Unit 3
1.5 Understanding JavaScript

1.5 JavaScript

JavaScript is THE scripting language of the Web and used in thousands of Web **pages to add functionality, validate forms, detect browsers, and much more**. It is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

1.5.1 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

JavaScript's official name is ECMAScript. ECMAScript is developed and maintained by the ECMA organization. ECMA-262 is the official JavaScript standard. The language was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996. The development of ECMA-262 started in 1996, and the first edition of was adopted by the ECMA General Assembly in June 1997. The standard was approved as an international ISO (ISO/IEC 16262) standard in 1998. The development of the standard is still in progress.

1.5.2 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 .

1.5.3 What can a JavaScript do?

- JavaScript gives HTML designers a programming tool - HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page** - A JavaScript statement like this:
  document.write("<h1>" + name + "</h1>");
  can write a variable text into an HTML page
- **JavaScript can react to events** - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element JavaScript can read and write HTML elements
- **A JavaScript can read and change the content** of an HTML element
- **JavaScript can be used to validate data** - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser – load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

1.5.4 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.

1.5.5 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 below 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. But 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.

1.5.6 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. Example 1.42 will write a heading and two paragraphs to a web page.

```
<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>
```

Example 1.42

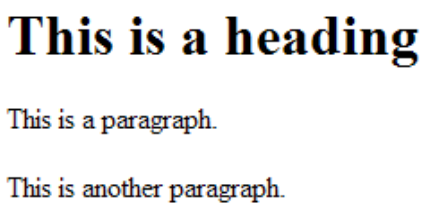

Figure 1.52

1.5.7 JavaScript Blocks

JavaScript statements can be grouped together in blocks. Blocks start with a left curly bracket {, and ends with a right curly bracket }. The purpose of a block is to make the sequence of statements execute together. This example will write a heading and two paragraphs to a web page.

```
<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>
```

Example 1.43

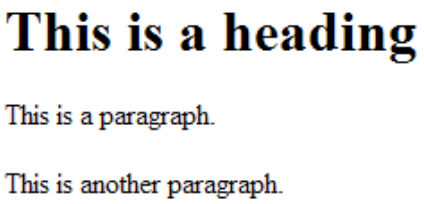

Figure 1.53

1.5.8 JavaScript cannot miss <script> and </script>

If you do not enter the **<script>** tag, the browser will treat the **document.write("Hello World!")** command as pure text, and just write the entire line on the page. See Example 1.44

```
<html>
<body>

document.write("Hello World!");

</body>
</html>
```

Example 1.44

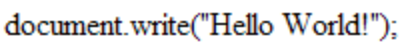

Figure 1.54

1.5.9 JavaScript Comments

Comments can be added to explain the JavaScript, or to make the code more readable. Single line comments start with //. Example 1.45 uses single line comments to explain the code.

```
<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>
```

Example 1.45

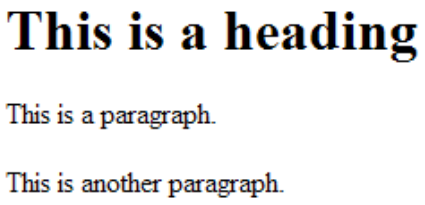

Figure 1.55

1.5.10 Using Comments to prevent execution

In the following examples show how the comment is used to prevent the execution of a single and multiple lines of code (can be suitable for debugging). See Example 1.46

```
<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>
```

Example 1.46

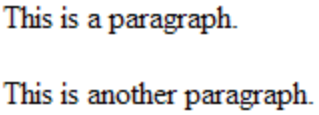

Figure 1.56

```
<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>
```

Example 1.47

Figure 1.57

Nothing in the output as everything is commented out

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

Example 1.48 – End of statement comment

Hello Dolly!

Figure 1.58

1.5.11 How to Handle Simple Browsers

Browsers that do not support JavaScript, will display JavaScript as page content. To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript. Just add an HTML comment tag <!-- before the first JavaScript statement, and a --> (end of comment. (See example 1.49) The two forward slashes at the end of comment line (//) is the JavaScript comment symbol. This prevents JavaScript from executing the --> tag.

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

Example 1.49

1.5.12 Where to Put the JavaScript

- JavaScripts **in the body** section will be executed WHILE the page loads.
- JavaScripts **in the head** section will be executed when CALLED.

1.5.12.1 Scripts in <head>

Scripts to be executed when they are called, or when an event is triggered, go in the head section. If you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

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

Example 1.50

Figure 1.59

1.5.12.2 Scripts in <body>

Scripts to be executed when the page loads go in the body section. If you place a script in the body section, it generates the content of a page.

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

Example 1.51

This message is written by JavaScript

Figure 1.60

1.5.12.3 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.

1.5.12.4 Using an External JavaScript

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

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

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

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

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

<p>
The actual script is in an external script file called "xxx.js".
</p>

</body>
</html>
```

Example 1.52

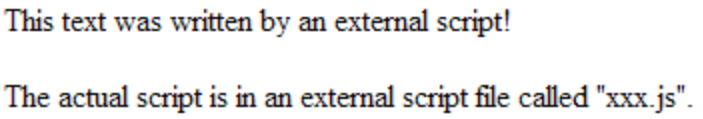

Figure 1.61

See this example:
http://labs.adobe.com/technologies/spry/articles/best_practices/separating_behavior.html

1.5.13 JavaScript variables and Operations

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

1.5.13.1 Rules for JavaScript variable names.

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

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

Examples:

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

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

1.5.13.2 Assigning Values to Undeclared JavaScript Variables

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

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

Have the same effect as

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

1.5.13.3 Re-declaring JavaScript variables.

If you re-declare a JavaScript variable, it will not lose its original value.

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

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

1.5.14 JavaScript variables and Operations

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

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

Figure 1.62 shows all possible arithmetic operators.

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

Figure 1.62

Figure 1.63 shows examples of 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

Figure 1.63

1.5.14.1 The + Operator can be used on Strings

Examples:

```
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".

Strings and numbers can be added directly using the + operator.

```
<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>
```

```
10
55
55
55

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

Figure 1.64

Example 1.53

1.5.14.2 Comparison Operations

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

Given that x=5, the table in Figure 1.65 explains the comparison operators:

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

Figure 1.65

You can do comparisons like this: `if (age<18) document.write("Too young");`

1.5.14.3 Logical Operations

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

Figure 1.64

1.5.14.4 Conditional Operations

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

Syntax

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

Example

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

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

1.5.15 JavaScript Conditional Statements

1.5.15.1 `if , it-else` statements

As in any other high level language, you can use a if and if-else statements in JavaScript at ease for conditional statements. the JavaScript if/else has several varieties.

- 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

** Note that all these syntaxes are lowercase and of you use upper case (such as IF or If) you will get JavaScript error.

```
<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>
```

Example 1.54

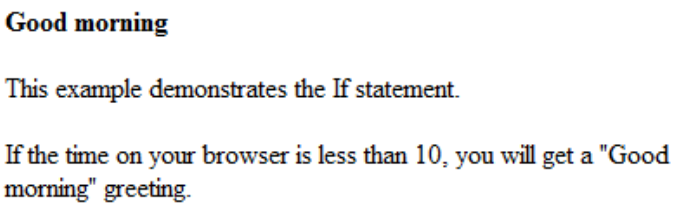

Figure 1.65

1.5.15.2 `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
}
```

1.5.15.3`if/else` statement

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

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
 }
```

```
<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>
```

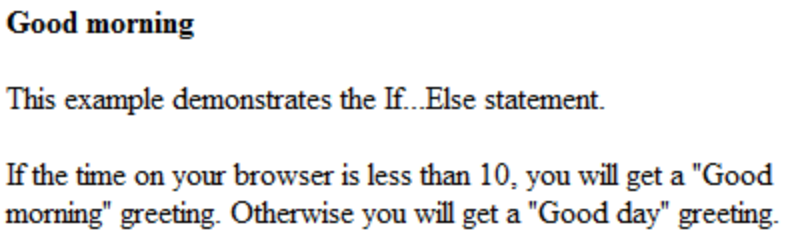

Figure 1.66

1.5.15.4 `if..else if…else` Statement

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

Syntax

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

```
<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>
```

Example 1.56

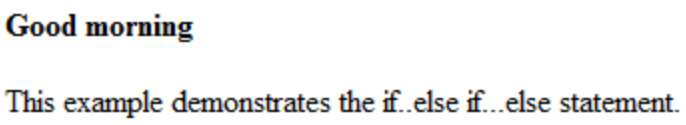

Figure 1.67

1.5.17 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 match, the block of code associated with that case is executed. compared with the values for each case in the structure. If there is a Use break to prevent the code from running into the next case automatically.

```
<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>
```

Example 1.57

Super Saturday

This JavaScript will generate a different greeting based on what day it is. Note that Sunday=0, Monday=1, Tuesday=2, etc.

Figure 1.68

1.5.18 JavaScript Popup Boxes

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

1.5.18.1 Alert Box

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

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

Syntax

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

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

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

</body>
</html>
```

Example 1.58

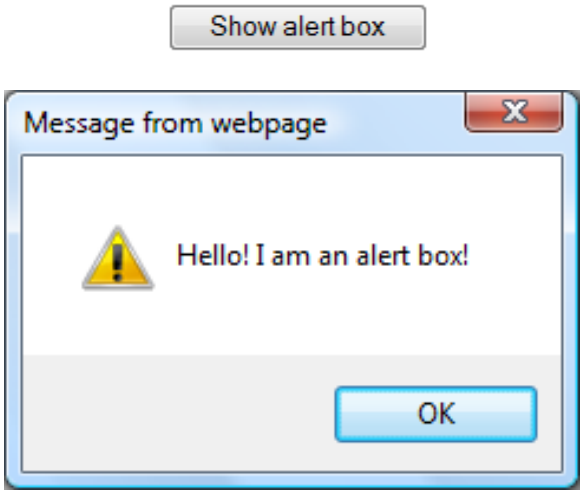

Figure 1.69

1.5.18.2 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.

Syntax

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

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

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

</body>
</html>
```

Example 1.59

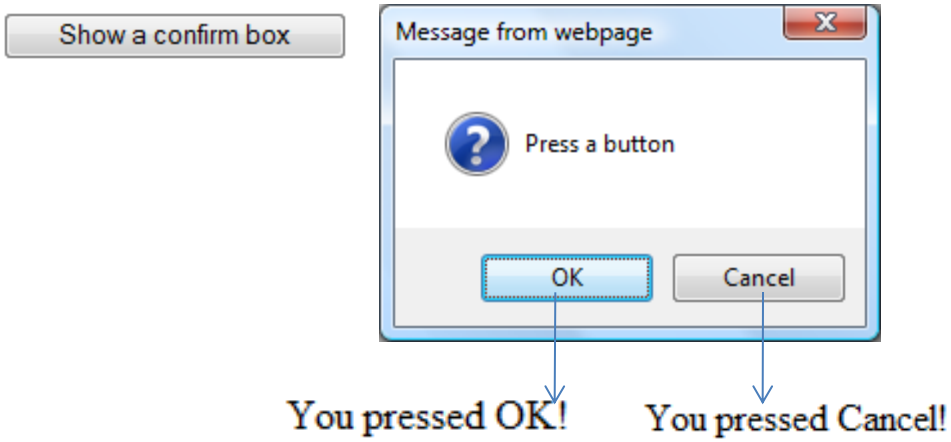

Figure 1.70

1.5.18.3 Prompt Box

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

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

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

Syntax

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

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

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

</body>
</html>
```

Example 1.60

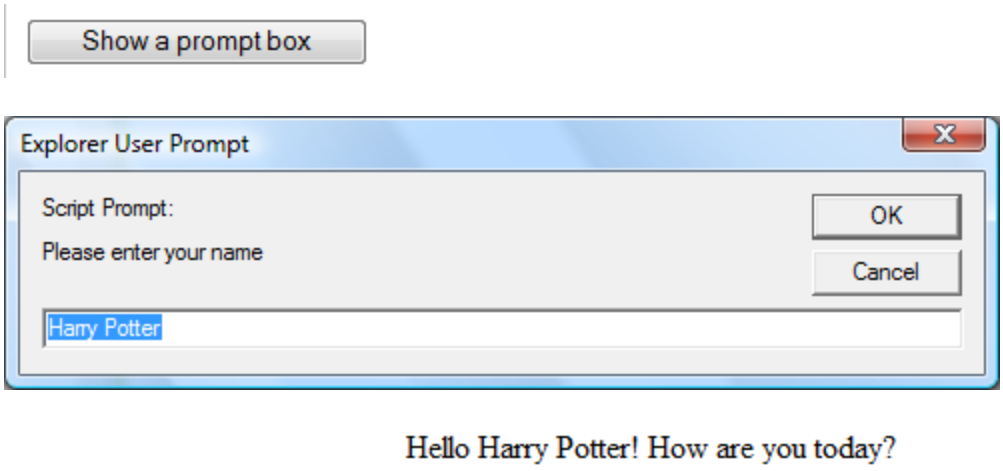

Figure 1.71

1.5.19 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

1.5.19.1 `for` loop

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

Syntax

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

```
<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>
```

Example 1.61

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

Explanation:

This for loop starts with i=0.

As long as **i** is less than, or equal to 5, the loop will continue to run.

i will increase by 1 each time the loop runs.

Figure 1.72

The example shown 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.

1.5.19.1 `while` loop

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

Syntax

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

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

```
<html>
<body>

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

Example 1.62

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

Explanation:

This for loop starts with i=0.

As long as **i** is less than, or equal to 5, the loop will continue to run.

i will increase by 1 each time the loop runs.

Figure 1.73

1.5.19.2 do - while loop

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

Syntax

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

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:

```
<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>
```

Example 1.63

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

Explanation:

This for loop starts with i=0.

As long as **i** is less than, or equal to 5, the loop will continue to run.

i will increase by 1 each time the loop runs.

Figure 1.74

1.5.19.3 Some Valuable Examples

A **timer, calendar** and a **page counter** are three common javascript examples we normally see in many web pages. Here are two sites you can find the javascript code for these client side controls. Take a look.

A Java Script Timer Example:
http://javascript.internet.com/time-date/add-time.html
Another One:
http://www.javascriptsearch.com/scripts/Counters/simple_counter.html

A javascript Calendar Example:
http://www.dynamicdrive.com/dynamicindex7/basiccalendar.htm

A Javascript Visitor Counter Example:
http://www.javascriptkit.com/script/script2/counter.shtml

If you are working with javascript keep a browser open to
http://www.w3schools.com/jsref