||value=="")
    {
     return false;
     }

   var value = document.forms["myform"].elements["comment"].value;
  if (value==null||value=="")
    {
     return false;
     }
  return true;

}
<html>
<head> </head>
<body>
<!-- Favorites -->
<table border="0" cellpadding="10" cellspacing="0" width="100%">
  <tr>
    <td>

    <div dojoType="jatin.TitlePane" title="Feedback Form">
    So how did you like this webpage?  <BR>
    <FORM id=myform " NAME="myform"
ACTION="http://www.veinstitution.com/FeedbackForm/FeedbackSave" method="get"
```

```
    onclick="return updateFormAction()"><input TYPE="hidden"
    NAME="webpage" VALUE="test">

   <table>
    <tr>
     <td></td>
    </tr>
    <tr>
     <td align="right">User: (*)</td>
     <td><input name="user" type="text" size="30" maxlength="30"></td>
    </tr>
    <tr>
     <td align="right">Email:(*)</td>
     <td><input name="email" type="text" size="30" maxlength="30">
     (will not be visible)</td>
    </tr>
    <tr>
     <td align="right" valign="top">Summary:(*)</td>
     <td><input name="summary" type="text" size="30" maxlength="30"></td>
    </tr>
    <tr>
     <td align="right" valign="top">Comment:(*)</td>
     <td><textarea name="comment" rows="6" cols="40"></textarea></td>
    </tr>
    <tr>
     <td></td>
     <td><input type="submit" value="Submit"></td>
    </tr>
    <tr> <td colspan="2"> <div id="replacefeedback"> (*) are required </div></div> </td></tr>
   </table>
   </FORM>
   </div>
   </td>
  </tr>
</table>
</body>
</html>
```