Syllabus Covered (HTML)

- ✓ Introduction
- ✓ HTML Elements
- ✓ HTML Attributes
- ✓ HTML Headings
- ✓ HTML Paragraphs
- ✓ HTML Formatting
- ✓ HTML Fonts
- ✓ HTML Styles
- ✓ HTML Links
- ✓ HTML Images
- ✓ HTML Tables
- ✓ HTML Lists
- ✓ HTML Forms
- ✓ HTML Iframes
- ✓ HTML Colors
- ✓ HTML Colornames
- ✓ HTML Colorvalues
- ✓ HTML Layout
- ✓ HTML Doctypes
- ✓ HTML Head
- ✓ HTML Meta
- ✓ HTML Scripts
- ✓ HTML Media
- ✓ HTML Audio
- ✓ HTML Video
- ✓ HTML YouTube
- ✓ HTML Media Tags
- ✓ HTML Summary

Syllabus Covered (CSS)

- ○ CSS Introduction
- ○ CSS Syntax
- ○ CSS Id & Class
- ○ CSS Styling
- ○ Styling Backgrounds
- ○ Styling Text
- ○ Styling Fonts
- ○ Styling Links
- ○ Styling Lists
- ○ Styling Tables
- ○ CSS Box Model
- ○ CSS Box Model
- ○ CSS Border
- ○ CSS Outline
- ○ CSS Margin
- ○ CSS Padding
- ○ CSS Advanced
- ○ CSS Grouping/Nesting
- ○ CSS Dimension
- ○ CSS Display
- ○ CSS Positioning
- ○ CSS Floating
- ○ CSS Align
- ○ CSS Navigation Bar
- ○ CSS Image Gallery
- ○ CSS Image Opacity
- ○ CSS Image Sprites
- ○ CSS Media Types
- ○ CSS Summary

	Description
<!--...-->	Defines a comment
<!DOCTYPE>	Defines the document type
<a>	Defines a hyperlink
<abbr>	Defines an abbreviation or an acronym
<address>	Defines contact information for the author/owner of a document
<article>	Defines an article
<audio>	Defines sound content
<b>	Defines bold text
<body>	Defines the document's body
 	Defines a single line break
<button>	Defines a clickable button

<canvas>	Used to draw graphics, on the fly, via scripting (usually JavaScript)
<caption>	Defines a table caption
	Not supported in HTML5. Use CSS instead. Defines centered text
<dd>	Defines a description/value of a term in a description list
	Defines a section in a document
<dl>	Defines a description list
<dt>	Defines a term/name in a description list
	Defines emphasized text
<embed>	Defines a container for an external (non-HTML) application
<fieldset>	Groups related elements in a form
<figcaption>	Defines a caption for a <figure> element
<figure>	Specifies self-contained content

<font>	Not supported in HTML5. Use CSS instead. Defines font, color, and size for text
<footer>	Defines a footer for a document or section
<form>	Defines an HTML form for user input
<h1> to <h6>	Defines HTML headings
<head>	Defines information about the document
<header>	Defines a header for a document or section
<hr>	Defines a thematic change in the content
<html>	Defines the root of an HTML document
<i>	Italic font
<iframe>	Defines an inline frame
<img>	Defines an image
<input>	Defines an input control

<label>	Defines a label for an <input> element
<legend>	Defines a caption for a <fieldset> element
	Defines a list item
<link>	Defines the relationship between a document and an external resource (most used to link to style sheets)
<main>	Specifies the main content of a document
<meta>	Defines metadata about an HTML document
<nav>	Defines navigation links
	Defines an ordered list
<option>	Defines an option in a drop-down list
<p>	Defines a paragraph
<script>	Defines a client-side script
<section>	Defines a section in a document
<select>	Defines a drop-down list

<small>	Defines smaller text
	Defines a section in a document
<strike>	Not supported in HTML5. Use <del> or <s> instead. Defines strikethrough text
	Defines important text
<style>	Defines style information for a document
<table>	Defines a table
<tbody>	Groups the body content in a table
<td>	Defines a cell in a table
<template>	Defines a template
<textarea>	Defines a multiline input control (text area)
<th>	Defines a header cell in a table
<time>	Defines a date/time
<title>	Defines a title for the document

<tr>	Defines a row in a table
<u>	Defines text that should be stylistically different from normal text
	Defines an unordered list
<var>	Defines a variable
<video>	Defines a video or movie

HTML

Definitions

- ✓ W W W – World Wide Web.
- ✓ HTML – **HyperText Markup Language** – The Language of Web Pages on the World Wide Web.
  - HTML is a text formatting language.
- ✓ URL – Uniform Resource Locator.
- ✓ Browser – A software program which is used to show web pages.

- ✓ The <!DOCTYPE html> declaration defines this document to be HTML5
- ✓ The <html> element is the root element of an HTML page
- ✓ The <head> element contains meta information about the document
- ✓ The <title> element specifies a title for the document
- ✓ The <body> element contains the visible page content
- ✓ The <h1> element defines a large heading
- ✓ The <p> element defines a paragraph

HTML Tags

HTML tags are element names surrounded by angle brackets:

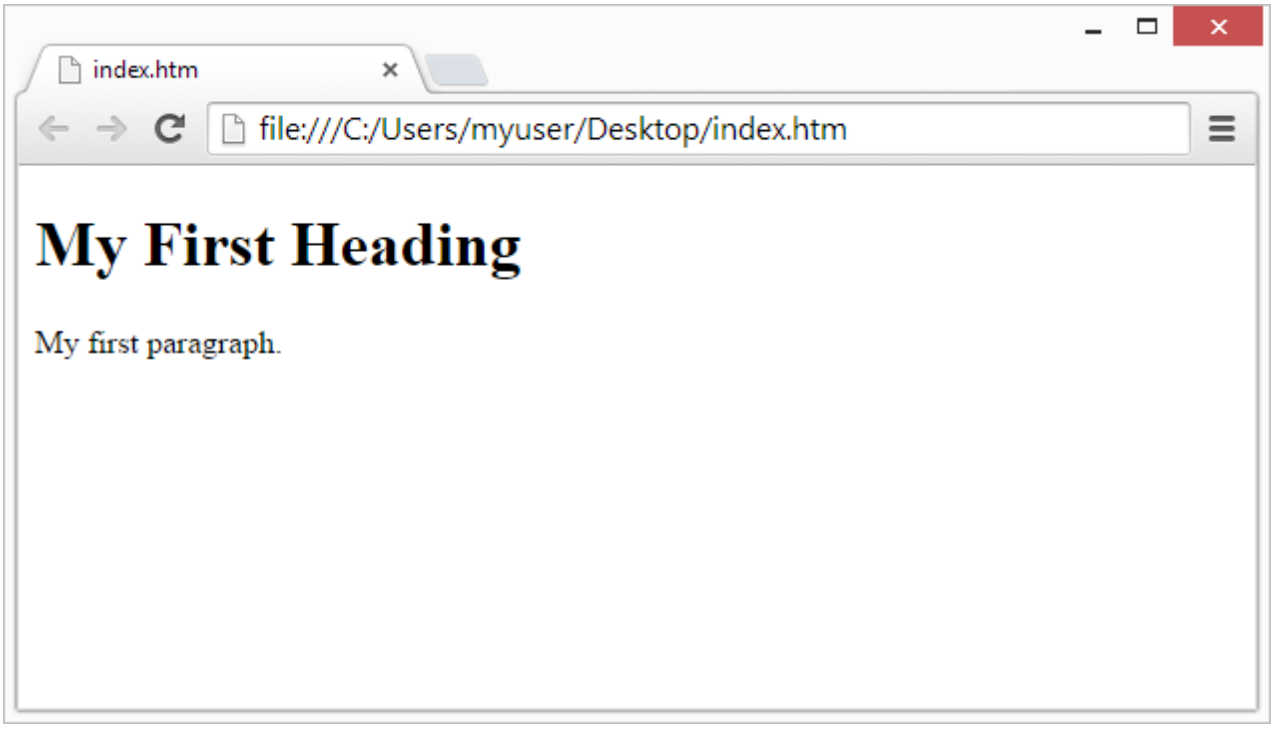
<tagname>content goes here...</tagname>

- HTML tags normally come **in pairs** like and
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

Web Browsers

The purpose of a web browser (Chrome, IE, Firefox, Safari) is to read HTML documents and display them.

The browser does not display the HTML tags, but uses them to determine how to display the document:



HTML Page Structure

Below is a visualization of an HTML page structure:

```
<html>
<head>
<title>Page title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
</body>
</html>
```

Note: Only the content inside the <body> section (the white area above) is displayed in a browser.

The <!DOCTYPE> Declaration

The <!DOCTYPE> declaration represents the document type, and helps browsers to display web pages correctly.

It must only appear once, at the top of the page (before any HTML tags).

The <!DOCTYPE> declaration is not case sensitive.

The <!DOCTYPE> declaration for HTML5 is:

<!DOCTYPE html>

HTML Versions

Since the early days of the web, there have been many versions of HTML:

Version	Year
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

HTML Editors

Write HTML Using Notepad or TextEd

Web pages can be created and modified by using professional HTML editors.

However, for learning HTML we recommend a simple text editor like Notepad (PC) or TextEdit (Mac).

We believe using a simple text editor is a good way to learn HTML.

Follow the four steps below to create your first web page with Notepad or TextEdit.

Step 1: Open Notepad (PC)

Windows 8 or later:
Open the **Start Screen** (the window symbol at the bottom left on your screen).
Type **Notepad**.
Windows 7 or earlier:
Open **Start** > **Programs > Accessories > Notepad**

Step 1: Open TextEdit (Mac)

Open **Finder > Applications > TextEdit**

Also change some preferences to get the application to save files correctly. In **Preferences > Format >** choose **"Plain Text"**

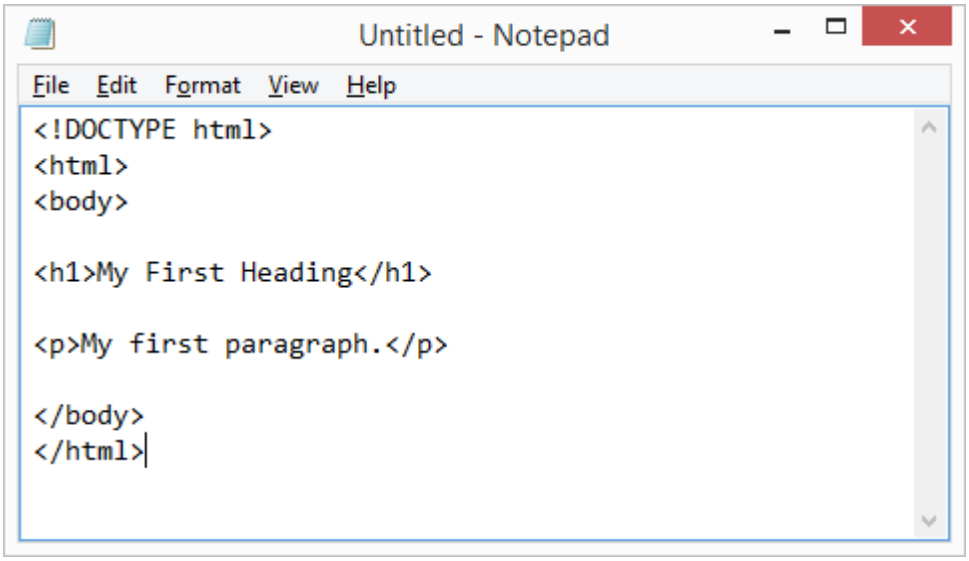
Then under "Open and Save", check the box that says "Display HTML files as HTML code instead of formatted text".

Then open a new document to place the code.

Step 2: Write Some HTML

Write or copy some HTML into Notepad.

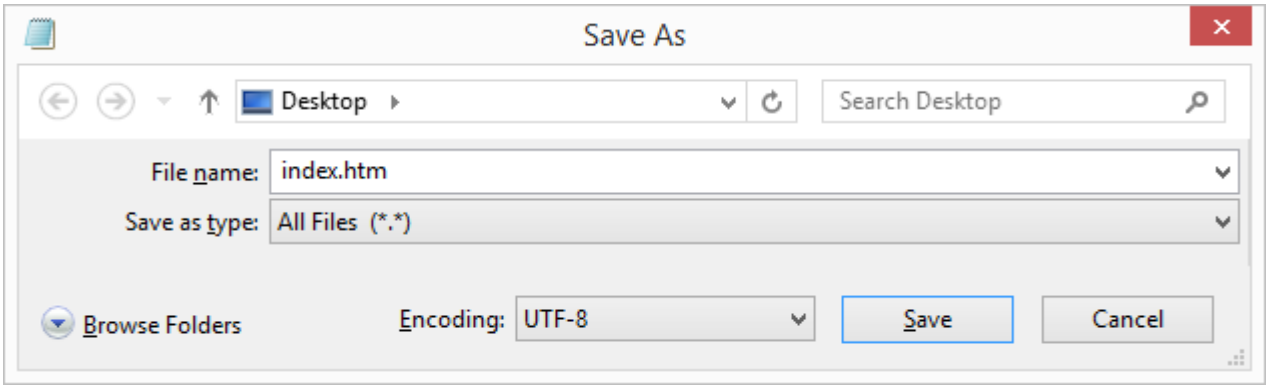
```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```



Step 3: Save the HTML Page

Save the file on your computer. Select **File > Save as** in the Notepad menu.

Name the file **"index.htm"** and set the encoding to **UTF-8** (which is the preferred encoding for HTML files).

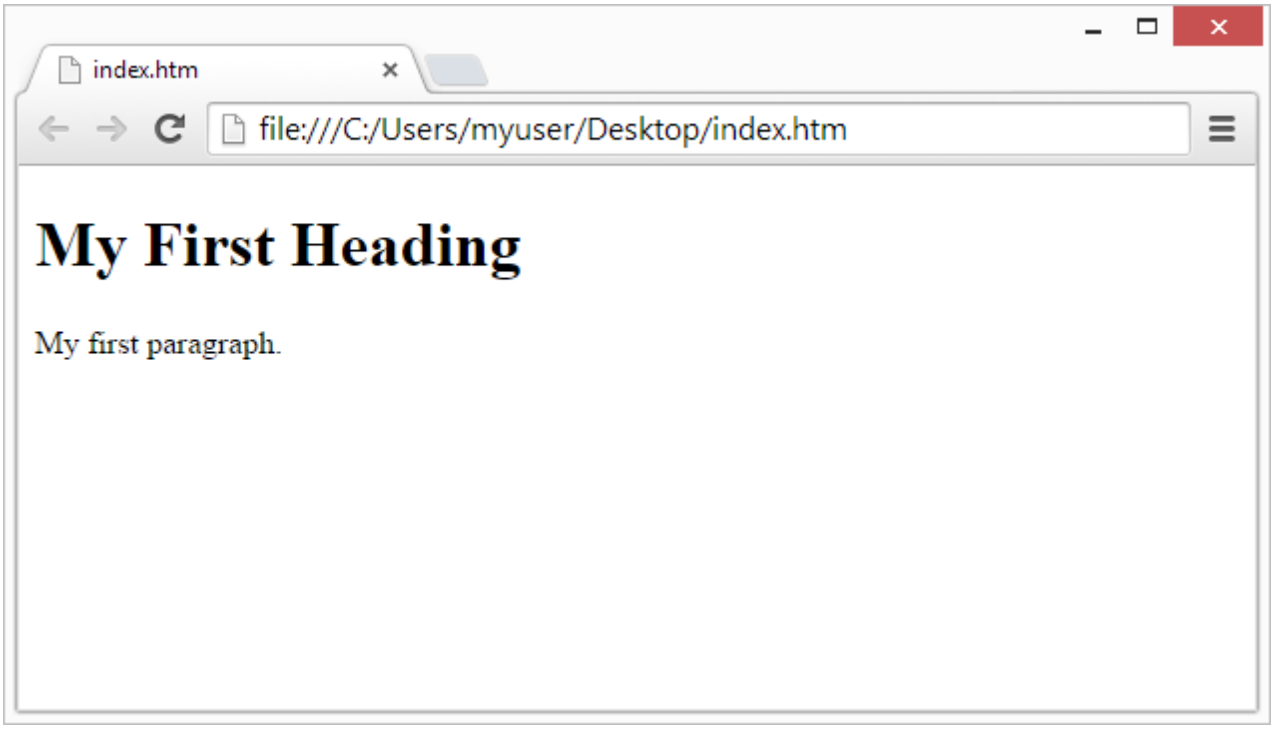


You can use either .htm or .html as file extension. There is no difference, it is up to you.

Step 4: View the HTML Page in Your Browser

Open the saved HTML file in your favorite browser (double click on the file, or right-click - and choose "Open with").

The result will look much like this:



“Normal text” surrounded by bracketed *tags* that tell browsers how to display web pages
Pages end with “.htm” or “.html”
HTML Editor – A word processor that has been specialized to make the writing of HTML documents more effortless.

Tags
Codes enclosed in brackets
Usually paired
<TITLE>My Web Page</TITLE>
Not case sensitive
<TITLE> = <title> = <TITLE>

Creating a Basic Starting Document

```
<HTML>
    <HEAD>
    <TITLE>Al al-Bayt University</TITLE>
    </HEAD>
    <BODY>
        This is what is displayed.
    </BODY>
</HTML>
```

Creating a Basic Starting Document

- The HEAD of your document point to above window part. The TITLE of your document appears in the very top line of the user’s browser. If the user chooses to “Bookmark” your page or save as a “Favorite”; it is the TITLE that is added to the list.
- The text in your TITLE should be as descriptive as possible because this is what many search engines, on the internet, use for indexing your site.

Setting Document Properties

- Document properties are controlled by attributes of the BODY element. For example, there are color settings for the background color of the page, the document’s text and different states of links.

Color Codes

- Colors are set using “RGB” color codes, which are, represented as hexadecimal values. Each 2-digit section of the code represents the amount, in sequence, of red, green or blue

that forms the color. For example, a RGB value with 00 as the first two digits has no red in the color.

- If you require more information about color values, there is an excellent site entitled "VGDesign's Interactive Color Cube" that displays the background color code when you put your cursor over a small color sample. The Web address is :

https://htmlcolorcodes.com/

THE BODY ELEMENT

The BODY element of a web page is an important element in regards to the page's appearance Here are the attributes of the **BODY** tag to control all the levels:

TEXT="#RRGGBB" to change the color of **all the text** on the page (**full page text color.**)

This element contains information about the page's background color, the background image, as well as the text and link colors.

BACKGROUND COLOR

- It is very common to see web pages with their background color set to white or some other colors.
- To set your document's background color, you need to edit the <BODY> element by adding the BGCOLOR attribute. The following example will display a document with a white background color:

<BODY BGCOLOR="#FFFFFF"></BODY>

TEXT COLOR

- The TEXT attribute is used to control the color of all the normal text in the document. The default color for text is black. The TEXT attribute would be added as follows:

<BODY BGCOLOR="#FFFFFF" TEXT="#FF0000"></BODY>

In this example the document's page
color is white and the text would be red.

LINK, VLINK, and ALINK

These attributes control the colors of the different link states:
1. LINK – initial appearance – default = Blue.
2. VLINK – visited link – default = Purple.
3. ALINK –active link being clicked–default= Yellow.

The Format for setting these attributes is:

<BODY BGCOLOR="#FFFFFF" TEXT="#FF0000" LINK="#0000FF"
VLINK="#FF00FF"
ALINK="FFFF00"> </BODY>

Using Image Background

- The BODY element also gives you ability of setting an image as the document's background.
- An example of a background image's HTML code is as follows:

<BODY BACKGROUND="hi.gif" BGCOLOR="#FFFFFF"></BODY>

HEADINGS, PARAGRAPHS, BREAKS & HORIZONTAL RULES

In this chapter you will add headings to your page, insert paragraphs, add some breaks, and add horizontal rules.

Objectives

- Upon completing this section, you should be able to
- List and describe the different Heading elements.
- Use Paragraphs to add text to a document.
- Insert breaks where necessary.
- Add a Horizontal Rule.

Headings, <Hx> </Hx>

- Inside the **BODY** element, heading elements **H1** through **H6** are generally used for major divisions of the document. Headings are permitted to appear in any order, but you will obtain the best results when your documents are displayed in a browser if you follow these guidelines:
- **H1**: should be used as the highest level of heading, **H2** as the next highest, and so forth.
- You should not skip heading levels: e.g., an **H3** should not appear after an **H1**, unless there is an **H2** between them.

Headings, <Hx> </Hx>

```
<HTML>
<HEAD>
<TITLE> Example Page</TITLE>
</HEAD>
<BODY>
<H1> Heading 1 </H1>
<H2> Heading 2 </H2>
<H3> Heading 3 </H3>
<H4> Heading 4 </H4>
<H5> Heading 5 </H5>
<H6> Heading 6 </H6>
</BODY>
```

</HTML>

Paragraphs, <P> </P>

- Paragraphs allow you to add text to a document in such a way that it will automatically adjust the end of line to suite the window size of the browser in which it is being displayed. Each line of text will stretch the entire length of the window.

<HTML><HEAD>
<TITLE> Example Page</TITLE>
</HEAD>
<BODY></H1> Heading 1 </H1>
<P> Paragraph 1, ….</P>
<H2> Heading 2 </H2>
<P> Paragraph 2, ….</P>
<H3> Heading 3 </H3>
<P> Paragraph 3, ….</P>
<H4> Heading 4 </H4>
<P> Paragraph 4, ….</P>
<H5> Heading 5 </H5>
<P> Paragraph 5, ….</P>
<H6> Heading 6</H6>
<P> Paragraph 6, ….</P>
</BODY></HTML>

Output
Heading 1
Paragraph 1,….
Heading 2
Paragraph 2,….
Heading 3
Paragraph 3,….
Heading 4
Paragraph 4,….
Heading 5
Paragraph 5,….
Heading 6
Paragraph 6,….

**Break,
**

- Line breaks allow you to decide where the text will break on a line or continue to the end of the window.
- A
 is an empty Element, meaning that it may contain attributes but it does not contain content.
- The
 element does not have a closing tag.

```
<HTML>
<HEAD>
<TITLE> Example Page</TITLE>
</HEAD>
<BODY>
<H1> Heading 1 </H1>
<P>Paragraph 1, <BR>
Line 2 <BR> Line 3 <BR>….
</P>
</BODY>
</HTML>
```

OUTPUT

Heading 1
Paragraph 1,….
Line 2
Line 3
….

Horizontal Rule - <HR>

- The <HR> element causes the browser to display a horizontal line (rule) in your document.
- <HR> does not use a closing tag, </HR>.

```
<HTML>
<HEAD>
<TITLE> Example Page</TITLE>
</HEAD>
<BODY>
<H1> Heading 1 </H1>
<P>Paragraph 1, <BR>
Line 2 <BR>
<HR>Line 3 <BR>
</P>
</BODY>
</HTML>
```

Heading 1
Paragraph 1,….
Line 2

Line 3

HTML Formatting Elements

In the previous chapter, you learned about the HTML **style attribute**.

HTML also defines special **elements** for defining text with a special **meaning**.

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

Formatting elements were designed to display special types of text:

- <b> - Bold text
- - Important text
- <i> - Italic text
- - Emphasized text
- <mark> - Marked text
- <small> - Small text
- <del> - Deleted text
- <ins> - Inserted text
- <sub> - Subscript text
- <sup> - Superscript text

HTML <b> and Elements

The HTML <b> element defines **bold** text, without any extra importance.

Example

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

The HTML element defines **strong** text, with added semantic "strong" importance.

Example

```
<strong>This text is strong</strong>
```

HTML <i> and Elements

The HTML <i> element defines *italic* text, without any extra importance.

Example

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

The HTML element defines *emphasized* text, with added semantic importance.

Example

```
<em>This text is emphasized</em>
```

Note: Browsers display as <b>, and as <i>. However, there is a difference in the meaning of these tags: <b> and <i> defines bold and italic text, but and means that the text is "important".

HTML <small> Element

The HTML <small> element defines smaller text:

Example

```
<h2>HTML <small>Small</small> Formatting</h2>
```

HTML <mark> Element

The HTML <mark> element defines marked or highlighted text:

Example

```
<h2>HTML <mark>Marked</mark> Formatting</h2>
```

HTML <del> Element

The HTML <del> element defines ~~deleted~~ (removed) text.

Example

```
<p>My favorite color is <del>blue</del> red.</p>
```

HTML <ins> Element

The HTML <ins> element defines inserted (added) text.

Example

```
<p>My favorite <ins>color</ins> is red.</p>
```

HTML <sub> Element

The HTML <sub> element defines subscripted text.

Example

```
<p>This is <sub>subscripted</sub> text.</p>
```

HTML <sup> Element

The HTML <sup> element defines superscripted text.

Example

```
<p>This is <sup>superscripted</sup> text.</p>
```

HTML Text Formatting Elements

Tag	Description
<b>	Defines bold text
	Defines emphasized text
<i>	Defines italic text

<small>	Defines smaller text
	Defines important text
<sub>	Defines subscripted text
<sup>	Defines superscripted text
<ins>	Defines inserted text
<del>	Defines deleted text
<mark>	Defines marked/highlighted text

Character Formatting

In this chapter you will learn how to enhance your page with Bold, Italics, and other character formatting options.

Bold, Italic and other Character Formatting Elements

- **<FONT SIZE="+2">** Two sizes bigger**</FONT>**
- The size attribute can be set as an absolute value from 1 to 7 or as a relative value using the "+" or "-" sign. Normal text size is 3 (from -2 to +4).
- **<B> Bold </B>**
- **<I>** ***Italic*** **</I>**
- **<U> Underline </U>**
- Color = "#RRGGBB" The COLOR attribute of the FONT element. E.g., **<FONT COLOR="#RRGGBB">this text has color</FONT>**.

    - Bold, Italic and other Character Formatting Elements

- **<EM>** ***Emphasis*** **</EM>** Browsers usually display this as italics.
- **<STRONG> STRONG </STRONG>** Browsers display this as bold.

<P> <FONT SIZE="+1"> One Size Larger </FONT> - Normal –
<FONT SIZE="-1"> One Size Smaller </FONT>
<B> Bold</B> - <I> italics</I> - <U> Underlined </U> -
<FONT COLOR="#FF0000"> Colored </FONT>
<EM> Emphasized</EM> - <STRONG> Strong </STRONG> - <TT> Tele Type </TT>

One Size Larger - Normal – One Size Smaller
Bold - *italics* - Underlined - Colored
Emphasized - **Strong** - Tele Type

Alignment

- Some elements have attributes for alignment (ALIGN) e.g. Headings, Paragraphs and Horizontal Rules.
- The Three alignment values are : LEFT, RIGHT, CENTER.
- <CENTER></CENTER> Will center elements.

Alignment

- **<DIV ALIGN="value"></DIV>** Represents a division in the document and can contain most other element type. The alignment attribute of the DIV element is well supported.
- **<TABLE></TABLE>** Inside a TABLE, alignment can be set for each individual cell.

Additional Character Formatting Elements

- **<STRIKE>** strike-through text**</STRIKE>**

DEL is used for STRIKE at the latest browsers

- **<BIG>** places text in a big font**</BIG>**
- **<SMALL>** places text in a small font**</SMALL>**

Example
<P><STRIKE> strike-through text </STRIKE></BR>

<BIG>places text in a big font </BIG>

<SMALL> places text in a small font</SMALL>

Lists

In this chapter you will learn how to create a variety of lists.

Objectives

Upon completing this section, you should be able to

- Create an unordered list.

- Create an ordered list.
- Create a defined list.
- Nest Lists.

List Elements

- HTML supplies several list elements. Most list elements are composed of one or more <LI> (List Item) elements.
- UL : Unordered List. Items in this list start with a list mark such as a bullet. Browsers will usually change the list mark in nested lists.

<UL>
<LI> List item **…</LI>**
<LI> List item **…</LI>**
</UL>
OUTPUT

- List item …
- List item …

List Elements

- You have the choice of three bullet types: **disc(default), circle, square.**
- These are controlled in Netscape Navigator by the "TYPE" attribute for the <UL> element.

<UL TYPE="square">
<LI> List item …</LI>
<LI> List item …</LI>
<LI> List item …</LI>
</UL>

- List item …
- List item …
- List item …

OL: Ordered List. Items in this list are numbered automatically by the browser.

<OL>
<LI> List item …</LI>
<LI> List item …</LI>
<LI> List item …</LI>
</OL>

1. **List item …**
2. **List item …**
3. **List item**

- You have the choice of setting the TYPE Attribute to one of five numbering styles.
- You can specify a starting number for an ordered list.

<OL TYPE =“i”>
<LI> List item …</LI>
<LI> List item …</LI>
</OL>
<P> text ….</P>
<OL TYPE=“i” START=“3”>
<LI> List item …</LI>
</OL>
List item …
List item …

Text ….

List item …

List Elements

DL: Definition List. This kind of list is different from the others. Each item in a DL consists of one or more **Definition Terms (DT elements),** followed by one or more **Definition Description (DD elements).**

<DL>
<DT> HTML </DT>
<DD> Hyper Text Markup Language </DD>
<DT> DOG </DT>
<DD> A human’s best friend!</DD>
</DL>

HTML

Hyper Text Markup Language

DOG

A human’s best friend!

Nesting Lists

You can nest lists by inserting a UL, OL, etc., inside a list item (LI).

EXample

<UL TYPE = “square”>
<LI> List item …</LI>
<LI> List item …
<OL TYPE=“i” START=“3”>
<LI> List item …</LI>
<LI> List item …</LI>
<LI> List item …</LI>
<LI> List item …</LI>

<LI> List item …</LI>
</OL>
</LI>
<LI> List item …</LI>
</UL>

In this chapter you will learn about images and how to place images in your pages.

Objectives
Upon completing this section, you should be able to.Add images to your pages.
I**MAGES**

- **<IMG>**This element defines a graphic image on the page.
- **Image File (SRC:source):** This value will be a URL (location of the image) E.g. http://www.domain.com/dir/file.ext or /dir/file.txt.
- **Alternate Text (ALT):** This is a text field that describes an image or acts as a label. It is displayed when they position the cursor over a graphic image.
- **Alignment (ALIGN):** This allows you to align the image on your page.
- **Width (WIDTH):** is the width of the image in pixels.
- **Height (HEIGHT):** is the height of the image in pixels.
- **Border (BORDER):** is for a border around the image, specified in pixels.
- **HSPACE:** is for Horizontal Space on both sides of the image specified in pixels. A setting of 5 will put 5 pixels of invisible space on both sides of the image.
- **VSPACE:** is for Vertical Space on top and bottom of the image specified in pixels. A setting of 5 will put 5 pixels of invisible space above and bellow the image.

2) <IMG SRC=" jordan.gif" width="60" height="60">
3) <IMG SRC="jordan.gif" ALT="This is a text that goes with the image">
4) <IMG SRC=" jordan.gif " Hspace="30" Vspace="10" border=20>
5) < IMG SRC =" jordan.gif" align="left">

A**nchors,** URL**s and** I**mage** M**aps**
In this chapter you will learn about Uniform Resource Locator, and how to add them as Anchor or Links inside your web pages.

- **Objectives**
- Upon completing this section, you should be able to
- Insert links into documents.
- Define Link Types.
- Define URL.
- List some commonly used URLs.
- Plan an Image Map.

HOW TO MAKE A LINK
1) The tags used to produce links are the <A>

and </A>. The <A> tells where the link should start and the </A> indicates where the link ends. Everything between these two will work as a link.

2) The example below shows how to make the word Here work as a link to yahoo.

Click <A HREF="http://www.yahoo.com">here</A> to go to yahoo.

HTML <meta> Tag

Example

Describe metadata within an HTML document:

```
<head>
  <meta charset="UTF-8">
  <meta name="description" content="Free Web tutorials">
  <meta name="keywords" content="HTML,CSS,XML,JavaScript">
  <meta name="author" content="John Doe">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
```

Definition and Usage

Metadata is data (information) about data.

The <meta> tag provides metadata about the HTML document. Metadata will not be displayed on the page, but will be machine parsable.

Meta elements are typically used to specify page description, keywords, author of the document, last modified, and other metadata.

The metadata can be used by browsers (how to display content or reload page), search engines (keywords), or other web services.

HTML5 introduced a method to let web designers take control over the viewport (the user's visible area of a web page), through the <meta> tag (See "Setting The Viewport" example below).

Tips and Notes

Note: <meta> tags always go inside the <head> element.

Note: Metadata is always passed as name/value pairs.

Note: The content attribute MUST be defined if the name or the http-equiv attribute is defined. If none of these are defined, the content attribute CANNOT be defined.

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.

The viewport is the user's visible area of a web page. It varies with the device, and will be smaller on a mobile phone than on a computer screen.

You should include the following <meta> viewport element in all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

A <meta> viewport element gives the browser instructions on how to control the page's dimensions and scaling.

The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

Here is an example of a web page *without* the viewport meta tag, and the same web page *with* the viewport meta tag:

Tip: If you are browsing this page with a phone or a tablet, you can click on the two links below to see the difference.

Without the viewport meta tag

With the viewport meta tag

T**ABLES**

In this chapter you will learn that tables have many uses in HTML.

Objectives:

Upon completing this section, you should be able to:

- Insert a table.
- Explain a table's attributes.
- Edit a table.
- Add a table header.

Tables

- The <TABLE></TABLE> element has four sub-elements:
- Table Row<TR></TR>.
- Table Header <TH></TH>.
- Table Data <TD></TD>.
- Caption <CAPTION></CAPTION>.

The table row elements usually contain table header elements or table data elements.

Tables

```
<table border="1">
<tr>
<th> Column 1 header </th>
<th> Column 2 header </th>
</tr>
<tr>
<td> Row1, Col1 </td>
<td> Row1, Col2 </td>
</tr>
<tr>
<td> Row2, Col1 </td>
<td> Row2, Col2 </td>
</tr>
</table>
```

Tables Attributes

- **BGColor:** Some browsers support background colors in a table.
- **Width:** you can specify the table width as an absolute number of pixels or a percentage of the document width. You can set the width for the table cells as well.
- **Border:** You can choose a numerical value for the border width, which specifies the border in pixels.

- **CellSpacing:** Cell Spacing represents the space between cells and is specified in pixels.
- **CellPadding:** Cell Padding is the space between the cell border and the cell contents and is specified in pixels.
- **Align:** tables can have left, right, or center alignment.
- **Background:** Background Image, will be titled in IE3.0 and above.
- BorderColor, BorderColorDark.

Table Caption

A table caption allows you to specify a line of text that will appear centered above or bellow the table.

<TABLE BORDER=1 CELLPADDING=2>
<CAPTION ALIGN="BOTTOM"> Label For My Table </CAPTION>

- The Caption element has one attribute ALIGN that can be either TOP (Above the table) or BOTTOM (below the table).
- Table Header
- Table Data cells are represented by the TD element. Cells can also be TH (Table Header) elements which results in the contents of the table header cells appearing centered and in bold text.

Table Data and Table Header Attributes

- **Colspan:** Specifies how many cell columns of the table this cell should span.
- **Rowspan*:*** Specifies how many cell rows of the table this cell should span.
- **Align*:*** cell data can have left, right, or center alignment.
- **Valign*:*** cell data can have top, middle, or bottom alignment.
- **Width*:*** you can specify the width as an absolute number of pixels or a percentage of the document width.
- **Height*:*** You can specify the height as an absolute number of pixels or a percentage of the document height.

Basic Table Code

```
<TABLE BORDER=1 width=50%>
<CAPTION>  <h1>Spare Parts </h1> </Caption>
<TR>
    <TH>Stock Number</TH>
    <TH>Description</TH>
    <TH>List Price</TH>
</TR>
<TR>
    <TD bgcolor=red>3476-AB</TD>
    <TD>76mm Socket</TD>
    <TD>45.00</TD>
</TR>
```

```
<TR>
        <TD >3478-AB</TD>
        <TD><font color=blue>78mm Socket</font> </TD>
        <TD>47.50</TD>
</TR>
<TR>
        <TD>3480-AB</TD>
        <TD>80mm Socket</TD>
        <TD>50.00</TD>
</TR>
</TABLE>
```

Table Data and Table Header Attributes

```
<Table border=1 cellpadding =2>
<tr>
        <th> Column 1 Header</th>
        <th> Column 2 Header</th>
</tr>
<tr>
        <td colspan=2> Row 1 Col 1</td>
</tr>
<tr>
        <td rowspan=2>Row 2 Col 1</td>
        <td> Row 2 Col2</td>
 </tr>
<tr>
        <td> Row 3 Col2</td>
</tr>
</table>
```

- **TH, TD and TR should always have end tags.**

Although the end tags are formally optional, many browsers will mess up the formatting of the table if you omit the end tags. In particular, you should ***always*** use end tags if you have a TABLE within a TABLE -- in this situation, the table parser gets hopelessly confused if you don't close your TH, TD and TR elements.

- **A default TABLE has no borders**
  By default, tables are drawn without border lines. You need the BORDER attribute to draw the lines.

- **By default, a table is flush with the left margin**
  TABLEs are plopped over on the left margin. If you want centered tables, You can either: place the table inside a DIV element with attribute ALIGN="center".
  Most current browsers also supports table alignment, using the ALIGN attribute. Allowed values are "left", "right", or "center", for example: <TABLE ALIGN="left">. The values "left" and "right" float the table to the left or right of the page, with text flow allowed around the table. This is entirely equivalent to IMG alignment

What will be the output?

```
<TABLE BORDER width="750">
<TR>
        <TD colspan="4" align="center">Page Banner</TD>
</TR>
 <TR>
        <TD rowspan="2" width="25%">Nav Links</TD>
        <TD colspan="2">Feature Article</TD>
        <TD rowspan="2" width="25%">Linked Ads</TD>
 </TR>

<TR>
      <TD width="25%">News Column 1 </TD>
      <TD width="25%"><News Column 2 </TD>
</TR>
</TABLE>
```

FORMS

- Forms add the ability to web pages to not only provide the person viewing the document with dynamic information but also to obtain information from the person viewing it, and process the information.

Objectives:

Upon completing this section, you should be able to

- Create a FORM.
- Add elements to a FORM.
- Define CGI (Common Gateway Interface).
- Describe the purpose of a CGI Application.
- Specify an action for the FORM.
- Forms work in all browsers.
- Forms are Platform Independent.

To insert a form we use the <FORM></FORM> tags. The rest of the form elements must be inserted in between the form tags.

```
<HTML> <HEAD>
<TITLE> Sample Form</TITLE>
</HEAD>
<BODY BGCOLOR="FFFFFF">
<FORM ACTION = http://www.xnu.com/formtest.asp>
<P> First Name: <INPUT TYPE="TEXT" NAME="fname" MAXLENGTH="50"> </P>
<P> <INPUT TYPE="SUBMIT" NAME="fsubmit1" VALUE="Send Info"> </P>
</FORM>
</BODY> </HTML>
```

- ✓ **ACTION:** is the **URL** of the **CGI** (Common Gateway Interface) program that is going to accept the data from the form, process it, and send a response back to the browser. (mandatory)
- ✓ **METHOD: GET** (default) or **POST** specifies which **HTTP** method will be used to send the form's contents to the web server. The CGI application should be written to accept the data from either method. (mandatory)

- ✓ **NAME:** is a form name used by VBScript or JavaScripts.
- ✓ **TARGET:** is the target frame where the response page will show up.

Form elements have properties:

- Text boxes,
- Password boxes,
- Checkboxes,
- Option(Radio) buttons,
- Submit,
- Reset,
- File,
- Hidden and

- Image.

- **The properties are specified in the TYPE Attribute of the HTML element <INPUT></INPUT>.**

- **Text boxes*:*** Used to provide input fields for text, phone numbers, dates, etc.
  <INPUT TYPE= " TEXT " >

  Textboxes use the following attributes:

✓ **TYPE:** text.
✓ **SIZE:** determines the size of the textbox in characters. **Default=20** characters.
✓ **MAXLENGHT *:*** determines the maximum number of characters that the field will accept.
✓ **NAME:** is the name of the variable to be sent to the CGI application.
✓ **VALUE:** will display its contents as the default value.

```
<TITLE>Form_Text_Type</TITLE>
</HEAD> <BODY>
<h1> <font color=blue>Please enter the following bioData</font></h1>
<FORM name="fome1"  Method= " get " Action= " URL " >
First Name: <INPUT TYPE="TEXT" NAME="FName"
SIZE="15" MAXLENGTH="25"><BR>
Last Name: <INPUT TYPE="TEXT" NAME="LName"
SIZE="15" MAXLENGTH="25"><BR>
Nationality: <INPUT TYPE="TEXT" NAME="Country"
SIZE="25" MAXLENGTH="25"><BR>
The Phone Number: <INPUT TYPE="TEXT" NAME="Phone"
SIZE="15" MAXLENGTH="12"><BR>
</FORM> </BODY> </HTML>
```

- **Password:** Used to allow entry of passwords.
- **<INPUT TYPE= " PASSWORD " >**
- Browser will display
- Text typed in a password box is starred out in the browser
- display.
- Password boxes use the following attributes:
- **TYPE:** password.
- **SIZE:** determines the size of the textbox in characters.
- **MAXLENGHT:** determines the maximum size of the password in characters.
- **NAME:** is the name of the variable to be sent to the CGI application.

- **VALUE:** is usually blank.
- **<HTML><HEAD>**
- **<TITLE>Form_Password_Type</TITLE></HEAD>**
- **<BODY>**
- **<h1> To Access, Please**
- **enter:</h1>**
- **<FORM name="fome2" Action="url" method="get">**
- **User Name: <INPUT TYPE="TEXT" Name="FName"**
- **SIZE="15" MAXLENGTH="25">
**
- **Password: <INPUT TYPE="PASSWORD"**
- **NAME="PWord" value="" SIZE="15"**
- **MAXLENGTH="25">
**
- **</FORM></BODY> </HTML>**

- **Hidden:** Used to send data to the CGI application that you don't want the web surfer to see, change or have to enter but is necessary for the application to process the form correctly.
- **<INPUT TYPE="HIDDEN">**
- **Nothing is displayed in the browser.**
- Hidden inputs have the following attributes:
- **TYPE:** hidden.
- **NAME:** is the name of the variable to be sent to the CGI application.
- **VALUE:** is usually set a value expected by the CGI application.
- **Check Box:** Check boxes allow the users to select more than one option.
- **<INPUT TYPE="CHECKBOX">**
- Browser will display

- Checkboxes have the following attributes:
- **TYPE:** checkbox.
- **CHECKED:** is blank or CHECKED as the initial
- status.
- **NAME:** is the name of the variable to be sent to the
- CGI application.
- **VALUE:** is usually set to a value.

- **Radio Button:** Radio buttons allow the users to select
- only one option.
- **<INPUT TYPE="RADIO">**
- Browser will display

- Radio buttons have the following attributes:
- **TYPE:** radio.
- **CHECKED:** is blank or CHECKED as the initial

- status. Only one radio button can be
- checked

- **NAME:** is the name of the variable to be sent to the
  - CGI application.
- **VALUE:** usually has a set value.

- **Push Button:** This element would be used with
- JavaScript to cause an action to take place.
- **<INPUT TYPE="BUTTON">**
- Browser will display

- Push Button has the following attributes:
- **TYPE:** button.
- **NAME:** is the name of the button to be used
- in scripting.
- **VALUE:** determines the text label on the button.

```
<DIV align=center><BR><BR>
<FORM>
<FONT Color=red>
<h1>Press Here to see a baby crying:<BR>
<INPUT TYPE="button" VALUE="PressMe"><BR><BR>
<FONT Color=blue>
Click Here to see a baby shouting:<BR>
<INPUT TYPE="button" VALUE="ClickMe" > <BR><BR>
<FONT Color=green>
Hit Here to see a baby eating:<BR>
<INPUT TYPE="button" VALUE="HitME" > <BR><BR>
<FONT Color=yellow>
</FORM></DIV>
```

Submit: Every set of Form tags requires a Submit button. This is the element causes the browser to send the names and values of the other elements to the CGI Application specified by the ACTION attribute of the FORM element.

- **<INPUT TYPE="SUBMIT">**
- The browser will display
- Submit has the following attributes:
- **TYPE:** submit.
- **NAME:** value used by the CGI script for processing.
- **VALUE:** determines the text label on the button, usually Submit Query.

- **Reset:** It is a good idea to include one of these for each form where users are entering data. It allows the surfer to clear all the input in the form.

- **<INPUT TYPE="RESET">**

- Browser will display
-
- Reset buttons have the following attributes:
- **TYPE:** reset.
- **VALUE:** determines the text label on the button, usually Reset.

Image Submit Button: Allows you to substitute an image for the standard submit button.

<INPUT TYPE="IMAGE" SRC="jordan.gif">

Image submit button has the following attributes:

- **TYPE:** Image.
- **NAME:** is the name of the button to be used in scripting.
- **SRC:** URL of the Image file.

```
<form>
<H1><font color=blue>
Click to go Jordan's Map:
<input  type="image"  src="jordan.gif">
</form>
```

File

- **File Upload:** You can use a file upload to allow surfers to upload files to your web server.
- **<INPUT TYPE="FILE">**
- Browser will display

- File Upload has the following attributes:
- **TYPE:** file.
- **SIZE:** is the size of the text box in characters.
- **NAME:** is the name of the variable to be sent to the
- CGI application.
- **MAXLENGHT:** is the maximum size of the input in the
- textbox in characters.

```
<BODY bgcolor=lightblue>
<form>
<H3><font color=forestgreen>
Please attach your file here to for uploading to
My <font color =red>SERVER...<BR>

<input  type="file"  name="myfile"  size="30">

<input  type="submit"  value="submitfile">
</form>
</body>
```

<TEXTAREA></TEXTAREA>: is an element that allows for free form text entry.

- Browser will display

- Textarea has the following attributes:
- **NAME:** is the name of the variable to be sent to the CGI application.
- **ROWS:** the number of rows to the textbox.
- **COLS:** the number of columns to the textbox.

The two following examples are **<SELECT></SELECT>** elements, where the attributes are set differently.

- The Select elements attributes are:
- **NAME:** is the name of the variable to be sent to the CGI application.
- **SIZE:** this sets the number of **visible** choices.
- **MULTIPLE:** the presence of this attribute signifies that the user can make multiple selections. By default only one selection is allowed.
- Drop Down List

- Name: is the name of the variable to be sent to the CGI application.
- Size: 1.

- Other Elements used in Forms
- List Box

- Name: is the name of the variable to be sent to the CGI application.
- SIZE: is greater than one.
- Other Elements used in Forms
- Option

The list items are added to the **<SELECT>** element by inserting **<OPTION></OPTION>**

elements.

The Option Element's attributes are:

- **SELECTED:** When this attribute is present, the option is selected when the document is initially loaded. **It is an error for more than one option to be selected.**
- **VALUE:** Specifies the value the variable named in the select element.

```
</head>
<body>
<h2><font color=blue>what type of computer do you have?</font><h2>
<form>
<select name="computertype" size=4>
        <option  value="ibm" selected> ibm</option>
        <option  value="intel"> intel</option>
        <option value=" apple"> apple</option>
        <option value="compaq"> compaq</option>
</select>
</form></body></html>
```

```
<head> <title>select with mutiple </title> </head>
<body>
<h2><font color=blue>what type of computer do you have?</font><h2>
<form>
<select name="computertype" size=5   multiple>
        <option  value="ibm" > ibm</option>
        <option  value="intel"> intel</option>
        <option value=" apple"> apple</option>
        <option value="compaq" selected> compaq</option>
        <option value=" other"> other</option>
</select>
</form></body></html>
```

HTML5 Canvas

The HTML <canvas> element is used to draw graphics on a web page.

The graphic to the left is created with <canvas>. It shows four elements: a red rectangle, a gradient rectangle, a multicolor rectangle, and a multicolor text.

What is HTML Canvas?

The HTML <canvas> element is used to draw graphics, on the fly, via JavaScript.

The <canvas> element is only a container for graphics. You must use JavaScript to actually draw the graphics.

Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

Browser Support

The numbers in the table specify the first browser version that fully supports the<canvas> element.

Canvas Examples

A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content.

The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Note: Always specify an id attribute (to be referred to in a script), and a widthand height attribute to define the size of the canvas. To add a border, use the style attribute.

Here is an example of a basic, empty canvas:

Example

```
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;">
</canvas>
```

HTML Multimedia

Multimedia on the web is sound, music, videos, movies, and animations.

What is Multimedia?

Multimedia comes in many different formats. It can be almost anything you can hear or see.

Examples: Images, music, sound, videos, records, films, animations, and more. Web pages often contain multimedia elements of different types and formats. In this chapter you will learn about the different multimedia formats.

Browser Support

The first web browsers had support for text only, limited to a single font in a single color.

Later came browsers with support for colors and fonts, and images!

Audio, video, and animation have been handled differently by the major browsers. Different formats have been supported, and some formats require extra helper programs (plug-ins) to work.

Hopefully this will become history. HTML5 multimedia promises an easier future for multimedia.

Multimedia Formats

Multimedia elements (like audio or video) are stored in media files.

The most common way to discover the type of a file, is to look at the file extension.

Multimedia files have formats and different extensions like: .swf, .wav, .mp3, .mp4, .mpg, .wmv, and .avi.

Common Video Formats

MP4 is the new and upcoming format for internet video.

MP4 is recommended by YouTube.

MP4 is supported by Flash Players.

MP4 is supported by HTML5.

Format	File	Description
MPEG	.mpg .mpeg	MPEG. Developed by the Moving Pictures Expert Group. The first popular video format on the web. Used to be supported by all browsers, but it is not supported in HTML5 (See MP4).
AVI	.avi	AVI (Audio Video Interleave). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
WMV	.wmv	WMV (Windows Media Video). Developed by Microsoft. Commonly used in video cameras and TV hardware. Plays well on Windows computers, but not in web browsers.
QuickTime	.mov	QuickTime. Developed by Apple. Commonly used in video cameras and TV hardware. Plays well on Apple computers, but not in web browsers. (See MP4)
RealVideo	.rm .ram	RealVideo. Developed by Real Media to allow video streaming with low bandwidths. It is still used for online video and Internet TV, but does not play in web browsers.
Flash	.swf .flv	Flash. Developed by Macromedia. Often requires an extra component (plug-in) to play in web browsers.

Ogg	.ogg	Theora Ogg. Developed by the Xiph.Org Foundation. Supported by HTML5.
WebM	.webm	WebM. Developed by the web giants, Mozilla, Opera, Adobe, and Google. Supported by HTML5.
MPEG-4 or MP4	.mp4	MP4. Developed by the Moving Pictures Expert Group. Based on QuickTime. Commonly used in newer video cameras and TV hardware. Supported by all HTML5 browsers. Recommended by YouTube.

Only MP4, WebM, and Ogg video are supported by the HTML5 standard.

Audio Formats

MP3 is the newest format for compressed recorded music. The term MP3 has become synonymous with digital music.

If your website is about recorded music, MP3 is the choice.

Format	File	Description
MIDI	.mid .midi	MIDI (Musical Instrument Digital Interface). Main format for all electronic music devices like synthesizers and PC sound cards. MIDI files do not contain sound, but digital notes that can be played by electronics. Plays well on all computers and music hardware, but not in web browsers.

RealAudio	.rm .ram	RealAudio. Developed by Real Media to allow streaming of audio with low bandwidths. Does not play in web browsers.
WMA	.wma	WMA (Windows Media Audio). Developed by Microsoft. Commonly used in music players. Plays well on Windows computers, but not in web browsers.
AAC	.aac	AAC (Advanced Audio Coding). Developed by Apple as the default format for iTunes. Plays well on Apple computers, but not in web browsers.
WAV	.wav	WAV. Developed by IBM and Microsoft. Plays well on Windows, Macintosh, and Linux operating systems. Supported by HTML5.
Ogg	.ogg	Ogg. Developed by the Xiph.Org Foundation. Supported by HTML5.
MP3	.mp3	MP3 files are actually the sound part of MPEG files. MP3 is the most popular format for music players. Combines good compression (small files) with high quality. Supported by all browsers.
MP4	.mp4	MP4 is a video format, but can also be used for audio. MP4 video is the upcoming video format on the internet. This leads to automatic support for MP4 audio by all browsers.

Only MP3, WAV, and Ogg audio are supported by the HTML5 standard.

Playing Videos in HTML

Before HTML5, a video could only be played in a browser with a plug-in (like flash).

The HTML5 <video> element specifies a standard way to embed a video in a web page.

Browser Support

The numbers in the table specify the first browser version that fully supports the<video> element.

The HTML <video> Element

To show a video in HTML, use the <video> element:

Example

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

How it Works

The controls attribute adds video controls, like play, pause, and volume.

It is a good idea to always include width and height attributes. If height and width are not set, the page might flicker while the video loads.

The <source> element allows you to specify alternative video files which the browser may choose from. The browser will use the first recognized format.

The text between the <video> and </video> tags will only be displayed in browsers that do not support the <video> element.

HTML <video> Autoplay

To start a video automatically use the autoplay attribute:

Example

```
<video width="320" height="240" autoplay>
  <source src="movie.mp4" type="video/mp4">
```

```
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

HTML5 Audio

Audio on the Web

Before HTML5, audio files could only be played in a browser with a plug-in (like flash). The HTML5 <audio> element specifies a standard way to embed audio in a web page. The numbers in the table specify the first browser version that fully supports the<audio> element.

The HTML <audio> Element

To play an audio file in HTML, use the <audio> element:

Example

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

HTML Audio - How It Works

The controls attribute adds audio controls, like play, pause, and volume.

The <source> element allows you to specify alternative audio files which the browser may choose from. The browser will use the first recognized format.

The text between the <audio> and </audio> tags will only be displayed in browsers that do not support the <audio> element.

HTML Audio - Browser Support

In HTML5, there are 3 supported audio formats: MP3, WAV, and OGG.

The browser support for the different formats is:

Browser	MP3	WAV	OGG

Internet Explorer	YES	NO	NO
Chrome	YES	YES	YES
Firefox	YES	YES	YES
Safari	YES	YES	NO
Opera	YES	YES	YES

Playing a YouTube Video in HTML

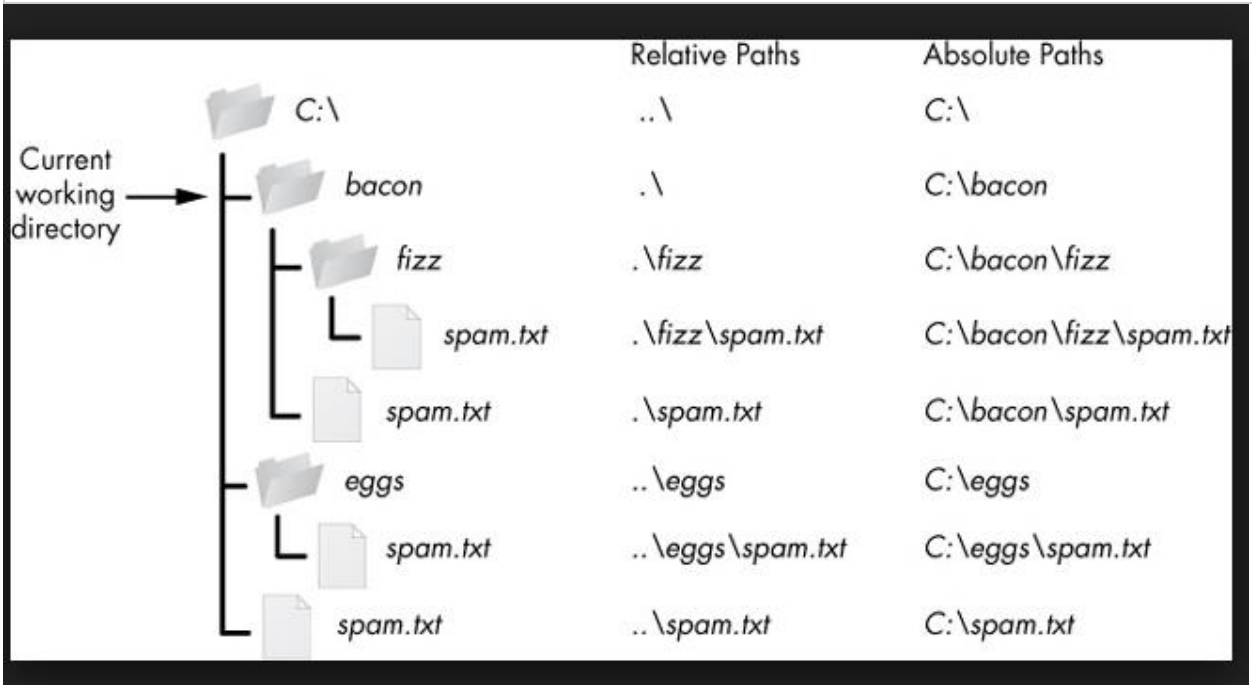
To play your video on a web page, do the following:

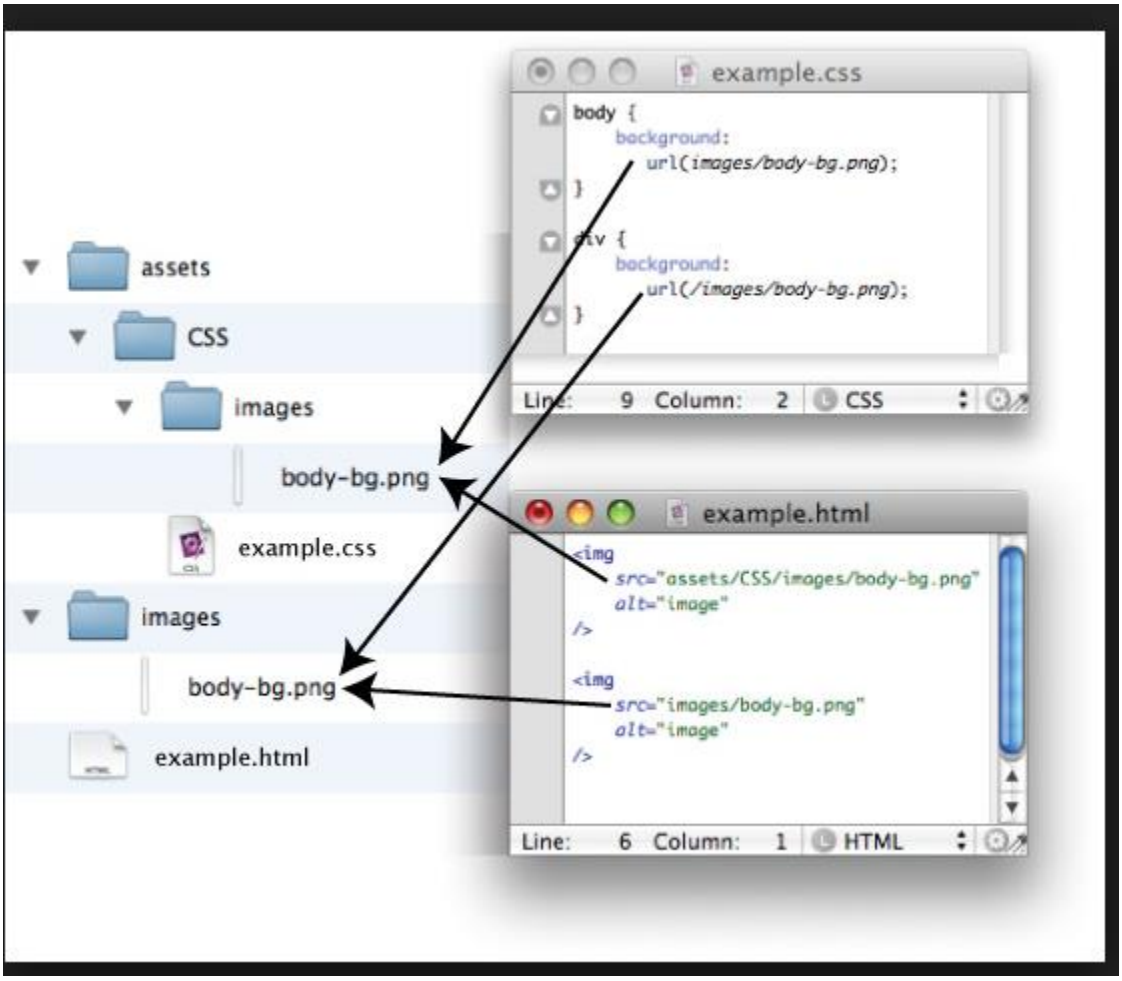
- Upload the video to YouTube
- Take a note of the video id
- Define an <iframe> element in your web page
- Let the src attribute point to the video URL
- Use the width and height attributes to specify the dimension of the player
- Add any other parameters to the URL (see below)

Example - Using iFrame (recommended)

```
<iframe width="420" height="315"
src="https://www.youtube.com/embed/tgbNymZ7vqY">
</iframe>
```

Path	Description
<img src="picture.jpg">	picture.jpg is located in the same folder as the current page
<img src="images/picture.jpg">	picture.jpg is located in the images folder in the current folder
<img src="/images/picture.jpg">	picture.jpg is located in the images folder at the root of the current web
<img src="../picture.jpg">	picture.jpg is located in the folder one level up from the current folder





3 common tags are empty: they have no end tags.

(There are 11 tags of this kind, but you only need to learn 3.)

- These tags are called "empty tags" because they don't accommodate data that would be typed between a start tag and an end tag.

Nevertheless - always close your tags:
Type a space plus forward slash just before the closing angle bracket.

- This tag creates a line break:
- This tag creates a horizontal rule: <hr />
- This tag embeds an image: <img />

- To be useful, the img tag requires additional information inside the tag. Here is a basic img tag:

```
<img src ="web-address-of-the-image" />
```

FORM type option

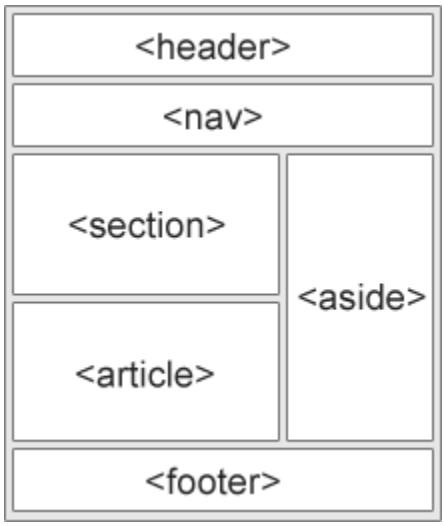
type	button checkbox date email file hidden image number password radio reset search submit text time url

HTML Layout

HTML Layout Elements

Websites often display content in multiple columns (like a magazine or newspaper).

HTML5 offers new semantic elements that define the different parts of a web page:



- <header> - Defines a header for a document or a section
- <nav> - Defines a container for navigation links
- <section> - Defines a section in a document
- <article> - Defines an independent self-contained article
- <aside> - Defines content aside from the content (like a sidebar)
- <footer> - Defines a footer for a document or a section
- <details> - Defines additional details
- <summary> - Defines a heading for the <details> element

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>CSS Template</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
* {
  box-sizing: border-box;
}

body {
  font-family: Arial, Helvetica, sans-serif;
}
```

```
/* Style the header */
header {
  background-color: #666;
  padding: 30px;
  text-align: center;
  font-size: 35px;
  color: white;
}

/* Container for flexboxes */
section {
  display: -webkit-flex;
  display: flex;
}

/* Style the navigation menu */
nav {
  -webkit-flex: 1;
  -ms-flex: 1;
  flex: 1;
  background: #ccc;
  padding: 20px;
}

/* Style the list inside the menu */
nav ul {
  list-style-type: none;
  padding: 0;
}

/* Style the content */
article {
  -webkit-flex: 3;
  -ms-flex: 3;
  flex: 3;
  background-color: #f1f1f1;
  padding: 10px;
}

/* Style the footer */
footer {
  background-color: #777;
  padding: 10px;
  text-align: center;
```

```
  color: white;
}

/* Responsive layout - makes the menu and the content (inside the section) sit on top of each other instead of next to each other */
@media (max-width: 600px) {
  section {
    -webkit-flex-direction: column;
    flex-direction: column;
  }
}
</style>
</head>
<body>

<h2>CSS Layout Flexbox</h2>
<p>In this example, we have created a header, two columns/boxes and a footer. On smaller screens, the columns will stack on top of each other.</p>
<p>Resize the browser window to see the responsive effect.</p>
<p><strong>Note:</strong> Flexbox is not supported in Internet Explorer 10 and earlier versions.</p>

<header>
  <h2>Cities</h2>
</header>

<section>
  <nav>
    <ul>
      <li><a href="#">London</a></li>
      <li><a href="#">Paris</a></li>
      <li><a href="#">Tokyo</a></li>
    </ul>
  </nav>

  <article>
    <h1>London</h1>
    <p>London is the capital city of England. It is the most populous city in the  United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>
    <p>Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.</p>
  </article>
</section>

<footer>
```

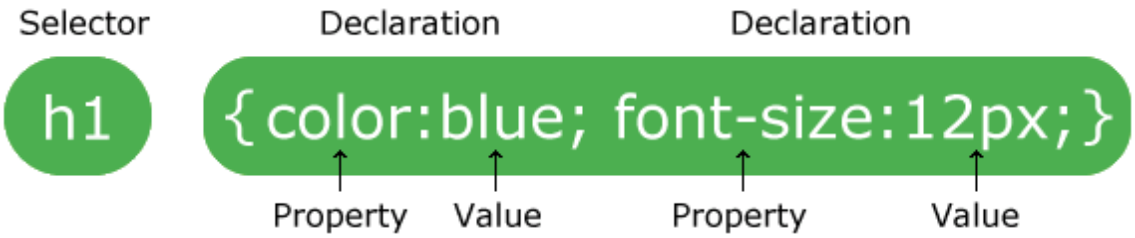
```
  <p>Footer</p>
</footer>

</body>
</html>
```

What is CSS?

- **CSS** stands for **C**ascading **S**tyle **S**heets
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
- External stylesheets are stored in **CSS files**

CSS Syntax

A CSS rule-set consists of a selector and a declaration block:



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a CSS property name and a value, separated by a colon.

A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.

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

CSS Selectors

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

The element Selector

The element selector selects elements based on the element name.

You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color):

Example

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

The id Selector

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element should be unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

The style rule below will be applied to the HTML element with id="para1":

Example

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

The class Selector

The class selector selects elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the name of the class.

In the example below, all HTML elements with class="center" will be red and center-aligned:

Example

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

CSS Comments

Comments are used to explain the code, and may help when you edit the source code at a later date.

Comments are ignored by browsers.

A CSS comment starts with /* and ends with */. Comments can also span multiple lines:

Example

```
p {
  color: red;
  /* This is a single-line comment */
  text-align: center;
}

/* This is
a multi-line
comment */
```

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

External Style Sheet

With an external style sheet, you can change the look of an entire website by changing just one file!

Each page must include a reference to the external style sheet file inside the <link> element. The <link> element goes inside the <head> section:

Example

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

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a .css extension.

Here is how the "mystyle.css" looks:

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;
  margin-left: 20px;
}
```

Internal Style Sheet

An internal style sheet may be used if one single page has a unique style.

Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

Example

```
<head>
<style>
body {
  background-color: linen;
}

h1 {
  color: maroon;
  margin-left: 40px;
}
</style>
</head>
```

Inline Styles

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

The example below shows how to change the color and the left margin of a <h1> element:

Example

```
<h1 style="color:blue;margin-left:30px;">This is a heading</h1>
```

1) **Example**

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  border: 2px solid #a1a1a1;
  padding: 10px 40px;
  background: #dddddd;
  width: 300px;
  border-radius: 25px;
}
</style>
</head>
<body>

<div>The border-radius property allows you to add rounded corners to elements.</div>

</body>
</html>
```

2)

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 300px;
  height: 100px;
  background-color: yellow;
  box-shadow: 10px 10px 5px #888888;
}
</style>
</head>
<body>
```

```
<div></div>

</body>
</html>
```

3)

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
   background: url(img_flwr.gif);
   background-size: 400px 600px;
   background-repeat: no-repeat;
   padding-top: 40px;
}
</style>
</head>
<body>

<p>
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.
</p>
<p>Original image: <img src="img_flwr.gif" alt="Flowers" width="224" height="162"></p>

</body>
</html>
```

4)

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
   background: url(img_flwr.gif);
   background-size: 100% 100%;
   background-repeat: no-repeat;
}
</style>
</head>
<body>

<div>
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation
```

ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

</body>
</html>

5)
```
<!DOCTYPE html>
<html>
<head>
<style>
body {
   background: url(img_tree.gif), url(img_flwr.gif);
   background-size: 50% 50%;
   background-repeat: no-repeat;
}
</style>
</head>
<body>

<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.</p>
<p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.</p>
<p>Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.</p>

</body>
</html>
```

6)
```
<!DOCTYPE html>
<html>
<head>
<style>
#grad1 {
   height: 200px;
   background: -webkit-linear-gradient(left top, red , blue); /* For Safari 5.1 to 6.0 */
   background: -o-linear-gradient(bottom right, red, blue); /* For Opera 11.1 to 12.0 */
   background: -moz-linear-gradient(bottom right, red, blue); /* For Firefox 3.6 to 15 */
   background: linear-gradient(to bottom right, blue  , red); /* Standard syntax (must be last) */
}
</style>
```

```
</head>
<body>

<h3>Linear Gradient - Diagonal</h3>
<p>This linear gradient starts at top left. It starts red, transitioning to blue:</p>

<div id="grad1"></div>

<p><strong>Note:</strong> Internet Explorer 9 and earlier versions do not support gradients.</p>

</body>
</html>
```

Linear Gradient - Top to Bottom (this is default)
Linear Gradient - Left to Right
Linear Gradient - Diagonal

Example
A linear gradient from top to bottom:

```
#grad {
  background: -webkit-linear-gradient(red, blue); /* For Safari 5.1 to 6.0 */
  background: -o-linear-gradient(red, blue); /* For Opera 11.1 to 12.0 */
  background: -moz-linear-gradient(red, blue); /* For Firefox 3.6 to 15 */
  background: linear-gradient(red, blue); /* Standard syntax */
}
```

Linear Gradient - Left to Right
The following example shows a linear gradient that starts from the left. It starts red, transitioning to blue:
Example
A linear gradient from left to right:

```
#grad {
  background: -webkit-linear-gradient(left, red , blue); /* For Safari 5.1 to 6.0 */
  background: -o-linear-gradient(right, red, blue); /* For Opera 11.1 to 12.0 */
  background: -moz-linear-gradient(right, red, blue); /* For Firefox 3.6 to 15 */
  background: linear-gradient(to right, red , blue); /* Standard syntax */
}
```

7)TEXT SHADOW EFFECT

```
<!DOCTYPE html>
<html>
<head>
<style>
h1 {
```

```
   text-shadow: 5px 5px 5px #FF0000;
}
</style>
</head>
<body>

<h1>Text-shadow effect!</h1>

<p><b>Note:</b> Internet Explorer 9 and earlier versions, does not support the text-shadow property.</p>

</body>
</html>
```

8)CSS3 WORD WRAPPING

```
<!DOCTYPE html>
<html>
<head>
<style>
p.test {
   width: 11em;
   border: 1px solid #000000;
   word-wrap: break-word;
}
</style>
</head>
<body>

<p class="test"> This paragraph contains a very long word: thisisaveryveryveryveryveryverylongword. thisisaveryveryveryveryveryverylongword.The long word will break and wrap to the next line.</p>

</body>
</html>
```

CSS Comments

Comments are used to explain the code, and may help when you edit the source code at a later date.

Comments are ignored by browsers.

A CSS comment starts with `/*` and ends with `*/`:

Example

```
/* This is a single-line comment */
p {
  color: red;
}
```

You can add comments wherever you want in the code:

Example

```
p {
  color: red;  /* Set text color to red */
}
```

Comments can also span multiple lines:

Example

```
/* This is
a multi-line
comment */

p {
  color: red;
}
```

CSS Colors

Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

CSS Color Names

In CSS, a color can be specified by using a color name:

LightGray

CSS/HTML support 140 standard color names.

CSS Background Color

You can set the background color for HTML elements:

Hello World

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Example

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

CSS Text Color

You can set the color of text:

Hello World

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

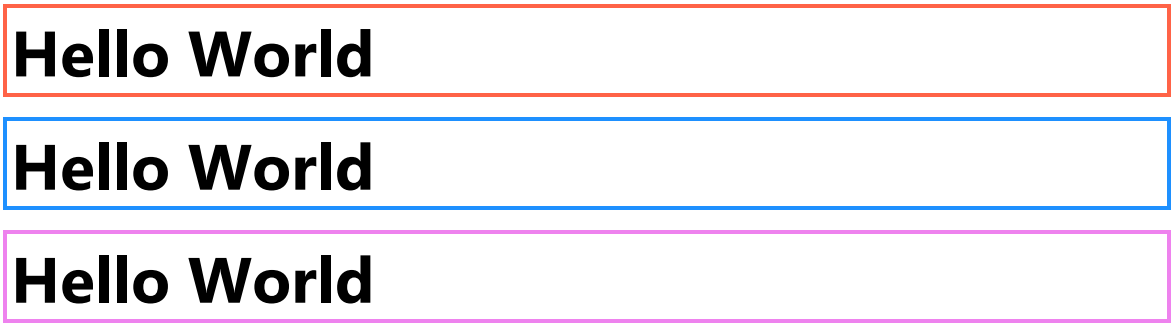
Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Example

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Lorem ipsum...</p>
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

CSS Border Color

You can set the color of borders:



Example

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
<h1 style="border:2px solid DodgerBlue;">Hello World</h1>
<h1 style="border:2px solid Violet;">Hello World</h1>
```

CSS Color Values

In CSS, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

Same as color name "Tomato":

```
rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)
```

Same as color name "Tomato", but 50% transparent:

Example

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>
```

CSS Backgrounds

The CSS background properties are used to define the background effects for elements.

In these chapters, you will learn about the following CSS background properties:

- background-color
- background-image
- background-repeat
- background-attachment
- background-position

CSS background-color

The `background-color` property specifies the background color of an element.

Example

The background color of a page is set like this:

```
body {
  background-color: lightblue;
}
```

With CSS, a color is most often specified by:

- a valid color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

Look at CSS Color Values for a complete list of possible color values.

Example

Here, the <h1>, <p>, and elements will have different background colors:

```
h1 {
  background-color: green;
}

div {
  background-color: lightblue;
}

p {
  background-color: yellow;
}
```

CSS Borders

CSS Border Properties

The CSS `border` properties allow you to specify the style, width, and color of an element's border.

CSS Border Style

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

The following values are allowed:

- `dotted` - Defines a dotted border
- `dashed` - Defines a dashed border
- `solid` - Defines a solid border
- `double` - Defines a double border
- `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
- `none` - Defines no border
- `hidden` - Defines a hidden border

The `border-style` property can have from one to four values (for the top border, right border, bottom border, and the left border).

Example

Demonstration of the different border styles:

```
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.none {border-style: none;}
p.hidden {border-style: hidden;}
p.mix {border-style: dotted dashed solid double;}
```

Result:

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

A hidden border.

A mixed border.

CSS Margins

This element has a margin of 70px.

CSS Margins

The CSS `margin` properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

Margin - Individual Sides

CSS has properties for specifying the margin for each side of an element:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

All the margin properties can have the following values:

- auto - the browser calculates the margin
- *length* - specifies a margin in px, pt, cm, etc.
- *%* - specifies a margin in % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element

Tip: Negative values are allowed.

Example

Set different margins for all four sides of a <p> element:

```
p {
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}
```

Margin - Shorthand Property

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

The `margin` property is a shorthand property for the following individual margin properties:

- `margin-top`

- `margin-right`
- `margin-bottom`
- `margin-left`

So, here is how it works:

If the `margin` property has four values:

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

Example

Use the margin shorthand property with four values:

```
p {
  margin: 25px 50px 75px 100px;
}
```

If the `margin` property has three values:

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

Example

Use the margin shorthand property with three values:

```
p {
  margin: 25px 50px 75px;
}
```

If the `margin` property has two values:

- **margin: 25px 50px;**
  - top and bottom margins are 25px
  - right and left margins are 50px

Example

Use the margin shorthand property with two values:

```
p {
  margin: 25px 50px;
}
```

If the `margin` property has one value:

- **margin: 25px;**
  - all four margins are 25px

Example

Use the margin shorthand property with one value:

```
p {
  margin: 25px;
}
```

The auto Value

You can set the margin property to `auto` to horizontally center the element within its container.

The element will then take up the specified width, and the remaining space will be split equally between the left and right margins.

Example

Use margin: auto:

```
div {
  width: 300px;
  margin: auto;
  border: 1px solid red;
}
```

The inherit Value

This example lets the left margin of the <p class="ex1"> element be inherited from the parent element ( ):

Example

Use of the inherit value:

```
div {
  border: 1px solid red;
  margin-left: 100px;
}

p.ex1 {
  margin-left: inherit;
}
```

Margin Collapse

Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.

This does not happen on left and right margins! Only top and bottom margins!

Look at the following example:

Example

Demonstration of margin collapse:

```
h1 {
  margin: 0 0 50px 0;
}

h2 {
  margin: 20px 0 0 0;
}
```

In the example above, the <h1> element has a bottom margin of 50px and the <h2> element has a top margin set to 20px.

Common sense would seem to suggest that the vertical margin between the <h1> and the <h2> would be a total of 70px (50px + 20px). But due to margin collapse, the actual margin ends up being 50px.

All CSS Margin Properties

Property	Description

margin	A shorthand property for setting the margin properties in one declaration
margin-bottom	Sets the bottom margin of an element
margin-left	Sets the left margin of an element
margin-right	Sets the right margin of an element
margin-top	Sets the top margin of an element

CSS Padding

The CSS `padding` properties are used to generate space around an element's content, inside of any defined borders.

With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).

Padding - Individual Sides

CSS has properties for specifying the padding for each side of an element:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

All the padding properties can have the following values:

- *length* - specifies a padding in px, pt, cm, etc.
- *%* - specifies a padding in % of the width of the containing element
- inherit - specifies that the padding should be inherited from the parent element

Note: Negative values are not allowed.

Example

Set different padding for all four sides of a element:

```
div {
  padding-top: 50px;
  padding-right: 30px;
  padding-bottom: 50px;
  padding-left: 80px;
}
```

Padding - Shorthand Property

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

The `padding` property is a shorthand property for the following individual padding properties:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

So, here is how it works:

If the `padding` property has four values:

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

Example

Use the padding shorthand property with four values:

```
div {
  padding: 25px 50px 75px 100px;
}
```

If the `padding` property has three values:

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

Example

Use the padding shorthand property with three values:

```
div {
  padding: 25px 50px 75px;
}
```

If the `padding` property has two values:

- **padding: 25px 50px;**
  - top and bottom paddings are 25px
  - right and left paddings are 50px

Example

Use the padding shorthand property with two values:

```
div {
  padding: 25px 50px;
}
```

If the `padding` property has one value:

- **padding: 25px;**
  - all four paddings are 25px

Example

Use the padding shorthand property with one value:

```
div {
  padding: 25px;
}
```

Padding and Element Width

The CSS `width` property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element (the box model).

So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result.

Example

Here, the element is given a width of 300px. However, the actual width of the element will be 350px (300px + 25px of left padding + 25px of right padding):

```
div {
  width: 300px;
  padding: 25px;
}
```

To keep the width at 300px, no matter the amount of padding, you can use the `box-sizing` property. This causes the element to maintain its width; if you increase the padding, the available content space will decrease.

Example

Use the box-sizing property to keep the width at 300px, no matter the amount of padding:

```
div {
  width: 300px;
  padding: 25px;
  box-sizing: border-box;
}
```

All CSS Padding Properties

Property	Description
padding	A shorthand property for setting all the padding properties in one declaration
padding-bottom	Sets the bottom padding of an element
padding-left	Sets the left padding of an element
padding-right	Sets the right padding of an element

padding-top	Sets the top padding of an element

CSS Setting height and width

The `height` and `width` properties are used to set the height and width of an element.

The height and width properties do not include padding, borders, or margins. It sets the height/width of the area inside the padding, border, and margin of the element.

CSS height/width Values

The `height` and `width` properties may have the following values:

- `auto` - This is default. The browser calculates the height and width
- `length` - Defines the height/width in px, cm etc.
- `%` - Defines the height/width in percent of the containing block
- `initial` - Sets the height/width to its default value
- `inherit` - The height/width will be inherited from its parent value

CSS height/width Examples

This element has a height of 200 pixels and a width of 50%

Example

Set the height and width of a element:

```
div {
  height: 200px;
  width: 50%;
  background-color: powderblue;
}
```

This element has a height of 100 pixels and a width of 500 pixels.

Example

Set the height and width of another element:

```
div {
  height: 100px;
  width: 500px;
  background-color: powderblue;
}
```

Note: Remember that the `height` and `width` properties do not include padding, borders, or margins! They set the height/width of the area inside the padding, border, and margin of the element!

Setting max-width

The `max-width` property is used to set the maximum width of an element.

The `max-width` can be specified in *length values*, like px, cm, etc., or in percent (%) of the containing block, or set to none (this is default. Means that there is no maximum width).

The problem with the `<div>` above occurs when the browser window is smaller than the width of the element (500px). The browser then adds a horizontal scrollbar to the page.

Using `max-width` instead, in this situation, will improve the browser's handling of small windows.

Tip: Drag the browser window to smaller than 500px wide, to see the difference between the two divs!

This element has a height of 100 pixels and a max-width of 500 pixels.

Note: The value of the `max-width` property overrides `width`.

Example

This element has a height of 100 pixels and a max-width of 500 pixels:

```
div {
  max-width: 500px;
  height: 100px;
  background-color: powderblue;
}
```

All CSS Dimension Properties

Property	Description
height	Sets the height of an element
max-height	Sets the maximum height of an element
max-width	Sets the maximum width of an element
min-height	Sets the minimum height of an element
min-width	Sets the minimum width of an element

width	Sets the width of an element

The CSS Box Model

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

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:

Explanation of the different parts:

- **Content** - The content of the box, where text and images appear
- **Padding** - Clears an area around the content. The padding is transparent
- **Border** - A border that goes around the padding and content
- **Margin** - Clears an area outside the border. The margin is transparent

The box model allows us to add a border around elements, and to define space between elements.

Example

Demonstration of the box model:

```
div {
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
```

Width and Height of an Element

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

Important: When you set the width and height properties of an element with CSS, you just set the width and height of the **content area**. To calculate the full size of an element, you must also add padding, borders and margins.

Example

This element will have a total width of 350px:

```
div {
  width: 320px;
  padding: 10px;
  border: 5px solid gray;
  margin: 0;
}
```

Here is the calculation:

320px (width)
+ 20px (left + right padding)
+ 10px (left + right border)
+ 0px (left + right margin)
= 350px

The total width of an element should be calculated like this:

Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

---

CSS Outline Style

The `outline-style` property specifies the style of the outline, and can have one of the following values:

- `dotted` - Defines a dotted outline
- `dashed` - Defines a dashed outline
- `solid` - Defines a solid outline
- `double` - Defines a double outline
- `groove` - Defines a 3D grooved outline
- `ridge` - Defines a 3D ridged outline
- `inset` - Defines a 3D inset outline
- `outset` - Defines a 3D outset outline
- `none` - Defines no outline
- `hidden` - Defines a hidden outline

The following example shows the different `outline-style` values:

Example

Demonstration of the different outline styles:

```
p.dotted {outline-style: dotted;}
p.dashed {outline-style: dashed;}
p.solid {outline-style: solid;}
p.double {outline-style: double;}
p.groove {outline-style: groove;}
p.ridge {outline-style: ridge;}
p.inset {outline-style: inset;}
p.outset {outline-style: outset;}
```

Result:

A dotted outline.

A dashed outline.

A solid outline.

A double outline.

A groove outline. The effect depends on the outline-color value.

A ridge outline. The effect depends on the outline-color value.

An inset outline. The effect depends on the outline-color value.

An outset outline. The effect depends on the outline-color value.

CSS Text

TEXT FORMATTING

This text is styled with some of the text formatting properties. The heading uses the text-align, text-transform, and color properties. The paragraph is indented, aligned, and the space between characters is specified. The underline is removed from this colored "Try it Yourself" link.

Text Color

The `color` property is used to set the color of the text. The color is specified by:

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

Look at CSS Color Values for a complete list of possible color values.

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

Example

```
body {
  color: blue;
}

h1 {
  color: green;
}
```

Note: For W3C compliant CSS: If you define the `color` property, you must also define the `background-color`.

Text Alignment

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

A text can be left or right aligned, centered, or justified.

The following example shows center aligned, and left and right aligned text (left alignment is default if text direction is left-to-right, and right alignment is default if text direction is right-to-left):

Example

```
h1 {
  text-align: center;
}

h2 {
  text-align: left;
}

h3 {
```

```
  text-align: right;
}
```

When the `text-align` property 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

```
div {
  text-align: justify;
}
```

Text Decoration

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

The value `text-decoration: none;` is often used to remove underlines from links:

Example

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

The other `text-decoration` values are used to decorate text:

Example

```
h1 {
  text-decoration: overline;
}

h2 {
  text-decoration: line-through;
}
```

```
h3 {
  text-decoration: underline;
}
```

Note: It is not recommended to underline text that is not a link, as this often confuses the reader.

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-indent` property is used to specify the indentation of the first line of a text:

Example

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

Letter Spacing

The `letter-spacing` property is used to specify the space between the characters in a text.

The following example demonstrates how to increase or decrease the space between characters:

Example

```
h1 {
  letter-spacing: 3px;
}

h2 {
  letter-spacing: -3px;
}
```

Line Height

The `line-height` property is used to specify the space between lines:

Example

```
p.small {
  line-height: 0.8;
}

p.big {
  line-height: 1.8;
}
```

Text Direction

The `direction` and `unicode-bidi` properties can be used to change the text direction of an element:

Example

```
p {
  direction: rtl;
  unicode-bidi: bidi-override;
}
```

Word Spacing

The `word-spacing` property is used to specify the space between the words in a text.

The following example demonstrates how to increase or decrease the space between words:

Example

```
h1 {
  word-spacing: 10px;
}

h2 {
  word-spacing: -5px;
}
```

Text Shadow

The `text-shadow` property adds shadow to text.

The following example specifies the position of the horizontal shadow (3px), the position of the vertical shadow (2px) and the color of the shadow (red):

Example

```
h1 {
  text-shadow: 3px 2px red;
}
```

All CSS Text Properties

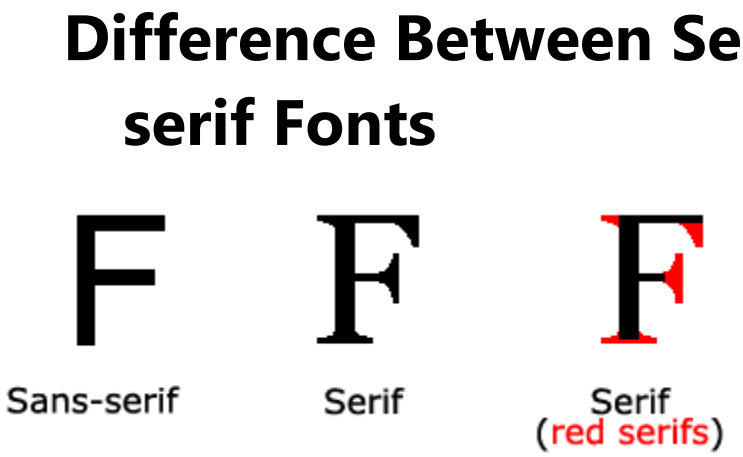
Property	Description
color	Sets the color of text
direction	Specifies the text direction/writing direction
letter-spacing	Increases or decreases the space between characters in a text
line-height	Sets the line height

text-align	Specifies the horizontal alignment of text
text-decoration	Specifies the decoration added to text
text-indent	Specifies the indentation of the first line in a text-block
text-shadow	Specifies the shadow effect added to text
text-transform	Controls the capitalization of text
text-overflow	Specifies how overflowed content that is not displayed should be signaled to the user
unicode-bidi	Used together with the direction property to set or return whether the text should be overridden to support multiple languages in the same document
vertical-align	Sets the vertical alignment of an element

white-space	Specifies how white-space inside an element is handled
word-spacing	Increases or decreases the space between words in a text

CSS Fonts

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



CSS Font Families

In CSS, there are 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 have the same width

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

Font Family

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

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

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: "Times New Roman".

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

Example

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

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

Set Font Size With Pixels

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

Example

```css
h1 {
  font-size: 40px;
}

h2 {
  font-size: 30px;
}

p {
  font-size: 14px;
}
```

Tip: If you use pixels, you can still use the zoom tool to resize the entire page.

Set Font Size With Em

To allow users to resize the text (in the browser menu), 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

```css
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 older versions of IE. The text becomes larger than it should when made larger, and smaller than it should when made smaller.

Use 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;
}
```

Our code now works great! It shows the same text size in all browsers, and allows all browsers to zoom or resize the text!

Font Weight

The `font-weight` property specifies the weight of a font:

Example

```
p.normal {
  font-weight: normal;
}

p.thick {
  font-weight: bold;
}
```

Responsive Font Size

The text size can be set with a `vw` unit, which means the "viewport width".

That way the text size will follow the size of the browser window:

Hello World

Resize the browser window to see how the font size scales.

Example

```
<h1 style="font-size:10vw">Hello World</h1>
```

Viewport is the browser window size. 1vw = 1% of viewport width. If the viewport is 50cm wide, 1vw is 0.5cm.

Font Variant

The `font-variant` property specifies whether or not a text should be displayed in a small-caps font.

In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text.

Example

```
p.normal {
  font-variant: normal;
}

p.small {
  font-variant: small-caps;
}
```

All CSS Font Properties

Property	Description
font	Sets all the font properties in one declaration
font-family	Specifies the font family for text
font-size	Specifies the font size of text
font-style	Specifies the font style for text

font-variant	Specifies whether or not a text should be displayed in a small-caps font
font-weight	Specifies the weight of a font

Google Icons

To use the Google icons, add the following line inside the `<head>` section of your HTML page:

```
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons">
```

Note: No downloading or installation is required!

Example

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Mate
rial+Icons">
</head>
<body>

<i class="material-icons">cloud</i>
<i class="material-icons">favorite</i>
<i class="material-icons">attachment</i>
<i class="material-icons">computer</i>
<i class="material-icons">traffic</i>

</body>
</html>
```

CSS Links

With CSS, links can be styled in different ways.

Text Link Text Link Link Button Link Button

Styling Links

Links can be styled with any CSS property (e.g. `color`, `font-family`, `background`, etc.).

Example

```
a {
  color: hotpink;
}
```

In addition, links can be styled differently depending on what **state** they are in.

The four links states are:

- `a:link` - a normal, unvisited link
- `a:visited` - a link the user has visited
- `a:hover` - a link when the user mouses over it
- `a:active` - a link the moment it is clicked

Example

```
/* unvisited link */
a:link {
  color: red;
}

/* visited link */
```

```
a:visited {
  color: green;
}

/* mouse over link */
a:hover {
  color: hotpink;
}

/* selected link */
a:active {
  color: blue;
}
```

When setting the style for several link states, there are some order rules:

- a:hover MUST come after a:link and a:visited
- a:active MUST come after a:hover

Text Decoration

The `text-decoration` property is mostly used to remove underlines from links:

Example

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

a:visited {
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

a:active {
```

```
  text-decoration: underline;
}
```

Background Color

The `background-color` property can be used to specify a background color for links:

Example

```
a:link {
  background-color: yellow;
}

a:visited {
  background-color: cyan;
}

a:hover {
  background-color: lightgreen;
}

a:active {
  background-color: hotpink;
}
```

Link Buttons

This example demonstrates a more advanced example where we combine several CSS properties to display links as boxes/buttons:

Example

```
a:link, a:visited {
  background-color: #f44336;
  color: white;
```

```
  padding: 14px 25px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
}

a:hover, a:active {
  background-color: red;
}
```

CSS Lists

Unordered Lists:

- Coffee
- Tea
- Coca Cola

- Coffee
- Tea
- Coca Cola

Ordered Lists:

1. Coffee
2. Tea
3. Coca Cola

I. Coffee
II. Tea
III. Coca Cola

HTML Lists and CSS List Properties

In HTML, there are two main types of lists:

- unordered lists (<ul>) - the list items are marked with bullets
- ordered lists (<ol>) - the list items are marked with numbers or letters

The CSS list properties allow you to:

- Set different list item markers for ordered lists
- Set different list item markers for unordered lists
- Set an image as the list item marker
- Add background colors to lists and list items

Different List Item Markers

The `list-style-type` property specifies the type of list item marker.

The following example shows some of the available list item markers:

Example

```
ul.a {
  list-style-type: circle;
}

ul.b {
  list-style-type: square;
}

ol.c {
  list-style-type: upper-roman;
}

ol.d {
  list-style-type: lower-alpha;
}
```

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

An Image as The List Item Marker

The `list-style-image` property specifies an image as the list item marker:

Example

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

Position The List Item Markers

The `list-style-position` property specifies the position of the list-item markers (bullet points).

"list-style-position: outside;" means that the bullet points will be outside the list item. The start of each line of a list item will be aligned vertically. This is default:

- Coffee - A brewed drink prepared from roasted coffee beans...
- Tea
- Coca-cola

"list-style-position: inside;" means that the bullet points will be inside the list item. As it is part of the list item, it will be part of the text and push the text at the start:

- Coffee - A brewed drink prepared from roasted coffee beans...
- Tea
- Coca-cola

Example

```
ul.a {
  list-style-position: outside;
}

ul.b {
```

```
  list-style-position: inside;
}
```

Remove Default Settings

The `list-style-type:none` property can also be used to remove the markers/bullets. Note that the list also has default margin and padding. To remove this, add `margin:0` and `padding:0` to <ul> or <ol>:

Example

```
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

List - Shorthand property

The `list-style` property is a shorthand property. It is used to set all the list properties in one declaration:

Example

```
ul {
  list-style: square inside url("sqpurple.gif");
}
```

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

- `list-style-type` (if a list-style-image is specified, the value of this property will be displayed if the image for some reason cannot be displayed)
- `list-style-position` (specifies whether the list-item markers should appear inside or outside the content flow)
- `list-style-image` (specifies an image as the list item marker)

If one of the property values above are missing, the default value for the missing property will be inserted, if any.

Styling List With Colors

We can also style lists with colors, to make them look a little more interesting.

Anything added to the <ol> or <ul> tag, affects the entire list, while properties added to the <li> tag will affect the individual list items:

Example

```
ol {
  background: #ff9999;
  padding: 20px;
}

ul {
  background: #3399ff;
  padding: 20px;
}

ol li {
  background: #ffe5e5;
  padding: 5px;
  margin-left: 35px;
}

ul li {
  background: #cce5ff;
  margin: 5px;
}
```

Result:

1. Coffee
2. Tea
3. Coca Cola

- Coffee

- Tea
- Coca Cola

All CSS List Properties

Property	Description
list-style	Sets all the properties for a list in one declaration
list-style-image	Specifies an image as the list-item marker
list-style-position	Specifies the position of the list-item markers (bullet points)
list-style-type	Specifies the type of list-item marker

CSS Tables

The look of an HTML table can be greatly improved with CSS:

Company	Contact	Country

Alfreds Futterkiste	Maria Anders	Germany
Berglunds snabbköp	Christina Berglund	Sweden
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Königlich Essen	Philip Cramer	Germany
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

Table Borders

To specify table borders in CSS, use the `border` property.

The example below specifies a black border for <table>, <th>, and <td> elements:

Example

```
table, th, td {
  border: 1px solid black;
}
```

Notice that the table in the example above has double borders. This is because both the table and the <th> and <td> elements have separate borders.

Collapse Table Borders

The `border-collapse` property sets whether the table borders should be collapsed into a single border:

Example

```
table {
  border-collapse: collapse;
}

table, th, td {
  border: 1px solid black;
}
```

If you only want a border around the table, only specify the `border` property for <table>:

Example

```
table {
  border: 1px solid black;
}
```

Table Width and Height

Width and height of a table are defined by the `width` and `height` properties.

The example below sets the width of the table to 100%, and the height of the <th> elements to 50px:

Example

```
table {
  width: 100%;
}

th {
  height: 50px;
}
```

Horizontal Alignment

The `text-align` property sets the horizontal alignment (like left, right, or center) of the content in <th> or <td>.

By default, the content of <th> elements are center-aligned and the content of <td> elements are left-aligned.

The following example left-aligns the text in <th> elements:

Example

```
th {
  text-align: left;
}
```

Vertical Alignment

The `vertical-align` property sets the vertical alignment (like top, bottom, or middle) of the content in <th> or <td>.

By default, the vertical alignment of the content in a table is middle (for both <th> and <td> elements).

The following example sets the vertical text alignment to bottom for <td> elements:

Example

```
td {
  height: 50px;
  vertical-align: bottom;
}
```

Table Padding

To control the space between the border and the content in a table, use the `padding` property on <td> and <th> elements:

Example

```
th, td {
  padding: 15px;
  text-align: left;
}
```

Horizontal Dividers

First Name	Last Name	Savings
Peter	Griffin	$100
Lois	Griffin	$150
Joe	Swanson	$300

Add the `border-bottom` property to <th> and <td> for horizontal dividers:

Example

```
th, td {
  border-bottom: 1px solid #ddd;
}
```

Hoverable Table

Use the `:hover` selector on <tr> to highlight table rows on mouse over:

First Name	Last Name	Savings
Peter	Griffin	$100
Lois	Griffin	$150
Joe	Swanson	$300

Example

```
tr:hover {background-color: #f5f5f5;}
```

Striped Tables

First Name	Last Name	Savings

Peter	Griffin	$100
Lois	Griffin	$150
Joe	Swanson	$300

For zebra-striped tables, use the `nth-child()` selector and add a `background-color` to all even (or odd) table rows:

Example

```
tr:nth-child(even) {background-color: #f2f2f2;}
```

Table Color

The example below specifies the background color and text color of <th> elements:

First Name	Last Name	Savings
Peter	Griffin	$100

Lois	Griffin	$150
Joe	Swanson	$300

Example

```
th {
  background-color: #4CAF50;
  color: white;
}
```

Responsive Table

A responsive table will display a horizontal scroll bar if the screen is too small to display the full content:

First Name	Last Name	Points	Points	Points	Points	Points	Points	Points	Points	Points	Points	Points	Points
Jill	Smith	50	50	50	50	50	50	50	50	50	50	50	50
Eve	Jackson	94	94	94	94	94	94	94	94	94	94	94	94
Adam	Johnson	67	67	67	67	67	67	67	67	67	67	67	67

Add a container element (like) with `overflow-x:auto` around the <table> element to make it responsive:

Example

```
<div style="overflow-x:auto;">

<table>
... table content ...
</table>

</div>
```

Try it Yourself »

Note: In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set).

CSS Table Properties

Property	Description
border	Sets all the border properties in one declaration
border-collapse	Specifies whether or not table borders should be collapsed
border-spacing	Specifies the distance between the borders of adjacent cells

caption-side	Specifies the placement of a table caption
empty-cells	Specifies whether or not to display borders and background on empty cells in a table
table-layout	Sets the layout algorithm to be used for a table

CSS Layout - The position Property

The `position` property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

The position Property

The `position` property specifies the type of positioning method used for an element.

There are five different position values:

- `static`
- `relative`
- `fixed`
- `absolute`
- `sticky`

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the `position` property is set first. They also work differently depending on the position value.

---

position: static;

HTML elements are positioned static by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with `position: static;` is not positioned in any special way; it is always positioned according to the normal flow of the page:

This element has position: static;

Here is the CSS that is used:

Example

```
div.static {
  position: static;
  border: 3px solid #73AD21;
}
```

---

position: relative;

An element with `position: relative;` is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

This element has position: relative;

Here is the CSS that is used:

Example

```
div.relative {
  position: relative;
```

```
  left: 30px;
  border: 3px solid #73AD21;
}
```

position: fixed;

An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

Notice the fixed element in the lower-right corner of the page. Here is the CSS that is used:

Example

```
div.fixed {
  position: fixed;
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
```

This element has `position: fixed;`

---

position: absolute;

An element with `position: absolute;` is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: A "positioned" element is one whose position is anything except `static`.

Here is a simple example:

This element has position: relative;

This element has position: absolute;

Here is the CSS that is used:

Example

```
div.relative {
  position: relative;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}

div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #73AD21;
}
```

---

position: sticky;

An element with `position: sticky;` is positioned based on the user's scroll position.

A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

Note: Internet Explorer, Edge 15 and earlier versions do not support sticky positioning. Safari requires a -webkit- prefix (see example below). You must also specify at least one of `top`, `right`, `bottom` or `left` for sticky positioning to work.

In this example, the sticky element sticks to the top of the page (`top: 0`), when you reach its scroll position.

Example

```
div.sticky {
  position: -webkit-sticky; /* Safari */
  position: sticky;
  top: 0;
  background-color: green;
  border: 2px solid #4CAF50;
}
```

Overlapping Elements

When elements are positioned, they can overlap other elements.

The `z-index` property specifies the stack order of an element (which element should be placed in front of, or behind, the others).

An element can have a positive or negative stack order:

This is a heading

Because the image has a z-index of -1, it will be placed behind the text.

Example

```
img {
  position: absolute;
```

```
  left: 0px;
  top: 0px;
  z-index: -1;
}
```

An element with greater stack order is always in front of an element with a lower stack order.

Note: If two positioned elements overlap without a `z-index` specified, the element positioned last in the HTML code will be shown on top.

Positioning Text In an Image

How to position text over an image:

Example

Bottom Left

Top Left

Top Right

Bottom Right

Centered

Try it Yourself:

Top Left » Top Right » Bottom Left » Bottom Right » Centered »

.

All CSS Positioning Properties

Property	Description
bottom	Sets the bottom margin edge for a positioned box
clip	Clips an absolutely positioned element
left	Sets the left margin edge for a positioned box
position	Specifies the type of positioning for an element
right	Sets the right margin edge for a positioned box
top	Sets the top margin edge for a positioned box
z-index	Sets the stack order of an element

CSS Layout - Overflow

The CSS `overflow` property controls what happens to content that is too big to fit into an area.

This text is really long and the height of its container is only 100 pixels. Therefore, a scrollbar is added to help the reader to scroll the content. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem.

Try it Yourself »

CSS Overflow

The `overflow` property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

The `overflow` property has the following values:

- `visible` - Default. The overflow is not clipped. The content renders outside the element's box
- `hidden` - The overflow is clipped, and the rest of the content will be invisible
- `scroll` - The overflow is clipped, and a scrollbar is added to see the rest of the content
- `auto` - Similar to `scroll`, but it adds scrollbars only when necessary

Note: The `overflow` property only works for block elements with a specified height.

Note: In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set).

---

overflow: visible

By default, the overflow is `visible`, meaning that it is not clipped and it renders outside the element's box:

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

```
div {
  width: 200px;
  height: 50px;
  background-color: #eee;
  overflow: visible;
}
```

overflow: hidden

With the `hidden` value, the overflow is clipped, and the rest of the content is hidden:

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

```
div {
  overflow: hidden;
}
```

overflow: scroll

Setting the value to `scroll`, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

```
div {
  overflow: scroll;
}
```

Try it Yourself »

---

overflow: auto

The `auto` value is similar to `scroll`, but it adds scrollbars only when necessary:

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

```
div {
  overflow: auto;
}
```

overflow-x and overflow-y

The `overflow-x` and `overflow-y` properties specifies whether to change the overflow of content just horizontally or vertically (or both):

`overflow-x` specifies what to do with the left/right edges of the content.
`overflow-y` specifies what to do with the top/bottom edges of the content.

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

Example

```css
div {
  overflow-x: hidden; /* Hide horizontal scrollbar */
  overflow-y: scroll; /* Add vertical scrollbar */
}
```

All CSS Overflow Properties

Property	Description
overflow	Specifies what happens if content overflows an element's box
overflow-x	Specifies what to do with the left/right edges of the content if it overflows the element's content area
overflow-y	Specifies what to do with the top/bottom edges of the content if it overflows the element's content area

CSS Layout - float and clear

The CSS `float` property specifies how an element should float.

The CSS `clear` property specifies what elements can float beside the cleared element and on which side.

The float Property

The `float` property is used for positioning and formatting content e.g. let an image float left to the text in a container.

The `float` property can have one of the following values:

- left - The element floats to the left of its container
- right - The element floats to the right of its container
- none - The element does not float (will be displayed just where it occurs in the text). This is default
- inherit - The element inherits the float value of its parent

In its simplest use, the `float` property can be used to wrap text around images.

Example - float: right;

The following example specifies that an image should float to the **right** in a text:

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet.

Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

Example

```
img {
  float: right;
}
```

Example - float: left;

The following example specifies that an image should float to the **left** in a text:

 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

Example

```
img {
  float: left;
}
```

Example - No float

In the following example the image will be displayed just where it occurs in the text (float: none;):


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio, vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est, ultrices nec congue eget, auctor vitae massa. Fusce luctus vestibulum augue ut aliquet. Mauris ante ligula, facilisis sed ornare eu, lobortis in odio. Praesent convallis urna a lacus interdum ut hendrerit risus congue. Nunc sagittis dictum nisi, sed ullamcorper ipsum dignissim ac...

Example

```
img {
  float: none;
}
```

Try it Yourself »

The clear Property

The `clear` property specifies what elements can float beside the cleared element and on which side.

The `clear` property can have one of the following values:

- none - Allows floating elements on both sides. This is default
- left - No floating elements allowed on the left side
- right- No floating elements allowed on the right side
- both - No floating elements allowed on either the left or the right side
- inherit - The element inherits the clear value of its parent

The most common way to use the `clear` property is after you have used a `float` property on an element.

When clearing floats, you should match the clear to the float: If an element is floated to the left, then you should clear to the left. Your floated element will continue to float, but the cleared element will appear below it on the web page.

The following example clears the float to the left. Means that no floating elements are allowed on the left side (of the div):

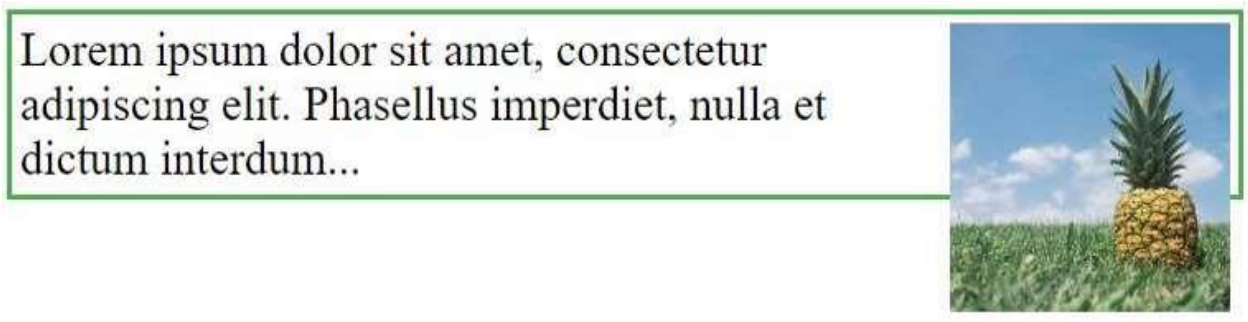
Example

```
div {
  clear: left;
}
```

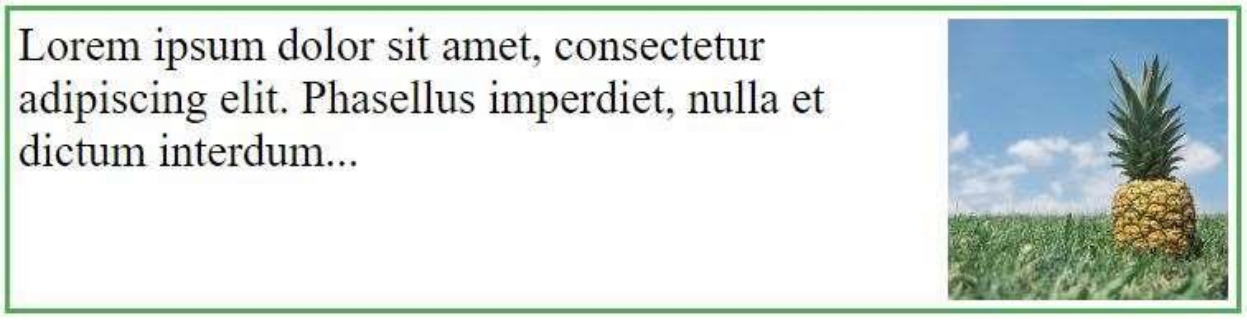
The clearfix Hack

If an element is taller than the element containing it, and it is floated, it will "overflow" outside of its container:

Without Clearfix



With Clearfix



Then we can add `overflow: auto;` to the containing element to fix this problem:

Example

```
.clearfix {
  overflow: auto;
}
```

The `overflow: auto` clearfix works well as long as you are able to keep control of your margins and padding (else you might see scrollbars). The **new, modern clearfix hack** however, is safer to use, and the following code is used for most webpages:

Example

```
.clearfix::after {
  content: "";
  clear: both;
  display: table;
}
```

You will learn more about the `::after` pseudo-element in a later chapter.

Grid of Boxes / Equal Width Boxes

Box 1

Box 2

Box 1

Box 2

Box 3

With the `float` property, it is easy to float boxes of content side by side:

Example

```
* {
  box-sizing: border-box;
}

.box {
  float: left;
  width: 33.33%; /* three boxes (use 25% for four, and 50% for two, etc)
*/
  padding: 50px; /* if you want space between the images */
}
```

Try it Yourself »

What is box-sizing?

You can easily create three floating boxes side by side. However, when you add something that enlarges the width of each box (e.g. padding or borders), the box will break. The `box-sizing` property allows us to include the padding and border in the box's total width (and height), making sure that the padding stays inside of the box and that it does not break.

You can read more about the box-sizing property in our CSS Box Sizing Chapter.

Equal Height Boxes

In the previous example, you learned how to float boxes side by side with an equal width. However, it is not easy to create floating boxes with equal heights. A quick fix however, is to set a fixed height, like in the example below:

Box 1

Some content, some content, some content

Box 2

Some content, some content, some content

Some content, some content, some content

Some content, some content, some content

Example

```
.box {
  height: 500px;
}
```

However, this is not very flexible. It is ok if you can guarantee that the boxes will always have the same amount of content in them. But many times, the content is not the same. If you try the example above on a mobile phone, you will see that the second box's content will be displayed outside of the box. This is where CSS3 Flexbox comes in handy - as it can automatically stretch boxes to be as long as the longest box:

Example

Using **Flexbox** to create flexible boxes:

Box 1 - This is some text to make sure that the content gets really tall. This is some text to make

sure that the content gets really tall. This is some text to make sure that the content gets really tall.

Box 2 - My height will follow Box 1.

The only problem with Flexbox is that it does not work in Internet Explorer 10 or earlier versions. You can read more about the Flexbox Layout Module in our CSS Flexbox Chapter.

Navigation Menu

Use `float` with a list of hyperlinks to create a horizontal menu:

Example

- Home
- News
- Contact
- About

Web Layout Example

It is also common to do entire web layouts using the `float` property:

Example

```
.header, .footer {
  background-color: grey;
  color: white;
  padding: 15px;
}

.column {
```

```
  float: left;
  padding: 15px;
}

.clearfix::after {
  content: "";
  clear: both;
  display: table;
}

.menu {
  width: 25%;
}

.content {
  width: 75%;
}
```

All CSS Float Properties

Property	Description
box-sizing	Defines how the width and height of an element are calculated: should they include padding and borders, or not
clear	Specifies what elements can float beside the cleared element and on which side
float	Specifies how an element should float

overflow	Specifies what happens if content overflows an element's box
overflow-x	Specifies what to do with the left/right edges of the content if it overflows the element's content area
overflow-y	Specifies what to do with the top/bottom edges of the content if it overflows the element's content area

CSS Layout - display: inline-block

The display: inline-block Value

Compared to `display: inline`, the major difference is that `display: inline-block` allows to set a width and height on the element.

Also, with `display: inline-block`, the top and bottom margins/paddings are respected, but with `display: inline` they are not.

Compared to `display: block`, the major difference is that `display: inline-block` does not add a line-break after the element, so the element can sit next to other elements.

The following example shows the different behavior of `display: inline`, `display: inline-block` and `display: block`:

Example

```
span.a {
  display: inline; /* the default for span */
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
```

```css
span.b {
  display: inline-block;
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}

span.c {
  display: block;
  width: 100px;
  height: 100px;
  padding: 5px;
  border: 1px solid blue;
  background-color: yellow;
}
```

Using inline-block to Create Navigation Links

One common use for `display: inline-block` is to display list items horizontally instead of vertically. The following example creates horizontal navigation links:

Example

```css
.nav {
  background-color: yellow;
  list-style-type: none;
  text-align: center;
  padding: 0;
  margin: 0;
}

.nav li {
  display: inline-block;
  font-size: 20px;
```

```
  padding: 20px;
}
```

CSS Layout - Horizontal & Vertical Align

▲▼

◧▶

Center elements horizontally and vertically

Center Align Elements

To horizontally center a block element (like), use `margin: auto;`

Setting the width of the element will prevent it from stretching out to the edges of its container.

The element will then take up the specified width, and the remaining space will be split equally between the two margins:

This div element is centered.

Example

```css
.center {
  margin: auto;
  width: 50%;
  border: 3px solid green;
  padding: 10px;
}
```

Note: Center aligning has no effect if the `width` property is not set (or set to 100%).

Center Align Text

To just center the text inside an element, use `text-align: center;`

This text is centered.

Example

```css
.center {
  text-align: center;
  border: 3px solid green;
}
```

Center an Image

To center an image, set left and right margin to `auto` and make it into a `block` element:

Example

```
img {
  display: block;
  margin-left: auto;
  margin-right: auto;
  width: 40%;
}
```

Left and Right Align - Using position

One method for aligning elements is to use `position: absolute;`:

In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since.

Example

```
.right {
  position: absolute;
  right: 0px;
  width: 300px;
  border: 3px solid #73AD21;
  padding: 10px;
}
```

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

Left and Right Align - Using float

Another method for aligning elements is to use the `float` property:

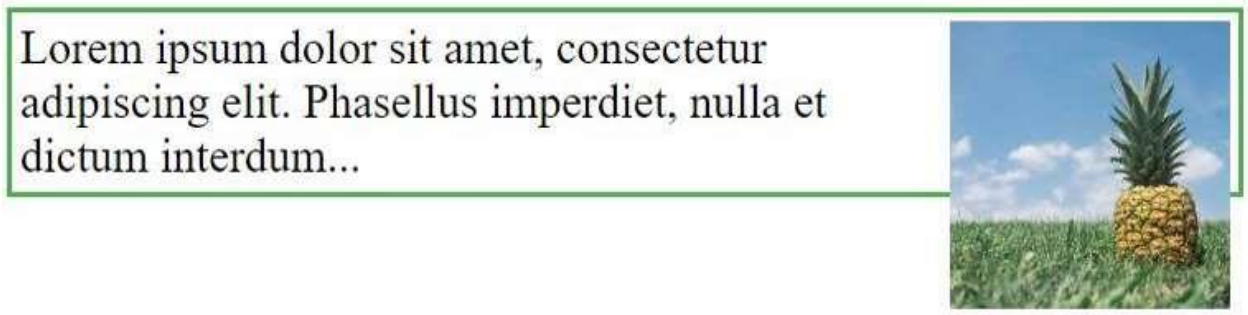
Example

```
.right {
  float: right;
  width: 300px;
  border: 3px solid #73AD21;
  padding: 10px;
}
```

Note: If an element is taller than the element containing it, and it is floated, it will overflow outside of its container. You can use the "**clearfix**" hack to fix this (see example below).

The clearfix Hack

Without Clearfix



With Clearfix

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...

Then we can add `overflow: auto;` to the containing element to fix this problem:

Example

```
.clearfix {
  overflow: auto;
}
```

Center Vertically - Using padding

There are many ways to center an element vertically in CSS. A simple solution is to use top and bottom `padding`:

I am vertically centered.

Example

```
.center {
  padding: 70px 0;
  border: 3px solid green;
}
```

To center both vertically and horizontally, use `padding` and `text-align: center`:

I am vertically and horizontally centered.

Example

```
.center {
  padding: 70px 0;
```

```
  border: 3px solid green;
  text-align: center;
}
```

Center Vertically - Using line-height

Another trick is to use the `line-height` property with a value that is equal to the `height` property.

I am vertically and horizontally centered.

Example

```
.center {
  line-height: 200px;
  height: 200px;
  border: 3px solid green;
  text-align: center;
}

/* If the text has multiple lines, add the following: */
.center p {
  line-height: 1.5;
  display: inline-block;
  vertical-align: middle;
}
```

Center Vertically - Using position & transform

If `padding` and `line-height` are not options, a third solution is to use positioning and the `transform` property:

I am vertically and horizontally centered.

Example

```
.center {
  height: 200px;
  position: relative;
  border: 3px solid green;
}

.center p {
  margin: 0;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

Tip: You will learn more about the transform property in our 2D Transforms Chapter.

JAVASCRIPT

In JavaScript you cannot use these reserved words as variables, labels, or function names:

abstract	**arguments**	**await***	**boolean**
break	byte	case	catch
char	class*	const	continue

debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

JavaScript Objects, Properties, and Methods

You should also avoid using the name of JavaScript built-in objects, properties, and methods:

Array	Date	eval	function
hasOwnProperty	Infinity	isFinite	isNaN
isPrototypeOf	length	Math	NaN
name	Number	Object	prototype
String	toString	undefined	valueOf

Java Reserved Words:JavaScript is often used together with Java. You should avoid using some Java objects and properties as JavaScript identifiers:

getClass	java	JavaArray	javaClass
JavaObject	JavaPackage		

Other Reserved Words

JavaScript can be used as the programming language in many applications.

You should also avoid using the name of HTML and Window objects and properties:

alert	all	anchor	anchors

area	assign	blur	button
checkbox	clearInterval	clearTimeout	clientInformation
close	closed	confirm	constructor
crypto	decodeURI	decodeURIComponent	defaultStatus
document	element	elements	embed
embeds	encodeURI	encodeURIComponent	escape
event	fileUpload	focus	form
forms	frame	innerHeight	innerWidth
layer	layers	link	location
mimeTypes	navigate	navigator	frames
frameRate	hidden	history	image
images	offscreenBuffering	open	opener
option	outerHeight	outerWidth	packages

pageXOffset	pageYOffset	parent	parseFloat
parseInt	password	pkcs11	plugin
prompt	propertyIsEnum	radio	reset
screenX	screenY	scroll	secure
select	self	setInterval	setTimeout
status	submit	taint	text
textarea	top	unescape	untaint
window			

HTML Event Handlers: In addition you should avoid using the name of all HTML event handlers.

Examples

onblur	onclick	Onerro	onfocus
onkeydown	onkeypress	onkeyup	onmouseover
onload	onmouseup	onmousedown	onsubmit

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using **innerHTML**.
- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.

Using innerHTML

To access an HTML element, JavaScript can use the **document.getElementById(id)** method.

The **id** attribute defines the HTML element. The **innerHTML** property defines the HTML content:

Example

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

<h1>My First Web Page</h1>
<p>My First Paragraph</p>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Changing the innerHTML property of an HTML element is a common way to display data in HTML.

Using document.write()

For testing purposes, it is convenient to use **document.write()**:

Example

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

```
<h1>My First Web Page</h1>
<p>My first paragraph.</p>
<script>
document.write(5 + 6);
</script>
</body>
</html>
```

Using document.write() after an HTML document is loaded, will **delete all existing HTML**:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button type="button" onclick="document.write(5 + 6)">Try it</button>
</body>
</html>
```

The document.write() method should only be used for testing.

Using window.alert()

You can use an alert box to display data:

Example

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

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

Using console.log()

For debugging purposes, you can use the **console.log()** method to display data.

You will learn more about debugging in a later chapter.

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

Datatypes in javascript

Variable	Explanation	Example
String	A sequence of text known as a string. To signify that the value is a string, you must enclose it in quote marks.	`var myVariable = 'Bob';`
Number	A number. Numbers don't have quotes around them.	`var myVariable = 10;`
Boolean	A True/False value. The words `true` and `false` are special keywords in JS, and don't need quotes.	`var myVariable = true;`
Array	A structure that allows you to store multiple values in one single reference.	`var myVariable = [1,'Bob','Steve',10];` Refer to each member of the array like this: `myVariable[0]`, `myVariable[1]`, etc.
Object	Basically, anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn.	`var myVariable = document.querySelector('h1');` All of the above examples too.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".
In programming, text values are called text strings.
JavaScript can handle many types of data, but for now, just think of numbers and strings.
Strings are written inside double or single quotes. Numbers are written without quotes.
If you put a number in quotes, it will be treated as a text string.

Example

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the **var** keyword:

```
var carName;
```

After the declaration, the variable has no value. (Technically it has the value of **undefined**)

To **assign** a value to the variable, use the equal sign:

```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:

```
var carName = "Volvo";
```

In the example below, we create a variable called carName and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

Example

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

<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

It's a good programming practice to declare all variables at the beginning of a script.

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with **var** and separate the variables by **comma**:

```
var person = "John Doe", carName = "Volvo", price = 200;
```

A declaration can span multiple lines:

```
var person = "John Doe",
carName = "Volvo",
price = 200;
```

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value **undefined**.

The variable carName will have the value undefined after the execution of this statement:

Example

```
var carName;
```

Re-Declaring JavaScript Variables

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

The variable carName will still have the value "Volvo" after the execution of these statements:

Example

```
var carName = "Volvo";
var carName;
```

JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +:

Example

```
var x = 5 + 2 + 3;
```

You can also add strings, but strings will be concatenated:

Example

```
var x = "John" + " " + "Doe";
```

Also try this:

Example

```
var x = "5" + 2 + 3;
```

Example 1

```
<!DOCTYPE html>
<html>
<body>
<script>
function changeImage() {
  var image = document.getElementById('myImage');
  if (image.src.match("bulbon")) {
    image.src = "pic_bulboff.gif";
  } else {
    image.src = "pic_bulbon.gif";
  }
}
</script>

<img id="myImage" onclick="changeImage()" src="pic_bulboff.gif" width="100" height="180">
```

```
<p>Click the light bulb to turn on/off the light</p>

</body>
</html>
```

Example 2

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

Example 3

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

<h1>My First Web Page</h1>

<p>My first paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
   document.write(Date());
}
```

```
</script>

</body>
</html>
```

Example 4

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

<h1>My First Web Page</h1>

<p>
Activate debugging in your browser (Chrome, IE, Firefox) with F12, and select "Console" in the debugger menu.
</p>

<script>
a = 5;
b = 6;
c = a + b;
console.log(c);
</script>

</body>
</html>
```

JavaScript variables are "containers" for storing information:
Example

```
var x = 5;
var y = 6;
var z = x + y;
```

Example 5

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

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

<script>
var x = 5;
var y = 6;
document.getElementById("demo").innerHTML = x + y;
```

```
//document.getElementById("demo").innerHTML = x;
//document.getElementById("demo").innerHTML = y;
</script>

<p>Try to experiment with the // comments.</p>

</body>
</html>
```

JAVASCRIPT DATATYPES

JavaScript Has Dynamic Types

JavaScript has dynamic types. This means that the same variable can be used as different types:

Example

```
var x;              // Now x is undefined
var x = 5;          // Now x is a Number
var x = "John";     // Now x is a String
```

JAVASCRIPT STRINGS

A string is a variable which stores a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example

```
var carName = "Volvo XC60";   // Using double quotes
var carName = 'Volvo XC60';   // Using single quotes
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

```
var answer = "It's alright";            // Single quote inside double quotes
var answer = "He is called 'Johnny'";   // Single quotes inside double quotes
var answer = 'He is called "Johnny"';   // Double quotes inside single quotes
```

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

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

<script>
var carName1 = "Volvo XC60";
var carName2 = 'Volvo XC60';
var answer1 = "It's alright";
var answer2 = "He is called 'Johnny'";
var answer3 = 'He is called "Johnny"';

document.getElementById("demo").innerHTML =
carName1 + "<br>" +
```

```
carName2 + "<br>" +
answer1 + "<br>" +
answer2 + "<br>" +
answer3;
</script>

</body>
</html>
```

JAVASCRIPT OBJECTS

JavaScript objects are written with curly braces.
Object properties are written as name: value pairs, separated by commas.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
var person = {
   firstName : "John",
   lastName  : "Doe",
   age      : 50,
   eyeColor  : "blue"
};

document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>

</body>
</html>
```

UNDEFINED AND NULL

The value of a variable with no value is **undefined**.
Variables can be emptied by setting the value to **null**.

```
<!DOCTYPE html>
<html>
<body>
<p>The value of a variable with no value is <b>undefined</b>.</p>
<p>Variables can be emptied by setting the value to <b>null</b>.</p>
<p id="demo"></p>
<script>
```

```
var person;
var car = "Volvo";
var x = null;
document.getElementById("demo").innerHTML =
person + "<br>" + car + "<br>" + x;
</script>
</body>
</html>
```

THE TYPE OF OPERATOR

You can use the JavaScript type of operator to find the type of a variable.

```
<!DOCTYPE html>
<html>
<body>
<p>The typeof operator returns the type of a variable or an expression.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  document.getElementById("demo").innerHTML =
  typeof "john" + "<br>" +
  typeof 3.14 + "<br>" +
  typeof false + "<br>" +
  typeof [1,2,3,4] + "<br>" +
  typeof {name:'john', age:34};
}
</script>

</body>
</html>
```

A JavaScript function is a block of code designed to perform a particular task.
A JavaScript function is executed when "something" invokes it (calls it).

```
<!DOCTYPE html>
<html>
<body>
<p>This example calls a function which performs a calculation, and returns the result:</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
```

```
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
</html>
```

JavaScript Function Syntax
A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.
The parentheses may include a list of parameter names: **(*parameter1, parameter2, ...*)**
The code to be executed by the function is placed inside curly brackets: **{}**

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

The code inside the function will execute when "something" invokes (calls) the function.
A function can be invoked:
When an event occurs (when a user clicks a button)
When it is invoked (called) from JavaScript code
Automatically (self invoked)

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

<p>This example calls a function to convert from Fahrenheit to Celcius:</p>
<p id="demo"></p>
<script>
function toCelcius(f) {
   return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelcius(32);
</script>
</body>
</html>
```

COMMON HTML EVENTS

```
<!DOCTYPE html>
<html>
<body>
<button onclick="this.innerHTML=Date()">The time is?</button>
</body>
</html>
```

Accessing Object Properties
You can access the object properties in two ways:

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

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

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

<h2>JavaScript Objects</h2>

<p>An object method is a function definition, stored as a property value.</p>

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

<script>
// Create an object:
var person = {
  firstName: "John",
  lastName : "Doe",
  id     : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>
</body>
</html>
```

The **this** Keyword

In a function definition, **this** refers to the "owner" of the function.

In the example above, **this** is the **person object** that "owns" the **fullName** function.

In other words, **this.firstName** means the **firstName** property of **this object**.

String Length
The length of a string (a string object) is found in the built in property **length**:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var txt="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
document.getElementById("demo").innerHTML = txt.length;
</script>
</body>
</html>
```

Finding a String in a String
The **indexOf()** method returns the position (a number) of the **first** occurrence of a specified text (string) in a string:
Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```

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

<p id="p1">Please locate where 'locate' occurs!.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
   var str = document.getElementById("p1").innerHTML;
   var pos = str.indexOf("locate");
   document.getElementById("demo").innerHTML = pos;
}
</script>

</body>
</html>
```

The **lastIndexOf()** method finds the **last** occurrence of a specified text inside a string:
Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.lastIndexOf("locate");
```

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

<p id="p1">Please locate where 'locate' occurs!.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
  var str = document.getElementById("p1").innerHTML;
  var pos = str.lastIndexOf("locate");
  document.getElementById("demo").innerHTML = pos;
}
</script>

</body>
</html>
```

Searching for String Content

The **search()** method searches a string for a specified value and returns the position of the match:

Example

```
var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
```

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

<p id="p1">Please locate where 'locate' occurs!.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
  var str = document.getElementById("p1").innerHTML;
```

```
  var pos = str.search("locate");
  document.getElementById("demo").innerHTML = pos;
}
</script>

</body>
</html>
```

Infinity

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

Example

```
var myNumber = 2;
while (myNumber != Infinity) {        // Execute until Infinity
  myNumber = myNumber * myNumber;
}
```

Division by 0 (zero) also generates Infinity:

Example

```
var x =  2 / 0;         // x will be Infinity
var y = -2 / 0;         // y will be -Infinity
```

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

<p>Infinity is returned if you calculate a number outside the largest possible number.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
  var myNumber = 2;
  var txt = "";
  while (myNumber != Infinity) {
    myNumber = myNumber * myNumber;
    txt = txt + myNumber + "<br>";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>
```

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

NaN - Not a Number
NaN is a JavaScript reserved word indicating that a value is not a number.
You can use the global JavaScript function isNaN() to find out if a value is a number.
Example

```
var x = 100 / "Apple";  // a number divided by a string is not a number
var y = 100 / "10";     // a number divided by a numeric string is a number
```

```
<!DOCTYPE html>
<html>
<body>
<p>A number divided by a string is not a number</p>
<p>A number divided by a numeric string is a number</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
   var x = 1000 / "Apple";
   var y = 1000 / "10";
   document.getElementById("demo").innerHTML = x + "<br>" + y;
}
</script>

</body>
</html>
```

Numbers Can be Objects
Normally JavaScript numbers are primitive values created from literals: **var x = 123**
But numbers can also be defined as objects with the keyword new: **var y = new Number(123)**
Example

```
var x = 123;
var y = new Number(123);
typeof x;            // returns number
typeof y;            // returns object
```

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

One common use of the Math object is to create a random number:
Example
```
Math.random();      // returns a random number
```

Math.round()
Math.round() rounds a number to the nearest integer:
Example
```
Math.round(4.7);              // returns 5
Math.round(4.4);               // returns 4
```

Math.ceil()
Math.ceil() rounds a number **up** to the nearest integer:
Example
```
Math.ceil(4.4);               // returns 5
```

Math.floor()
Math.floor() rounds a number **down** to the nearest integer:
Example
```
Math.floor(4.7);              // returns 4
```

Math Constants
JavaScript provides 8 mathematical constants that can be accessed with the Math object:
Example
```
Math.E;       // returns Euler's number
Math.PI       // returns PI
Math.SQRT2     // returns the square root of 2
Math.SQRT1_2   // returns the square root of 1/2
Math.LN2       // returns the natural logarithm of 2
Math.LN10      // returns the natural logarithm of 10
Math.LOG2E     // returns base 2 logarithm of E
Math.LOG10E    // returns base 10 logarithm of E
```

Array, Example1
```
<!DOCTYPE html>
<html>
<body>

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

<script>
var cars = [
  "Saab",
  "Volvo",
```

```
   "BMW"
];
document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

Ex 2.

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

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

<script>
var cars = new Array("Saab", "Volvo", "BMW");
document.getElementById("demo").innerHTML = cars[0];
</script>

</body>
</html>
```

Ex 3

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

<p>The length property returns the length of an array.</p>

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

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.length;
</script>

</body>
</html>
```

Ex 4

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

```
<p>The length property provides an easy way to append new elements to an array without using the
push() method.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits[fruits.length] = "Lemon";
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Ex 5

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

<p>Adding elements with high indexes can create undefined "holes" in an array.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
  var index;
  var text = "<ul>";
  var fruits = ["Banana", "Orange", "Apple", "Mango"];
  for (index = 0; index < fruits.length; ++index) {
    text += "<li>" + fruits[index] + "</li>";
  }
  text += "</ul>";
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

Ex6

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

<p>The pop method removes the last element from an array.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.pop()
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Ex 7

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

<p>The push method appends a new element to an array.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
  fruits.push("Kiwi")
  document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Sort and Reverse

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

<p>The sort() method sorts an array alphabetically.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;

function myFunction() {
 fruits.sort();
   fruits.reverse();
   document.getElementById("demo").innerHTML = fruits;
}
</script>

</body>
</html>
```

Concat

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

<p>Click the button to join three arrays.</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
   var arr1 = ["Cecilie", "Lone"];
   var arr2 = ["Emil", "Tobias", "Linus"];
   var arr3 = ["Robin", "Morgan"];
   document.getElementById("demo").innerHTML =
   arr1.concat(arr2, arr3);
}
</script>
```

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

If condition

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

<p>Click the button to get a time-based greeting:</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
  var greeting;
  var time = new Date().getHours();
  if (time < 10) {
    greeting = "Good morning";
  } else if (time < 20) {
    greeting = "Good day";
  } else {
    greeting ="Good evening";
  }
document.getElementById("demo").innerHTML = greeting;
}
</script>

</body>
</html>
```

Switch case:

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

<p>Click the button to display what day it is today:</p>

<button onclick="myFunction()">Try it</button>

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

<script>
function myFunction() {
  var day;
  switch (new Date().getDay()) {
```

```
      case 0:
        day = "Sunday";
        break;
      case 1:
        day = "Monday";
        break;
      case 2:
        day = "Tuesday";
        break;
      case 3:
        day = "Wednesday";
        break;
      case 4:
        day = "Thursday";
        break;
      case 5:
        day = "Friday";
        break;
      case  6:
        day = "Saturday";
        break;
    }
    document.getElementById("demo").innerHTML = "Today is " + day;
}
</script>

</body>
</html>
```

For loop

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

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

<script>
cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 2;
var len = cars.length;
var text = "";

for (; i < len; i++) {
   text += cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = text;
```

```
</script>

</body>
</html>
```

While loop

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

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

<script>
cars = ["BMW","Volvo","Saab","Ford"];
var i = 0;
var text = "";
while (cars[i]) {
   text += cars[i] + "<br>";
   i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Try-catch-throw

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

<script>
function myFunction() {
   var mess = document.getElementById("mess");
   mess.innerHTML = "";
   try {
      var x = document.getElementById("demo").value;
      if(x == "")  throw "Empty";
      if(isNaN(x)) throw "Not a number";
      if(x > 10)   throw "Too high";
      if(x < 5)    throw "Too low";
   }
   catch(err) {
      mess.innerHTML = "Error: " + err + ".";
   }
}
</script>
```

```
<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">

<button type="button" onclick="myFunction()">Test Input</button>

<p id="mess"></p>

</body>
</html>
```

Regular Expression
A regular expression is a sequence of characters that forms a search pattern.
The search pattern can be used for text search and text replace operations.
What Is a Regular Expression?
A regular expression is a sequence of characters that forms a **search pattern**.
When you search for data in a text, you can use this search pattern to describe what you are searching for.
A regular expression can be a single character, or a more complicated pattern.
Regular expressions can be used to perform all types of **text search** and **text replace** operations.
Syntax
/*pattern*/*modifiers*;

Example:
var patt = /w3schools/i
Example explained:
/w3schools/i is a regular expression.
w3schools is a pattern (to be used in a search).
i is a modifier (modifies the search to be case-insensitive).
Using String Methods
In JavaScript, regular expressions are often used with the two **string methods**: search() and replace().
The search() method uses an expression to search for a match, and returns the position of the match.
The replace() method returns a modified string where the pattern is replaced.

Using String search() With a Regular Expression
Example
Use a regular expression to do a case-insensitive search for "w3schools" in a string:
var str = "Visit W3Schools";
var n = str.search(/w3schools/i);
The result in n will be:
6

Use String replace() With a Regular Expression

Example
Use a case insensitive regular expression to replace Microsoft with W3Schools in a string:
var str = "Visit Microsoft!";
var res = str.replace(/microsoft/i, "W3Schools");
The result in res will be:
Visit W3Schools!

Regular Expression Modifiers
Modifiers can be used to perform case-insensitive more global searches:

Regular Expression Patterns
Brackets are used to find a range of characters:

Metacharacters are characters with a special meaning:

Quantifiers define quantities

In-built Methods in Javascript

Number Methods

The Number object contains only the default methods that are part of every object's definition.

Sr.No	Method & Description
1	**constructor()** Returns the function that created this object's instance. By default this is the Number object.
2	**toExponential()** Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation.
3	**toFixed()** Formats a number with a specific number of digits to the right of the decimal.
4	**toLocaleString()**

	Returns a string value version of the current number in a format that may vary according to a browser's locale settings.
5	**toPrecision()** Defines how many total digits (including digits to the left and right of the decimal) to display of a number.
6	**toString()** Returns the string representation of the number's value.
7	**valueOf()** Returns the number's value.

Boolean Methods

Here is a list of each method and its description.

Sr.No	Method & Description
1	**toSource()** Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	**toString()** Returns a string of either "true" or "false" depending upon the value of the object.
3	**valueOf()** Returns the primitive value of the Boolean object.

String Methods

Here is a list of each method and its description.

Sr.No	Method & Description
1	**charAt()** Returns the character at the specified index.
2	**charCodeAt()** Returns a number indicating the Unicode value of the character at the given index.
3	**concat()** Combines the text of two strings and returns a new string.
4	**indexOf()** Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	**lastIndexOf()** Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	**localeCompare()** Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	**length()** Returns the length of the string.
8	**match()** Used to match a regular expression against a string.

9	**replace()** Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
10	**search()** Executes the search for a match between a regular expression and a specified string.
11	**slice()** Extracts a section of a string and returns a new string.
12	**split()** Splits a String object into an array of strings by separating the string into substrings.
13	**substr()** Returns the characters in a string beginning at the specified location through the specified number of characters.
14	**substring()** Returns the characters in a string between two indexes into the string.
15	**toLocaleLowerCase()** The characters within a string are converted to lower case while respecting the current locale.
16	**toLocaleUpperCase()** The characters within a string are converted to upper case while respecting the current locale.
17	**toLowerCase()** Returns the calling string value converted to lower case.
18	**toString()**

	Returns a string representing the specified object.
19	**toUpperCase()** Returns the calling string value converted to uppercase.
20	**valueOf()** Returns the primitive value of the specified object.

String HTML wrappers

Here is a list of each method which returns a copy of the string wrapped inside the appropriate HTML tag.

Sr.No	Method & Description
1	**anchor()** Creates an HTML anchor that is used as a hypertext target.
2	**big()** Creates a string to be displayed in a big font as if it were in a <big> tag.
3	**blink()** Creates a string to blink as if it were in a <blink> tag.
4	**bold()** Creates a string to be displayed as bold as if it were in a <b> tag.
5	**fixed()** Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag
6	**fontcolor()** Causes a string to be displayed in the specified color as if it were in a <font color="color"> tag.

7	**fontsize()** Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag.
8	**italics()** Causes a string to be italic, as if it were in an <i> tag.
9	**link()** Creates an HTML hypertext link that requests another URL.
10	**small()** Causes a string to be displayed in a small font, as if it were in a <small> tag.
11	**strike()** Causes a string to be displayed as struck-out text, as if it were in a <strike> tag.
12	**sub()** Causes a string to be displayed as a subscript, as if it were in a <sub> tag
13	**sup()** Causes a string to be displayed as a superscript, as if it were in a <sup> tag

Array Methods

Here is a list of each method and its description.

Sr.No	Method & Description
1	**concat()** Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	**every()**

	Returns true if every element in this array satisfies the provided testing function.
3	**filter()** Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	**forEach()** Calls a function for each element in the array.
5	**indexOf()** Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
6	**join()** Joins all elements of an array into a string.
7	**lastIndexOf()** Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
8	**map()** Creates a new array with the results of calling a provided function on every element in this array.
9	**pop()** Removes the last element from an array and returns that element.
10	**push()** Adds one or more elements to the end of an array and returns the new length of the array.
11	**reduce()**

	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
12	**reduceRight()** Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13	**reverse()** Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
14	**shift()** Removes the first element from an array and returns that element.
15	**slice()** Extracts a section of an array and returns a new array.
16	**some()** Returns true if at least one element in this array satisfies the provided testing function.
17	**toSource()** Represents the source code of an object
18	**sort()** Sorts the elements of an array.
19	**splice()** Adds and/or removes elements from an array.
20	**toString()**

	Returns a string representing the array and its elements.
21	**unshift()** Adds one or more elements to the front of an array and returns the new length of the array.

Date Methods

Here is a list of each method and its description.

Sr.No	Method & Description
1	**Date()** Returns today's date and time
2	**getDate()** Returns the day of the month for the specified date according to local time.
3	**getDay()** Returns the day of the week for the specified date according to local time.
4	**getFullYear()** Returns the year of the specified date according to local time.
5	**getHours()** Returns the hour in the specified date according to local time.
6	**getMilliseconds()** Returns the milliseconds in the specified date according to local time.
7	**getMinutes()** Returns the minutes in the specified date according to local time.

8	**getMonth()** Returns the month in the specified date according to local time.
9	**getSeconds()** Returns the seconds in the specified date according to local time.
10	**getTime()** Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
11	**getTimezoneOffset()** Returns the time-zone offset in minutes for the current locale.
12	**getUTCDate()** Returns the day (date) of the month in the specified date according to universal time.
13	**getUTCDay()** Returns the day of the week in the specified date according to universal time.
14	**getUTCFullYear()** Returns the year in the specified date according to universal time.
15	**getUTCHours()** Returns the hours in the specified date according to universal time.
16	**getUTCMilliseconds()** Returns the milliseconds in the specified date according to universal time.
17	**getUTCMinutes()**

	Returns the minutes in the specified date according to universal time.
18	**getUTCMonth()** Returns the month in the specified date according to universal time.
19	**getUTCSeconds()** Returns the seconds in the specified date according to universal time.
20	**getYear()** **Deprecated** - Returns the year in the specified date according to local time. Use getFullYear instead.
21	**setDate()** Sets the day of the month for a specified date according to local time.
22	**setFullYear()** Sets the full year for a specified date according to local time.
23	**setHours()** Sets the hours for a specified date according to local time.
24	**setMilliseconds()** Sets the milliseconds for a specified date according to local time.
25	**setMinutes()** Sets the minutes for a specified date according to local time.
26	**setMonth()** Sets the month for a specified date according to local time.

27	**setSeconds()** Sets the seconds for a specified date according to local time.
28	**setTime()** Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
29	**setUTCDate()** Sets the day of the month for a specified date according to universal time.
30	**setUTCFullYear()** Sets the full year for a specified date according to universal time.
31	**setUTCHours()** Sets the hour for a specified date according to universal time.
32	**setUTCMilliseconds()** Sets the milliseconds for a specified date according to universal time.
33	**setUTCMinutes()** Sets the minutes for a specified date according to universal time.
34	**setUTCMonth()** Sets the month for a specified date according to universal time.
35	**setUTCSeconds()** Sets the seconds for a specified date according to universal time.
36	**setYear()**

	Deprecated - Sets the year for a specified date according to local time. Use setFullYear instead.
37	**toDateString()** Returns the "date" portion of the Date as a human-readable string.
38	**toGMTString()** **Deprecated -** Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
39	**toLocaleDateString()** Returns the "date" portion of the Date as a string, using the current locale's conventions.
40	**toLocaleFormat()** Converts a date to a string, using a format string.
41	**toLocaleString()** Converts a date to a string, using the current locale's conventions.
42	**toLocaleTimeString()** Returns the "time" portion of the Date as a string, using the current locale's conventions.
43	**toSource()** Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
44	**toString()** Returns a string representing the specified Date object.
45	**toTimeString()** Returns the "time" portion of the Date as a human-readable string.

46	**toUTCString()** Converts a date to a string, using the universal time convention.
47	**valueOf()** Returns the primitive value of a Date object.

Date Static Methods

In addition to the many instance methods listed previously, the Date object also defines two static methods. These methods are invoked through the Date( ) constructor itself −

Sr.No	Method & Description
1	**Date.parse( )** Parses a string representation of a date and time and returns the internal millisecond representation of that date.
2	**Date.UTC( )** Returns the millisecond representation of the specified UTC date and time.

Math Methods

Here is a list of each method and its description.

Sr.No	Method & Description
1	**abs()** Returns the absolute value of a number.
2	**acos()** Returns the arccosine (in radians) of a number.

3	**asin()** Returns the arcsine (in radians) of a number.
4	**atan()** Returns the arctangent (in radians) of a number.
5	**atan2()** Returns the arctangent of the quotient of its arguments.
6	**ceil()** Returns the smallest integer greater than or equal to a number.
7	**cos()** Returns the cosine of a number.
8	**exp()** Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	**floor()** Returns the largest integer less than or equal to a number.
10	**log()** Returns the natural logarithm (base E) of a number.
11	**max()** Returns the largest of zero or more numbers.
12	**min()** Returns the smallest of zero or more numbers.

13	**pow()** Returns base to the exponent power, that is, base exponent.
14	**random()** Returns a pseudo-random number between 0 and 1.
15	**round()** Returns the value of a number rounded to the nearest integer.
16	**sin()** Returns the sine of a number.
17	**sqrt()** Returns the square root of a number.
18	**tan()** Returns the tangent of a number.
19	**toSource()** Returns the string "Math".

RegExp Methods

Here is a list of each method and its description.

Sr.No	Method & Description
1	**exec()**-Executes a search for a match in its string parameter.
2	**test()**-Tests for a match in its string parameter.

3	**toSource()**-Returns an object literal representing the specified object; you can use this value to create a new object.
4	**toString()** -Returns a string representing the specified object.

Document object model.
The DOM is the way Javascript sees its containing pages' data. It is an object that includes how the HTML/XHTML/XML is formatted, as well as the browser state.
A DOM element is something like a DIV, HTML, BODY element on a page. You can add classes to all of these using CSS, or interact with them using JS.

The Document Object

When an HTML document is loaded into a web browser, it becomes a **document object**.

The document object is the root node of the HTML document.

Document Object Properties and Methodsss

The following properties and methods can be used on HTML documents:

Property / Method	Description
activeElement	Returns the currently focused element in the document
addEventListener()	Attaches an event handler to the document
adoptNode()	Adopts a node from another document

anchors	Returns a collection of all <a> elements in the document that have a name attribute
applets	Returns a collection of all <applet> elements in the document
baseURI	Returns the absolute base URI of a document
body	Sets or returns the document's body (the <body> element)
close()	Closes the output stream previously opened with document.open()
cookie	Returns all name/value pairs of cookies in the document
charset	Deprecated. Use characterSet instead. Returns the character encoding for the document
characterSet	Returns the character encoding for the document
createAttribute()	Creates an attribute node
createComment()	Creates a Comment node with the specified text
createDocumentFragment()	Creates an empty DocumentFragment node
createElement()	Creates an Element node

createEvent()	Creates a new event
createTextNode()	Creates a Text node
defaultView	Returns the window object associated with a document, or null if none is available.
designMode	Controls whether the entire document should be editable or not.
doctype	Returns the Document Type Declaration associated with the document
documentElement	Returns the Document Element of the document (the <html> element)
documentMode	Returns the mode used by the browser to render the document
documentURI	Sets or returns the location of the document
domain	Returns the domain name of the server that loaded the document
domConfig	Obsolete. Returns the DOM configuration of the document
embeds	Returns a collection of all <embed> elements the document
execCommand()	Invokes the specified clipboard operation on the element currently having focus.

forms	Returns a collection of all <form> elements in the document
fullscreenElement	Returns the current element that is displayed in fullscreen mode
fullscreenEnabled()	Returns a Boolean value indicating whether the document can be viewed in fullscreen mode
getElementById()	Returns the element that has the ID attribute with the specified value
getElementsByClassName()	Returns a NodeList containing all elements with the specified class name
getElementsByName()	Returns a NodeList containing all elements with a specified name
getElementsByTagName()	Returns a NodeList containing all elements with the specified tag name
hasFocus()	Returns a Boolean value indicating whether the document has focus
head	Returns the <head> element of the document
images	Returns a collection of all <img> elements in the document
implementation	Returns the DOMImplementation object that handles this document

importNode()	Imports a node from another document
inputEncoding	Returns the encoding, character set, used for the document
lastModified	Returns the date and time the document was last modified
links	Returns a collection of all <a> and <area> elements in the document that have a href attribute
normalize()	Removes empty Text nodes, and joins adjacent nodes
normalizeDocument()	Removes empty Text nodes, and joins adjacent nodes
open()	Opens an HTML output stream to collect output from document.write()
querySelector()	Returns the first element that matches a specified CSS selector(s) in the document
querySelectorAll()	Returns a static NodeList containing all elements that matches a specified CSS selector(s) in the document
readyState	Returns the (loading) status of the document
referrer	Returns the URL of the document that loaded the current document

removeEventListener()	Removes an event handler from the document (that has been attached with the addEventListener() method)
renameNode()	Renames the specified node
scripts	Returns a collection of <script> elements in the document
strictErrorChecking	Sets or returns whether error-checking is enforced or not
title	Sets or returns the title of the document
URL	Returns the full URL of the HTML document
write()	Writes HTML expressions or JavaScript code to a document
writeln()	Same as write(), but adds a newline character after each statement

Code Conventions for the JavaScript Programming Language

This is a set of coding conventions and rules for use in JavaScript programming.

The long-term value of software to an organization is in direct proportion to the quality of the codebase. Over its lifetime, a program will be handled by many pairs of hands and eyes. If a program is able to clearly communicate its structure and characteristics, it is less likely that it will break when modified in the never-too-distant future. Code conventions can help in reducing the brittleness of programs.

All of our JavaScript code is sent directly to the public. It should always be of publication quality. Neatness counts.

JavaScript Files

JavaScript programs should be stored in and delivered as .js files.

JavaScript code should not be embedded in HTML files unless the code is specific to a single session. Code in HTML adds significantly to pageweight with no opportunity for mitigation by caching and compression.

Whitespace

Where possible, these rules are consistent with centuries of good practice with literary style. Deviations from literary style should only be tolerated if there is strong evidence of a significant benefit.

Blank lines improve readability by setting off sections of code that are logically related.

Blank spaces should always be used in the following circumstances:

- A keyword followed by (*left parenthesis* should be separated by a space. Spaces are used to make things that are not invocations look less like invocations, so for example, there should be space after if or while.

```
while (true) {
```

- A blank space should not be used between a function value and its invoking (*left parenthesis*. This helps to distinguish between keywords and function invocations.
- The word function is always followed with one space.
- No space should separate a unary operator and its operand except when the operator is a word such as typeof.
- All binary operators should be separated from their operands by a space on each side except . *period* and (*left parenthesis* and [*left bracket*.
- Every , *comma* should be followed by a space or a line break.
- Each ; *semicolon* at the end of a statement should be followed with a line break.
- Each ; *semicolon* in the control part of a for statement should be followed with a space.

Every statement should begin aligned with the current indentation. The outermost level is at the left margin. The indentation increases by 4 spaces when the last token on the previous line is { *left brace*, [*left bracket*, (*left paren*. The matching closing token will be the first token on a line, restoring the previous indentation.

The ternary operator can be visually confusing, so wrap the entire ternary expression in parens. The condition, the ? *question mark*, and the : *colon* always begins a line, indented 4 spaces.

```
let integer = function (
    value,
    default_value
) {
    value = resolve(value);
    return (
        typeof value === "number"
        ? Math.floor(value)
        : (
            typeof value === "string"
            ? value.charCodeAt(0)
            : default_value
        )
    );
};
```

Clauses (case, catch, default, else, finally) are not statements and so should not be indented like statements.

Use of tabs invites confusion, argument,and crying, with little compensating value. Do not use tabs. Use space.

Comments

Be generous with comments. It is useful to leave information that will be read at a later time by people (possibly your future self) who will need to understand what you have done and why. The comments should be well-written and clear, just like the code they are annotating. An occasional nugget of humor might be appreciated. Frustrations and resentments will not.

It is important that comments be kept up-to-date. Erroneous comments can make programs even harder to read and understand.

Make comments meaningful. Focus on what is not immediately visible. Don't waste the reader's time with stuff like

```
// Set i to zero.

i = 0;
```

Use line comments, not block comments. The comments should start at the left margin.

Variable Declarations

All variables should be declared before used. JavaScript does not require this, but doing so makes the program easier to read and makes it easier to detect undeclared variables that may become implied. Implied global variables should never be used. Use of global variables should be minimized.

```
let currentEntry; // currently selected table entry
let level;       // indentation level
let size;        // size of table
```

Function Declarations

All functions should be declared before they are used. Inner functions should follow the let statement. This helps make it clear what variables are included in its scope.

There should be no space between the name of a function and the (*left parenthesis* of its parameter list. There should be one space between the) *right parenthesis*and the { *left curly brace* that begins the statement body. The body itself is indented four spaces. The } *right curly brace* is aligned with the line containing the beginning of the declaration of the function.

```
function outer(c, d) {
    let e = c * d;

    function inner(a, b) {
        return (e * a) + b;
    }
```

```
    return inner(0, 1);
  }
```

This convention works well with JavaScript because in JavaScript, functions and object literals can be placed anywhere that an expression is allowed. It provides the best readability with inline functions and complex structures.

```
  function getElementsByClassName(className) {
    let results = [];
    walkTheDOM(document.body, function (node) {
      let array;            // array of class names
      let ncn = node.className; // the node's classname

// If the node has a class name, then split it into a list of simple names.
// If any of them match the requested name, then append the node to the list of results.

      if (ncn && ncn.split(" ").indexOf(className) >= 0) {
        results.push(node);
      }
    });
    return results;
  }
```

If a function literal is anonymous, there should be one space between the word function and the (*left parenthesis*. If the space is omitted, then it can appear that the function's name is function, which is an incorrect reading.

```
  div.onclick = function (e) {
    return false;
  };

  that = {
    method: function () {
      return this.datum;
    },
    datum: 0
  };
```

Use of global functions should be minimized.

When a function is to be invoked immediately, the entire invocation expression should be wrapped in parens so that it is clear that the value being produced is the result of the function and not the function itself.

```
let collection = (function () {
  let keys = [];
  let values = [];

  return {
    get: function (key) {
      let at = keys.indexOf(key);
      if (at >= 0) {
```

```
            return values[at];
        }
    },
    set: function (key, value) {
        let at = keys.indexOf(key);
        if (at < 0) {
            at = keys.length;
        }
        keys[at] = key;
        values[at] = value;
    },
    remove: function (key) {
        let at = keys.indexOf(key);
        if (at >= 0) {
            keys.splice(at, 1);
            values.splice(at, 1);
        }
    }
};
}());
```

Names

Names should be formed from the 26 upper and lower case letters (A .. Z, a .. z), the 10 digits (0 .. 9), and _ *underbar*. Avoid use of international characters because they may not read well or be understood everywhere. Do not use $ *dollar sign* or \ *backslash* in names.

Do not use _ *underbar* as the first or last character of a name. It is sometimes intended to indicate privacy, but it does not actually provide privacy. If privacy is important, use closure. Avoid conventions that demonstrate a lack of competence.

Most variables and functions should start with a lower case letter.

Constructor functions that must be used with the new prefix should start with a capital letter. JavaScript issues neither a compile-time warning nor a run-time warning if a required new is omitted. Bad things can happen if new is not used, so the capitalization convention is the only defense we have.

Global variables in browsers should be in all caps.

Statements
Simple Statements

Each line should contain at most one statement. Put a ; *semicolon* at the end of every simple statement. Note that an assignment statement that is assigning a function literal or object literal is still an assignment statement and must end with a semicolon.

JavaScript allows any expression to be used as a statement. This can mask some errors, particularly in the presence of semicolon insertion. The only expressions that should be used as statements are assignments, invocations, and delete.

Compound Statements

Compound statements are statements that contain lists of statements enclosed in { } *curly braces*.

- The enclosed statements should be indented four more spaces.
- The { *left curly brace* should be at the end of the line that begins the compound statement.
- The } *right curly brace* should begin a line and be indented to align with the beginning of the line containing the matching { *left curly brace*.
- Braces should be used around all statements, even single statements, when they are part of a control structure, such as an if or for statement. This makes it easier to add statements without accidentally introducing bugs.

Labels

Statement labels should be avoided. Only these statements should be labeled: while, do, for, switch.

return Statement

The return value expression must start on the same line as the return keyword in order to avoid semicolon insertion.

if Statement

The if class of statements should have the following form:

```
if (condition) {
   statements
}

if (condition) {
   statements
} else {
   statements
}

if (condition) {
   statements
} else if (condition) {
   statements
} else {
   statements
}
```

for Statement

A for class of statements should have the following form:

```
for (initialization; condition; update) {
   statements
}
```

while Statement

A while statement should have the following form:

```
while (condition) {
    statements
}
```

do Statement

A do statement should have the following form:

```
do {
    statements
} while (condition);
```

Unlike the other compound statements, the do statement always ends with a ; *semicolon*.

switch Statement

A switch statement should have the following form:

```
switch (expression) {
case expression:
    statements
default:
    statements
}
```

Each case is aligned with the switch. This avoids over-indentation. A case label is not a statement, and should not be indented like one.

Each group of *statements* (except the default) should end with break, return, or throw. Do not fall through.

try Statement

The try class of statements should have the following form:

```
try {
    statements
} catch (variable) {
    statements
}

try {
    statements
} catch (variable) {
    statements
} finally {
```

statements
}

continue Statement

Avoid use of the continue statement. It tends to obscure the control flow of the function.

with Statement

The with statement should not be used.

{} and **[]**

Use {} instead of new Object(). Use [] instead of new Array().

Use arrays when the member names would be sequential integers. Use objects when the member names are arbitrary strings or names.

, *comma* **Operator**

Avoid the use of the comma operator. (This does not apply to the comma separator, which is used in object literals, array literals, and parameter lists.)

Assignment Expressions

Avoid doing assignments in the condition part of if and while statements.

Is

```
if (a = b) {
```

a correct statement? Or was

```
if (a == b) {
```

intended? Avoid constructs that cannot easily be determined to be correct.

=== and !== Operators.

Use the === and !== operators. The == and != operators do type coercion and should not be used.

Confusing Pluses and Minuses

Be careful to not follow a + with + or ++. This pattern can be confusing. Insert parens between them to make your intention clear.

```
total = subtotal + +myInput.value;
```

is better written as

```
    total = subtotal + Number(myInput.value);
```

so that the + + is not misread as ++. Avoid ++.

eval is Evil

The eval function is the most misused feature of JavaScript. Avoid it.

eval has aliases. Do not use the Function constructor. Do not pass strings to setTimeout or setInterval.