- ✓ **JavaScript How To and Where To**
- ✓ **JavaScript Statements and Comments**
- ✓ **JavaScript Variables**
- ✓ **JavaScript Operators**
- ✓ **JavaScript Comparisons**
- ✓ **JavaScript If…Else**
- ✓ **JavaScript Loops**
- ✓ **JavaScript Flow Control Statements**
- ✓ **JavaScript Switch and Popup Boxes**
- ✓ **JavaScript Functions**
- ✓ **JavaScript Events**
- ✓ **JavaScript Try…Catch and Throw Statements**
- ✓ **JavaScript Special Characters and Guidelines**

1. JavaScript How To and Where To

1.1 How to Put a JavaScript into an HTML

Example

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

Another example

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

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

1.2 How to Handle Simple Browsers

Just add an HTML comment tag <!-- before the first JavaScript statement, and an end-of–comment tag --> after the last JavaScript statement, like this:

Example

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

1.3 Where to Put the JavaScript

Scripts in <head>

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

Scripts in <body>

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

Scripts in <head> and <body>

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

Example

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script></head>
<body onload="message()">
<script type="text/javascript">
document.write("This message is written by JavaScript");
</script>
</body>
</html>
```

1.4 Using an External JavaScript

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

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

2. JavaScript Statements and Comments

2.1 JavaScript Code

Example

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

Other Example

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

2.2 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.

Example

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

2.3 JavaScript Comments

1. Single line comments start with //.

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

2. Multiline comments start with /* and end with */.

3. JavaScript Variables

3.1 Algebra in 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 given to calculate the value of z to be 11?

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

3.2 JavaScript Variables

JavaScript variables are used to hold values or expressions.

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 underscore character, or a dollar sign. (The $ character is used primarily by code-generation tools.)
- Subsequent characters may be letter, number, underscore, or dollar sign.

Example

```
<html>
<body>
<script type="text/javascript">
var firstname;
firstname="Hege";
document.write(firstname);
document.write("<br />");
firstname="Tove";
document.write(firstname);
</script>
<p>The script above declares a variable, assigns a value to it, displays the value, changes the value,and displays the value again.</p>
</body>
</html>
```

3.3 Declaring (Creating) JavaScript Variables

You can declare JavaScript variables with the var statement:

Example

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

Other Example

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

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

3.4 Assigning Values to Undeclared JavaScript Variables

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

Example

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

Have the same effect as these:

Other Example

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

3.5 Redeclaring JavaScript Variables

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

Example

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

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

3.6 JavaScript Arithmetic

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

Example

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

Note: - Sometimes the results seem unpredictable. If at least one variable on the right side of an assignment expression contains a string value, the result will be a string and the "+" operator is applied as the concatenation operator to the toString() values of the variables. Only if all the variables to the right of the assignment operator are numbers will the result be a number.

4. JavaScript Operators

4.1 JavaScript Arithmetic Operators

The assignment operator, =, is used to assign values to JavaScript variables

Example

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

4.2 JavaScript 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

4.3 The + Operator Used on Strings

Example

```
txt1="What a very";
```

```
txt2="nice day";
txt3=txt1+txt2;
```

4.4 Adding Strings and Numbers

Example

```
<html>
<body>
<script type="text/javascript">
x=5+5;
document.write(x);
document.write("<br />");
x="5"+"5";
document.write(x);
document.write("<br />");
x=5+"5";
document.write(x);
document.write("<br />");
x="5"+5;
document.write(x);
document.write("<br />");
</script>
<p>The rule is: If you add a number and a string, the result
will be a string.</p>
</body>
</html>
```

5. JavaScript Comparisons

5.1 Comparison Operators

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

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

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

5.2 How to Use Comparisons

Example

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

5.3 Logical Operators

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

Given that *x* = 6 and *y* = 3, the following table 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

5.4 Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition. The syntax is as follows:

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

For example,

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

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

6. JavaScript If...Else Statements

6.1 Conditional Statements

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

JavaScript has 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.

6.2 if Statement

Use the if statement to execute some code only if a specified condition is true.
The **syntax** is as follows:

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

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

Example

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

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

```
</script>
<p>This example demonstrates the If statement.</p>
<p>If the time on your browser is less than 10, you will get a "Good morning" greeting.</p>
</body>
</html>
```

6.3 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.

The **syntax** is as follows:

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

Example

```
<html>
<body>
<script type="text/javascript">
var d = new Date();
var time = d.getHours();
if (time < 10)
{
document.write("<b>Good morning</b>");
}
else
{
document.write("<b>Good day</b>");
}
</script>
<p>
This example demonstrates the If...Else statement.
</p>
<p>
If the time on your browser is less than 10,
you will get a "Good morning" greeting.
Otherwise you will get a "Good day" greeting.
</p>
</body>
</html>
```

6.4 if...else if...else Statement

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

The **syntax** is as follows:

```
if (condition1)
{
code to be executed if condition1 is true
}
```

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

Example

```
<html>
<body>
<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>
<p>
This example demonstrates the if..else if...else statement.
</p>
</body>
</html>
```

7. JavaScript Loops

7.1 The for Loop

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

The **syntax** is as follows:

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

The following example 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">
for (i = 0; i <= 5; i++)
{
document.write("The number is " + i);
```

```
document.write("<br />");
}
</script>
<p>Explanation:</p>
<p>This for loop starts with i=0.</p>
<p>As long as <b>i</b> is less than, or equal to 5, the loop will continue to run.</p>
<p><b>i</b> will increase by 1 each time the loop runs.</p>
</body>
</html>
```

Another example

```
<html>
<body>
<script type="text/javascript">
for (i = 1; i <= 6; i++)
{
document.write("<h" + i + ">This is heading " + i);
document.write("</h" + i + ">");
}
</script>
</body>
</html>
```

7.2 The while Loop

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

The **syntax** is as follows:

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

Note: - The distinction between the for and the while is that in the for loop, the conditions are known and can be specified beforehand. The while loop is used when the initial conditions are known, but the terminal condition is discovered as the block is executed.

Example

```
<html>
<body>
<script type="text/javascript">
i=0;
while (i<=5)
{
document.write("The number is " + i);
document.write("<br />");
i++;
}
</script>
<p>Explanation:</p>
<p><b>i</b> is equal to 0.</p>
<p>While <b>i</b> is less than, or equal to, 5, the loop will continue to run.</p>
<p><b>i</b> will increase by 1 each time the loop runs.</p>
</body>
</html>
```

7.3 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.
The **syntax** is as follows:

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

The following example 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.

Note: The difference between the while and do...while loops should be characterized by whether the condition is checked before or after the block is executed. In the case of the while loop, the condition is checked first, so if false, the block will not be executed. In the do...while loop, the condition is checked ***after*** the block is executed;

Therefore the block is always executed at least once.

Example

```
<html>
<body>
<script type="text/javascript">
i = 0;
do
{
document.write("The number is " + i);
document.write("<br />");
i++;
}
while (i <= 5)
</script>
<p>Explanation:</p>
<p><b>i</b> equal to 0.</p>
<p>The loop will run</p>
<p><b>i</b> will increase by 1 each time the loop runs.</p>
<p>While <b>i</b> is less than, or equal to, 5, the loop will continue to run.</p>
</body>
</html>
```

8. Additional JavaScript Flow Control Statements.

The break and continue statements are used to control loop execution. The break statement can be used to halt execution of a loop if, for example, an error condition is encountered. The continue statement is used to begin the next iteration of a loop without executing all the statements in the block.

8.1 The break Statement

The break statement will terminate execution of 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>
<p>Explanation: The loop will break when i=3.</p>
</body>
</html>
```

8.2 The continue Statement

The continue statement will terminate the current iteration and restart the loop 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>
<p>Explanation: The loop will break the current loop and continue with the next value when i=3.</p>
</body>
</html>
```

8.3 JavaScript for...in Statement

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

The **syntax** is as follows:

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

Example

```
<html>
<body>
<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";
for (x in mycars)
```

```
{
document.write(mycars[x] + "<br />");
}
</script>
</body>
</html>
```

8.4 JavaScript switch Statement

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

The **syntax** is as follows:

```
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
}
```

```
<html>
<body>
<script type="text/javascript">
var i = 1;
switch (i)
{
case 0:
document.write("<b>i == 0</b><br>");
case 1:
document.write("<b>i == 1</b><br>");
case 2:
document.write("<b>i == 2</b><br>");
break;
case 3:
document.write("<b>i == 3</b><br>");
break;
default:
document.write("<b>i > 3</b><br>");
}
</script>
<p>Note that when i == 1 execution begins with case 1: but continues until the break statement is
encountered.</>
</body>
</html>
```

Example

```
<html>
<body>
<script type="text/javascript">
var d = new Date();
theDay=d.getDay();
switch (theDay)
{
```

```
case 5:
document.write("<b>Finally Friday</b>");
break;
case 6:
document.write("<b>Super Saturday</b>");
break;
case 0:
document.write("<b>Sleepy Sunday</b>");
break;
default:
document.write("<b>I'm really looking forward to this weekend!</b>");
}
</script>
<p>This JavaScript will generate a different greeting based
on what day it is. Note that Sunday=0, Monday=1, Tuesday=2, etc.</p>

</body>
</html>
```

9. JavaScript Popup Boxes

JavaScript has three types of popup boxes: alert box, confirm box, and prompt box.

9.1 Popup Boxes

9.1.1 Alert Box

An alert box is often used when you want to display information to the user.
When an alert box pops up, the user will have to click OK to proceed.
The **syntax** is as follows:

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

Example

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

9.1.2 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*.
The **syntax** is as follows:

```
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 a confirm box" />
</body>
</html>
```

9.1.3 **Prompt Box**

A prompt box is often used if you want the user to input a value while on a page or from 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. The **syntax** is as follows:

```
prompt(“sometext”,”defaultvalue”);
```

Example

```
<html>
<head>
<script type="text/javascript">
function disp_prompt()
{
var fname=prompt("Please enter your name:","Your name")
document.getElementById("msg").innerHTML="Greetings, " +
fname
}
</script>
</head>
<body>
<input type="button" onclick="disp_prompt()" value="Display
a prompt box" />
<br /><br />
<div id="msg"></div>
</body>
</html>
```

10. JavaScript Functions

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 ensure that a function is read/loaded by the browser before it is called, it should be placed in the <head> section.

10.1 **How to Define a Function**

The **syntax** is as follows:

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

The parameters var1, var2, and so on, 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 capitalization 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 capitalization as in the function declaration.

10.2 JavaScript Function Examples

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>
<p>By pressing the button above, a function will be called. The function will alert a message.</p>
</body>
</html>
```

Other Example

```
<html>
<head>
<script type="text/javascript">
function myfunction(txt)
{
alert(txt);
}
</script>
</head>
<body>
<form>
<input type="button" onclick="myfunction('Hello')" value="Call function">
</form>
<p>By pressing the button above, a function will be called with "Hello" as a parameter. The function
will alert the parameter.</p>
</body>
</html>
```

The following example shows how to let a function return a value.

Example

```
<html>
```

```
<head>
<script type="text/javascript">
function myFunction()
{
return ("Hello world!");
}
</script>
</head>
<body>
<script type="text/javascript"> document.write(myFunction())
</script>
</body>
</html>
```

10.3 The return Statement

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

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

Example

```
<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>
<p>The script in the body section calls a function with two parameters (4 and 3).</p>
<p>The function will return the product of these two parameters.</p>
</body>
</html>
```

10.4 The Lifetime of JavaScript Variables.

Local variables:

If you declare a variable within a function, the variable can be accessed only 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.

Global variables:

If you declare a variable outside a function, all the functions on your page can access it. These variables are called **global variables.**

The lifetime of these variables starts when they are declared and ends when the page is closed.

11. JavaScript Events

Events are actions that can be detected by JavaScript.

By using JavaScript, we have the ability to create dynamic Web pages.

Every element on a Web page has certain events that 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 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

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

11.1 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 often are 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!".

11.2 onFocus, onBlur, and onChange

The onFocus, onBlur, and onChange events are often used in combination with validation of form fields. The onFocus and onBlur events are complementary and are caused by the user clicking outside of the current window, frame, or form element or using the Tab key to move among fields or elements. When the user leaves an element, that element triggers a blur event. When the user moves to a new element, that element triggers a focus event.

Following 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()">
```

11.3 onSubmit

The onSubmit event may be used to validate form fields before submitting the form.

Note: Because the programmer controls the function executed on onSubmit, he can validate any, all, or no inputs as he sees fit.

Following 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()">
```

11.4 onMouseOver and onMouseOut

onMouseOver and onMouseOut often are used to create Rollover buttons. Following is an example of an onMouseOver event. An alert box appears when an onMouseOver event is detected:

```
<a href="http://www.google.com" onmouseover="alert('An onMouseOver event');return
```

```
false"><img src="s.gif" alt="Google" /></a>
```

11.5 onClick

The onClick event occurs when the user mouse clicks on a visible element on the screen. The following example could be used to translate text on a page when requested by the user:

```
<input type="button" name="language" value="Click for Spanish" onclick="translate()">
```

12. Javascript try…catch and throw Statements

The try…catch statement enables you to trap errors that occur during the execution of a block of code.

12.1 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 messages like this may be useful for developers but not for users. When users see errors, they often leave the Web page.

This chapter teaches you how to catch and handle JavaScript error messages so you don't lose your audience.

12.2 The try...catch Statement

The try…catch statement enables you to trap errors that occur during the execution of a block of code. The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.

The **syntax** is as follows:

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

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

The following example 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 message: " + 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>
```

12.3 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.

The syntax is as follows:
`throw(exception)`
The exception can be a string, integer, Boolean, or an object.

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

The following example determines the value of a variable called x. If the value of x is higher than 10, lower than 0, or not a number, we are going to throw an error. The error is then caught by the catch argument, and the proper error message is displayed:

Example

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 0 and 10:","");
try
{
if(x>10)
{
throw "Err1";
}
else if(x<0)
{
throw "Err2";
}
else if(isNaN(x))
{
throw "Err3";
}
}
catch(er)
{
if(er=="Err1")
{
alert("Error! The value is too high");
}
if(er=="Err2")
{
alert("Error! The value is too low");
```

```
}
if(er=="Err3")
{
alert("Error! The value is not a number");
}
}
</script>
</body>
</html>
```

13. JavaScript Special Characters and Guidelines.

13.1 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 preceding string will be chopped to *We are the so-called*.

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

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

JavaScript will now output the proper text string: *We are the so-called “Vikings” from the north.*
Here is another example:

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

The previous example will produce the following output:

```
You & I are singing!
```

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

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.

13.2 White Space

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

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

13.3 Break up a Code Line

Text in code statements contained within quotes is called a **string literal**. A string literal may not be broken across lines except by inserting the backslash character (\) at the point where you want to break the string:

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

The following will generate an “unterminated string literal” script error:

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

Another option is to use the concatenate operator (+) to break the string:
document.write("Hello "+"World!");

Code statements may be broken across lines, but the backslash character must not be used in this case.

The following is legal JavaScript:
document.write("Hello "+World!");

As a rule, break code statements or string literals across lines only when the length of the line or literal makes it difficult to read.

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

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

However, you cannot break up a code line like this:
document.write \("Hello World!");