MODULE OF INTERNET AND WEB TECHNOLOGIES

TABLE OF CONTENTS

Chapter 1: HTML

1.1 Introduction

What is HTML?

HTML is a language for describing web pages.

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is not a programming language, it is a **markup language**
- A markup language is a set of **markup tags**
- HTML uses **markup tags** to describe web pages

HTML Tags

HTML markup tags are usually called HTML tags

- HTML tags are keywords surrounded by **angle brackets** like <html>
- HTML tags normally **come in pairs** like <b> and </b>
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- Start and end tags are called **opening tags** and **closing tags**.

HTML Documents = Web Pages

- HTML documents **describe web pages**
- HTML documents **contain HTML tags** and plain text
- HTML documents are also **called web pages**

The purpose of a web browser (like Internet Explorer or Firefox) is to read HTML documents and display them as web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page:

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

```
<h1>My First Heading</h1>

<p>My first paragraph</p>

</body>
</html>
```

Example Explained

- The text between <html> and </html> describes the web page
- The text between <body> and </body> is the visible page content
- The text between <h1> and </h1> is displayed as a heading
- The text between <p> and </p> is displayed as a paragraph

1.2 HTML Basic

Editing HTML

In this tutorial we use a plain text editor (like Notepad) to edit HTML. We believe this is the best way to learn HTML.

However, professional web developers often prefer HTML editors like FrontPage or Dreamweaver, instead of writing plain text.

HTML Headings

HTML headings are defined with the <h1> to <h6> tags.

Example

```
<h1>This is a heading</h1>
<h2>This is a heading</h2>
<h3>This is a heading</h3
```

HTML Paragraphs

HTML paragraphs are defined with the <p> tag.

Example

```
<p>This is a paragraph</p>
```

```
<p>This is another paragraph</p>
```

HTML Links

HTML links are defined with the <a> tag.

Example

```
<a href="http://www.yahoo.com">Yahoo!</a>
```

HTML Images

HTML images are defined with the <img> tag.

Example

```
<img src="usamap.jpg" width="104" height="142" />
```

1.3 HTML Elements

HTML documents are defined by HTML elements.

HTML Elements

An HTML element is everything from the start tag to the end tag:

Start tag *	Element content	End tag *
<p>	This is a paragraph	</p>
<a href="default.htm" >	This is a link	</a>
<br />		

* The start tag is often called the **opening tag**. The end tag is often called the **closing tag**.

HTML Element Syntax

- An HTML element starts with a **start tag / opening tag**
- An HTML element ends with an **end tag / closing tag**
- The **element content** is everything between the start and the end tag
- Some HTML elements have **empty content**
- Empty elements are **closed in the start tag**
- Most HTML elements can have **attributes**

Nested HTML Elements

Most HTML elements can be nested (can contain other HTML elements).

HTML documents consist of nested HTML elements.

HTML Document Example

```
<html>

<body>
<p>This is my first paragraph</p>
</body>

</html>
```

The example above contains 3 HTML elements.

Example Explained

The <p> element:

```
<p>This is my first paragraph</p>
```

The <p> element defines a paragraph in the HTML document
The element has a start tag <p> and an end tag </p>
The element content is: This is my first paragraph

The <body> element:

```
<body>
<p>This is my first paragraph</p>
</body>
```

The <body> element defines the body of the HTML document
The element has a start tag <body> and an end tag </body>
The element content is another HTML element (a paragraph)

The <html> element:

```
<html>

<body>
<p>This is my first paragraph</p>
```

```
</body>

</html>
```

The <html> element defines the whole HTML document.
The element has a start tag <html> and an end tag </html>
The element content is another HTML element (the body)

Don't Forget the End Tag

Most browsers will display HTML correctly even if you forget the end tag:

```
<p>This is a paragraph
<p>This is a paragraph
```

The example above will work in most browsers, but don't rely on it. Forgetting the end tag can produce unexpected results or errors.

Empty HTML Elements

HTML elements without content are called empty elements. Empty elements can be closed in the start tag.

<br> is an empty element without a closing tag (it defines a line break).

HTML Tip: Use Lowercase Tags

HTML tags are not case sensitive: <P> means the same as <p>. Plenty of web sites use uppercase HTML tags in their pages.

1.4 HTML Attributes

Attributes provide additional information about HTML elements.

HTML Attributes

- HTML elements can have **attributes**
- Attributes provide **additional information** about the element
- Attributes are always specified in **the start tag**
- Attributes come in name/value pairs like: **name="value"**

Attribute Example

HTML links are defined with the <a> tag. The link address is provided as an attribute:

Example

```
<a href="http://www.w3schools.com">This is a link</a>
```

Always Quote Attribute Values

Attribute values should always be enclosed in quotes.

Double style quotes are the most common, but single style quotes are also allowed.

In some rare situations, like when the attribute value itself contains quotes, it is necessary to use single quotes:

name='John "ShotGun" Nelson'

1.5 HTML Headings

Headings are important in HTML documents.

HTML Headings

Headings are defined with the <h1> to <h6> tags.

<h1> defines the largest heading. <h6> defines the smallest heading.

Example

```
<h1>This is a heading</h1>
<h2>This is a heading</h2>
<h3>This is a heading</h3>
```

Headings Are Important

Use HTML headings for headings only. Don't use headings to make text **BIG** or **bold**.

Search engines use your headings to index the structure and content of your web pages.

Since users may skim your pages by its headings, it is important to use headings to show the document structure.

H1 headings should be used as main headings, followed by H2 headings, then less important H3 headings, and so on.

HTML Rules (Lines)

The <hr /> tag is used to create a horizontal rule (line).

Example

```
<p>This is a paragraph</p>
<hr />
<p>This is a paragraph</p>
<hr />
<p>This is a paragraph</p>
```

HTML Comments

Comments can be inserted in the HTML code to make it more readable and understandable. Comments are ignored by the browser and are not displayed.

Comments are written like this:

Example

```
<!-- This is a comment -->
```

1.6 HTML Paragraphs

HTML documents are divided into paragraphs.

HTML Paragraphs

Paragraphs are defined with the <p> tag.

Example

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

Note: Browsers automatically adds an empty line before and after paragraphs.

Don't Forget the End Tag

Most browsers will display HTML correctly even if you forget the end tag:

Example

```
<p>This is a paragraph
<p>This is another paragraph
```

The example above will work in most browsers, but don't rely on it. Forgetting the end tag can produce unexpected results or errors.

HTML Line Breaks

Use the <br /> tag if you want a line break (a new line) without starting a new paragraph:

Example

```
<p>This is<br />a para<br />graph with line breaks</p>
```

The <br /> element is an empty HTML element. It has no end tag.

<br> or <br />

Even if <br> works in all browsers, writing <br /> instead is more **future proof**.

1.7 HTML **Text Formatting**

HTML Text Formatting

This text is bold

This text is big

This text is italic

`This is computer output`

This is $_{\text{subscript}}$ and $^{\text{superscript}}$

HTML Formatting Tags

HTML uses tags like <b> and <i> for formatting output, like **bold** or *italic* text.

These HTML tags are called formatting tags.

Text Formatting Tags

Tag	Description
<b>	Defines bold text
<big>	Defines big text
<em>	Defines emphasized text
<i>	Defines italic text
<small>	Defines small text
<strong>	Defines strong text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
<del>	Defines deleted text

1.8 HTML Styles

The HTML Style Attribute

The purpose of the style attribute is:

To provide a common way to style all HTML elements.

HTML Style Examples

style="background-color:yellow"

style="font-size:10px"

style="font-family:Times"

style="text-align:center"

Style Examples:

<body style="background-color:yellow">

The style attribute defines a style for the <body> element.

```
<html>
<body style="background-color:yellow">
<h2>Look: Colored Background!</h2>
</body>
</html>
```

Font Family, Color and Size

<p style="font-family:courier new; color:red; font-size:20px">

The style attribute defines a style for the <p> element.

```
<html>
<body>
<h1 style="font-family:verdana">A heading</h1>
<p style="font-family:courier new; color:red; font-size:20px;">A paragraph</p>
</body>
</html>
```

Text Alignment

<h1 style="text-align:center">

The style attribute defines a style for the <h1> element.

```
<html>
<body>

<h1 style="text-align:center">This is heading 1</h1>

<p>The heading above is aligned to the center of this page. The heading above is aligned
to the center of this page. The heading above is aligned to the center of this page.</p>

</body>
</html>
```

1.9 HTML Links

A link is the "address" to a document (or a resource) on the web.

Hyperlinks, Anchors, and Links

In web terms, a hyperlink is a reference (an address) to a resource on the web.

Hyperlinks can point to any resource on the web: an HTML page, an image, a sound file, a movie, etc.

An anchor is a term used to define a hyperlink destination inside a document.

The HTML anchor element <a>, is used to define both hyperlinks and anchors.

We will use the term HTML link when the <a> element points to a resource, and the term HTML anchor when the <a> elements defines an address inside a document..

The href Attribute

The **href attribute** defines the link "address".

This <a> element defines a link to W3Schools:

```
<a href="http://www.w3schools.com/">Visit W3Schools!</a>
```

The target Attribute

The **target attribute** defines **where** the linked document will be opened.

The code below will open the document in a new browser window:

Example

```
<a href="http://www.w3schools.com/"
target="_blank">Visit W3Schools!</a>
```

1.10 HTML Images

The Image Tag and the Src Attribute

In HTML, images are defined with the <img> tag.

The <img> tag is empty, which means that it contains attributes only and it has no closing tag.

To display an image on a page, you need to use the src attribute. Src stands for "source". The value of the src attribute is the URL of the image you want to display on your page.

The syntax of defining an image:

```
<img src="url" />
```

The URL points to the location where the image is stored. An image named "boat.gif" located in the directory "images" on "www.ulk.ac.rw" has the URL: http://www.ulk.ac.rw/images/boat.gif.

The browser puts the image where the image tag occurs in the document. If you put an image tag between two paragraphs, the browser shows the first paragraph, then the image, and then the second paragraph.

The Alt Attribute

The alt attribute is used to define an "alternate text" for an image. The value of the alt attribute is an author-defined text:

```
<img src="boat.gif" alt="Big Boat" />
```

The "alt" attribute tells the reader what he or she is missing on a page if the browser can't load images. The browser will then display the alternate text instead of the image. It is a good practice to include the "alt" attribute for each image on a page, to improve the display and usefulness of your document for people who have text-only browsers.

1.11 HTML Tables

HTML Tables

Apples	44%
Bananas	23%
Oranges	13%
Other	10%

Tables

Tables are defined with the <table> tag. A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). The letters td stands for "table data," which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc.

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

How it looks in a browser:

row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

Tables and the Border Attribute

If you do not specify a border attribute the table will be displayed without any borders. Sometimes this can be useful, but most of the time, you want the borders to show.

To display a table with borders, you will have to use the border attribute:

```
<table border="1">
<tr>
<td>Row 1, cell 1</td>
<td>Row 1, cell 2</td>
</tr>
</table>
```

Headings in a Table

Headings in a table are defined with the <th> tag.

```
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

How it looks in a browser:

Heading	Another Heading
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

Empty Cells in a Table

Table cells with no content are not displayed very well in most browsers.

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td></td>
</tr>
</table>
```

How it looks in a browser:

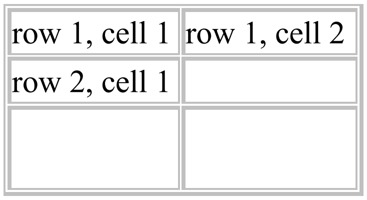
row 1, cell 1	row 1, cell 2
row 2, cell 1	

Note that the borders around the empty table cell are missing (NB! Mozilla Firefox displays the border).

To avoid this, add a non-breaking space (&nbsp;) to empty data cells, to make the borders visible:

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>&nbsp;</td>
</tr>
</table>
```

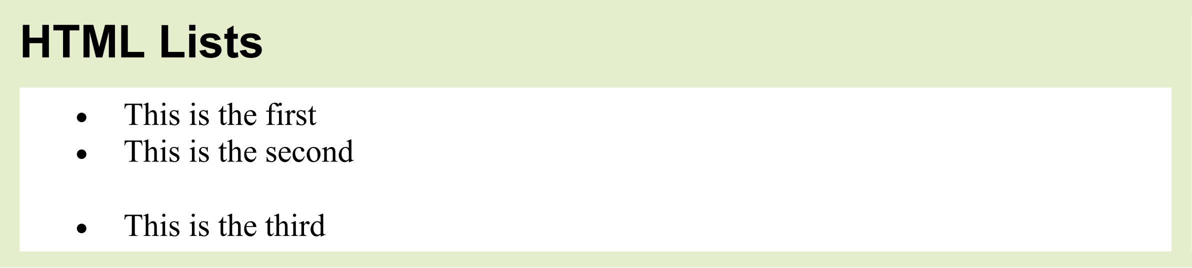
How it looks in a browser:



row 1, cell 1	row 1, cell 2
row 2, cell 1	

1.12 HTML Lists

HTML supports ordered, unordered and definition lists.



HTML Lists

- This is the first
- This is the second

- This is the third

Unordered Lists

An unordered list is a list of items. The list items are marked with bullets (typically small black circles).

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag.

```
<ul>
<li>Coffee</li>
<li>Milk</li>
</ul>
```

Here is how it looks in a browser:

- Coffee
- Milk

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

Ordered Lists

An ordered list is also a list of items. The list items are marked with numbers.

An ordered list starts with the <ol> tag. Each list item starts with the <li> tag.

```
<ol>
<li>Coffee</li>
<li>Milk</li>
</ol>
```

Here is how it looks in a browser:

1. Coffee
2. Milk

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

Definition Lists

A definition list is not a list of single items. It is a list of items (terms), with a description of each item (term).

A definition list starts with a <dl> tag (**d**efinition **l**ist).

Each term starts with a <dt> tag (**d**efinition **t**erm).

Each description starts with a <dd> tag (**d**efinition **d**escription).

```
<dl>
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink</dd>
</dl>
```

Here is how it looks in a browser:

```
Coffee
        Black hot drink
Milk
        White cold drink
```

Inside the <dd> tag you can put paragraphs, line breaks, images, links, other lists, etc.

List Tags

Tag	Description
<ol>	Defines an ordered list
<ul>	Defines an unordered list
<li>	Defines a list item
<dl>	Defines a definition list
<dt>	Defines a term (an item) in a definition list
<dd>	Defines a description of a term in a definition list

1.13 HTML Forms and Input

HTML Forms are used to select different kinds of user input.

Forms

A form is an area that can contain form elements.

Form elements are elements that allow the user to enter information (like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.

A form is defined with the <form> tag.

```
<form>
.
input elements
```

```
.
</form>
```

Input

The most used form tag is the <input> tag. The type of input is specified with the type attribute. The most commonly used input types are explained below.

Text Fields

Text fields are used when you want the user to type letters, numbers, etc. in a form.

```
<form>
First name:
<input type="text" name="firstname" />
<br />
Last name:
<input type="text" name="lastname" />
</form>
```

How it looks in a browser:

First name:

Last name:

Note that the form itself is not visible. Also note that in most browsers, the width of the text field is 20 characters by default.

Radio Buttons

Radio Buttons are used when you want the user to select one of a limited number of choices.

```
<form>
<input type="radio" name="sex" value="male" /> Male
<br />
<input type="radio" name="sex" value="female" /> Female
</form>
```

How it looks in a browser:

Male

Female

Note that only one option can be chosen.

Checkboxes

Checkboxes are used when you want the user to select one or more options of a limited number of choices.

```
<form>
I have a bike:
<input type="checkbox" name="vehicle" value="Bike" />
<br />
I have a car:
<input type="checkbox" name="vehicle" value="Car" />
<br />
I have an airplane:
<input type="checkbox" name="vehicle" value="Airplane" />
</form>
```

How it looks in a browser:

I have a bike: □

I have a car: □

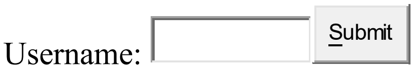
I have an airplane: □

The Form's Action Attribute and the Submit Button

When the user clicks on the "Submit" button, the content of the form is sent to the server. The form's action attribute defines the name of the file to send the content to. The file defined in the action attribute usually does something with the received input.

```
<form name="input" action="html_form_submit.asp" method="get">
Username:
<input type="text" name="user" />
<input type="submit" value="Submit" />
</form>
```

How it looks in a browser:



Username:

If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "html_form_submit.asp". The page will show you the received input.

1.14 HTML Colors

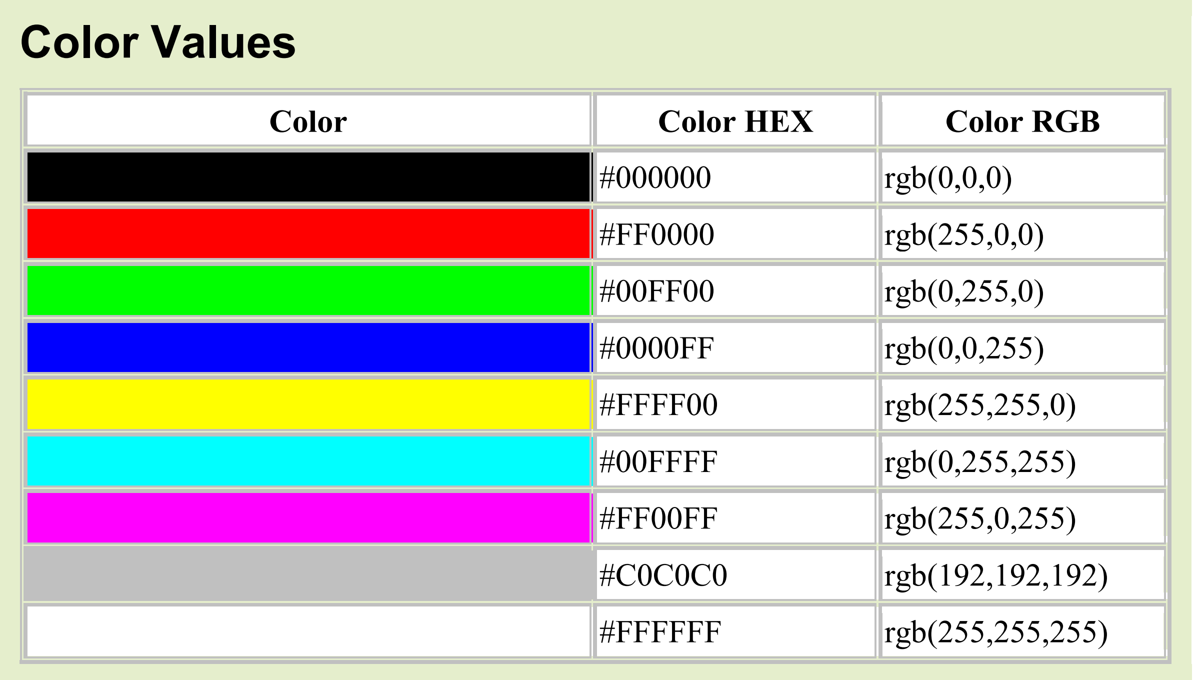
Colors are displayed combining RED, GREEN, and BLUE light.

Color Values

HTML colors are defined using a hexadecimal (hex) notation for the combination of Red, Green, and Blue color values (RGB).

The lowest value that can be given to one of the light sources is 0 (hex 00). The highest value is 255 (hex FF).

Hex values are written as 3 double digit numbers, starting with a # sign.



Color Values

Color	Color HEX	Color RGB
⬛	#000000	rgb(0,0,0)
🟥	#FF0000	rgb(255,0,0)
🟩	#00FF00	rgb(0,255,0)
🟦	#0000FF	rgb(0,0,255)
🟨	#FFFF00	rgb(255,255,0)
▩	#00FFFF	rgb(0,255,255)
🟪	#FF00FF	rgb(255,0,255)
▨	#C0C0C0	rgb(192,192,192)
□	#FFFFFF	rgb(255,255,255)

1.15 HTML Color Names

Sorted by Names

Link: Same list sorted by values

Color Name	Color HEX	Color
AliceBlue	#F0F8FF	■
AntiqueWhite	#FAEBD7	■
Aqua	#00FFFF	■
Aquamarine	#7FFFD4	■
Azure	#F0FFFF	■
Beige	#F5F5DC	■
Bisque	#FFE4C4	■
Black	#000000	■
BlanchedAlmond	#FFEBCD	■
Blue	#0000FF	■
BlueViolet	#8A2BE2	■
Brown	#A52A2A	■
BurlyWood	#DEB887	■
CadetBlue	#5F9EA0	■
Chartreuse	#7FFF00	■
Chocolate	#D2691E	■
Coral	#FF7F50	■
CornflowerBlue	#6495ED	■
Cornsilk	#FFF8DC	■
Crimson	#DC143C	■
Cyan	#00FFFF	■
DarkBlue	#00008B	■
DarkCyan	#008B8B	■
DarkGoldenRod	#B8860B	■
DarkGray	#A9A9A9	■
DarkGreen	#006400	■
DarkKhaki	#BDB76B	■
DarkMagenta	#8B008B	■
DarkOliveGreen	#556B2F	■
Darkorange	#FF8C00	■
DarkOrchid	#9932CC	■
DarkRed	#8B0000	■

DarkSalmon	#E9967A	🟧
DarkSeaGreen	#8FBC8F	🟩
DarkSlateBlue	#483D8B	🟪
DarkSlateGray	#2F4F4F	⬛
DarkTurquoise	#00CED1	🟦
DarkViolet	#9400D3	🟪
DeepPink	#FF1493	🟥
DeepSkyBlue	#00BFFF	🟦
DimGray	#696969	⬛
DodgerBlue	#1E90FF	🟦
FireBrick	#B22222	🟥
FloralWhite	#FFFAF0	⬜
ForestGreen	#228B22	🟩
Fuchsia	#FF00FF	🟪
Gainsboro	#DCDCDC	⬜
GhostWhite	#F8F8FF	⬜
Gold	#FFD700	🟨
GoldenRod	#DAA520	🟨
Gray	#808080	⬛
Green	#008000	🟩
GreenYellow	#ADFF2F	🟩
HoneyDew	#F0FFF0	⬜
HotPink	#FF69B4	🟥
IndianRed	#CD5C5C	🟥
Indigo	#4B0082	🟪
Ivory	#FFFFF0	⬜
Khaki	#F0E68C	🟨
Lavender	#E6E6FA	⬜
LavenderBlush	#FFF0F5	⬜
LawnGreen	#7CFC00	🟩
LemonChiffon	#FFFACD	🟨
LightBlue	#ADD8E6	🟦
LightCoral	#F08080	🟥
LightCyan	#E0FFFF	⬜
LightGoldenRodYellow	#FAFAD2	🟨
LightGrey	#D3D3D3	⬜
LightGreen	#90EE90	🟩
LightPink	#FFB6C1	🟥
LightSalmon	#FFA07A	🟧
LightSeaGreen	#20B2AA	🟩

LightSkyBlue	#87CEFA	🟦
LightSlateGray	#778899	⬛
LightSteelBlue	#B0C4DE	🟦
LightYellow	#FFFFE0	🟨
Lime	#00FF00	🟩
LimeGreen	#32CD32	🟩
Linen	#FAF0E6	⬜
Magenta	#FF00FF	🟪
Maroon	#800000	🟥
MediumAquaMarine	#66CDAA	🟩
MediumBlue	#0000CD	🟦
MediumOrchid	#BA55D3	🟪
MediumPurple	#9370D8	🟪
MediumSeaGreen	#3CB371	🟩
MediumSlateBlue	#7B68EE	🟪
MediumSpringGreen	#00FA9A	🟩
MediumTurquoise	#48D1CC	🟦
MediumVioletRed	#C71585	🟥
MidnightBlue	#191970	🟦
MintCream	#F5FFFA	⬜
MistyRose	#FFE4E1	⬜
Moccasin	#FFE4B5	🟨
NavajoWhite	#FFDEAD	🟨
Navy	#000080	🟦
OldLace	#FDF5E6	⬜
Olive	#808000	🟫
OliveDrab	#6B8E23	🟩
Orange	#FFA500	🟧
OrangeRed	#FF4500	🟧
Orchid	#DA70D6	🟪
PaleGoldenRod	#EEE8AA	🟨
PaleGreen	#98FB98	🟩
PaleTurquoise	#AFEEEE	🟦
PaleVioletRed	#D87093	🟥
PapayaWhip	#FFEFD5	⬜
PeachPuff	#FFDAB9	🟧
Peru	#CD853F	🟫
Pink	#FFC0CB	🟥
Plum	#DDA0DD	🟪
PowderBlue	#B0E0E6	🟦

Purple	#800080	🟪
Red	#FF0000	🟥
RosyBrown	#BC8F8F	🟫
RoyalBlue	#4169E1	🟦
SaddleBrown	#8B4513	🟫
Salmon	#FA8072	🟧
SandyBrown	#F4A460	🟧
SeaGreen	#2E8B57	🟩
SeaShell	#FFF5EE	⬜
Sienna	#A0522D	🟫
Silver	#C0C0C0	⬜
SkyBlue	#87CEEB	🟦
SlateBlue	#6A5ACD	🟪
SlateGray	#708090	⬛
Snow	#FFFAFA	⬜
SpringGreen	#00FF7F	🟩
SteelBlue	#4682B4	🟦
Tan	#D2B48C	🟫
Teal	#008080	🟦
Thistle	#D8BFD8	🟪
Tomato	#FF6347	🟥
Turquoise	#40E0D0	🟦
Violet	#EE82EE	🟪
Wheat	#F5DEB3	🟨
White	#FFFFFF	⬜
WhiteSmoke	#F5F5F5	⬜
Yellow	#FFFF00	🟨
YellowGreen	#9ACD32	🟩

Note: The names above are not a part of the W3C web standard.

The W3C HTML and CSS standards have listed only 16 valid color names:
aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow.

1.16 HTML Quick List

HTML Basic Document

```
<html>
<head>
<title>Document name goes here</title>
</head>

<body>
Visible text goes here
</body>

</html>
```

Heading Elements

```
<h1>Largest Heading</h1>

<h2> . . . </h2>
<h3> . . . </h3>
<h4> . . . </h4>
<h5> . . . </h5>

<h6>Smallest Heading</h6>
```

Text Elements

```
<p>This is a paragraph</p>
<br> (line break)
<hr> (horizontal rule)
<pre>This text is preformatted</pre>
```

Logical Styles

```
<em>This text is emphasized</em>
<strong>This text is strong</strong>
<code>This is some computer code</code>
```

Physical Styles

```
<b>This text is bold</b>
<i>This text is italic</i>
```

Links, Anchors, and Image Elements

```
<a href="http://www.example.com/">This is a Link</a>
<a href="http://www.example.com/"><img src="URL" alt="Alternate Text"></a>
<a href="mailto:webmaster@example.com">Send e-mail</a>
```

A named anchor:

```
<a name="tips">Useful Tips Section</a>
<a href="#tips">Jump to the Useful Tips Section</a>
```

Unordered list

```
<ul>
<li>First item</li>
<li>Next item</li>
</ul>
```

Ordered list

```
<ol>
<li>First item</li>
<li>Next item</li>
</ol>
```

Definition list

```
<dl>
<dt>First term</dt>
<dd>Definition</dd>
<dt>Next term</dt>
<dd>Definition</dd>
</dl>
```

Tables

```
<table border="1">
<tr>
<th>someheader</th>
<th>someheader</th>
</tr>
<tr>
<td>sometext</td>
<td>sometext</td>
</tr>
</table>
```

Frames

```
<frameset cols="25%,75%">
  <frame src="page1.htm">
  <frame src="page2.htm">
</frameset>
```

Forms

```
<form action="http://www.example.com/test.asp" method="post/get">

<input type="text" name="lastname" value="Nixon" size="30" maxlength="50">
<input type="password">
<input type="checkbox" checked="checked">
```

```
<input type="radio" checked="checked">
<input type="submit">
<input type="reset">
<input type="hidden">

<select>
<option>Apples
<option selected>Bananas
<option>Cherries
</select>

<textarea name="Comment" rows="60" cols="20"></textarea>

</form>
```

Entities

&lt; is the same as <
&gt; is the same as >
&#169; is the same as ©

Other Elements

```
<!-- This is a comment -->

<blockquote>
Text quoted from some source.
</blockquote>

<address>
Address 1<br>
Address 2<br>
City<br>
</address>
```

Chapter 2: Cascading Style Sheets

2.1 Introduction

What is CSS?

- **CSS** stands for **C**ascading **S**tyle **S**heets
- Styles define **how to display** HTML elements
- Styles are normally stored in **Style Sheets**
- Styles were added to HTML 4.0 **to solve a problem**
- **External Style Sheets** can save a lot of work
- External Style Sheets are stored in **CSS files**
- Multiple style definitions will **cascade** into one

CSS demo

Heading 1

This is some text in a paragraph.

This is another paragraph.

Heading 2

Name	E-mail	Phone
Doe, John	jdoe@example.com	555-789-7222
Smith, Eva	esmith@example.com	555-324-3693

Heading 3

Visit our Home Page or our CSS Tutorial.

What you should already know:

I. HTML
II. XHTML

Favorite drinks:

- Smoothie
- Green tea
- Coffee

```
body
{
font-size:75%;
font-family:verdana,arial,'sans serif';
background-color:#FFFFF0;
color:#000080;
margin:10px;
}

h1 {font-size:200%;}
h2 {font-size:140%;}
h3 {font-size:110%;}

th {background-color:#ADD8E6;}

ul {list-style:circle;}
ol {list-style:upper-roman;}

a:link {color:#000080;}
a:hover {color:red;}
```

Styles solved a big problem

The original HTML was never intended to contain tags for formatting a document. HTML tags were intended to define the content of a document, like:

```
<p>This is a paragraph.</p>

<h1>This is a heading</h1>
```

When tags like <font> and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large web sites where fonts and color information had to be added to every single Web page, became a long, expensive and unduly painful process.

To solve this problem, the World Wide Web Consortium (W3C) - responsible for standardizing HTML - created CSS in addition to HTML 4.0.

With HTML 4.0, all formatting can be removed from the HTML document and stored in a separate CSS file.

All browsers support CSS today.

Styles save a lot of work

Styles sheets define HOW HTML elements are to be displayed.

Styles are normally saved in external .css files. External style sheets enable you to change the appearance and layout of all the pages in a Web site, just by editing one single CSS document!

Multiple styles will cascade into one

Style sheets allow style information to be specified in many ways.

Styles can be specified:

- inside an HTML element
- inside the head section of an HTML page
- in an external CSS file

Tip: Even multiple external style sheets can be referenced inside a single HTML document.

Cascading order - What style will be used when there is more than one style specified for an HTML element?

Generally speaking we can say that all the styles will "cascade" into a new "virtual" style sheet by the following rules, where number four has the highest priority:

1. Browser default
2. External style sheet
3. Internal style sheet (in the head section)
4. Inline style (inside an HTML element)

So, an inline style (inside an HTML element) has the highest priority, which means that it will override a style defined inside the <head> tag, or in an external style sheet, or in a browser (a default value).

💡If the link to the external style sheet is placed after the internal style sheet in HTML <head>, the external style sheet will override the internal style sheet!

2.2 CSS Syntax

Syntax

The CSS syntax is made up of three parts: a selector, a property and a value:

```
selector {property:value}
```

The selector is normally the HTML element/tag you wish to define, the property is the attribute you wish to change, and each property can take a value. The property and value are separated by a colon, and surrounded by curly braces:

```
body {color:black}
```

Note: If the value is multiple words, put quotes around the value:

```
p {font-family:"sans serif"}
```

Note: If you want to specify more than one property, you must separate each property with a semicolon. The example below shows how to define a center aligned paragraph, with a red text color:

```
p {text-align:center;color:red}
```

To make the style definitions more readable, you can describe one property on each line, like this:

```
p
{
text-align:center;
color:black;
font-family:arial
}
```

Grouping

You can group selectors. Separate each selector with a comma. In the example below we have grouped all the header elements. All header elements will be displayed in green text color:

```
h1,h2,h3,h4,h5,h6
{
color:green
}
```

The class Selector

With the class selector you can define different styles for the same type of HTML element.

Say that you would like to have two types of paragraphs in your document: one right-aligned paragraph, and one center-aligned paragraph. Here is how you can do it with styles:

```
p.right {text-align:right}
p.center {text-align:center}
```

You have to use the class attribute in your HTML document:

```
<p class="right">This paragraph will be right-aligned.</p>
<p class="center">This paragraph will be center-aligned.</p>
```

Note: To apply more than one class per given element, the syntax is:

```
<p class="center bold">This is a paragraph.</p>
```

The paragraph above will be styled by the class "center" AND the class "bold".

You can also omit the tag name in the selector to define a style that will be used by all HTML elements that have a certain class. In the example below, all HTML elements with class="center" will be center-aligned:

```
.center {text-align:center}
```

In the code below both the h1 element and the p element have class="center". This means that both elements will follow the rules in the ".center" selector:

```
<h1 class="center">This heading will be center-aligned</h1>
<p class="center">This paragraph will also be center-aligned.</p>
```

💡Do **NOT** start a class name with a number! It will not work in Mozilla/Firefox.

Add Styles to Elements with Particular Attributes

You can also apply styles to HTML elements with particular attributes.

The style rule below will match all input elements that have a type attribute with a value of "text":

```
input[type="text"] {background-color:blue}
```

The id Selector

You can also define styles for HTML elements with the id selector. The id selector is defined as a #.

The style rule below will match the element that has an id attribute with a value of "green":

```
#green {color:green}
```

The style rule below will match the p element that has an id with a value of "para1":

```
p#para1
{
text-align:center;
color:red
}
```

Do **NOT** start an ID name with a number! It will not work in Mozilla/Firefox.

CSS Comments

Comments are used to explain your code, and may help you when you edit the source code at a later date. A comment will be ignored by browsers. A CSS comment begins with "/*", and ends with "*/", like this:

```
/*This is a comment*/
p
{
text-align:center;
/*This is another comment*/
color:black;
font-family:arial
}
```

2.3 CSS How to Implement

Examples 1:

The HTML file below links to an external style sheet with the <link> tag:

```
<html>
<head>
<link rel="stylesheet"
type="text/css" href="ex1.css" />
</head>

<body>
<h1>This header is 36 pt</h1>
<h2>This header is blue</h2>
<p>This paragraph has a left
margin of 50 pixels</p>
</body>
</html>
```

This is the style sheet file (ex1.css):

```
body {background-color: yellow}
h1 {font-size: 36pt}
h2 {color: blue}
p {margin-left: 50px}
```

The result is in the frame below:

This header is 36 pt

This header is blue

This paragraph has a left margin of 50 pixels

N.B The background color is not shown but it must be yellow

Example 2:

The HTML file below links to an external style sheet with the <link> tag:

```
<html>
<head>
<link rel="stylesheet" type="text/css"
href="ex2.css" />
</head>
<body>
<h1>This is a header 1</h1>
<hr />
<p>You can see that the style
sheet formats the text</p>
<p><a href="http://www.google.com"
target="_blank">This is a link</a></p>
</body>
</html>
```

This is the style sheet file (ex2.css):

```
body {background-color: tan}
h1 {color:maroon; font-size:20pt}
hr {color:navy}
p {font-size:11pt; margin-left: 15px}
a:link     {color:green}
a:visited  {color:yellow}
a:hover    {color:black}
a:active   {color:blue}
```

The result is in the frame below:

This is a header 1

You can see that the style sheet formats the text

This is a link

N.B the background color is

How to Insert a Style Sheet

When a browser reads a style sheet, it will format the document according to it. There are three ways of inserting a style sheet:

External Style Sheet

An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing one file. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the head section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

The browser will read the style definitions from the file mystyle.css, and format the document according to it.

An external style sheet can be written in any text editor. The file should not contain any html tags. Your style sheet should be saved with a .css extension. An example of a style sheet file is shown below:

```
hr {color:sienna}
p {margin-left:20px}
body {background-image:url("images/back40.gif")}
```

Do not leave spaces between the property value and the units! "margin-left:20 px" (instead of "margin-left:20px") will only work in IE6, but it will not work in Firefox or Opera.

Internal Style Sheet

An internal style sheet should be used when a single document has a unique style. You define internal styles in the head section by using the <style> tag, like this:

```
<head>
<style type="text/css">
hr {color:sienna}
p {margin-left:20px}
body {background-image:url("images/back40.gif")}
</style>
</head>
```

The browser will now read the style definitions, and format the document according to it.

Note: A browser normally ignores unknown tags. This means that an old browser that does not support styles, will ignore the <style> tag, but the content of the <style> tag will be displayed on the page. It is possible to prevent an old browser from displaying the content by hiding it in the HTML comment element:

```
<head>
<style type="text/css">
<!--
hr {color:sienna}
p {margin-left:20px}
body {background-image:url("images/back40.gif")}
-->
</style>
</head>
```

Inline Styles

An inline style loses many of the advantages of style sheets by mixing content with presentation. Use this method sparingly, such as when a style is to be applied to a single occurrence of an element.

To use inline styles you use the style attribute in the relevant tag. The style attribute can contain any CSS property. The example shows how to change the color and the left margin of a paragraph:

```
<p style="color:sienna;margin-left:20px">This is a paragraph.</p>
```

Multiple Style Sheets

If some properties have been set for the same selector in different style sheets, the values will be inherited from the more specific style sheet.

For example, an external style sheet has these properties for the h3 selector:

```
h3
{
color:red;
text-align:left;
font-size:8pt
}
```

And an internal style sheet has these properties for the h3 selector:

```
h3
{
text-align:right;
font-size:20pt
}
```

If the page with the internal style sheet also links to the external style sheet the properties for h3 will be:

```
color:red;
text-align:right;
font-size:20pt
```

The color is inherited from the external style sheet and the text-alignment and the font-size is replaced by the internal style sheet.

2.4 CSS Background

The CSS background properties define the background effects of an element.

```
<html>

<head>
<style type="text/css">
body
{
background-color:yellow;
}
h1
{
background-color:#00ff00;
}
```

```
p
{
background-color:rgb(255,0,255);
}
</style>
</head>

<body>
<h1>This is heading 1</h1>
<p>This is a paragraph.</p>
</body>

</html>
```

Your Result:

This is heading 1

This is a paragraph.

All CSS Background Properties

The number in the "CSS" column indicates in which CSS version the property is defined (CSS1 or CSS2).

Property	Description	Values	CSS
background	A shorthand property for setting all background properties in one declaration	*background-color* *background-image* *background-repeat* *background-attachment* *background-position*	1
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page	scroll fixed	1
background-color	Sets the background color of an element	*color-rgb* *color-hex* *color-name* transparent	1
background-image	Sets an image as the background	url(*URL*) none	1
background-position	Sets the starting position of a background image	top left top center top right center left center center center right	1

		bottom left bottom center bottom right *x% y%* *xpos ypos*	

background-repeat	Sets if/how a background image will be repeated	repeat
		repeat-x
		repeat-y
		no-repeat
			1

2.5 CSS Text

The CSS text properties define the appearance of text:

Text Color

The color property is used to set the color of the text. The color can be set by:

- name - specify a color name, like "red"
- RGB - specify an RGB value, like "rgb(255,0,0)"
- Hex - specify a hex value, like "#ff0000"

The default color for a page is defined in the body selector.

Example

```
body {color:blue}
h1 {color:#00ff00}
h2 {color:rgb(255,0,0)}
```

Text Alignment

The text-align property is used to set the horizontal alignment of a text.

Text can be centered, or aligned to the left or right, or justified.

When text-align is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight (like in magazines and newspapers).

Example

```
h1 {text-align:center}
p.date {text-align:right}
p.main {text-align:justify}
```

Text Decoration

The text-decoration property is used to set or remove decorations from text.

The text-decoration property is mostly used to remove underlines from links for design purposes:

Example

```
a {text-decoration:none}
```

It can also be used to decorate text:

Example

```
h1 {text-decoration:overline}
h2 {text-decoration:line-through}
h3 {text-decoration:underline}
h4 {text-decoration:blink}
```

Text Transformation

The text-transform property is used to specify uppercase and lowercase letters in a text.

It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word.

Example

```
p.uppercase {text-transform:uppercase}
p.lowercase {text-transform:lowercase}
p.capitalize {text-transform:capitalize
```

Text Indentation

The text-indentation property is used to specify the indentation of the first line of a text.

Example

```
p {text-indent:50px}
```

All CSS Text Properties

The number in the "CSS" column indicates in which CSS version the property is defined (CSS1 or CSS2).

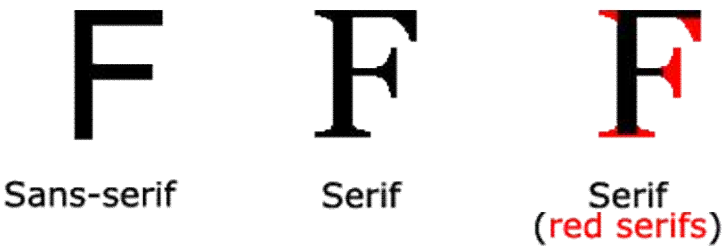
Property	Description	Values	CSS
color	Sets the color of a text	*color*	1
direction	Sets the text direction	ltr rtl	2
line-height	Sets the distance between lines	normal *number* *length* *%*	1
letter-spacing	Increase or decrease the space between characters	normal *length*	1
text-align	Aligns the text in an element	left right center justify	1
text-decoration	Adds decoration to text	none underline overline line-through blink	1
text-indent	Indents the first line of text in an element	*length* *%*	1
text-shadow		none *color* *length*	
text-transform	Controls the letters in an element	none capitalize uppercase lowercase	1
unicode-bidi		normal embed bidi-override	2

vertical-align	Sets the vertical alignment of an element	baseline sub super top text-top middle bottom text-bottom *length* *%*	1
white-space	Sets how white space inside an element is handled	normal pre nowrap	1
word-spacing	Increase or decrease the space between words	normal *length*	1

2.6 CSS Font

CSS font properties define the font family, boldness, size, and the style of a text.

Difference Between Serif and Sans-serif Fonts



On computer screens, sans-serif fonts are considered easier to read than serif fonts.

CSS Font Families

In CSS, there is two types of font family names:

- **generic family** - a group of font families with a similar look (like "Serif" or "Monospace")
- **font family** - a specific font family (like "Times New Roman" or "Arial")

Generic family	Font family	Description
Serif	Times New Roman Georgia	Serif fonts have small lines at the ends on some characters

Sans-serif	Arial Verdana	"Sans" means without - these fonts do not have the lines at the ends of characters
Monospace	Courier New Lucida Console	All monospace characters has the same width

Font Family

The font family of a text is set with the font-family property.

The font-family property can hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font.

Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

Note: If the name of a font family is more than one word, it must be in quotation marks, like font-family: "Times New Roman".

More than one font family is specified in a comma-separated list:

Example

```
p{font-family:"Times New Roman",Georgia,Serif}
```

Font Style

The font-style property is mostly used to specify italic text.

This property has three values:

- normal - The text is shown normally
- italic - The text is shown in italics
- oblique - The text is "leaning" (oblique is very similar to italic, but less supported)

Example

```
p.normal {font-style:normal}
p.italic {font-style:italic}
p.oblique {font-style:oblique}
```

Font Size

The font-size property sets the size of the text.

Being able to manage the text size is important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs.

Always use the proper HTML tags, like <h1> - <h6> for headings and <p> for paragraphs.

The font-size value can be an absolute, or relative size.

Absolute size:

- Sets the text to a specified size
- Does not allow a user to change the text size in all browsers (bad for accessibility reasons)
- Absolute size is useful when the physical size of the output is known

Relative size:

- Sets the size relative to surrounding elements
- Allows a user to change the text size in browsers

If you do not specify a font size, the default size for normal text, like paragraphs, is 16px (16px=1em).

Setting Text Size Using Pixels

Setting the text size with pixels, gives you full control over the text size:

Example

```
h1 {font-size:40px}
h2 {font-size:30px}
p {font-size:14px}
```

Setting Text Size Using Em

To avoid the resizing problem with Internet Explorer, many developers use em instead of pixels.

The em size unit is recommended by the W3C.

1em is equal to the current font size. The default text size in browsers is 16px. So, the default size of 1em is 16px.

The size can be calculated from pixels to em using this formula: *pixels*/16=*em*

Example

```
h1 {font-size:2.5em} /* 40px/16=2.5em */
h2 {font-size:1.875em} /* 30px/16=1.875em */
p {font-size:0.875em} /* 14px/16=0.875em */
```

In the example above, the text size in em is the same as the previous example in pixels. However, with the em size, it is possible to adjust the text size in all browsers.

Unfortunately, there is still a problem with IE. When resizing the text, it becomes larger than it should when made larger, and smaller than it should when made smaller.

Using a Combination of Percent and Em

The solution that works in all browsers, is to set a default font-size in percent for the body element:

Example

```
body {font-size:100%}
h1 {font-size:2.5em}
h2 {font-size:1.875em}
p {font-size:0.875em}
```

All CSS Font Properties

The number in the "CSS" column indicates in which CSS version the property is defined (CSS1 or CSS2).

Property	Description	Values	CSS
font	Sets all the font properties in one declaration	*font-style* *font-variant* *font-weight* *font-size/line-height* *font-family* caption icon menu message-box small-caption status-bar	1

		inherit	
font-family	Specifies the font family for text	*family-name* *generic-family* inherit	1
font-size	Specifies the font size of text	xx-small x-small small medium large x-large xx-large smaller larger *length* *%* inherit	1
font-style	Specifies the font style for text	normal italic oblique inherit	1
font-variant	Specifies whether or not a text should be displayed in a small-caps font	normal small-caps inherit	1
font-weight	Specifies the weight of a font	normal bold bolder lighter 100 200 300 400 500 600 700 800 900 inherit	1

2.7 CSS Box Model

Box Model in CSS

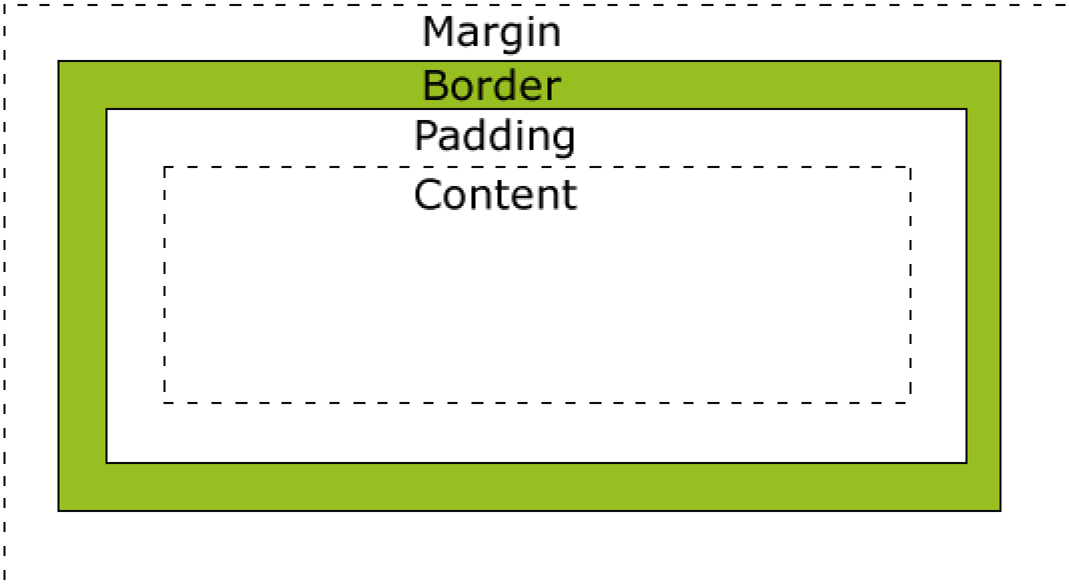
All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

The box model illustrates how the CSS properties: margin, border, and padding, affects the width and height of an element.

The Box Model

The image below illustrates the box model:



Explanation of the different parts:

- **Margin** - Clears an area around the border. The margin does not have a background color, and it is completely transparent
- **Border** - A border that lies around the padding and content. The border is affected by the background color of the box
- **Padding** - Clears an area around the content. The padding is affected by the background color of the box
- **Content** - The content of the box, where text and images appear

2.8 CSS Border

The CSS border properties define the borders around an element:

Border Style

The border-style property specifies what kind of border to display.

None of the other border properties will have any effect unless border-style is set.

border-style Values

none: Defines no border

dotted: Defines a dotted border

dashed: Defines a dashed border

solid: Defines a solid border

double: Defines two borders. The width of the two borders are the same as the border-width value

groove: Defines a 3D grooved border. The effect depends on the border-color value

ridge: Defines a 3D ridged border. The effect depends on the border-color value

inset: Defines a 3D inset border. The effect depends on the border-color value

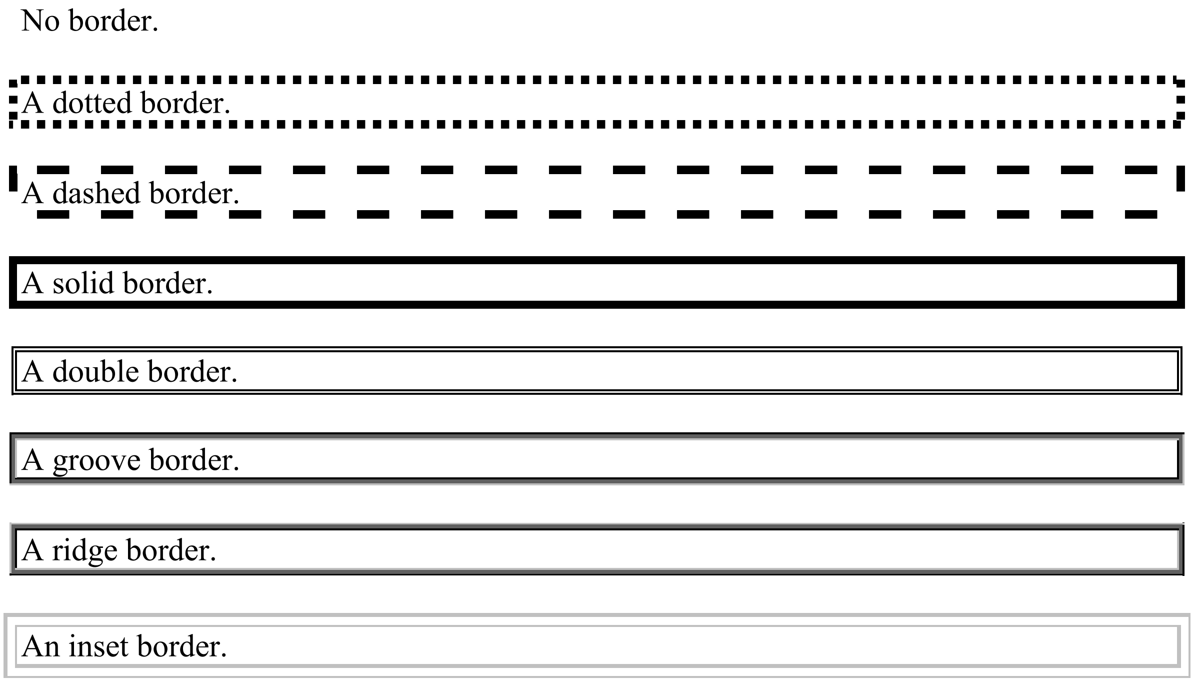
outset: Defines a 3D outset border. The effect depends on the border-color value

```
<html>
<head>
<style type="text/css">
p.none {border-style:none}
p.dotted {border-style:dotted}
p.dashed {border-style:dashed}
```

```
p.solid {border-style:solid}
p.double {border-style:double}
p.groove {border-style:groove}
p.ridge {border-style:ridge}
p.inset {border-style:inset}
p.outset {border-style:outset}
p.hidden {border-style:hidden}
</style>
</head>

<body>
<p class="none">No border.</p>
<p class="dotted">A dotted border.</p>
<p class="dashed">A dashed border.</p>
<p class="solid">A solid border.</p>
<p class="double">A double border.</p>
<p class="groove">A groove border.</p>
<p class="ridge">A ridge border.</p>
<p class="inset">An inset border.</p>
<p class="outset">An outset border.</p>
<p class="hidden">A hidden border.</p>
</body>

</html>
```



No border.

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border.

A ridge border.

An inset border.

An outset border.

A hidden border.

Border Width

The border-width property is used to set the width of the border.

The width is set in pixels, or by using one of the three pre-defined values: thin, medium, or thick.

Note: The "border-width" property does not work if it is used alone. Use the "border-style" property to set the borders first.

Example

```
p.one
{
border-style:solid;
border-width:5px;
}
p.two
{
border-style:solid;
border-width:medium;
}
```

Border Color

The border-color property is used to set the color of the border. The color can be set by:

- name - specify a color name, like "red"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- Hex - specify a hex value, like "#ff0000"

You can also set the border color to "transparent".

Note: The "border-color" property does not work if it is used alone. Use the "border-style" property to set the borders first.

Example

```
p.one
```

```
{
border-style:solid;
border-color:red;
}
p.two
{
border-style:solid;
border-color:#98bf21;
}
```

Border - Individual sides

In CSS it is possible to specify different borders for different sides:

Example

```
p
{
border-top-style:dotted;
border-right-style:solid;
border-bottom-style:dotted;
border-left-style:solid;
}
```

The example above can also be set with a single property:

Example

```
border-style:dotted solid;
```

The border-style property can have from one to four values.

- **border-style:dotted solid double dashed;**
  - top border is dotted
  - right border is solid
  - bottom border is double
  - left border is dashed

- **border-style:dotted solid double;**

    - top border is dotted
    - right and left borders are solid
    - bottom border is double

- **border-style:dotted solid;**
    - top and bottom borders are dotted
    - right and left borders are solid

- **border-style:dotted;**
    - all four borders are dotted

The border-style property is used in the example above. However, it also works with border-width and border-color.

Border - Shorthand property

As you can see from the examples above, there are many properties to consider when dealing with borders.

To shorten the code, it is also possible to specify all the border properties in one property. This is called a shorthand property.

The shorthand property for the border properties is "border":

Example

```
border:5px solid red;
```

All CSS Border Properties

The number in the "CSS" column indicates in which CSS version the property is defined (CSS1 or CSS2).

Property	Description	Values	CSS
border	Sets all the border properties in one declaration	*border-width* *border-style* *border-color*	1
border-bottom	Sets all the bottom border properties in one declaration	*border-bottom-width* *border-bottom-style* *border-bottom-*	1

		color	
border-bottom-color	Sets the color of the bottom border	*border-color*	2
border-bottom-style	Sets the style of the bottom border	*border-style*	2
border-bottom-width	Sets the width of the bottom border	*border-width*	1
border-color	Sets the color of the four borders	*color_name* *hex_number* *rgb_number* transparent inherit	1
border-left	Sets all the left border properties in one declaration	*border-left-width* *border-left-style* *border-left-color*	1
border-left-color	Sets the color of the left border	*border-color*	2
border-left-style	Sets the style of the left border	*border-style*	2
border-left-width	Sets the width of the left border	*border-width*	1
border-right	Sets all the right border properties in one declaration	*border-right-width* *border-right-style* *border-right-color*	1
border-right-color	Sets the color of the right border	*border-color*	2
border-right-style	Sets the style of the right border	*border-style*	2
border-right-width	Sets the width of the right border	*border-width*	1
border-style	Sets the style of the four borders	none hidden dotted dashed solid double groove ridge inset outset inherit	1
border-top	Sets all the top border properties in one declaration	*border-top-width* *border-top-style* *border-top-color*	1
border-top-color	Sets the color of the top border	*border-color*	2
border-top-style	Sets the style of the top border	*border-style*	2
border-top-width	Sets the width of the top border	*border-width*	1
border-width	Sets the width of the four borders	thin medium thick *length* inherit	1

2.9 CSS Outlines

An outline is a line that is drawn around elements, outside the border edge, to make the element "stand out".

CSS Outline Properties

The outline properties specifies the style, color, and width of an outline.

The number in the "CSS" column indicates in which CSS version the property is defined (CSS1 or CSS2).

Property	Description	Values	CSS
outline	A shorthand property for setting all the outline properties	*outline-color* *outline-style* *outline-width*	2
outline-color	Sets the color of the outline around an element	*color* invert	2
outline-style	Sets the style of the outline around an element	none dotted dashed solid double groove ridge inset outset	2
outline-width	Sets the width of the outline around an element	thin medium thick *length*	2

2.10 CSS Margin

Margin

The margin clears an area around an element (outside the border). The margin does not have a background color, and is completely transparent.

The top, right, bottom, and left margin can be changed independently using separate properties. A shorthand margin property can also be used, to change all margins at once.

Possible Values

Value	Description
auto	The browser sets the margin. The result of this is dependant of the browser
length	Defines a fixed margin (in pixels, pt, em, etc.)
%	Defines a margin in % of the containing element

💡It is possible to use negative values, to overlap content.

Margin - Individual sides

In CSS, it is possible to specify different margins for different sides:

Example

```
margin-top:100px;
margin-bottom:100px;
margin-right:50px;
margin-left:50px;
```

Margin - Shorthand property

To shorten the code, it is possible to specify all the margin properties in one property. This is called a shorthand property.

The shorthand property for all the margin properties is "margin":

Example

```
margin:100px 50px;
```

The margin property can have from one to four values.

- **margin:25px 50px 75px 100px;**
    - top margin is 25px
    - right margin is 50px
    - bottom margin is 75px
    - left margin is 100px
    -

- **margin:25px 50px 75px;**
    - top margin is 25px
    - right and left margins are 50px
    - bottom margin is 75px
    -
- **margin:25px 50px;**
    - top and bottom margins are 25px
    - right and left margins are 50px
    -
- **margin:25px;**
    - all four margins are 25px

2.11 CSS Padding

The CSS padding properties define the space between the element border and the element content.

Padding

The padding clears an area around the content (inside the border) of an element. The padding is affected by the background color of the element.

The top, right, bottom, and left padding can be changed independently using separate properties. A shorthand padding property can also be used, to change all paddings at once.

Possible Values

Value	Description
length	Defines a fixed padding (in pixels, pt, em, etc.)
%	Defines a padding in % of the containing element

Padding - Shorthand property

To shorten the code, it is possible to specify all the padding properties in one property. This is called a shorthand property.

The shorthand property for all the padding properties is "padding":

Example

```
padding:25px 50px;
```

The padding property can have from one to four values.

- **padding:25px 50px 75px 100px;**
    - top padding is 25px
    - right padding is 50px
    - bottom padding is 75px
    - left padding is 100px
    -
- **padding:25px 50px 75px;**
    - top padding is 25px
    - right and left paddings are 50px
    - bottom padding is 75px
    -
- **padding:25px 50px;**
    - top and bottom paddings are 25px
    - right and left paddings are 50px
    -
- **padding:25px;**
    - all four paddings are 25px

2.12 CSS List

The CSS list properties allow you to place the list item marker, change between different list item markers, or set an image as the list item marker.

List

In HTML, there are two types of lists:

- unordered list - the list items are marked with bullets (typically circles or squares)
- ordered list - the list items are marked with numbers or letters

With CSS, lists can be styled further, and images can be used as list item markers.

Different List Item Markers

It is possible to specify the type of list item marker with the list-style-type property:

Example

```
ul.circle {list-style-type:circle}
ul.square {list-style-type:square}

ol.upper-roman {list-style-type:upper-roman}
ol.lower-alpha {list-style-type:lower-alpha}
```

Some of the values are for unordered lists, and some for ordered lists.

Unordered List - Possible Values

Value	Description
none	No marker
disc	Default. The marker is a filled circle
circle	The marker is a circle
square	The marker is a square

Ordered List - Possible Values

Value	Description
none	No marker
circle	The marker is a circle
disc	The marker is a filled circle. This is default
square	The marker is a square
armenian	The marker is traditional Armenian numbering
decimal	The marker is a number
decimal-leading-zero	The marker is a number padded by initial zeros (01, 02, 03, etc.)
georgian	The marker is traditional Georgian numbering (an, ban, gan, etc.)
lower-alpha	The marker is lower-alpha (a, b, c, d, e, etc.)
lower-greek	The marker is lower-greek (alpha, beta, gamma, etc.)
lower-latin	The marker is lower-latin (a, b, c, d, e, etc.)
lower-roman	The marker is lower-roman (i, ii, iii, iv, v, etc.)
upper-alpha	The marker is upper-alpha (A, B, C, D, E, etc.)
upper-latin	The marker is upper-latin (A, B, C, D, E, etc.)
upper-roman	The marker is upper-roman (I, II, III, IV, V, etc.)

Internet Explorer does not support all property values for ordered lists.

Positioning the List

The list-style-position property specifies the indentation of a list.

"outside" is the default value. The "inside" value further indents the list:

Example

```
ul.inside {list-style-position:inside}
ul.outside {list-style-position:outside}
```

Using an Image as List Item Marker

It is also possible to use an image as a list item marker:

Example

```
ul
{
list-style-image:url('arrow.gif');
}
```

The example above will not show the exact same result in all browsers. IE and Opera will display the images slightly higher than in Firefox, Chrome, and Safari.

The example above will be fine for most occasions. However, there is a way to position the image more precisely.

For the same result in all browsers, you will have to use a background image on each list item, like this:

Example

```
ul
{
list-style-type:none;
padding:0px;
margin:0px;
}
li
{
background-image:url(arrow.gif);
background-repeat:no-repeat;
background-position:0px 5px;
padding-left:14px;
}
```

Example explained:

- For ul:
  - Set the list-style-type to none to remove the list item marker
  - Both padding and margin must be set to 0px for cross-browser compatibility

- For li:
  - Set the URL of the image, and show it only once (no-repeat)
  - Use the background-position property to place the image where you want it (left 0px and down 5px)
  - Use the padding-left property to position the text in the list

List - Shorthand property

It is possible to specify all the list properties in a single property. This is called a shorthand property.

The shorthand property for list is "list-style":

Example

```
list-style:square inside;
```

When using the shorthand property, the order of the values are:

- list-style-type
- list-style-position
- list-style-image

It does not matter if one of the values above are missing, as long as the rest are in the specified order.

All CSS List Properties

The number in the "CSS" column indicates in which CSS version the property is defined (CSS1 or CSS2).

Property	Description	Values	CSS
list-style	Sets all the properties for a list in one declaration	*list-style-type* *list-style-position* *list-style-image* inherit	1
list-style-image	Specifies an image as the list-item marker	URL none inherit	1
list-style-position	Specifies where to place the list-item marker	inside outside inherit	1
list-style-type	Specifies the type of list-item marker	none disc	1

		circle square decimal decimal-leading-zero armenian georgian lower-alpha upper-alpha lower-greek lower-latin upper-latin lower-roman upper-roman inherit	

2.13 CSS Table

The CSS table properties allow you to set the layout of a table.

Example:

```
html>
<head>
<style type="text/css">
table.ex1 {table-layout:auto}
table.ex2 {table-layout:fixed}
</style>
</head>

<body>
<table class="ex1" border="1" width="100%">
<tr>
<td width="5%">1000000000000000000000000000</td>
<td width="95%">10000000</td>
</tr>
</table>
<br />

<table class="ex2" border="1" width="100%">
<tr>
<td width="5%">1000000000000000000000000000</td>
<td width="95%">10000000</td>
```

```
</tr>
</table>

</body>
</html>
```

1000000000000000000000000000	10000000

1000000000000000000000000000	10000000

CSS Table Properties

The CSS table properties allow you to set the layout of a table.

Browser support: IE: Internet Explorer, M: Mac IE only, F: Firefox, N: Netscape.

W3C: The number in the "W3C" column indicates in which CSS recommendation the property is defined (CSS1 or CSS2).

Property	Description	Values	IE	F	N	W3C
border-collapse	Sets whether the table borders are collapsed into a single border or detached as in standard HTML	collapse separate	5	1	7	2
border-spacing	Sets the distance that separates cell borders (only for the "separated borders" model)	*length length*	5M	1	6	2
caption-side	Sets the position of the table caption	top bottom left right	5M	1	6	2
empty-cells	Sets whether or not to show empty cells in a table (only for the "separated borders" model)	show hide	5M	1	6	2
table-layout	Sets the algorithm used to display the table cells, rows, and columns	auto fixed	5	1	6	2

Chapter 3: JavaScript

3.1 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 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 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

3.2 How to Start

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

Put a JavaScript into an HTML page

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

Example

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

The example below shows how to add HTML tags to the JavaScript:

Example

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

Example Explained

To insert a JavaScript into an HTML page, we use the <script> tag. Inside the <script> tag we use the type attribute to define the scripting language.

So, the <script type="text/javascript"> and </script> tells where the JavaScript starts and ends:

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

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

By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write Hello World! to the page:

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

Note: If we had not entered the <script> tag, the browser would have treated the document.write("Hello World!") command as pure text, and just write the entire line on the page.

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) after the last JavaScript statement, like this:

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

```
//-->
</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.

3.3 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.

JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.

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.

Example

```
<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>

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.

Example

```
<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.

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

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:

Example

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

3.4 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 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:

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>
```

3.5 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>
```

3.6 JavaScript Variables

Variables are "containers" for storing information.

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 or the underscore character

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

Declaring (Creating) JavaScript Variables

Creating variables in JavaScript is most often referred to as "declaring" variables.

You can declare JavaScript variables with the **var statement**:

```
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, use quotes around the value.

Assigning Values to Undeclared JavaScript Variables

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

These statements:

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

have the same effect as:

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

Redeclaring JavaScript Variables

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

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

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

3.7 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
-	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

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);
```

3.8 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
===	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");
```

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

```
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".

3.9 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>
```

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 condition1 and condition2 are not 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>
```

3.10 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();
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>
```

3.11 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)
  {
  document.write("You pressed OK!");
  }
else
  {
  document.write("You pressed Cancel!");
  }
}
</script>
</head>
<body>

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

</body>
</html>
```

Prompt Box

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

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

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

Syntax

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

Example

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

```
  document.write("Hello " + name + "! How are you today?");
  }
}
</script>
</head>
<body>

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

</body>
</html>
```

3.12 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>
```

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>
```

3.13 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 (var=startvalue;var<=endvalue;var=var+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>
```

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

Note: The <= could be any comparing statement.

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 (var<=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>
```

3.15 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>
```

3.16 For...In Statement

JavaScript For...In Statement

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

Syntax

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

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

Note: The variable argument can be a named variable, an array element, or a property of an object.

Example

Use the for...in statement to loop through an array:

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>
```

3.17 JavaScript Events

Events are actions that can be detected by JavaScript.

Events

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.

Every element on a web page has certain events which can trigger a JavaScript. For example, we can use the onClick event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

- A mouse click
- A web page or an image loading

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

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

onLoad and onUnload

The onLoad and onUnload events are triggered when the user enters or leaves the page.

The onLoad event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

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

onFocus, onBlur and onChange

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

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

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

onSubmit

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

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

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

onMouseOver and onMouseOut

onMouseOver and onMouseOut are often used to create "animated" buttons.

Below is an example of an onMouseOver event. An alert box appears when an onMouseOver event is detected:

```
<a href="http://www.w3schools.com" onmouseover="alert('An onMouseOver
event');return false"><img src="w3s.gif" alt="W3Schools" /></a
```

3.18 JavaScript Objects Introduction

JavaScript is an Object Oriented Programming (OOP) language.

An OOP language allows you to define your own objects and make your own variable types.

Object Oriented Programming

JavaScript is an Object Oriented Programming (OOP) language. An OOP language 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!
```

3.19 JavaScript Date Object

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

Create a Date Object

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

The following code create a Date object called myDate:

```
var myDate=new Date()
```

Note: The Date object will automatically hold the current date and time as its initial value!

Set Dates

We can easily manipulate the date by using the methods available for the Date object.

In the example below we set a Date object to a specific date (14th January 2010):

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

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

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

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

Compare Two Dates

The Date object is also used to compare two dates.

The following example compares today's date with the 14th January 2010:

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

if (myDate>today)
  {
  alert("Today is before 14th January 2010");
  }
else
  {
  alert("Today is after 14th January 2010");
  }
```

3.20 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:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="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

The following code creates an Array object called myCars:

```
var myCars=new Array();
```

There are two ways of adding values to an array (you can add as many values as you need to define as many variables you require).

1:

```
var myCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

You could also pass an integer argument to control the array's size:

```
var myCars=new Array(3);
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

2:

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

Note: If you specify numbers or true/false values inside the array then the type of variables will be numeric 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
```

3.21 JavaScript Form Validation

JavaScript Form Validation

JavaScript can be used to validate data in HTML forms before sending off the content to a server.

Form data that typically are checked by a JavaScript could be:

- has the user left required fields empty?
- has the user entered a valid e-mail address?
- has the user entered a valid date?

- has the user entered text in a numeric field?

Required Fields

The function below checks if a required field has been left empty. If the required field is blank, an alert box alerts a message and the function returns false. If a value is entered, the function returns true (means that data is OK):

```
function validate_required(field,alerttxt)
{
with (field)
  {
  if (value==null||value=="")
    {
    alert(alerttxt);return false;
    }
  else
    {
    return true;
    }
  }
}
```

The entire script, with the HTML form could look something like this:

```
<html>
<head>
<script type="text/javascript">
function validate_required(field,alerttxt)
{
with (field)
  {
  if (value==null||value=="")
    {
    alert(alerttxt);return false;
    }
  else
    {
    return true;
    }
  }
}

function validate_form(thisform)
```

```
{
with (thisform)
  {
  if (validate_required(email,"Email must be filled out!")==false)
  {email.focus();return false;}
  }
}
</script>
</head>

<body>
<form action="submit.htm" onsubmit="return validate_form(this)" method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>

</html>
```

E-mail Validation

The function below checks if the content has the general syntax of an email.

This means that the input data must contain at least an @ sign and a dot (.). Also, the @ must not be the first character of the email address, and the last dot must at least be one character after the @ sign:

```
function validate_email(field,alerttxt)
{
with (field)
  {
  apos=value.indexOf("@");
  dotpos=value.lastIndexOf(".");
  if (apos<1||dotpos-apos<2)
    {alert(alerttxt);return false;}
  else {return true;}
  }
}
```

The entire script, with the HTML form could look something like this:

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

```
function validate_email(field,alerttxt)
{
with (field)
  {
  apos=value.indexOf("@");
  dotpos=value.lastIndexOf(".");
  if (apos<1||dotpos-apos<2)
    {alert(alerttxt);return false;}
  else {return true;}
  }
}

function validate_form(thisform)
{
with (thisform)
  {
  if (validate_email(email,"Not a valid e-mail address!")==false)
    {email.focus();return false;}
  }
}
</script>
</head>

<body>
<form action="submit.htm" onsubmit="return validate_form(this);" method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>

</html>
```

3.22 JavaScript Animation

With JavaScript we can create animated images.

JavaScript Animation

It is possible to use JavaScript to create animated images.

The trick is to let a JavaScript change between different images on different events.

In the following example we will add an image that should act as a link button on a web page. We will then add an onMouseOver event and an onMouseOut event that will run two JavaScript functions that will change between the images.

The HTML Code

The HTML code looks like this:

```
<a href="http://www.w3schools.com" target="_blank">
<img border="0" alt="Visit W3Schools!" src="b_pink.gif" id="b1"
onmouseOver="mouseOver()" onmouseOut="mouseOut()" /></a>
```

Note that we have given the image an id, to make it possible for a JavaScript to address it later.

The onMouseOver event tells the browser that once a mouse is rolled over the image, the browser should execute a function that will replace the image with another image.The onMouseOut event tells the browser that once a mouse is rolled away from the image, another JavaScript function should be executed. This function will insert the original image again.

The JavaScript Code

The changing between the images is done with the following JavaScript:

```
<script type="text/javascript">
function mouseOver()
{
document.getElementById("b1").src ="b_blue.gif";
}
function mouseOut()
{
document.getElementById("b1").src ="b_pink.gif";
}
</script>
```

The function mouseOver() causes the image to shift to "b_blue.gif".

The function mouseOut() causes the image to shift to "b_pink.gif".

The Entire Code

Example

```
<html>
<head>
<script type="text/javascript">
function mouseOver()
{
document.getElementById("b1").src ="b_blue.gif";
}
function mouseOut()
{
document.getElementById("b1").src ="b_pink.gif";
}
</script>
</head>

<body>
<a href="http://www.w3schools.com" target="_blank">
<img border="0" alt="Visit W3Schools!" src="b_pink.gif" id="b1"
onmouseover="mouseOver()" onmouseout="mouseOut()" /></a>
</body>
</html>
```

Chapter 4: Introduction to PHP, MySQL and Apache

4.1 PHP Introduction

What is PHP?

- PHP stands for **P**HP: **H**ypertext **P**reprocessor
- PHP is a server-side scripting language, like ASP
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software
- PHP is free to download and use

What is a PHP File?

- PHP files can contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3", or ".phtml"

What is MySQL?

- MySQL is a database server
- MySQL is ideal for both small and large applications
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

PHP + MySQL

- PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

Where to Start?

To get access to a web server with PHP support, you can:

- Install Apache (or IIS) on your own server, install PHP, and MySQL
- Or find a web hosting plan with PHP and MySQL support

4.2 PHP Syntax

PHP code is executed on the server, and the plain HTML result is sent to the browser.

Basic PHP Syntax

A PHP scripting block always starts with **<?php** and ends with **?>**. A PHP scripting block can be placed anywhere in the document.

On servers with shorthand support enabled you can start a scripting block with <? and end with ?>.

For maximum compatibility, we recommend that you use the standard form (<?php) rather than the shorthand form.

```
<?php
?>
```

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.

Below, we have an example of a simple PHP script which sends the text "Hello World" to the browser:

```
<html>
<body>

<?php
echo "Hello World";
?>

</body>
</html>
```

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

There are two basic statements to output text with PHP: **echo** and **print**. In the example above we have used the echo statement to output the text "Hello World".

Note: The file must have a .php extension. If the file has a .html extension, the PHP code will not be executed.

Comments in PHP

In PHP, we use // to make a single-line comment or /* and */ to make a large comment block.

```
<html>
<body>

<?php
//This is a comment

/*
This is
a comment
block
*/
?>

</body>
</html>
```

4.3 PHP Variables

A variable is used to store information.

Variables in PHP

Variables are used for storing a values, like text strings, numbers or arrays.

When a variable is declared, it can be used over and over again in your script.

All variables in PHP start with a $ sign symbol.

The correct way of declaring a variable in PHP:

```
$var_name = value;
```

New PHP programmers often forget the $ sign at the beginning of the variable. In that case it will not work.

Let's try creating a variable containing a string, and a variable containing a number:

```
<?php
$txt="Hello World!";
$x=16;
?>
```

PHP is a Loosely Typed Language

In PHP, a variable does not need to be declared before adding a value to it.

In the example above, you see that you do not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on its value.

In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.

In PHP, the variable is declared automatically when you use it.

Naming Rules for Variables

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _)
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore ($my_string), or with capitalization ($myString)

4.4 PHP String Variables

A string variable is used to store and manipulate text.

String Variables in PHP

String variables are used for values that contains characters.

In this chapter we are going to look at the most common functions and operators used to manipulate strings in PHP.

After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

Below, the PHP script assigns the text "Hello World" to a string variable called $txt:

```
<?php
$txt="Hello World";
echo $txt;
?>
```

The output of the code above will be:

```
Hello World
```

Now, lets try to use some different functions and operators to manipulate the string.

The Concatenation Operator

There is only one string operator in PHP.

The concatenation operator (.) is used to put two string values together.

To concatenate two string variables together, use the concatenation operator:

```
<?php
```

```
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

```
Hello World! What a nice day!
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string (a space character), to separate the two strings.

The strlen() function

The strlen() function is used to return the length of a string.

Let's find the length of a string:

```
<?php
echo strlen("Hello world!");
?>
```

The output of the code above will be:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string).

The strpos() function

The strpos() function is used to search for character within a string.

If a match is found, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

```
<?php
echo strpos("Hello world!","world");
?>
```

The output of the code above will be:

```
6
```

The position of the string "world" in our string is position 6. The reason that it is 6 (and not 7), is that the first position in the string is 0, and not 1.

4.5 PHP Operators

Operators are used to operate on values.

PHP Operators

This section lists the different operators used in PHP.

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Logical Operators

Operator	Description	Example

&&	and	x=6
		y=3
		
		(x < 10 && y > 1) returns true

||	or	x=6
		y=3
		
		(x==5 || y==5) returns false

!	not	x=6
		y=3
		
		!(x==y) returns true

4.6 PHP If...Else Statements

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 PHP 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 a condition is true and another code if the condition is false
- **if...elseif....else statement** - use this statement to select one of several blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

The 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;
```

The following example will output "Have a nice weekend!" if the current day is Friday:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri") echo "Have a nice weekend!";
?>

</body>
</html>
```

The if...else Statement

Use the if....else statement to execute some code if a condition is true and another code if a condition is false.

Syntax

```
if (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>

</body>
</html>
```

The if...elseif....else Statement

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

Syntax

```
if (condition)
  code to be executed if condition is true;
elseif (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

4.7 PHP Switch Statement

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

The PHP Switch Statement

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

Syntax

```
switch (n)
{
case label1:
  code to be executed if n=label1;
  break;
case label2:
  code to be executed if n=label2;
  break;
default:
  code to be executed if n is different from both label1 and label2;
}
```

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. The default statement is used if no match is found.

Example

```
<html>
<body>

<?php
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>

</body>
</html>
```

4.8 PHP Arrays

An array stores multiple values in a single variable.

What is an Array?

You have already learnt that a variable is a storage area holding numbers and text. The problem is, a variable will hold only one value.

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:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="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 index so that it can be easily accessed.

In PHP, there are three kind of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

Numeric Arrays

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

2. In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

Example

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
```

```
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output:

```
Saab and Volvo are Swedish cars.
```

Associative Arrays

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

Example 1

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Example 2

This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
  (
  "Griffin"=>array
  (
  "Peter",
  "Lois",
  "Megan"
  ),
  "Quagmire"=>array
  (
  "Glenn"
  ),
  "Brown"=>array
  (
  "Cleveland",
  "Loretta",
  "Junior"
  )
  );
```

The array above would look like this if written to the output:

```
Array
(
[Griffin] => Array
  (
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] => Array
```

```
  (
  [0] => Glenn
  )
[Brown] => Array
  (
  [0] => Cleveland
  [1] => Loretta
  [2] => Junior
  )
)
```

Example 2

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

4.9 PHP Looping - While Loops

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

PHP 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 PHP, we have the following looping statements:

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

The while Loop

The while loop executes a block of code while a condition is true.

Syntax

```
while (condition)
  {
  code to be executed;
  }
```

Example

The example below defines a loop that starts with i=1. 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>

<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br />";
  $i++;
  }
?>

</body>
</html>
```

Output:

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

The do...while Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

Syntax

```
do
```

```
  {
  code to be executed;
  }
while (condition);
```

Example

The example below defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:

```
<html>
<body>

<?php
$i=1;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<=5);
?>

</body>
</html>
```

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

The for loop and the foreach loop will be explained in the next chapter

4.10 PHP Looping - For Loops

Loops execute a block of code a specified number of times, or 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 (init; condition; increment)
  {
  code to be executed;
  }
```

Parameters:

- *init*: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- *condition*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment*: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

Note: Each of the parameters above can be empty, or have multiple expressions (separated by commas).

Example

The example below defines a loop that starts with i=1. 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>

<?php
for ($i=1; $i<=5; $i++)
  {
  echo "The number is " . $i . "<br />";
  }
?>

</body>
</html>
```

Output:

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

The foreach Loop

The foreach loop is used to loop through arrays.

Syntax

```
foreach ($array as $value)
  {
  code to be executed;
  }
```

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

Example

The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>

<?php
$x=array("one","two","three");
foreach ($x as $value)
  {
  echo $value . "<br />";
  }
?>

</body>
</html>
```

Output:

```
one
two
three
```

4.11 PHP Functions

The real power of PHP comes from its functions.

In PHP, there are more than 700 built-in functions.

PHP Functions

In this chapter we will show you how to create your own functions.

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

A function will be executed by a call to the function.

You may call a function from anywhere within a page.

Create a PHP Function

A function will be executed by a call to the function.

Syntax

```
function functionName()
{
code to be executed;
}
```

PHP function guidelines:

- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

Example

A simple function that writes my name when it is called:

```
<html>
<body>

<?php
function writeName()
{
echo "Kai Jim Refsnes";
}
```

```
echo "My name is ";
writeName();
?>

</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes
```

PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable.

Parameters are specified after the function name, inside the parentheses.

Example 1

The following example will write different first names, but equal last name:

```
<html>
<body>

<?php
function writeName($fname)
{
echo $fname . " Refsnes.<br />";
}

echo "My name is ";
writeName("Kai Jim");
echo "My sister's name is ";
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>

</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes.
My brother's name is Stale Refsnes.
```

Example 2

The following function has two parameters:

```
<html>
<body>

<?php
function writeName($fname,$punctuation)
{
echo $fname . " Refsnes" . $punctuation . "<br />";
}

echo "My name is ";
writeName("Kai Jim",".");
echo "My sister's name is ";
writeName("Hege","!");
echo "My brother's name is ";
writeName("Ståle","?");
?>

</body>
</html>
```

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes!
My brother's name is Ståle Refsnes?
```

PHP Functions - Return values

To let a function return a value, use the return statement.

Example

```
<html>
<body>

<?php
```

```
function add($x,$y)
{
$total=$x+$y;
return $total;
}

echo "1 + 16 = " . add(1,16);
?>

</body>
</html>
```

Output:

```
1 + 16 = 17
```

4.12 PHP Forms and User Input

The PHP $_GET and $_POST variables are used to retrieve information from forms, like user input.

PHP Form Handling

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will **automatically** be available to your PHP scripts.

Example

The example below contains an HTML form with two input fields and a submit button:

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>

</body>
</html>
```

When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called "welcome.php":

"welcome.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_POST["fname"]; ?>!<br />
You are <?php echo $_POST["age"]; ?> years old.

</body>
</html>
```

Output could be something like this:

```
Welcome John!
You are 28 years old.
```

The PHP $_GET and $_POST functions will be explained in the next chapters.

Form Validation

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.

You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

4.13 PHP $_GET Function

The built-in $_GET function is used to collect values in a form with method="get".

The $_GET Function

The built-in $_GET function is used to collect values from a form sent with method="get".

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send (max. 100 characters).

Example

```
<form action="welcome.php" method="get">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

```
http://www.w3schools.com/welcome.php?fname=Peter&age=37
```

The "welcome.php" file can now use the $_GET function to collect form data (the names of the form fields will automatically be the keys in the $_GET array):

```
Welcome <?php echo $_GET["fname"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

When to use method="get"?

When using method="get" in HTML forms, all variable names and values are displayed in the URL.

Note: This method should not be used when sending passwords or other sensitive information!

However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

Note: The get method is not suitable for large variable values; the value cannot exceed 100 characters.

4.14 PHP $_POST Function

The built-in $_POST function is used to collect values in a form with method="post".

The $_POST Function

The built-in $_POST function is used to collect values from a form sent with method="post".

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

Note: However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post_max_size in the php.ini file).

Example

```
<form action="welcome.php" method="post">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL will look like this:

```
http://www.w3schools.com/welcome.php
```

The "welcome.php" file can now use the $_POST function to collect form data (the names of the form fields will automatically be the keys in the $_POST array):

```
Welcome <?php echo $_POST["fname"]; ?>!<br />
You are <?php echo $_POST["age"]; ?> years old.
```

When to use method="post"?

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

The PHP $_REQUEST Function

The PHP built-in $_REQUEST function contains the contents of both $_GET, $_POST, and $_COOKIE.

The $_REQUEST function can be used to collect form data sent with both the GET and POST methods.

Example

```
Welcome <?php echo $_REQUEST["fname"]; ?>!<br />
You are <?php echo $_REQUEST["age"]; ?> years old.
```

4.15 PHP Cookies

A cookie is often used to identify a user.

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

How to Create a Cookie?

The setcookie() function is used to set a cookie.

Note: The setcookie() function must appear BEFORE the <html> tag.

Syntax

```
setcookie(name, value, expire, path, domain);
```

Example 1

In the example below, we will create a cookie named "user" and assign the value "Alex Porter" to it. We also specify that the cookie should expire after one hour:

```
<?php
setcookie("user", "Alex Porter", time()+3600);
?>

<html>
.....
```

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Example 2

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.

```
<?php
$expire=time()+60*60*24*30;
setcookie("user", "Alex Porter", $expire);
?>
```

```
<html>
.....
```

In the example above the expiration time is set to a month (*60 sec * 60 min * 24 hours * 30 days*).

How to Retrieve a Cookie Value?

The PHP $_COOKIE variable is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "user" and display it on a page:

```
<?php
// Print a cookie
echo $_COOKIE["user"];

// A way to view all cookies
print_r($_COOKIE);
?>
```

In the following example we use the isset() function to find out if a cookie has been set:

```
<html>
<body>

<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br />";
else
  echo "Welcome guest!<br />";
?>

</body>
</html>
```

How to Delete a Cookie?

When deleting a cookie you should assure that the expiration date is in the past.

Delete example:

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time()-3600);
?>
```

What if a Browser Does NOT Support Cookies?

If your application deals with browsers that do not support cookies, you will have to use other methods to pass information from one page to another in your application. One method is to pass the data through forms (forms and user input are described earlier in this tutorial).

The form below passes the user input to "welcome.php" when the user clicks on the "Submit" button:

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>

</body>
</html>
```

Retrieve the values in the "welcome.php" file like this:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old.

</body>
</html>
```

4.16 PHP Sessions

A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

PHP Session Variables

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Starting a PHP Session

Before you can store user information in your PHP session, you must first start up the session.

Note: The session_start() function must appear BEFORE the <html> tag:

```
<?php session_start(); ?>

<html>
<body>

</body>
</html>
```

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

Storing a Session Variable

The correct way to store and retrieve session variables is to use the PHP $_SESSION variable:

```
<?php
```

```
session_start();
// store session data
$_SESSION['views']=1;
?>

<html>
<body>

<?php
//retrieve session data
echo "Pageviews=". $_SESSION['views'];
?>

</body>
</html>
```

Output:

```
Pageviews=1
```

In the example below, we create a simple page-views counter. The isset() function checks if the "views" variable has already been set. If "views" has been set, we can increment our counter. If "views" doesn't exist, we create a "views" variable, and set it to 1:

```
<?php
session_start();

if(isset($_SESSION['views']))
$_SESSION['views']=$_SESSION['views']+1;
else
$_SESSION['views']=1;
echo "Views=". $_SESSION['views'];
?>
```

Destroying a Session

If you wish to delete some session data, you can use the unset() or the session_destroy() function.

The unset() function is used to free the specified session variable:

```
<?php
```

```
unset($_SESSION['views']);
?>
```

You can also completely destroy the session by calling the session_destroy() function:

```
<?php
session_destroy();
?>
```

Note: session_destroy() will reset your session and you will lose all your stored session data.

4.17 PHP MySQL Introduction

MySQL is the most popular open-source database system.

What is MySQL?

MySQL is a database.

The data in MySQL is stored in database objects called tables.

A table is a collections of related data entries and it consists of columns and rows.

Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

Queries

A query is a question or a request.

With MySQL, we can query a database for specific information and have a recordset returned.

Look at the following query:

```
SELECT LastName FROM Persons
```

The query above selects all the data in the "LastName" column from the "Persons" table, and will return a recordset like this:

LastName
Hansen
Svendson
Pettersen

PHP MySQL Connect to a Database

The free MySQL database is very often used with PHP.

Create a Connection to a MySQL Database

Before you can access data in a database, you must create a connection to the database.

In PHP, this is done with the mysql_connect() function.

Syntax

```
mysql_connect(servername,username,password);
```

Parameter	Description
servername	Optional. Specifies the server to connect to. Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default is ""

Note: There are more available parameters, but the ones listed above are the most important. Visit our full PHP MySQL Reference for more details.

Example

In the following example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// some code
?>
```

Closing a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the mysql_close() function:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// some code

mysql_close($con);
?>
```

PHP MySQL Create Database and Tables

A database holds one or multiple tables.

Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.

Syntax

```
CREATE DATABASE database_name
```

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example creates a database called "my_db":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }

mysql_close($con);
?>
```

Example

The following example creates a table named "Persons", with three columns. The column names will be "FirstName", "LastName" and "Age":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
```

```
// Create database
if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }

// Create table
mysql_select_db("my_db", $con);
$sql = "CREATE TABLE Persons
(
FirstName varchar(15),
LastName varchar(15),
Age int
)";

// Execute query
mysql_query($sql,$con);

mysql_close($con);
?>
```

Important: A database must be selected before a table can be created. The database is selected with the mysql_select_db() function.

Note: When you create a database field of type varchar, you must specify the maximum length of the field, e.g. varchar(15).

Primary Keys and Auto Increment Fields

Each table should have a primary key field.

A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key field cannot be null because the database engine requires a value to locate the record.

The following example sets the personID field as the primary key field. The primary key field is often an ID number, and is often used with the AUTO_INCREMENT setting. AUTO_INCREMENT automatically increases the value of the field by 1 each time a new record is added. To ensure that the primary key field cannot be null, we must add the NOT NULL setting to the field.

Example

```
$sql = "CREATE TABLE Persons
(
personID int NOT NULL AUTO_INCREMENT,
PRIMARY KEY(personID),
FirstName varchar(15),
LastName varchar(15),
Age int
)";

mysql_query($sql,$con);
```

PHP MySQL Insert Into

The INSERT INTO statement is used to insert new records in a table.

Insert Data Into a Database Table

The INSERT INTO statement is used to add new records to a database table.

Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...)
```

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

Example

In the previous chapter we created a table named "Persons", with three columns; "Firstname", "Lastname" and "Age". We will use the same table in this example. The following example adds two new records to the "Persons" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

mysql_query("INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('Peter', 'Griffin', '35')");

mysql_query("INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('Glenn', 'Quagmire', '33')");

mysql_close($con);
?>
```

Insert Data From a Form Into a Database

Now we will create an HTML form that can be used to add new records to the "Persons" table.

Here is the HTML form:

```
<html>
<body>

<form action="insert.php" method="post">
Firstname: <input type="text" name="firstname" />
Lastname: <input type="text" name="lastname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>

</body>
</html>
```

When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php".

The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $_POST variables.

Then, the mysql_query() function executes the INSERT INTO statement, and a new record will be added to the "Persons" table.

Here is the "insert.php" page:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$sql="INSERT INTO Persons (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";

if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());
  }
echo "1 record added";

mysql_close($con)
?>
```

PHP MySQL Select

The SELECT statement is used to select data from a database.

Select Data From a Database Table

The SELECT statement is used to select data from a database.

Syntax

```
SELECT column_name(s)
FROM table_name
```

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all the data stored in the "Persons" table (The * character selects all the data in the table):

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
  }

mysql_close($con);
?>
```

The example above stores the data returned by the mysql_query() function in the $result variable.

Next, we use the mysql_fetch_array() function to return the first row from the recordset as an array. Each call to mysql_fetch_array() returns the next row in the recordset. The while loop loops through all the records in the recordset. To print the value of each row, we use the PHP $row variable ($row['FirstName'] and $row['LastName']).

The output of the code above will be:

```
Peter Griffin
Glenn Quagmire
```

Display the Result in an HTML Table

The following example selects the same data as the example above, but will display the data in an HTML table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons");

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>";

while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "</tr>";
  }
echo "</table>";

mysql_close($con);
?>
```

The output of the code above will be:

Firstname	Lastname
Glenn	Quagmire
Peter	Griffin

PHP MySQL The Where Clause

The WHERE clause is used to filter records.

The WHERE clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all rows from the "Persons" table where "FirstName='Peter':

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons
WHERE FirstName='Peter'");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
  }
?>
```

The output of the code above will be:

```
Peter Griffin
```

PHP MySQL Order By Keyword

The ORDER BY keyword is used to sort the data in a recordset.

The ORDER BY Keyword

The ORDER BY keyword is used to sort the data in a recordset.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

Example

The following example selects all the data stored in the "Persons" table, and sorts the result by the "Age" column:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons ORDER BY age");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'];
  echo " " . $row['LastName'];
  echo " " . $row['Age'];
  echo "<br />";
  }

mysql_close($con);
?>
```

The output of the code above will be:

```
Glenn Quagmire 33
Peter Griffin 35
```

PHP MySQL Update

The UPDATE statement is used to modify data in a table.

Update Data In a Database

The UPDATE statement is used to update existing records in a table.

Syntax

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

Earlier in the tutorial we created a table named "Persons". Here is how it looks:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

The following example updates some data in the "Persons" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

mysql_query("UPDATE Persons SET Age = '36'
WHERE FirstName = 'Peter' AND LastName = 'Griffin'");

mysql_close($con);
```

```
?>
```

After the update, the "Persons" table will look like this:

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

PHP MySQL Delete

The DELETE statement is used to delete records in a table.

Delete Data In a Database

The DELETE FROM statement is used to delete records from a database table.

Syntax

```
DELETE FROM table_name
WHERE some_column = some_value
```

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Example

Look at the following "Persons" table:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

The following example deletes all the records in the "Persons" table where LastName='Griffin':

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);
```

```
mysql_query("DELETE FROM Persons WHERE LastName='Griffin'");

mysql_close($con);
?>
```

After the deletion, the table will look like this:

FirstName	**LastName**	**Age**
Glenn	Quagmire	33