Gaurish Technologies Private Limited

Javascript

Module-6A

JavaScript Tutorial

JavaScript is *THE* scripting language of the Web.
JavaScript is used in billions of Web pages to add functionality, validate forms, communicate with the server, and much more.
JavaScript is easy to learn. You will enjoy it.

JavaScript Introduction

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

What You Should Already Know

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

- HTML and CSS

What is JavaScript?

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

Are Java and JavaScript the same?

NO!
Java and JavaScript are two completely different languages in both concept and design!
Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What Can JavaScript do?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

---

JavaScript = ECMAScript

JavaScript is an implementation of the ECMAScript language standard. ECMA-262 is the official JavaScript standard.
JavaScript was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all browsers since 1996.
The official standardization was adopted by the ECMA organization (an industry standardization association) in 1997.
The ECMA standard (called ECMAScript-262) was approved as an international ISO (ISO/IEC 16262) standard in 1998.
The development is still in progress.

JavaScript How To

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

Writing to The HTML Document

The example below writes a <p> element with current date information to the HTML document:

Example

```
<html>
	<body>
		<h1>My First Web Page</h1>
		<script type="text/javascript">
			document.write("<p>" + Date() + "</p>");
		</script>
	</body>
</html>
```

Note: Try to avoid using document.write() in real life JavaScript code. The entire HTML page will be overwritten if document.write() is used inside a function, or after the page is loaded. However, document.write() is an easy way to demonstrate JavaScript output in a tutorial.

Changing HTML Elements

The example below writes the current date into an existing <p> element:

Example

```
<html>
	<body>
		<h1>My First Web Page</h1>
		<p id="demo"></p>
		<script type="text/javascript">
			document.getElementById("demo").innerHTML=Date();
		</script>
	</body>
</html>
```

Note: To manipulate HTML elements JavaScript uses the DOM method **getElementById()**. This method accesses the element with the specified id.

Examples Explained

To insert a JavaScript into an HTML page, use the <script> tag.
Inside the <script> tag use the type attribute to define the scripting language.
The <script> and </script> tells where the JavaScript starts and ends:

```
<html>
	<body>
		<h1>My First Web Page</h1>
		<p id="demo">This is a paragraph.</p>
		<script type="text/javascript">
		... some JavaScript code ...
		</script>
	</body>
</html>
```

The lines between the <script> and </script> contain the JavaScript and are executed by the browser.
In this case the browser will replace the content of the HTML element with id="demo", with the current date:

```
<html>
	<body>
		<h1>My First Web Page</h1>
		<p id="demo">This is a paragraph.</p>
		<script type="text/javascript">
			document.getElementById("demo").innerHTML=Date();
		</script>
	</body>
</html>
```

Without the <script> tag(s), the browser will treat "document.getElementById("demo").innerHTML=Date();" as pure text and just write it to the page:

Some Browsers do Not Support JavaScript

Browsers that do not support JavaScript, will display JavaScript as page content.
To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript.
Just add an HTML comment tag <!-- before the first JavaScript statement, and a --> (end of comment) after the last JavaScript statement, like this:

```
<html>
	<body>
		<script type="text/javascript">
		<!—
			document.getElementById("demo").innerHTML=Date();
		//-->
		</script>
```

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

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

JavaScript Where To

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

JavaScript in <body>

The example below writes the current date into an existing <p> element when the page loads:

Example

```
<html>
    <body>

        <h1>My First Web Page</h1>

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

        <script type="text/javascript">

            document.getElementById("demo").innerHTML=Date();
        </script>

    </body>
</html>
```

Note that the JavaScript is placed at the bottom of the page to make sure it is not executed before the <p> element is created.

JavaScript Functions and Events

JavaScripts in an HTML page will be executed when the page loads. This is not always what we want.

Sometimes we want to execute a JavaScript when an **event** occurs, such as when a user clicks a button. When this is the case we can put the script inside a **function**.

Events are normally used in combination with functions (like calling a function when an event occurs).

You will learn more about JavaScript functions and events in later chapters.

JavaScript in <head>

The example below calls a function when a button is clicked:

Example

```
<html>
    <head>
        <script type="text/javascript">
            function displayDate()
```

```
                {
                        document.getElementById("demo").innerHTML=Date();
                }
        </script>
    </head>
    <body>
        <h1>My First Web Page</h1>
        <p id="demo"></p>
        <button type="button" onclick="displayDate()">Display Date</button>
    </body>
</html>
```

Scripts in <head> and <body>

You can place an unlimited number of scripts in your document, and you can have scripts in both the body and the head section at the same time.
It is a common practice to put all functions in the head section, or at the bottom of the page. This way they are all in one place and do not interfere with page content.

Using an External JavaScript

JavaScript can also be placed in external files.
External JavaScript files often contain code to be used on several different web pages.
External JavaScript files have the file extension .js.
Note: External script cannot contain the <script></script> tags!
To use an external script, point to the .js file in the "src" attribute of the <script> tag:

Example

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

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

JavaScript Statements

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

JavaScript is Case Sensitive

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

JavaScript Statements

A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do.
This JavaScript statement tells the browser to write "Hello Dolly" to the web page:

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

It is normal to add a semicolon at the end of each executable statement. Most people think this is a good programming practice, and most often you will see this in JavaScript examples on the web.
The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. Because of this you will often see examples without the semicolon at the end.
Note: Using semicolons makes it possible to write multiple statements on one line.

JavaScript Code

JavaScript code (or just JavaScript) is a sequence of JavaScript statements.
Each statement is executed by the browser in the sequence they are written.
This example will write a heading and two paragraphs to a web page:

Example

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

JavaScript Blocks

JavaScript statements can be grouped together in blocks.
Blocks start with a left curly bracket {, and end with a right curly bracket }.

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

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

Example

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

The example above is not very useful. It just demonstrates the use of a block. Normally a block is used to group statements together in a function or in a condition (where a group of statements should be executed if a condition is met).
You will learn more about functions and conditions in later chapters.

JavaScript Comments

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

JavaScript Comments

Comments can be added to explain the JavaScript, or to make the code more readable.
Single line comments start with //.

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

Example

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

JavaScript Multi-Line Comments

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

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

Example

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

Using Comments to Prevent Execution

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

Example

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

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

Example

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

```
        document.write("<p>This is another paragraph.</p>");
        */
</script>
```

Using Comments at the End of a Line

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

Example

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

JavaScript Variables

Variables are "containers" for storing information.

Do You Remember Algebra From School?

Do you remember algebra from school? x=5, y=6, z=x+y
Do you remember that a letter (like x) could be used to hold a value (like 5), and that you could use the information above to calculate the value of z to be 11?
These letters are called **variables**, and variables can be used to hold values (x=5) or expressions (z=x+y).

JavaScript Variables

As with algebra, JavaScript variables are used to hold values or expressions.
A variable can have a short name, like x, or a more descriptive name, like carname.
Rules for JavaScript variable names:

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

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

Example

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

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables.
You declare JavaScript variables with the **var** keyword:

```
var x;
var carname;
```

After the declaration shown above, the variables are empty (they have no values yet).
However, you can also assign values to the variables when you declare them:

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

After the execution of the statements above, the variable **x** will hold the value **5**, and **carname** will hold the value **Volvo**.
Note: When you assign a text value to a variable, put quotes around the value.

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

Local JavaScript Variables

A variable declared within a JavaScript function becomes **LOCAL** and can only be accessed within that function. (the variable has local scope).

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.
Local variables are deleted as soon as the function is completed.
You will learn more about functions in a later chapter of this tutorial.

Global JavaScript Variables

Variables declared outside a function become **GLOBAL**, and all scripts and functions on the web page can access it.
Global variables are deleted when you close the page.

Assigning Values to Undeclared JavaScript Variables

If you assign values to variables that have not yet been declared, the variables will automatically be declared as global variables.
These statements:

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

will declare the variables x and carname as global variables (if they don't already exist).

JavaScript Arithmetic

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

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

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

JavaScript Operators

= is used to assign values.
+ is used to add values.

The assignment operator = is used to assign values to JavaScript variables.
The arithmetic operator + is used to add values together.

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

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

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

Given that y=5, the table below explains the arithmetic operators:

Operator	Description	Example	Result

+	Addition	x=y+2	x=7	y=5
-	Subtraction	x=y-2	x=3	y=5
*	Multiplication	x=y*2	x=10	y=5
/	Division	x=y/2	x=2.5	y=5
%	Modulus (division remainder)	x=y%2	x=1	y=5
++	Increment	x=++y	x=6	y=6
		x=y++	x=5	y=6
--	Decrement	x=--y	x=4	y=4
		x=y--	x=5	y=4

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

Given that x=10 and y=5, the table below explains the assignment operators:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

The + Operator Used on Strings

The + operator can also be used to add string variables or text values together.
To add two or more string variables together, use the + operator.

```
txt1="What a very";
txt2="nice day";
txt3=txt1+txt2;
```

After the execution of the statements above, the variable txt3 contains "What a verynice day".
To add a space between the two strings, insert a space into one of the strings:

```
txt1="What a very ";
txt2="nice day";
txt3=txt1+txt2;
```

or insert a space into the expression:

```
txt1="What a very";
txt2="nice day";
txt3=txt1+" "+txt2;
```

After the execution of the statements above, the variable txt3 contains:
"What a very nice day"

Adding Strings and Numbers

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

Example

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

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

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

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

JavaScript Comparison and Logical Operators

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

Comparison Operators

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

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

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

>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

How Can it be Used

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

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

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

Logical Operators

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

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

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

Conditional Operator

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

Syntax

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

Example

If the variable **visitor** has the value of "PRES", then the variable **greeting** will be assigned the value "Dear President " else it will be assigned "Dear":

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

var visitor="PRES";
var greeting=(visitor=="PRES")?"Dear President ":"Dear ";
document.write(greeting);

</script>
```

JavaScript If...Else Statements

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

Conditional Statements

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

In JavaScript we have the following conditional statements:

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

If Statement

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

Syntax

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

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

Example

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

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

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

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

If...else Statement

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

Syntax

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

Example

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

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

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

If...else if...else Statement

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

Syntax

```
if (condition1)
 {
        code to be executed if condition1 is true
 }
else if (condition2)
 {
        code to be executed if condition2 is true
 }
else
 {
        code to be executed if neither condition1 nor condition2 is true
 }
```

Example

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
 {
```

```
        document.write("<b>Good morning</b>");
 }
else if (time>=10 && time<16)
 {
        document.write("<b>Good day</b>");
 }
else
 {
        document.write("<b>Hello World!</b>");
 }
</script>
```

JavaScript Switch Statement

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

The JavaScript Switch Statement

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

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

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

Example

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

var d=new Date();
var theDay=d.getDay();
```

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

JavaScript Popup Boxes

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

Alert Box

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

alert("*sometext*");

Example

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

Confirm Box

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

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

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

Syntax

confirm("*sometext*");

Example

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

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.
When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

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

Example

```
<html>
	<head>
		<script type="text/javascript">
			function show_prompt()
			{
			var name=prompt("Please enter your name","Harry Potter");
```

```
                    if (name!=null && name!="")
                     {
                            document.write("<p>Hello " + name + "! How are you
                    today?</p>");
                     }
                    }
            </script>
        </head>
        <body>
            <input type="button" onclick="show_prompt()" value="Show prompt box" />
        </body>
</html>
```

JavaScript Functions

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

JavaScript Functions

To keep the browser from executing a script when the page loads, you can put your script into a function.

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

You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).
Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

How to Define a Function

Syntax

function *functionname*(*var1,var2,...,varX*)
{
some code
}
The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function.
Note: A function with no parameters must include the parentheses () after the function name.
Note: Do not forget about the importance of capitals in JavaScript! The word *function* must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

JavaScript Function Example

Example

<html>

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

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

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

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

The return Statement

The return statement is used to specify the value that is returned from the function.
So, functions that are going to return a value must use the return statement.
The example below returns the product of two numbers (a and b):

Example

```
<html>
	<head>
		<script type="text/javascript">
			function product(a,b)
			{
				return a*b;
			}
		</script>
	</head>
	<body>
		<script type="text/javascript">
			document.write(product(4,3));
		</script>
	</body>
</html>
```

The Lifetime of JavaScript Variables

If you declare a variable, using "var", within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.
If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

JavaScript For Loop

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

JavaScript Loops

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

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

The for Loop

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

Syntax

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

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs.
Note: The increment parameter could also be negative, and the <= could be any comparing statement.

Example

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

The while loop

The while loop will be explained in the next chapter.

JavaScript While Loop

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

The while Loop

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

Syntax

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

Note: The <= could be any comparing operator.

Example

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

Example

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

The do...while Loop

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

Syntax

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

Example

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

Example

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

JavaScript Break and Continue Statements

The break Statement

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

Example

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

The continue Statement

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

Example

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

JavaScript For...In Statement

JavaScript For...In Statement

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

Syntax

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

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

Example

Looping through the properties of an object:

Example

```
var person={fname:"John",lname:"Doe",age:25};
var x;

for (x in person)
{
document.write(person[x] + " ");
}
```

JavaScript Events

Events are actions that can be detected by JavaScript.

Acting to an Event

The example below displays the date when a button is clicked:

Example

```
<html>
	<head>
		<script type="text/javascript">
			function displayDate()
			{
				document.getElementById("demo").innerHTML=Date();
			}
		</script>
	</head>
	<body>
		<h1>My First Web Page</h1>
		<p id="demo"></p>
		<button type="button" onclick="displayDate()">Display Date</button>
	</body>
</html>
```

Events

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.
Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.
Examples of events:

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

Note: Events are normally used in combination with functions, and the function will not be executed before the event occurs!
For a complete reference of the events recognized by JavaScript, go to our complete

onLoad and onUnload

The onLoad and onUnload events are triggered when the user enters or leaves the page.
The onLoad event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

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

onFocus, onBlur and onChange

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

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

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

onSubmit

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

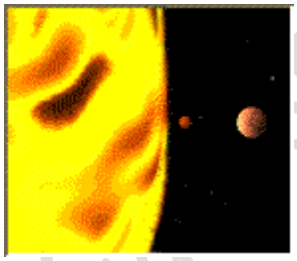
Below is an example of how to use the onSubmit event. The checkForm() function will be called when the user clicks the submit button in the form. If the field values are not accepted, the submit should be cancelled. The function checkForm() returns either true or false. If it returns true the form will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

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

onMouseOver

The onmouseover event can be used to trigger a function when the user mouses over an HTML element:

Example



The Sun and the gas giant planets like Jupiter are by far the largest objects in our Solar System.

JavaScript Try...Catch Statement

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

JavaScript - Catching Errors

When browsing Web pages on the internet, we all have seen a JavaScript alert box telling us there is a runtime error and asking "Do you wish to debug?". Error message like this may be useful for developers but not for users. When users see errors, they often leave the Web page. This chapter will teach you how to catch and handle JavaScript error messages, so you don't lose your audience.

The try...catch Statement

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

Syntax

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

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

Examples

The example below is supposed to alert "Welcome guest!" when the button is clicked. However, there's a typo in the message() function. alert() is misspelled as adddlert(). A JavaScript error occurs. The catch block catches the error and executes a custom code to handle it. The code displays a custom error message informing the user what happened:

Example

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

```
                }
        </script>
    </head>
    <body>
        <input type="button" value="View message" onclick="message()" />
    </body>
</html>
```

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

Example

```
<html>
    <head>
        <script type="text/javascript">
            var txt="";
            function message()
            {
                Try
                 {
                    adddlert("Welcome guest!");
                 }
                catch(err)
                 {
                 txt="There was an error on this page.\n\n";
                 txt+="Click OK to continue viewing this page,\n";
                 txt+="or Cancel to return to the home page.\n\n";
                 if(!confirm(txt))
                  {
                    document.location.href="http://www.w3schools.com/";
                  }
                 }
            }
        </script>
    </head>

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

The throw Statement

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

JavaScript Throw Statement

The throw statement allows you to create an exception.

The Throw Statement

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

Syntax

throw *exception*

The exception can be a string, integer, Boolean or an object.
Note that *throw* is written in lowercase letters. Using uppercase letters will generate a JavaScript error!

Example

The example below determines the value of a variable called x. If the value of x is higher than 10, lower than 5, or not a number, we are going to throw an error. The error is then caught by the catch argument and the proper error message is displayed:

Example

```
<html>
	<body>
		<script type="text/javascript">
			var x=prompt("Enter a number between 5 and 10:","");
			try
			 {
				if(x>10)
				 {
					throw "Err1";
				 }
					else if(x<5)
				 {
					throw "Err2";
				 }
					 else if(isNaN(x))
				 {
					throw "Err3";
				 }
			}
			catch(err)
			 {
				if(err=="Err1")
```

```
                {
                        document.write("Error! The value is too high.");
                }
                if(err=="Err2")
                {
                        document.write("Error! The value is too low.");
                }
                if(err=="Err3")
                {
                        document.write("Error! The value is not a number.");
                }
                }
        </script>
    </body>
</html>
```

JavaScript Special Characters

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

Insert Special Characters

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

Look at the following JavaScript code:

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

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

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

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

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

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

Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line

\r	carriage return
\t	tab
\b	backspace
\f	form feed

JavaScript Guidelines

Some other important things to know when scripting with JavaScript.

JavaScript is Case Sensitive

A function named "myfunction" is not the same as "myFunction" and a variable named "myVar" is not the same as "myvar".
JavaScript is case sensitive - therefore watch your capitalization closely when you create or call variables, objects and functions.

White Space

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

```
var name="Hege";
var name = "Hege";
```

Break up a Code Line

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

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

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

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

JavaScript Objects Introduction

JavaScript is an Object Based Programming language.
An Object Based Programming language allows you to define your own objects and make your own variable types.

Object Based Programming

JavaScript is an Object Based Programming language, and allows you to define your own objects and make your own variable types.
However, creating your own objects will be explained later, in the Advanced JavaScript section. We will start by looking at the built-in JavaScript objects, and how they are used. The next pages will explain each built-in JavaScript object in detail.

Note that an object is just a special kind of data. An object has properties and methods.

Properties

Properties are the values associated with an object.
In the following example we are using the length property of the String object to return the number of characters in a string:

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

The output of the code above will be:

```
12
```

Methods

Methods are the actions that can be performed on objects.

In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

```
<script type="text/javascript">
var str="Hello world!";
document.write(str.toUpperCase());
</script>
```

The output of the code above will be:

```
HELLO WORLD!
```

JavaScript String Object

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

Complete String Object Reference

For a complete reference of all the properties and methods that can be used with the String object, go to our complete String object reference.
The reference contains a brief description and examples of use for each property and method!

String object

The String object is used to manipulate a stored piece of text.
Examples of use:
The following example uses the length property of the String object to find the length of a string:

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

The code above will result in the following output:

```
12
```

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

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

The code above will result in the following output:

```
HELLO WORLD!
```

JavaScript Date Object

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

Complete Date Object Reference

For a complete reference of all the properties and methods that can be used with the Date object, go to our complete Date object reference.
The reference contains a brief description and examples of use for each property and method!

Create a Date Object

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

Date objects are created with the Date() constructor.
There are four ways of initiating a date:

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

Most parameters above are optional. Not specifying, causes 0 to be passed in.
Once a Date object is created, a number of methods allow you to operate on it. Most methods allow you to get and set the year, month, day, hour, minute, second, and milliseconds of the object, using either local time or UTC (universal, or GMT) time.
All dates are calculated in milliseconds from 01 January, 1970 00:00:00 Universal Time (UTC) with a day containing 86,400,000 milliseconds.
Some examples of initiating a date:

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

Set Dates

We can easily manipulate the date by using the methods available for the Date object.
In the example below we set a Date object to a specific date (14th January 2010):

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

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

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

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

Compare Two Dates

The Date object is also used to compare two dates.
The following example compares today's date with the 14th January 2100:

```
var x=new Date();
x.setFullYear(2100,0,14);
var today = new Date();

if (x>today)
 {
        alert("Today is before 14th January 2100");
 }
else
 {
        alert("Today is after 14th January 2100");
 }
```

JavaScript Array Object

The Array object is used to store multiple values in a single variable.

What is an Array?

An array is a special variable, which can hold more than one value, at a time.
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1="Saab";
var car2="Volvo";
var car3="BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
The best solution here is to use an array!
An array can hold all your variable values under a single name. And you can access the values by referring to the array name.
Each element in the array has its own ID so that it can be easily accessed.

Create an Array

An array can be defined in three ways.
The following code creates an Array object called myCars:
1:

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

2:
var myCars=new Array("Saab","Volvo","BMW"); // condensed array
3:
var myCars=["Saab","Volvo","BMW"]; // literal array
Note: If you specify numbers or true/false values inside the array then the variable type will be Number or Boolean, instead of String.

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

The following code line:

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

will result in the following output:
Saab

Modify Values in an Array
To modify a value in an existing array, just add a new value to the array with a specified index number:
myCars[0]="Opel";
Now, the following code line:
document.write(myCars[0]);
will result in the following output:
Opel

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

Create a Boolean Object
The Boolean object represents two values: "true" or "false".
The following code creates a Boolean object called myBoolean:
var myBoolean=new Boolean();
If the Boolean object has no initial value, or if the passed value is one of the following:

- 0
- -0
- null
- ""
- false
- undefined
- NaN

the object is set to false. For any other value it is set to true (even with the string "false")!

JavaScript Math Object

The Math object allows you to perform mathematical tasks.

Math Object

The Math object allows you to perform mathematical tasks.
The Math object includes several mathematical constants and methods.

Syntax for using properties/methods of Math:

```
var x=Math.PI;
var y=Math.sqrt(16);
```

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

Mathematical Constants

JavaScript provides eight mathematical constants that can be accessed from the Math object. These are: E, PI, square root of 2, square root of 1/2, natural log of 2, natural log of 10, base-2 log of E, and base-10 log of E.
You may reference these constants from your JavaScript like this:

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

Mathematical Methods

In addition to the mathematical constants that can be accessed from the Math object there are also several methods available.
The following example uses the round() method of the Math object to round a number to the nearest integer:

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

The code above will result in the following output:

```
5
```

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

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

The code above can result in the following output:

```
0.8913913895179748
```

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

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

The code above can result in the following output:

```
3
```

JavaScript RegExp Object

RegExp, is short for regular expression.

What is RegExp?

A regular expression is an object that describes a pattern of characters.
When you search in a text, you can use a pattern to describe what you are searching for.
A simple pattern can be one single character.
A more complicated pattern can consist of more characters, and can be used for parsing, format checking, substitution and more.
Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

Syntax

```
var patt=new RegExp(pattern,modifiers);
```

or more simply:

```
var patt=/pattern/modifiers;
```

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

RegExp Modifiers

Modifiers are used to perform case-insensitive and global searches.
The i modifier is used to perform case-insensitive matching.
The g modifier is used to perform a global match (find all matches rather than stopping after the first match).

Example 1

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

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

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

Visit W3Schools

Example 2

Do a global search for "is":

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

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

Is this all there is?

Example 3

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

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

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

Is this all there is?

test()

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

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

Example

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

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

true

exec()

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

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

Example 1

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

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

e