

Cascading Style Sheets

Contents

1. [Introduction](#)
2. [Applying CSS to \(X\)HTML](#) - linking, embedding and inlining
3. [Applying CSS to XML](#)
4. CSS Construction
5. [CSS Syntax](#)
6. [Defining Style Rules](#)
 1. [CSS Lengths and Units](#)
 2. [CSS Selectors](#)
 3. [CSS Inheritance Rules](#)
 4. [The !important Keyword](#)
7. CSS Presentation
 1. [Color](#)
 2. [Fonts and Text](#)
 3. [Hyperlinks](#)
 4. [Lists](#)
 5. [Box Model](#) — setting the size and shape of elements
 6. [Background](#)
 7. [CSS Shorthand Properties](#)
8. CSS Layout
 1. [Positioning Elements \(includes floating elements\)](#)
 2. [Tables](#)
 3. [Floating Elements](#)
 4. [Media Types](#)
9. CSS and Divs - in progress
10. Troubleshooting
 1. [Standards Mode and Quirks Mode](#)
 2. [Browser Compatibility](#)
 3. [CSS order of Rules](#)
11. Appendices
 1. [Hacks and Filters](#)
 2. [Links](#)
 3. [Index of Properties](#)
 4. [Index](#)
 5. [Contributors](#)

Introduction

[CSS](#), or cascading style sheets, are used to describe the visual style and presentation of a document, most commonly web sites. One of the purposes of CSS is to separate the presentation of a document from the structure and content (although it is possible to embed CSS within the structure as well), and is a [W3C](#) recommendation. Separation of style from structure and content increases maintainability for the author(s) of a document as well as accessibility for the audience. CSS is commonly applied to [HTML](#), [XHTML](#) and [XML](#) documents, though it is possible, albeit rare, to apply it to other types of documents as well.

CSS is one way of describing the "style" -- the colors, fonts, layout, and other "presentation" aspects of a document. A single CSS file can describe a common "style" to be applied to many HTML, XHTML, and XML documents (which describe the content and structure of the elements of each document). Typically a particular element in a XHTML file has a "cascade" of CSS style rules that can be applied to it. The highest priority style rule is applied to each element.

Why use CSS?

Why would I want to use CSS rather than something else?

CSS is a powerful tool that gives web designers flexibility and modularity in the presentation layer of a web site.

CSS allows you to have *every* format rule defined for later use (here "format" means how things appear). So if you are writing a large website and you want a consistent appearance for every title, sub-title, how examples of code appear, how paragraphs are aligned, (I could go on, CSS covers a wide range of presentation options) then CSS is the way to go.

Let's say you have a 1200 page website that took you months to complete. Your current boss gets a promotion and another person fills his place. Your new boss says to change the font, the size, the background, the appearance of tables, etc. *everywhere* on your 1200-page site to comply with some new corporate policy. If you engineered your site appropriately with CSS, you could do this by editing one [linked](#) CSS file that has *all* your appearance (format) rules in one place. Or you could do it the hard way, and hammer the appearance changes on each and every one of your 1200 HTML pages. By using CSS, changes can be made fast and easy simply by editing a few rules and lines in the global stylesheet. Before CSS was used, these changes were more difficult, expensive, and very time-consuming.

Some server-side template systems can largely be used for the same purpose. Unlike CSS, however, they most often separate the structure from the content and not the presentation from the structure, making it much more difficult for users who wish to disable or ignore styling to do so.

Applying CSS to (X)HTML

CSS can be applied to HTML or XHTML using three methods: linked, embedded, and inline. In the linked method, the CSS is stored in a separate file, instead of directly in the HTML page. In the embedded method, CSS is stored as part of the HTML page, in the header section. In the inline method, CSS is stored directly in the *style* attributes of the HTML tags, as `<div style="font-weight:bold">Bold Font</div>`.

The neatest method is probably the linked one, but the other ones are convenient and quick in the phases of prototyping a web page. The embedded and inline methods do not require having a separate file. The inline method saves you the trouble of considering what CSS classes your document should have. For a larger site, in which many web pages share the same styling, and in which the styling should be customizable by the user, the linked method is the only viable option.

The methods are treated in detail in the following sections.

Linking

With linked CSS, CSS rules are stored in a separate file. To refer to that file from the HTML page, add the link element (and its closing element within XHTML) to the head element, as shown in the following example, which assumes that the stylesheet is stored in the file named "style.css".

```
<head>
  <title>Example Web Page</title>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
```

The link element in the example has three attributes. The first, *rel*, tells the browser the type of the target of the link. The second, *type*, tells the browser what type of stylesheet it is. And the third, *href*, tells the browser under which URL to find the stylesheet. In the example, the URL is relative, but it can also be absolute.

The "style.css" with a single rule contains only the following text:

```
p {
  font-weight:bold;
color:red;
}
```

This tells the browser that the text in the paragraph (p) element should be rendered as bold.

Example rendering:

Text that will be formatted.

The source code for the complete HTML document is thus as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

CSS

```
<html lang="en">
<head>
  <title>Example Web Page</title>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <p>Text that will be formatted.</p>
</body>
</html>
```

Embedding

Dynamically generated webpages may need to use embedded CSS but this should be kept to a minimum. Even in dynamic pages any CSS that is common to multiple pages should generally be moved to a linked stylesheet.

Embedded CSS is CSS that is located in the header of the HTML document that requires styling. For example we would like the text in this HTML document to appear bold.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
<head>
  <title>Example Web Page</title>
</head>
<body>
  <p>
    Text that will be formatted.
  </p>
</body>
</html>
```

The CSS must be placed in the document's header section:

```
<head>
  <title>Example Web Page</title>
  <style type="text/css">
    p {
      font-weight:bold;
    }
  </style>
</head>
```

The CSS is contained in a style element. Setting the element's type attribute to text/css tells the browser that the enclosed sheetstyle is written in CSS and should be used to format the page. If the attribute type is missing or set to an unrecognized value the CSS will not be applied to the page.

The little bit of CSS in this example tells the browser to make all the text found in any paragraph (p) elements bold. The text on the page would look like this:

Text that will be formatted.

Here is the complete document including the CSS.

CSS

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
<head>
<title>Example Web Page</title>
<style type="text/css">
p {
font-weight:bold;
color:green;
}
</style>
</head>
<body>
<p>
Text that will be formatted.
</p>
</body>
</html>
```

Remember, you should use [linked](#) stylesheets in preference to embedded stylesheets whenever possible; this allows you to most easily replace the general style information without having to keep track of styles within the various documents and files.

Inlining

Inline CSS is specified directly in the opening tag of the element you want it to apply to. It is entered into the style attribute within HTML or XHTML 1.1.

For example

```
<div style="font-weight:bold; color:blue;">Bold Font</div>
```

is rendered as:

Bold Font

As mentioned, you should in general prefer linked CSS to inline CSS.

Applying CSS to XML

To apply a stylesheet to an XML document, place the modified following example into the *prolog* of the XML document.

```
<?xml-stylesheet href="style.css" type="text/css"?>
```

Any *XML declaration*, such as `<?xml version="1.0" encoding="UTF-8"?>`, must precede the stylesheet processing instruction.

This applies to XHTML too, if it is served with the correct MIME type: `application/xhtml+xml`.

A complete example, with an XML declaration and a stylesheet processing instruction:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="rss_style.css" type="text/css"?>
<rss version="2.0">
  <channel>
    <title>My news feed</title>
    <link>http://my.news/</link>
    <description>Latest news</description>
    <language>en-gb</language>
    <copyright>Me</copyright>
    <pubDate>Wed, 06 Sep 2006 00:00:00 GMT</pubDate>
    <docs>http://blogs.law.harvard.edu/tech/rss</docs>
  </channel>
</rss>
```

Syntax

Cascading Style Sheets are most often linked to a web page so that they can be used in an entire site. Because of this they are an independent file and have their own construction.

A stylesheet should begin with a declaration.

```
@charset "UTF-8";
```

After this declaration each CSS rule is independent and made of three parts: a selector, a property and an argument (i.e. a value):

Example:

```
@charset "UTF-8";
```

```
body {
background-color : #000000;
font-family : Georgia, "Times New Roman", Times, serif;
font-size : 100%;
color : #ffffff;
margin : 0;
padding : 0;
```

CSS

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

```
h1 {
font-family : Georgia, "Times New Roman", Times, serif;
font-size : 16px;
color : #ffffff;
font-style : normal;
font-weight : normal;
letter-spacing : 0.1em;
}
```

```
h2 {
font-family : Georgia, "Times New Roman", Times, serif;
font-size : 12px;
font-style : italic;
font-weight : bold;
color : #ffffff;
text-decoration : underline;
}
```

```
p {
font-family : Georgia, "Times New Roman", Times, serif;
font-size : 12px;
font-style : normal;
color : #ffffff;
}
```

etc

What this breaks down to here is this:

```
@rule argument(s) {
  selector {
    property: argument(s);
    another-property: argument(s);
    property-the-third: argument(s);
    fourth-property: argument(s);
  }
}
```

Each selector can have as many properties as you would like. The properties and values are enclosed in { } tags.

Rules

CSS rules are preceded by the @ symbol and are often the start of blocks of code. Rules take at least 1 argument. Some examples of rules are @charset, for defining the document character set; @media, for setting properties for a type of media, often all, screen and print; and @font for setting up a [web font](#).

Selectors

Selectors are where most of the power in CSS lies. Selectors allow the author to specifically target an element to apply any given property to. Many different selector types can be combined for the precision application of styles.

Elements

Elements that are specified in CSS definitions (abbreviated E, F, etc.) correspond to the (X)HTML or XML elements in the document. In HTML or XHTML, common elements include p, span, and div. In XML, the element names will vary with the type of document to be displayed.

Class

Class is the most basic of the selectors. The class operator is a "." and the syntax is E.className.

ID

ID specifies a unique element in the document and in (X)HTML it is also the fragment identifier. The ID operator is a "#" and the syntax is E#IDname.

Attribute

Attribute selection is a newer feature in CSS that allows selection based on any attribute of the element. The syntax is E[Attribute="value"]. Attribute supports several different matching operators, including =, !=, ~=, ^= and \$=.

Pseudo-class

Pseudo-classes are special classes based on the state of an element and uses the : operator. The syntax is E:pseudo-class. Commonly used pseudo-classes include :hover, :link and :visited...

Blocks

Blocks of code are everywhere in CSS. A block is started with { and closed with }. Blocks are used to group CSS statements into logical groups based on the author's needs. Blocks are most commonly used by Rules and to group properties to a selector.

Properties

Properties are the meat of CSS. The syntax is quite simple, property: argument(s);. Properties are always after a selector and always inside a {} block. A property must be ended with a ; to close it.

Defining Style Rules

You can associate formatting instructions with a web page by defining CSS rules in either an [embedded](#), [linked](#), or [inline](#) stylesheet. Although there are many ways of defining styles, the main difference between them is where they are located. Linked stylesheets are preferred for live websites. Embedded and inline stylesheets are useful for prototyping but do not have the maintenance savings associated with linked stylesheets.

Syntax

A stylesheet consists of one or more rules.

Each rule has two parts: a selector and a group of one or more declarations surrounded by braces (curly brackets):

```
selector { declaration; declaration2; ... }
```

Each declaration is formed like this:

```
property: value
```

Remember that there can be several declarations in one rule. A common mistake is to mix up colons, which separate the property and value of a declaration, and semicolons, which separate declarations. A selector chooses the elements for which the rule applies and the declaration sets the value for the different properties of the elements that are chosen.

The property is one of many properties. w3.org has a list of all of the [CSS Properties](#).

Basic Uses

Here is an example of how you would make all span elements appear bold:

```
span {  
    font-weight: bold;  
}
```

The font-weight: bold; declaration has the property font-weight and the value bold which makes the font bold.

That would appear like this:

Bold Text

A slightly more complex example is

```
span {  
    font-weight: bold;  
    color: yellow;  
    background-color: black;  
}
```

CSS

That would appear like this:

Yellow Bold Text

You can use many other properties, and values, but this is just a beginning as to how to use CSS.

CSS Lengths and Units

To specify widths, heights and other lengths, you can use various units, listed in the following table.

		Units in CSS
Code	Definition	Note
em	The height of the element's font.	
ex	The height of the letter 'x' in the element's font.	
px	pixel	
mm	millimeter	
cm	centimeter	
pt	point (1/72 inch)	
pc	pica (12 points = 1/6 inch)	
in	inch	

Lengths may also be specified as a percentage of another length. Using percentages can be complicated because the base length varies from property to property. For example, when percentages are used with the margin property the calculation is based on the width of the containing block. When percentages are used with the font-size property the calculation is based on the font-size of the parent element but with line-height it is based on the font-size of the current element.

Font sizes on screen are best given as a percentage or in ems. (See the note on [Using ems in Internet Explorer](#).) This means that the page will interact gracefully with users' font preferences. Using pixels (px) for font sizes causes a number of problems and should be avoided.

The absolute units mm, cm, pt, pc and in do not work well on screen and cause problems in many older browsers. It is safest to only use them for print media. Even in print media they may interact badly with users' preferences.

Relative units

The three units em, ex and px are called relative units. These units do not specify a fixed length. Instead they scale relative to some other quantity. In the case of em and ex they scale relative to the font size of some element.

Screen pixels, printer pixels and CSS pixels

The px unit specifies a length in *CSS pixels*. These are not the same as the pixels on the screen that the document is rendered on or the dots on the printer that it is printed on. It is perfectly possible, for example, for there to be two screen pixels or five printer dots to one CSS pixel. Many web browsers use the rule that one screen pixel equals one CSS pixel for simplicity but you can not rely on this. Opera and Internet Explorer version 7 allow the user to change the number of screen pixels per CSS pixel. It is common for users with poor vision who use these browsers to choose to have five or six screen pixels to the CSS pixel.

The [formal definition](#) of the CSS pixel is quite complicated but the basic idea is that a CSS pixel appears to be the same size when the document is at a comfortable reading distance. So a CSS pixel would be physically bigger on a monitor than on a mobile phone because the monitor would normally be further from the user's eyes than the mobile phone.

Calculations

Divide the desired margin width/height by the width/height of its container to convert it to ems. Multiply by 100 to convert that to percentages.

Divide the desired container width by 1.62 to use the Golden Ratio as the size of the content block. Subtract the content width from the container width to use the Golden Ratio as the size of the sidebar

CSS Selectors

In CSS, a **selector** is the part of a rule that comes before the first "{", such as "p" in the rule " p { font-weight:bold; } ". A selector specifies to which elements a rule should apply, by naming the type of the element, such as "div", the class of the element, or the id of the element. Selectors can only be used in linked and embedded CSS, not in inlined one.

The following table provides an overview of selectors. The sections that follow discuss each type of selector in detail.

Overview of selectors				
Type of Selector	Example Selector	Example Rule	Note	
type	div	div { margin:7px; padding:7px; }		
class	.important	.important { font-weight:bold; color:red; }		
id	#onlyThisOne	#onlyThisOne { font-family:courier; }		
universal	*	* { color:green; }		
pseudo-class	a:link	a:link { color:blue; }		
pseudo-element	p:first-letter	p:first-letter { color:red }		
descendant	td span	td span { font-weight:bold; }		
grouped	h1, h2, h3	h1, h2, h3 { text-align:center; }		

Type

These selectors match elements based on the name of their element type. The example above is using a type selector to make every instance of p have bold text. You can use the type selector with any element. Here are a few examples:

```
div{
  margin:7px;
  padding:7px;
}
```

```
body{
  background-image:url("image.gif");
  font-size:.9em;
}
```

Class

The class selector works with (X)HTML documents. It does not work with general XML documents unless the User Agent (web browser or other document reader) can determine which attribute is the class attribute for elements in the document. In (X)HTML documents class is defined to be the class attribute.

HTML permits the class attribute for all elements that are valid in the body section of an HTML document, including the body element itself. This allows the web designer to distinguish between elements of the same type used in different contexts. For example, the designer could differentiate between HTML elements and HTML attributes in a technical document on HTML.

The `<code class="attribute">alt</code>` attribute of the `<code class="element">img</code>`

The class selector allows you to apply CSS rules based on the class of an element.

The first way is to make a global class, which can be added to any element. You do that like this:

```
.important {  
  font-weight:bold; color:red;  
}
```

That will make any element that has the class of **important** be bold and red.

Sample HTML:

```
<p class="important">Important Text</p>  
<p>Ordinary Text</p>  
<div class="important">Important Footnote</div>
```

Example rendering:

Important Text
Ordinary Text
Important Footnote

The second way is to attach it to a type selector. The rule is only applied to elements of the given type which are of the specified class.

CSS rule:

```
p.right {  
  float:right;  
  clear:both  
}
```

Sample HTML:

```
<p class="right">Righty Righty</p>
```

Example rendering (look right):

Righty Righty

CSS

An element may belong to more than one class, e.g.

```
<p class="right">This paragraph belongs to the class 'right'.</p>  
<p class="important">This paragraph belongs to the class 'important'.</p>  
<p class="right important">This paragraph belongs to both classes.</p>
```

Example rendering:

This paragraph belongs to the class 'right'.

This paragraph belongs to the class 'important'.

This paragraph belongs to both classes.

Class names should describe the purpose of the class, e.g. important, not the effect of the class, e.g. red. If you name classes by effect and then change your mind about the appearance you want you can end up with rules like:

```
.red {color:blue}
```

If necessary multiple class selectors can be used to select only elements that belong to all the specified classes, e.g.

```
p.important.right {  
  border: 2px dashed #666  
}
```

Example rendering:

This paragraph belongs to the class 'right'.

This paragraph belongs to the class 'important'.

This paragraph belongs to both classes.

Internet Explorer 6 bug

Multiple class selectors as shown in the previous example do not work in Internet Explorer version 6. It treats the selector as though only the last class selector was present, e.g. p.important.right is treated as equivalent to p.right.

Previous example rendered in Internet Explorer 6:

This paragraph belongs to the class 'right'.

This paragraph belongs to the class 'important'.

This paragraph belongs to both classes.

CSS

The complete HTML document and CSS stylesheet to test this bug are given below.

classBug.css:

```
.important {
  font-weight:bold; color:red;
}

p.right {
  float:right;
  clear:both
}

p.important.right {
  border: 2px dashed #666
}
```

classBug.htm:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
<head>
  <title>IE6 multiple class bug</title>
  <link rel="stylesheet" type="text/css" href="classBug.css">
</head>
<body>
  <p class="right">This paragraph belongs to the class 'right'.</p>
  <p class="important">This paragraph belongs to the class 'important'.</p>
  <p class="right important">This paragraph belongs to both classes.</p>
</body>
</html>
```

ID

The ID selector works with (X)HTML documents. It does not work with general XML documents unless the User Agent (web browser or other document reader) can determine which attribute is the ID attribute for elements in the document. In (X)HTML documents id is defined to be the ID attribute.

The ID selector is used to select a single item on a page. It matches the id attribute of an element. Two elements on the same page must not have the same id attribute.

However, several different webpages within the same site might reuse an id. It is commonly used for the major divisions of a page, e.g. mainContent, navigationBar. It is often used in conjunction with descendant selectors, e.g. to style all list items in the navigation bar whilst not affecting lists in the main content.

CSS rule:

```
#onlyThisOne {
  font-family:courier;
}
```

CSS

Sample HTML:

```
<p id="onlyThisOne">Courier</p>
```

Example rendering:

Courier

Universal

These selectors apply a style to all the items on the page. For example to make all the text on the page green use the rule:

```
* {  
  color:green;  
}
```

Pseudo-Classes

Pseudo-classes and pseudo-elements allow elements to be formatted on the basis of information other than the position of the element in the document tree. Pseudo-class and pseudo-element selectors are prefixed by a colon, :, in CSS1 and CSS2.1. In CSS3 pseudo-elements are prefixed by ::.

CSS level 1 defines three pseudo-classes:

link
 unvisited links
visited
 visited links
active
 active links

These can only be applied to the anchor (a) elements.

```
a:link{  
  color:blue;  
}
```

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

```
a:active{  
  color:red;  
}
```

CSS level 2.1 introduces several additional pseudo-classes, including hover. hover can be used to create rollover effects without resorting to Javascript. CSS2.1 also allows active to apply to any element that can be active, e.g. a button.

CSS

The additional CSS 2.1 pseudo-classes are:

first-child

matches any element that is the first child of its parent.

lang(...)

matches the language of the element. The language may be set directly on the element or inherited from an ancestor element. A valid language code must appear in the parentheses, e.g. en for any variant of English or en-gb for British English.

hover

applies whilst the user hovers over the element with the mouse.

active

now allowed on any element capable of being 'active' – applies whilst the element is active.

focus

applies whilst the element has the keyboard focus.

Although CSS2.1 allows hover on any element Internet Explorer only allows it on anchor elements.

Examples:

```
p:first-child {
  font-size:120%
}
```

```
span:lang(la) { /* render Latin text, e.g. per se, in italics */
  font-style:italic
}
```

```
li:hover { /* doesn't work in Internet Explorer */
  background-color:blue
}
```

```
input:active {
  color:red
}
```

```
input:focus {
  background-color:yellow
}
```

An example of the first-child pseudo-class is given at the end of the [descendant](#) selector section.

Multiple pseudo-classes may be specified, e.g. to warn the user that they are hovering over a link they have already visited:

```
a:visited:hover {
  background-color:red
}
```

CSS

Care should be taken over the order of rules using pseudo-classes. For example, if you want visited links to be green except whilst the user hovers over them, when they should be yellow, the rules must be in the following order:

```
a:visited{
  color:green
}

a:hover{
  color:yellow
}
```

If the rules were reversed the [cascade](#) would mean that the visited color would take precedence over the hover color. Visited links would remain green even when the user hovered over them.

Pseudo-Elements

Pseudo-classes and pseudo-elements allow elements to be formatted on the basis of information other than the position of the element in the document tree. Pseudo-class and pseudo-element selectors are prefixed by a colon, :, in CSS1 and CSS2.1. In CSS3 pseudo-elements are prefixed by ::.

CSS level 1 defines two pseudo-elements, first-letter and first-line, which select the first letter and line of the rendered element respectively. These can be used to apply typographic effects, e.g. to create a drop cap at the start of a paragraph.

```
p:first-letter { color:red }
```

Only one pseudo-element selector may be used and it must be the last part of the selector chain. The first-line selector may only be applied to block-level elements, inline-blocks, table captions and table cells.

CSS2.1 adds two more pseudo-elements, before and after.

Starting in CSS3 pseudo-elements are prefixed with :: rather than :.

Simple selectors

The selectors described above (type, class, ID, universal, pseudo-class and pseudo-element) are all examples of simple selectors. The full syntax for a simple selector is:

- a type selector or the universal selector
- zero, one or more class, ID and pseudo-class selectors (CSS2.1 also allows attribute selectors)
- zero or one pseudo-element selectors

The following are all examples of simple selectors:

```
p
p.important
p#navigation
a:link
```

CSS

```
p:first-line  
a:visited#homePage.menu2:first-letter  
*  
*#footer  
*.content.abstract  
*#mainArticle:first-letter
```

As a shorthand notation the universal selector, *, may be omitted if it is not the only component of the simple selector. For example, #footer is equivalent to *#footer and :first-line is equivalent to *:first-line.

Combining simple selectors: Simple selectors can be combined to take the context of the element in to account. For example you might want to apply a rule only to list elements in the navigation bar.

Descendant

Descendant selectors allow you to apply style to elements that are nested within another specified element. For example, you could have every span element inside of every p element be bold. The span can be a child element of the p element, or a grandchild, or a great-grandchild, etc.

CSS rule:

```
p span{  
  font-weight:bold;  
}
```

Sample HTML:

```
<p>Start of paragraph. <span>This span is bold.</span> Rest of paragraph.</p>  
<div>Start of division. <span>This span is normal.</span> Rest of division.</div>
```

Example rendering:

Start of paragraph. **This span is bold.** Rest of paragraph.
Start of division. This span is normal. Rest of division.

The next example changes the color of visited links in the navigation section. Visited links in other parts of the document would be unaffected.

CSS rule:

```
ul#navigation a:visited {  
  color:red  
}
```

Sample HTML:

```
<ul id="navigation">  
  <li><a href="HomePage">Home</a></li>  
  <li><a href="AnotherPage">A page you haven't visited yet.</a></li>  
</ul>
```

CSS

Example rendering:

- **Home**
- [A page you haven't visited yet.](#)

An example using the first-child selector.

CSS rule:

```
div.content strong:first-child {
  color:red
}
```

Sample HTML:

```
<div class="content">
  <p>Some <em>emphasized</em> text and some <strong>strongly emphasized</strong> text.</p>
  <p>Some <strong>strongly emphasized</strong> text and some <em>emphasized</em> text.</p>
</div>
```

Example rendering:

Some *emphasized* text and some **strongly emphasized** text.

Some **strongly emphasized** text and some *emphasized* text.

Two important points to note:

- an element is still the first child if it is preceded by text, so the em element in the first paragraph and the strong element in the second paragraph are the first children of their respective paragraphs.
- a rule is only applied to an element if it matches all parts of the selector. The strong element in the first paragraph is a second child, so it is not matched by the strong:first-child selector.

Grouping selectors

Selectors can be grouped into comma separated lists.

```
h1, h2, h3 { text-align:center; }
```

is equivalent to

```
h1 { text-align:center; }
h2 { text-align:center; }
h3 { text-align:center; }
```

An element may be selected by more than one rule:

```
h1, h2, h3, h4, h5, h6 { color: white; background-color: red; }
h1 { color:yellow }
```

All headings will have a red background. Level 2–6 headings will have white text. Level 1 headings will have yellow text.

CSS

The order of the rules is significant. In the following example all headings would have white text.

```
h1 { color:yellow }  
h1, h2, h3, h4, h5, h6 { color: white; background-color: red; }
```

When more than one rule applies to an element the [cascade](#) is used to determine the resulting style applied to the element.

Later CSS versions

Additional selectors in CSS 2.1:

- Child selectors
- Adjacent sibling selectors
- Attribute selectors
- More pseudo-classes and pseudo-elements.

Additional selectors in CSS 3:

- More sibling selectors
- More attribute selectors
- More pseudo-classes and pseudo-elements.

CSS Inheritance Rules

[Cascading Style Sheets/Inheritance](#)

The !important Keyword

Overview

The important keyword makes a declaration take precedence over normal declarations—those that are not tagged with the important keyword. So "p { color: red ! important }" takes precedence over "p { color: green }".

The syntax for an important declaration is

```
property: value ! important
```

The relative importance of a rule also depends on its the source: whether it comes from a style sheet specified by the document author, the user or the user agent.

The order of declarations from least important to most important:

1. user agent declarations

CSS

2. user normal declarations
3. author normal declarations
4. author important declarations
5. user important declarations

User important declarations take precedence over everything else. This allows users to override the page designer's presentation to meet their needs. For example a user could override the text colours to be yellow on black in a large font to make the text easier to read.

The process of determining which of the rules from one or more style sheets applies to a given element is called the *cascade*, hence the name *Cascading Style Sheets*.

An Example of the Cascade

The HTML document used for this example is:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
<head>
  <title>'important' CSS declarations</title>
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <p>Paragraph 1.</p>
  <p id="para2">Paragraph 2.</p>
  <p>Paragraph 3.</p>
</body>
</html>
```

The document is styled with the style sheet, style.css:

```
p {
  background-color:#fff ;
  color:#000 ! important
}

#para2 {
  background-color:#fc9 ;
  color:#00f ;
  border:1px solid black
}

p {
  background-color:#9cf ;
  color:#f00
}
```

How will the three paragraphs be presented?

First paragraph

Consider the first paragraph element (<p>Paragraph 1.</p>). The first and third selectors (the p selectors) match the element. The second selector (#para2) does not match the element. The

CSS

rules for the two selectors that match are applied to the element. The rules, listed in the order they appear in the source, are:

Declaration		Selector	Weighting					
Property	Value		Importance*	Specificity				Source order**
		style attribute		ID selector	Class selector	Element selector		
background-color	#fff	p	author normal (rank 3)	0	0	0	1	1
color	#000	p	author important (rank 4)	0	0	0	1	2
background-color	#9cf	p	author normal (rank 3)	0	0	0	1	6
color	#00f	p	author normal (rank 3)	0	0	0	1	7

* The ranks for importance range from 1 (least important) to 5 (most important). Later examples on this page show other ranks.

** The source order column is the position of the declaration in the file, e.g. background-color:#9cf is the sixth declaration in the file.

The rules that apply to the element are sorted by property. For each property the rules are sorted in ascending order by importance, then by specificity (style attribute, ID selector, class selector, element selector), then by source order. The last value for the property from the sorted list wins (i.e. the value for the most important, most specific rule).

Declaration		Selector	Weighting					
Property	Value		Importance	Specificity				Source order
		style attribute		ID selector	Class selector	Element selector		
background-color	#fff	p	author normal (rank 3)	0	0	0	1	1
background-color	#9cf	p	author normal (rank 3)	0	0	0	1	6
color	#00f	p	author normal (rank 3)	0	0	0	1	7
color	#000	p	author important (rank 4)	0	0	0	1	2

The declarations that are used are shown in bold along with the winning tie-breaker that

CSS

decided between the rules.

The background-color is set to #9cf (a pale blue). The two rules for the background-color property have equal importance and specificity so the second rule wins because it occurs later in the source.

The color is set to #000 (black). The rules for the color property have different importance so the most important wins.

The first paragraph is rendered:

Paragraph 1.

Second paragraph

Consider the second paragraph element (<p id="para2">Paragraph 2.</p>). The first and third selectors (the p selectors) match the element. The second selector (#para2) also matches the element. The rules for all three selectors are applied to the element. The rules, listed in the order they appear in the source, are:

Declaration		Selector	Importance	Weighting				Source order
Property	Value			Specificity				
		style attribute	ID selector	Class selector	Element selector			
background-color	#fff	p	author normal (rank 3)	0	0	0	1	1
color	#000	p	author important (rank 4)	0	0	0	1	2
background-color	#fc9	#para2	author normal (rank 3)	0	1	0	0	3
color	#00f	#para2	author normal (rank 3)	0	1	0	0	4
border	1px solid black	#para2	author normal (rank 3)	0	1	0	0	5
background-color	#9cf	p	author normal (rank 3)	0	0	0	1	6
color	#00f	p	author normal (rank 3)	0	0	0	1	7

Sorting the rules gives:

Declaration		Selector	Importance	Weighting				Source order
Property	Value			Specificity				
		style attribute	ID selector	Class selector	Element selector			

CSS

background-color	#fff	p	author normal (rank 3)	0	0	0	1	1
background-color	#9cf	p	author normal (rank 3)	0	0	0	1	6
background-color	#fc9	#para2	author normal (rank 3)	0	1	0	0	3
border	1px solid black	#para2	author normal (rank 3)	0	1	0	0	5
color	#00f	#para2	author normal (rank 3)	0	1	0	0	4
color	#00f	p	author normal (rank 3)	0	0	0	1	7
color	#000	p	author important (rank 4)	0	0	0	1	2

The background-color is set to #fc9 (a pale orange). The rule with an ID selector has a higher specificity than the other two rules so it wins.

There is only one rule for the border property so it is set to 1px solid black.

The color is set to #000 (black). The rules for the color property have different importance so the most important wins.

The second paragraph is rendered:

Paragraph 2.

Third paragraph

The third paragraph is matched by exactly the same selectors as the first paragraph and so exactly the same values are applied.

The third paragraph is rendered:

Paragraph 3.

Result

Paragraph 1.

Paragraph 2.

Paragraph 3.

Example with a User Style sheet

In this example we will examine how the appearance of the document in the previous example is altered when the following user style sheet, user.css, is combined with the style sheet from the previous example. (See the chapter [User Style Sheets](#) for information on how to set a user style sheet.)

```
p {
  color:#ff0 ! important ;
  background-color:#808080 ;
  font-family:cursive
}
```

Consider the first paragraph element. The single selector from the user style sheet matches the element. The first and third selectors (the p selectors) from style.css also match the element. The second selector (#para2) from style.css does not match the element. The rules that apply to the element, listed in the order they appear in the source, are:

Declaration		Selector	Weighting					
Property	Value		Importance *	Specificity				Source order**
		style attribute		ID selector	Class selector	Element selector		
color	#ff0	p	user important (rank 5)	0	0	0	1	1:1
background-color	#808080	p	user normal (rank 2)	0	0	0	1	1:2
font-family	cursive	p	user normal (rank 2)	0	0	0	1	1:3
background-color	#fff	p	author normal (rank 3)	0	0	0	1	2:1
color	#000	p	author important (rank 4)	0	0	0	1	2:2
background-color	#9cf	p	author normal (rank 3)	0	0	0	1	2:6
color	#00f	p	author normal (rank 3)	0	0	0	1	2:7

* The ranks for importance range from 1 (least important) to 5 (most important). Notice that rules from the user style sheet are either rank 2 for normal declarations or rank 5 for important declarations.

** The first number in the source order column gives the order the file containing the rule

CSS

was loaded in. In this case user.css was loaded before style.css. The second number is the position of the declaration within the file. E.g. background-color:#9cf is the sixth declaration in the second file.

Sorting the rules gives:

Declaration		Selector	Weighting					Source order**
Property	Value		Specificity					
		Importance*	style attribute	ID selector	Class selector	Element selector		
background-color	#808080	p	user normal (rank 2)	0	0	0	1	1:2
background-color	#fff	p	author normal (rank 3)	0	0	0	1	2:1
background-color	#9cf	p	author normal (rank 3)	0	0	0	1	2:6
color	#00f	p	author normal (rank 3)	0	0	0	1	2:7
color	#000	p	author important (rank 4)	0	0	0	1	2:2
color	#ff0	p	user important (rank 5)	0	0	0	1	1:1
font-family	cursive	p	user normal (rank 2)	0	0	0	1	1:3

There are two rules with equal highest importance for the background-color property. These two rules also have equal specificity so they are split by source order. The background color is set to the author's choice, not the user's choice.

The important declaration for color in the user style sheet has the highest importance for this property so the user's choice of text color is used, not the page author's choice.

The only rule for font-family is the user's so the user's choice is used.

The paragraphs are rendered:

Paragraph 1.

Paragraph 2.

Paragraph 3.

Internet Explorer !important Bug

This bug is known to occur in Internet Explorer versions 6.0 and 7.0 on Windows XP; however, version 8.0 is unaffected.

There is a bug in Internet Explorer's handling of !important. If an important declaration for a property is followed by a normal declaration for the same property within the same block Internet Explorer treats both declarations as normal. For example in IE v6.0 the following style sheet incorrectly renders paragraphs as yellow on red. The paragraphs should be white on black because the first two declarations are important.

```
p {
  background-color: black ! important ;
  color: white ! important;
  background-color: red ;
  color: yellow
}
```

You can check that the declarations revert to normal by adding an extra rule to the style sheet.

```
p {
  background-color: black ! important ;
  color: white ! important;
  background-color: red ;
  color: yellow
}
```

```
p {
  color: green
}
```

In IE v6.0 this gives green on red indicating that all the color declarations are being treated as normal.

If you remove the second color declaration from the first block:

```
p {
  background-color: black ! important ;
  color: white ! important;
  background-color: red ;
}
```

```
p {
  color: green
}
```

the paragraphs become white on red. This shows that the color declaration in the first block is now correctly considered important.

A number of people suggest using this bug as a means of passing Internet Explorer different values from those given to other browsers. A better solution if you need to do this is to use [conditional comments](#).

Color

Colors can be specified for various objects. These include text ("color: white"), background ("background-color: white"), and borders ("border-color: gray").

An example CSS rule that sets all h1 elements to have white text on a red background:

```
h1 { color: white; background-color: red; }
```

Methods of specification of colors, an overview:

- English name, such as color: white
- Hexadecimal RGB value, such as color: #ff0000
- Decimal RGB value, such as color: rgb(255, 0, 0)
- Decimal RGBA value, such as color: rgba(255, 0, 0, 0.2)
- HSL value, such as color: hsl(120, 100%, 50%)
- HSLA value, such as color: hsl(0, 100%, 50%, 0.5)

Specification of colors is detailed in the following sections.

If you set any colors in your web page, you should set both the background and text color for the body element of the page. Imagine if you set the text color to black and did not set the background color. A user has their preferred colors set to yellow text on a black background, a fairly common combination for users with low vision. The page is rendered with your black text on their black background and is unusable.

Using English names

The following 16 values are defined:

- aqua
- black
- blue
- fuchsia
- gray
- green
- lime
- maroon
- navy
- olive
- purple
- red
- silver
- teal
- yellow
- white

CSS does not define the exact shade that should be used for the named colours. Use RGB-

values if the exact shade is important.

Hexadecimal RGB value

	Hex	Bin	Dec
	0	0000	0
	1	0001	1
	2	0010	2
	3	0011	3
	4	0100	4
	5	0101	5
	6	0110	6
	7	0111	7
	8	1000	8
	9	1001	9
	A	1010	10
	B	1011	11
	C	1100	12
	D	1101	13
	E	1110	14
	F	1111	15

The mixture ratio of a color to be displayed is specified in [hexadecimal](#) notation. That is, they are written in base-16 as opposed to the more familiar base 10. A reference table is included, courtesy [Wikipedia](#).

The two first hexadecimal digits specify the amount of red in the color, the third and fourth specify the amount of green and the last two figures specify the amount of blue.

```
h1 { color: #ff0000; } /* All h1 headings are printed in bright red. */
```

A short-hand notation is permitted: #rgb is equivalent to #rrgbb, e.g. #3cf is equivalent to #33ccff.

Note that the range of values possible is hexadecimal 00 (= decimal 0) to hexadecimal ff (= decimal 255). This is the same range that is available using the rgb notation from the next section.

RGB value

RGB is a abbreviation for red, green and blue – the three colors that are mixed to create all the other colors on a computer screen.

The basic syntax is rgb(red-value, green-value, blue-value).

The different values can be set using two different approaches.

CSS

A number from 0 to 255

```
h1 { color: rgb(255, 0, 0); } /* All h1 headings are printed in bright red. */
```

A decimal figure from 0% to 100%

```
h1 { color: rgb(100%, 0, 0); } /* All h1 headings are printed in bright red. */
```

Note that you can use either integer (0-255) **or** percentage (0-100%) values, you **cannot mix them**.

RGBA value

RGBA is RGB with an added [alpha channel](#) as its 4th argument. The alpha channel is a value between 0 (fully transparent) and 1 (opaque). RGBA is part of CSS3.

```
div { background-color: rgba(255, 0, 0, 0.5); } /* All divs are in bright red with 50% opacity. */
```

```
background-color: rgba(255, 0, 0, 0);
background-color: rgba(255, 0, 0, 0.1);
background-color: rgba(255, 0, 0, 0.2);
background-color: rgba(255, 0, 0, 0.3);
background-color: rgba(255, 0, 0, 0.4);
background-color: rgba(255, 0, 0, 0.5);
background-color: rgba(255, 0, 0, 0.6);
background-color: rgba(255, 0, 0, 0.7);
background-color: rgba(255, 0, 0, 0.8);
background-color: rgba(255, 0, 0, 0.9);
background-color: rgba(255, 0, 0, 1);
```

Please note that [MediaWiki](#) blocks the use of the background-image property, so you must copy the code used below to a file or your snippet editor to see the full effect.

```
<div style="background: url('http://upload.wikimedia.org/wikipedia/commons/1/1f/Wallpaper.FALA-S.gif');">
  <div style="background-color: rgba(255, 0, 0, 0); padding: .25em;">background-color: rgba(255, 0, 0, 0);</div>
  <div style="background-color: rgba(255, 0, 0, 0.1); padding: .25em;">background-color: rgba(255, 0, 0, 0.1);</div>
  <div style="background-color: rgba(255, 0, 0, 0.2); padding: .25em;">background-color: rgba(255, 0, 0, 0.2);</div>
  <div style="background-color: rgba(255, 0, 0, 0.3); padding: .25em;">background-color: rgba(255, 0, 0, 0.3);</div>
  <div style="background-color: rgba(255, 0, 0, 0.4); padding: .25em;">background-color: rgba(255, 0, 0, 0.4);</div>
  <div style="background-color: rgba(255, 0, 0, 0.5); padding: .25em;">background-color: rgba(255, 0, 0, 0.5);</div>
  <div style="background-color: rgba(255, 0, 0, 0.6); padding: .25em;">background-color: rgba(255, 0, 0, 0.6);</div>
  <div style="background-color: rgba(255, 0, 0, 0.7); padding: .25em;">background-color: rgba(255, 0, 0, 0.7);</div>
  <div style="background-color: rgba(255, 0, 0, 0.8); padding: .25em;">background-color: rgba(255, 0, 0, 0.8);</div>
  <div style="background-color: rgba(255, 0, 0, 0.9); padding: .25em;">background-color: rgba(255, 0, 0, 0.9);</div>
```

CSS

```
<div style="background-color: rgba(255, 0, 0, 1); padding: .25em;">background-color: rgba(255, 0, 0, 1);</div>
</div>
```

Here is the example again, with a silver background:

```
background-color: rgba(255, 0, 0, 0);
background-color: rgba(255, 0, 0, 0.1);
background-color: rgba(255, 0, 0, 0.2);
background-color: rgba(255, 0, 0, 0.3);
background-color: rgba(255, 0, 0, 0.4);
background-color: rgba(255, 0, 0, 0.5);
background-color: rgba(255, 0, 0, 0.6);
background-color: rgba(255, 0, 0, 0.7);
background-color: rgba(255, 0, 0, 0.8);
background-color: rgba(255, 0, 0, 0.9);
background-color: rgba(255, 0, 0, 1);, which is the: rgb(255, 0, 0)
```

HSL value

HSL stands for [hue, saturation and lightness](#). It is the color value system used by many [cathode-ray tube](#) devices. HSL is part of CSS3.

- `hsl(color-angle, saturation%, lightness%);`

```
div.red { background-color: hsl(0, 100%, 50%); } /* red in HSL */
div.green { background-color: hsl(120, 100%, 50%); } /* green in HSL */
div.blue { background-color: hsl(240, 100%, 50%); } /* blue in HSL */
```

- Red:
- Green:
- Blue:

HSLA value

HSLA is the HSL color with an alpha channel. Like RGBA, the 4th argument is a value between 0 and 1. HSLA is part of CSS3.

```
div.red { background-color: hsl(0, 100%, 50%, 0.5); } /* red in HSL with 50% opacity*/
```

```
div { background-color: hsla(0, 100%, 50%, 0.5); } /* All divs are in bright red with 50% opacity. */
```

```
background:rgba(255,255,255,0.9);
background-color: hsla(0, 100%, 50%, 0.1);
background-color: hsla(0, 100%, 50%, 0.2);
background-color: hsla(0, 100%, 50%, 0.3);
background-color: hsla(0, 100%, 50%, 0.4);
background-color: hsla(0, 100%, 50%, 0.5);
background-color: hsla(0, 100%, 50%, 0.6);
background-color: hsla(0, 100%, 50%, 0.7);
```


CSS

```
background-color: hsla(0, 100%, 50%, 0.8);
background-color: hsla(0, 100%, 50%, 0.9);
background-color: hsla(0, 100%, 50%, 1);
```

Please note that [MediaWiki](#) blocks the use of the background-image property, so you must copy the code used below a file or your snippet editor to see the full effect.

```
<div style="background: url('http://upload.wikimedia.org/wikipedia/commons/1/1f/Wallpaper.FALA-S.gif');">
  <div style="background-color: hsla(0, 100%, 50%, 0); padding: .25em;">background-color: hsla(0, 100%, 50%,
0);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.1); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.1);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.2); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.2);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.3); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.3);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.4); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.4);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.5); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.5);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.6); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.6);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.7); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.7);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.8); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.8);</div>
  <div style="background-color: hsla(0, 100%, 50%, 0.9); padding: .25em;">background-color: hsla(0, 100%,
50%, 0.9);</div>
  <div style="background-color: hsla(0, 100%, 50%, 1); padding: .25em;">background-color: hsla(0, 100%, 50%,
1);</div>
</div>
```

Here is the example again, with a silver background:

```
background-color: hsla(0, 100%, 50%, 0);
background-color: hsla(0, 100%, 50%, 0.1);
background-color: hsla(0, 100%, 50%, 0.2);
background-color: hsla(0, 100%, 50%, 0.3);
background-color: hsla(0, 100%, 50%, 0.4);
background-color: hsla(0, 100%, 50%, 0.5);
background-color: hsla(0, 100%, 50%, 0.6);
background-color: hsla(0, 100%, 50%, 0.7);
background-color: hsla(0, 100%, 50%, 0.8);
background-color: hsla(0, 100%, 50%, 0.9);
background-color: hsla(0, 100%, 50%, 1);, which is the: hsl(0, 100%, 50%)
```

Fonts and Text

Font Properties

- font-family — Sets the typeface.
- font-style — Sets the style of the text, common values are normal and italic.
- font-variant — Modifies the text, e.g. small-caps.
- font-weight — Sets the thickness the text will be, values like bold and normal.
- font-size — Sets the font size.
- line-height — Sets the spacing between the baselines of adjacent lines of text.

Each of these is examined below in detail.

font-family

W3C Specification [CSS1](#) [CSS2.1](#)

The following is a simplified description of CSS font matching. See the [CSS1 font matching](#) or [CSS2.1 font matching](#) specification for the definitive description.

The value for the font-family property is a comma separated list of fonts in a prioritized order. The web browser will attempt to use the first font in the list. If the first font isn't available to the web browser it will attempt to use the second, and so on through the list. If none of the fonts are matched the web browser will use a default font. This might be a font selected by the user.

In the example below the web browser would try to use Verdana. If Verdana was not installed (or not available in an appropriate variant if font-variant is also specified) the web browser would try to use Helvetica. If Helvetica was not available it would try to use Arial.

```
p { font-family: Verdana, Helvetica, Arial, sans-serif }
```

Since there isn't a standard set of fonts that is available in all web browsers and operating systems it is possible that none of the named fonts will be matched. Five generic font families are defined to allow for this possibility. Web browsers will replace a generic font with the closest available font. The five generic families are: serif, sans-serif, monospace, cursive and fantasy.

You should always end a font-family list with a generic text-family. Otherwise you might, for example, end up with the user's default serif font where you wanted a sans-serif font. A generic font is always guaranteed to match so there is no point specifying alternative fonts after a generic font. On systems with a limited range of fonts it is possible that two or more generic fonts may be mapped to the same font. You can check which fonts your web browser uses for the generic fonts by looking at the example below. Are there five distinct fonts?

serif sans-serif monospace cursive fantasy

An ongoing problem for web designers is that they are not able to specify exactly the fonts that they require, because that they don't know which fonts the user will have installed. The way around this is to specify in the font-family list a more-or-less equivalent font from each possible client system. There are a number of useful references for determining this equivalence ([here](#),

CSS

for example).

For example, to use the font known on Windows as Impact, the following declaration is used.

```
font-family: Impact, Charcoal, sans-serif
```

Where 'Impact' is the Windows name, 'Charcoal' is the MacOS name, and 'sans-serif' is the generic fall-back for systems without either of these two.

font-style

W3C Specification [CSS1](#) [CSS2.1](#)

CSS1 and CSS2.1 define three values for this property: normal, italic and oblique. The initial value for this property is normal. Many font families do not define a separate italic version of the font, in which case the oblique version is used when font-style is set to italic.

CSS rules:

```
#para1 {  
  font-style:normal  
}
```

```
#para2 {  
  font-style:oblique  
}
```

```
#para3 {  
  font-style:italic  
}
```

Sample HTML:

```
<p id="para1">This sentence should be in an upright version of the font.</p>  
<p id="para2">This sentence should be in an oblique version of the font.</p>  
<p id="para3">This sentence should be in an italic version of the font.  
  Is it different to the oblique version?  
</p>
```

Example rendering:

This sentence should be in an upright version of the font.

This sentence should be in an oblique version of the font.

This sentence should be in an italic version of the font. Is it different to the oblique version?

font-variant

W3C Specification [CSS1](#) [CSS2.1](#)

CSS1 and CSS2.1 define two values for this property: normal and small-caps. The initial value

CSS

for this property is normal. When small-caps is applied lowercase letters appear as scaled down uppercase letters. Uppercase letters and other characters are unaltered. The effects of these values is shown below.

CSS rules:

```
#para1 {
  font-variant:normal
}

#para2 {
  font-variant:small-caps
}
```

Sample HTML:

```
<p id="para1">This sentence shows the effect of setting the
  Cascading Style Sheets property <code>font-variant</code>.
  Other characters: 1 2 3 % !
</p>
<p id="para2">This sentence shows the effect of setting the
  Cascading Style Sheets property <code>font-variant</code>.
  Other characters: 1 2 3 % !
</p>
```

Example rendering:

This sentence shows the effect of setting the Cascading Style Sheets property font-variant.
Other characters: 1 2 3 % !

THIS SENTENCE SHOWS THE EFFECT OF SETTING THE CASCADING STYLE SHEETS PROPERTY FONT-VARIANT. OTHER CHARACTERS: 1 2 3 % !

font-weight

W3C Specification [CSS1](#) [CSS2.1](#)

The font-weight property sets the darkness of the font. CSS currently defines nine levels of darkness from 100, the lightest, to 900, the darkest. The number of distinct levels depends on the font-family. Some families may define all nine levels while most will only define one or two. The value normal is a synonym for 400 and bold is a synonym for 700.

The list below shows the different weights for the current family:

- 100
- 200
- 300
- 400 or normal
- **500**
- **600**
- **700 or bold**
- **800**

CSS

- **900**

The final two possible values for font-weight are lighter and bolder. When these values are used the web browser tries to pick a font from the current family that is lighter or darker than the font of the parent element.

Since most font families only have one or two weights it is usual to only use the values normal and bold.

For example to make strong elements red instead of the bold used by most web browsers apply the following rule.

```
strong {
  font-weight:normal; /* remove the bold defined by the web browser */
  color:red
}
```

As another example you can make all hyperlinks bold.

```
a {
  font-weight:bold
}
```

Sample HTML:

```
<p><strong>Note:</strong> in this text
  <a href="glossary#hyperlink">hyperlinks</a> are in boldface.
</p>
```

Example rendering:

Note: in this text **hyperlinks** are in boldface.

font-size

W3C Specification [CSS1](#) [CSS2.1](#)

Before reading this section make sure you know how to change the default font size in all the browsers you use for testing web pages.

Make sure you understand the difference between screen pixels and CSS pixels. See [CSS Lengths and units](#).

Font size is one of the more problematic areas of web design. The most common complaint of web surfers is that the font size is too small on many pages. (See Jakob Nielsen's [Top Ten Web Design Mistakes of 2005](#).) A frequent request from customers buying a web site is for the font to be made smaller so more information can be crammed on to the page. To make matters worse, most users aren't taught how to change the default font size used by their web browser. (If you ever teach a class how to use a web browser, changing the font size and colors is one of the most important things you can teach.) Another issue is that web designers tend to be young with comparatively good eyesight. Just because you prefer to read text that is 60% of the out-of-the-box default font size doesn't mean your target audience can read text at that size.

Some users do change their default font size and web pages should respect their choices.

CSS

Ideally all the important information on a page should be at least as large as the user's default font size. Important information includes publication dates, links to the site's privacy policy, etc. Small print should be reserved for unimportant text such as statements that the page is valid HTML, conforms to WCAG Double-AA, etc. and effects such as superscripts that are normally rendered in a smaller size.

Lengths in CSS pixels (px) or absolute units (pt, pc, mm, cm and in) do not respect users' preferences. In fact using these units prevents users from easily changing the size of the font if they find it too small or too large. For example in Internet Explorer for Windows the user has to select 'Internet Options...' from the 'Tools' menu, then click on the 'Accessibility...' button on the 'General' tab and check the box to 'Ignore font sizes specified on web pages' if they want to change the size of a font given in any of these units. This isn't user friendly.

The values that do respect users' preferences are length in ems or exs, percentages and keyword values. There are seven 'absolute' keyword values that scale the font relative to the user's default font size and two 'relative' keyword values that scale the font relative to the parent element. The 'absolute' keywords are

- xx-small
- x-small
- small – the user's default font size in IE/win v6 [Quirks Mode](#) and IE/win v5 regardless of rendering mode.
- medium – the user's default font size in [Standards Mode](#).
- large
- x-large
- xx-large

The two relative keywords are smaller and larger.

Most web designers prefer to use ems. Font sizes in ems, exs and percentages are calculated relative to the font size of the parent element. 1em is equal to 100%. The following example demonstrates this.

```
#span1 {  
  font-size:0.5em;  
  color:red  
}
```

```
#span2 {  
  font-size:5em;  
  color:green  
}
```

```
#span3 {  
  font-size:0.4em;  
  color:magenta  
}
```

Sample HTML:

```
<p>The <span id="span1">text <span id="span2">size  
<span id="span3">changes </span></span></span>a lot.</p>
```

CSS

Example rendering:

The text size changes a lot.

Since 0.5 times 5 times 0.4 equals 1 the text in the last span should be the same size as the text in the parent paragraph.

Using ems in Internet Explorer

If you use ems for the font-size in a page viewed with Internet Explorer you need to be aware of a quirk of Internet Explorer. On a Windows PC save the two files given below in the same directory and view *IE_font_size_bug.htm* with Internet Explorer. (This example has been tested in IE/win v5.0, v5.5 and v6.0.)

IE_font_size_bug.css

```
p {
  font-size:1em
}
```

IE_font_size_bug.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html lang="en">
  <head>
    <title>IE font size bug</title>
    <link rel="stylesheet" type="text/css" href="IE_font_size_bug.css">
  </head>
  <body>
    <p>Internet Explorer has a little problem with ems.</p>
  </body>
</html>
```

Set the default font size to medium, smaller and smallest. (Use the 'Text Size' submenu of the 'View' menu if you don't have the Size button on your toolbar.) The height of the text is shown in the following table.

Default font size	Height of a capital letter	Height of an em
Medium	11 screen pixels	16 screen pixels
Smaller	9 screen pixels	13 screen pixels
Smallest	6 screen pixels	12 screen pixels

At six pixels high letters become blurred splotches on the page and are effectively unreadable.

There is a simple fix. Add an extra rule to *IE_font_size_bug.css* like so:

```
body {
  font-size:100% /* Keeps IE happy when the user sets the default size to 'smallest'. */
}
```

CSS

```
p {
  font-size:1em
}
```

In theory setting the font-size to 100% shouldn't change anything. However if you view the page with the new stylesheet you will find the size of 'smallest' text has changed.

Default font size	Height of a capital letter	Height of an em
Medium	11 screen pixels	16 screen pixels
Smaller	9 screen pixels	13 screen pixels
Smallest	9 screen pixels	12 screen pixels

Nine pixel high letters are quite readable if you have good eyesight and/or a large monitor.

It is a good idea to always set the font-size of the body to a percentage to deal with this quirk. It is also a good idea to leave a comment explaining why the font size declaration is there so that someone else doesn't remove an apparently redundant rule when they take over responsibility for your stylesheet.

Problems with nesting

You can run into problems using ems or percentages when you set the font-size property using a type selector. In the example below each of the nested div elements matches the selector and so the size of the text is halved each time. The text in the innermost division is $1/16$ of the user's default font size. You will probably struggle to read the example rendering unless you increase your default font size significantly.

CSS rules:

```
body {
  font-size:100% /* Keeps IE happy when the user sets the default size to 'smallest'. */
}

div {
  font-size:0.5em;
  border:1px dotted green;
  padding:1em
}
```

Sample HTML:

```
<div>
  This division contains
  <div>
    another division that contains
    <div>
      another division and so on
      <div>
        until the text is tiny.
      </div>
    </div>
  </div>
</div>
```


CSS

</div>

Example rendering:

This division contains
another division that contains
another division and so on
until the text is tiny.

This problem is often found with lists where it is tempting to set the font-size of the li element. When someone adds a sub-list in the future the text of the sub-list is far too small or far too large depending on the scale factor. The solution is to use an ID selector to target the outermost element of the nested group, e.g. the outermost ul element for nested lists.

Problems with tables and forms

If you are using tables or forms you may find you need to use the following rule to get the text in these elements to scale properly with the user's preferences:

```
table, input, select {  
  font-size:100%  
}
```

line-height

W3C Specification [CSS1](#) [CSS2.1](#)

The line-height property defines the minimum distance between the baselines of adjacent lines of text. The distance may be increased if necessary to fit images, etc. on the line. It is expressed as a ratio between the height of the font and the line spacing. The height of a font is defined as the distance between the baseline and the top of a capital letter. Some letters may have *descenders* that drop below the baseline. In some fonts there may also be *ascenders* the rise above the top of a capital letter.

In the example below the green band behind each line runs from the baseline up to the font height. The first line shows capital letters. The second line shows lowercase letters with descenders. The third line shows capital letters with accents.



The line-height can be given as either a simple number, a length or a percentage. If it is given as a simple number or a percentage this is multiplied by the font height to determine the space between the lines. The next example demonstrates how this paragraph appears with four different values for the line-height. The font is enlarged in the example to make the line spacing easier to see.

CSS

```
#para1 {  
  line-height: 1  
}  
  
#para2 {  
  line-height: 1.2em  
}  
  
#para3 {  
  line-height: 160%  
}  
  
#para4 {  
  line-height: 2  
}
```

line-height = 1 or 100% or 1em

The line-height can be given as either a simple number, a length or a percentage. If it is given as a simple number or a percentage this is multiplied by the font height to determine the space between the lines. The next example demonstrates four different values for the line-height. The font is enlarged in the example to make the line spacing easier to see.

line-height = 1.2 or 120% or 1.2em

The line-height can be given as either a simple number, a length or a percentage. If it is given as a simple number or a percentage this is multiplied by the font height to determine the space between the lines. The next example demonstrates four different values for the line-height. The font is enlarged in the example to make the line spacing easier to see.

line-height = 1.6 or 160% or 1.6em

The line-height can be given as either a simple number, a length or a percentage. If it is given as a simple number or a percentage this is multiplied by the font height to determine the space between the lines. The next example demonstrates four different values for the line-height. The font is enlarged in the example to make the line spacing easier to see.

line-height = 2 or 200% or 2em

The line-height can be given as either a simple number, a length or a percentage. If it is given as a simple number or a percentage this is multiplied by the font height to determine the space between the lines. The next example demonstrates four different values for the line-height. The font is enlarged in the example to make the line spacing easier to see.

Look closely at the descenders in the example with a line-height of 1. You will probably notice at

CSS

least one point where a descender touches a letter in the line below. Most web browsers default to a line-height of 1.2. A number of web designers feel that a line-height of 1.6 is more readable, particularly if the font size is relatively small.

Shorthand property

W3C Specification [CSS1](#) [CSS2.1](#)

The font shorthand property can be used to set the five individual font properties and the line-height property in a single declaration. The font has two methods for setting the individual properties, by specifying a combination of individual properties or by giving a single keyword that selects a predefined set of individual values. The second option is only available in CSS2.1.

Combination of individual properties

The syntax for combining individual properties is a little complicated. The properties are given in the following order:

- none, one or more of font-style, font-variant and font-weight in any order;
- optionally followed by font-size;
- if font-size is present it may be followed by a slash, /, followed by the line-height;
- followed by the font-family, which must be present.

An example containing all the components is:

```
p {
  font: bold italic small-caps 150%/1.4 Arial, Helvetica, sans-serif
}
```

Notice that the 150% means that the font size is one and half time the font size of the parent element. The /1.4 then sets the line height to 1.4 times the font size of the current element. Also notice that the font family is a comma separated list of families in order of preference.

If some of the properties are omitted they are set to their initial values. For example,

```
p {
  font: monospace
}
```

is equivalent to

```
p {
  font-family: monospace;
  font-style:normal;
  font-variant:normal;
  font-weight:normal;
  font-size:medium;
  line-height:normal
}
```

CSS

Keyword

The keyword values refer to the system fonts used by the web browser for items such as menus. The keyword values are:

- caption ; the font used for buttons, drop-down lists, etc.
- icon ; the font used to label icons.
- menu ; the font used by drop-down menus.
- message-box ; the font used by dialog boxes.
- small-caption ; the font used for labelling small controls.
- status-bar ; the font used in the status bar.

The keyword values set all the individual properties to match the corresponding system font. The individual properties can then be overridden by additional declarations.

Example:

```
ul {  
  font:menu; /* Set lists to use a system font. */  
  font-style:italic /* Override an individual property. */  
}
```

Text Spacing

text-align

W3C Specification [CSS1](#) [CSS2.1](#)

This property is used to justify blocks of text. The four possible values are left, right, center and justify. The values are demonstrated below.

CSS rules:

```
div {  
  width:50% /* Make the column narrower so the alignment is more noticeable. */  
}
```

```
p {  
  border:1px solid red;  
  padding:1em  
}
```

```
#para1 {  
  text-align:left  
}
```

```
#para2 {  
  text-align:right  
}
```

```
#para3 {  
  text-align:center  
}
```

CSS

```
#para4 {  
  text-align:justify  
}
```

Sample HTML:

```
<div>  
<p id="para1">Web designers and web surfers are divided over the relative merits of left versus  
  full justification of text. Many state a preference for left justified text despite the  
  fact that most print media is fully justified.  
</p>  
<p id="para2">Web designers and web surfers are divided over the relative merits of left versus  
  full justification of text. Many state a preference for left justified text despite the  
  fact that most print media is fully justified.  
</p>  
<p id="para3">Web designers and web surfers are divided over the relative merits of left versus  
  full justification of text. Many state a preference for left justified text despite the  
  fact that most print media is fully justified.  
</p>  
<p id="para4">Web designers and web surfers are divided over the relative merits of left versus  
  full justification of text. Many state a preference for left justified text despite the  
  fact that most print media is fully justified.  
</p>  
</div>
```

Example rendering:

Web designers and web surfers are divided over the relative merits of left versus full justification of text. Many state a preference for left justified text despite the fact that most print media is fully justified.

Web designers and web surfers are divided over the relative merits of left versus full justification of text. Many state a preference for left justified text despite the fact that most print media is fully justified.

Web designers and web surfers are divided over the relative merits of left versus full justification of text. Many state a preference for left justified text despite the fact that most print media is fully justified.

Web designers and web surfers are divided over the relative merits of left versus full justification of text. Many state a preference for left justified text despite the fact that most print media is fully justified.

This property interacts with the following other properties:

white-space

If the value of text-align is justify when white-space is pre or pre-line the web browser must treat text-align as being left if the current text direction is left to right (this is the normal for English) or right if the current text direction is right to left.

letter-spacing

If this is set to anything other than normal the web browser may only adjust the inter-word spacing when justifying text.

CSS

word-spacing

If this is set to anything other than normal the web browser must try to honor the specified value when justifying text.

letter-spacing

W3C Specification [CSS1](#) [CSS2.1](#)

The letter-spacing property is used to increase or decrease the space between letters. Web browsers are not required to support decreasing the letter spacing or may only allow a limited decrease.

The value normal can be used to remove an inherited value from an element. If the value is set to anything other than normal the spacing between letters can not be altered when using text-align to justify the text. You can set letter-spacing to zero if you don't want the letter spacing altered by justification but don't want to add any extra space.

CSS rules:

```
.increase {  
  letter-spacing:0.5em  
}
```

```
.decrease {  
  letter-spacing:-0.05em  
}
```

Sample HTML:

```
<p class="increase">Letter spacing.</p>  
<p>Letter spacing.</p>  
<p class="decrease">Letter spacing.</p>
```

Example rendering:

Letter spacing.

Letter spacing.

Letter spacing.

word-spacing

W3C Specification [CSS1](#) [CSS2.1](#)

The word-spacing property is used to increase or decrease the space between words. Web browsers are not required to support decreasing the word spacing or may only allow a limited decrease.

The value normal can be used to remove an inherited value from an element.

CSS rules:

```
.increase {
```

CSS

```
word-spacing:1em
}

.decrease {
  word-spacing:-0.2em
}
```

Sample HTML:

```
<p class="increase">The word spacing can be altered.</p>
<p>The word spacing can be altered.</p>
<p class="decrease">The word spacing can be altered.</p>
```

Example rendering:

The word spacing can be altered.

The word spacing can be altered.

The word spacing can be altered.

Other Text Properties

text-decoration

W3C Specification [CSS1](#) [CSS2.1](#)

The text-decoration may either be set to none or any combination of underline, overline, line-through and blink. It is considered bad practice by many web designers to use blink. It is also considered bad practice to underline anything other than a hyperlink. Consider the second and third examples below as examples of what not to do.

If you use line-through to indicate deleted text consider using a semantic element such as (X)HTML's del element either instead of or in combination with this property.

The most common use of text-decoration is to change the appearance of hyperlinks to remove the underline from links in the main content whilst preserving them in navigation menus or vice versa.

CSS rules:

```
a {
  text-decoration:none
}

li a {
  text-decoration:underline
}
```

Sample HTML:

```
<p>The next chapter is rather boring so all but the most enthusiastic readers
  should skip to <a href="Chapter7.htm">the interesting bit</a>.
</p>
```

CSS

```
<ul>
  <li><a href="Chapter4.htm">Previous</a></li>
  <li><a href="Contents.htm">Contents</a></li>
  <li><a href="Chapter6.htm">Next</a></li>
</ul>
```

Example rendering:

The next chapter is rather boring so all but the most enthusiastic readers should skip to [the interesting bit](#).

- [Previous](#)
- [Contents](#)
- [Next](#)

An example of using all four values together. blink is not supported by some browsers so you may be spared that effect.

CSS rule:

```
#overDone {
  text-decoration:underline overline line-through blink
}
```

Sample HTML:

```
<p id="overDone">Can you actually read this?</p>
```

Example rendering:

~~Can you actually read this?~~

Any text-decoration applied to an element is drawn over all of its content including any child elements, regardless of the value text-decoration set on those elements. Any additional decoration specified for child elements will be applied as well as the parent's decoration.

CSS rule:

```
.notImportant {
  text-decoration:line-through;
  color:black
}
```

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

```
strong {
  text-decoration:underline;
  color:red
}
```

Sample HTML:

```
<p class="notImportant">Most of the text in this sentence is not <em>important</em>.</p>
```


CSS

```
<p class="notImportant">The last bit of text in this sentence is  
<strong>very important</strong>.  
</p>
```

Example rendering:

```
Most of the text in this sentence is not important.  
The last bit of text in this sentence is very important.
```

Notice that the underlining in the second paragraph is in red, the color assigned to the strong element that is underlined. However, the line through the strong element is black because it belongs to the parent p element, which has its color set to black.

text-indent

W3C Specification [CSS1](#) [CSS2.1](#)

The text-indent property specifies the indent for the first line of a block level element. Use margin-left, margin-right, padding-left or padding-right to indent an entire block. The indent may be negative but web browsers are not required to support negative indents or may limit the size of a negative indent. The text-indent property is inherited. In the example below the text-indent is applied to the div elements containing the paragraphs rather than directly to the paragraphs themselves.

Note that the indent is only applied at the start of a block level element. Line breaks caused by inline elements do not cause any additional indents to appear within a block.

CSS rules:

```
p {  
  width:50%; /* Narrow the column so that there are more  
             * lines of text in the example. */  
  margin-left:4em /* Apply a margin to the left of all paragraphs  
                * so there is room for the undent example. */  
  margin-top:0.5em /* Separate the two halves of the example slightly. */  
}  
  
.indented {  
  text-indent:2em;  
  border:1px solid red /* Put a border around the first half of the example. */  
}  
  
.undented {  
  text-indent:-2em;  
  border:1px solid blue /* Put a border around the second half of the example. */  
}
```

Sample HTML:

```
<div class="indented">  
<p>This paragraph should have an indent similar to that found in novels.  
However there will still be a vertical gap between the paragraphs because that  
is controlled by the <code>margin</code> property. If you want to recreate a novel-like  
appearance you will need to set both properties.
```

```
</p>
<p>This paragraph has a line break in it, created with the <code>br</code> element.
  Since this doesn't start a new block level element there is no indent or unindent after
  the line break.<br> The line break was immediately before this sentence. You may need
  to adjust the width of your browser window to make the line break obvious.
</p>
</div>
<div class="undented">
<p>This paragraph should have an unindent. This is a more unusual style but it can be useful.
  It is most often used with lists to emphasize the start of a list item either instead of
  or in conjunction with a bullet.
</p>
<p>This paragraph has a line break in it, created with the <code>br</code> element.
  Since this doesn't start a new block level element there is no indent or unindent after
  the line break.<br> The line break was immediately before this sentence. You may need
  to adjust the width of your browser window to make the line break obvious.
</p>
</div>
```

Example rendering:

This paragraph should have an indent similar to that found in novels. However there will still be a vertical gap between the paragraphs because that is controlled by the margin property. If you want to recreate a novel-like appearance you will need to set both properties.

This paragraph has a line break in it, created with the `br` element. Since this doesn't start a new block level element there is no indent or unindent after the line break. The line break was immediately before this sentence. You may need to adjust the width of your browser window to make the line break obvious.

This paragraph should have an unindent. This is a more unusual style but it can be useful. It is most often used with lists to emphasize the start of a list item either instead of or in conjunction with a bullet.

This paragraph has a line break in it, created with the `br` element. Since this doesn't start a new block level element there is no indent or unindent after the line break. The line break was immediately before this sentence. You may need to adjust the width of your browser window to make the line break obvious.

text-transform

W3C Specification [CSS1](#) [CSS2.1](#)

There are four values for the text-transform property:

none

do nothing.

uppercase

converts any lowercase characters to uppercase. Other characters are unaltered.

lowercase

converts any uppercase characters to lowercase. Other characters are unaltered.

capitalize

CSS

converts the first character of each word to uppercase. Other characters are unaltered. In particular uppercase characters elsewhere in a word remain uppercase.

The exact effect of text-transform depends on the language of the element. How the language is determined depends on the document being styled. HTML documents use the `lang` attribute to specify the language for an element. The language is inherited by child elements unless a `lang` attribute is used to alter the language. XML documents use `xml:lang` in the same manner. XHTML 1.0 documents may use either the HTML `lang` attribute or the XML `xml:lang` attribute.

CSS rules:

```
#para1 {  
  text-transform:uppercase  
}
```

```
#para2 {  
  text-transform:lowercase  
}
```

```
#para3 {  
  text-transform:capitalize  
}
```

```
#para4 {  
  text-transform:none  
}
```

Sample HTML:

```
<p id="para1">this TEXT is NOT iN aNy PaRtlcULaR CaSe.</p>  
<p id="para2">this TEXT is NOT iN aNy PaRtlcULaR CaSe.</p>  
<p id="para3">this TEXT is NOT iN aNy PaRtlcULaR CaSe.</p>  
<p id="para4">this TEXT is NOT iN aNy PaRtlcULaR CaSe.</p>
```

Example rendering:

```
THIS TEXT IS NOT IN ANY PARTICULAR CASE.  
this TEXT is NOT iN aNy PaRtlcULaR CaSe.  
This TEXT Is NOT IN ANY PaRtlcULaR CaSe.  
this TEXT is NOT iN aNy PaRtlcULaR CaSe.
```

Compare the effect of setting text-transform to uppercase with setting font-variant to small-caps.

CSS rules:

```
.uppercase {  
  text-transform:uppercase  
}
```

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

CSS

Sample HTML:

```
<p>Compare <span class="uppercase">uppercase</span>  
to <span class="small-caps">small-caps</span>.  
</p>
```

Example rendering:

Compare UPPERCASE to SMALL-CAPS.

vertical-align

W3C Specification [CSS1](#) [CSS2.1](#)

The vertical-align property only applies to elements whose display value is inline or table-cell. The behaviour of vertical-align is significantly different for table-cells and is described in the section on [Cascading Style Sheets/Tables](#).

What not to do

The most common mistake when using vertical-align is to apply it to a block-level element and then wonder why it doesn't work. The following example shows what goes wrong. CSS rules:

```
div#outer {  
  width:100px;  
  height:100px;  
  background-color:#00f;  
  text-align:center  
}  
  
div#inner {  
  width:50px;  
  height:50px;  
  background-color:#f00;  
  margin:auto;  
  vertical-align:middle /* This is ignored because this is a block level element. */  
}
```

Sample HTML:

```
<div id="outer">  
  <div id="inner">?</div>  
</div>
```

Example rendering:



As you can see the red inner division is firmly aligned with the top of the blue division.

Correct usage

The correct usage of vertical-align is to adjust the relative positions of sections of text and embedded content within a line of text. To understand vertical-align you need to understand

CSS

CSS *line boxes*. The following example shows two block-level elements, paragraphs. Each block-level paragraph contains an inline element, span. A border and background are applied to the inline elements so the line boxes are visible.

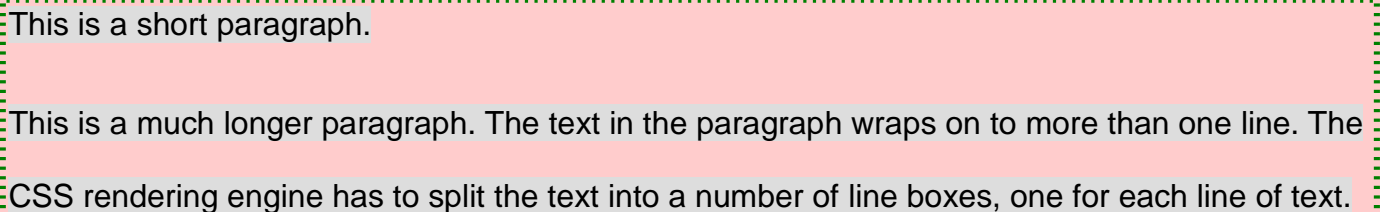
CSS rules:

```
p {
  width:20em; /* Narrow the paragraphs to ensure that the second paragraph wraps. */
  font-size:120%;
  line-height:2; /* Increase the line height to make the gaps between lines clearer. */
  border:3px solid red;
  padding:2px;
  background-color:#fcc
}

span {
  background-color:#ddd
  border:3px dashed #060
}
```

Sample HTML:

```
<p><span>This is a short paragraph.</span></p>
<p><span>This is a much longer paragraph. The text in the paragraph wraps on to
  more than one line. The CSS rendering engine has to split the text into a number
  of line boxes, one for each line of text.</span>
</p>
```



This is a short paragraph.

This is a much longer paragraph. The text in the paragraph wraps on to more than one line. The CSS rendering engine has to split the text into a number of line boxes, one for each line of text.

The vertical-align property adjusts the alignment of inline elements relative to other items in the same line box. The next example illustrates how it can be used to create superscripts and subscripts.

CSS rules:

```
p {
  font-size:150% /* Enlarge the example so it is easier to see. */
}

span {
  font-size:60% /* Superscripts and subscripts are usually shrunk by about 60%. */
}

.super {
  vertical-align:super
}

.sub {
```

CSS

```
vertical-align:sub  
}
```

Sample HTML:

```
<p>Text <span>shrunk</span>.</p>  
<p>Text <span class="super">superscript</span>.</p>  
<p>Text <span class="sub">subscript</span>.</p>
```

Example rendering:

Text shrunk.

Text superscript.

Text subscript.

The full set of values for vertical-align is:

- baseline
- superscript
- subscript
- middle
- text-top
- text-bottom
- top
- bottom

CSS2.1

CSS2.1 also allows the value of vertical-align to be a length or percentage.

white-space

W3C Specification [CSS1](#) [CSS2.1](#)

The white-space property controls the handling of whitespace (normal spaces, tabs, form feeds and newlines but not non-breaking spaces or other special spaces). In the absence of CSS, HTML and XML documents are rendered with runs of whitespace replaced by a single whitespace character. Blocks of text are word-wrapped so that the text fits within the width of the containing block. For example the two paragraphs in the sample HTML below would be rendered in the same way.

```
<p>A run of whitespace is normally replaced by a single whitespace character,  
  e.g. a run of tabs might be replaced by a one normal space.  
</p>  
<p>A run  
of whitespace is normally replaced by a single whitespace character,  
  e.g. a run of tabs might  
be  
replaced  
by a one normal space.  
</p>
```

CSS

Example rendering:

A run of whitespace is normally replaced by a single whitespace character, e.g. a run of tabs might be replaced by a one normal space.

The white-space property has five values. The value normal specifies that text should be handles as usual for an HTML or XML document. The table below shows how the other values compare to normal rendering.

Value	Replace runs of whitespace	Wrap text to fit the container	Start newlines at newlines in the source
normal	Replace	Wrap	No
pre	Preserve	Overflow	Yes
nowrap	Replace	Overflow	No
pre-wrap CSS2.1	Preserve	Wrap	Yes
pre-line CSS2.1	Replace	Wrap	Yes

The two values added in CSS2.1, pre-wrap and pre-line, aren't widely supported at the moment.

Internet Explorer for Windows versions 5.0 and 5.5 does not support pre. IE/win v6.0 only supports pre in [standards mode](#).

The HTML pre element behaves as though the following CSS rule is applied.

```
pre {
  white-space:pre;
  font-family:monospace
}
```

The example below compares normal and nowrap.

CSS rules:

```
p {
  width:15em; /* Make the paragraph narrower so wrapping is more obvious. */
  border:1px solid black; /* Highlight the boundary of the paragraph so overflow is more obvious. */
  background:#cfc
}

#para1 {
  white-space:normal
}

#para2 {
  white-space:nowrap
}
```

Sample HTML:

```
<p id="para1">This first sentence.
  The second sentence.
  The third sentence.
</p>
<p id="para2">This first sentence.
  The second sentence.
  The third sentence.
```

CSS

</p>

This first sentence. The second sentence. The third sentence.

This first sentence. The second sentence. The third sentence.

text-shadow

Text shadow is a new effect that is part of CSS3. It's arguments are the same as box-shadow's and supports multiple shadows.

- text-shadow: x-offset y-offset blur/diffusion color [, repeat for multiple];

```
p {
  text-shadow: 4px 4px 4px black;
  font-size: x-large;
}
```

Text with Shadow! **Bold!** *Italic!* Underline!
Strike!

```
p {
  text-shadow: 2px 2px 1px black;
  font-size: x-large;
}
```

Text with Sharp Shadow! **Bold!** *Italic!* Underline!
Strike!

Now let's try multiple shadows, note that you must keep moving the shadows to prevent them from all being drawn in the same place (unless that is what you want ;)):

```
p {
  text-shadow: 2px 2px 2px red, 4px 4px 2px blue, 6px 6px 2px green;
  font-size: x-large;
}
```

Text with Many Shadows! **Bold!** *Italic!*
Underline! Strike!

Web Fonts

Web fonts are a feature of CSS2 that allow a rule to be set to allow an embedable font in a web

CSS

page, freeing a designer from having to rely on a subset of fonts that ship with most OSes. Fonts supported by this feature are either [TrueType](#) (.tff) or [OpenType](#) (.otf). Web fonts are currently only supported by WebKit (Safari, Shiira, Chrome, *et al.*). Internet Explorer supports web fonts only with the [Embedded Open Type](#) (.eot) format, something unused by any other browser.

Sample CSS:

```
@font-face {
  font-family: WikiFont;
  src: url('wikifont.otf');
}

@font-face {
  font-family: WikiFont;
  font-weight: bold;
  src: url('wikifont-bold.otf');
}

@font-face {
  font-family: WikiFont;
  font-style: italic;
  src: url('wikifont-italic.otf');
}
```

Now, to use the font we simply go about business as usual:

```
h1, h2, h3, h4, h5, h6 {
  font-family: WikiFont;
}
```

Then any headings in that document would display in our example "WikiFont" in a supporting browser.

Hyperlinks

The `a` selector matches hyperlinks. There are the four pseudo-classes that are commonly used with it: `link`, `visited`, `hover` and `active`. Rules using these pseudo-classes should be given in this order so that the dynamic effects take precedence over the visited/unvisited state. The following example will display a link that is black, changes to green when the mouse is over it, and changes to pink when active (whilst the mouse is pressed over it).

```
a:link
{ color:black; background:gray }
a:visited
{ color:black; background:red }
a:hover
{ color:green }
a:active
{ color:pink }
```

When the mouse is over an unvisited link both the `a:link` selector and the `a:hover` selector are matched. Since the hover rule appears last its color declaration takes precedence over the link rule's color declaration. Since the hover rule does not alter the background property the result is green text on a gray background.

The `:link` and `:visited` pseudo-classes can only be used with the `a` selector. CSS2.1 allows the `:hover` and `:active` pseudo-classes to be used with other elements. However, Internet Explorer only allows `:hover` on anchor elements.

Lists

list-style-type

W3C Specification [CSS1](#) [CSS2.1](#)

The `list-style-type` property sets the marker used at the start of a list item. The marker may be a counter or a bullet point. Several different bullet points are available. The values defined in CSS1 or CSS2.1 are shown in the list below. The marker for each value in the list should be rendered in that style. Your browser may not support all the bullet types, in which case you will probably see a disc in place of any unsupported values.

- disc
- circle
- square
- decimal
- lower-roman
- upper-roman
- lower-alpha
- upper-alpha
- decimal-leading-zero, defined in CSS2.1.
- lower-latin, defined in CSS2.1.
- upper-latin, defined in CSS2.1.
- armenian, defined in CSS2.1 but [liable to be removed](#)
- georgian, defined in CSS2.1 but [liable to be removed](#).
- lower-greek, defined in CSS2.1 but [liable to be removed](#).
- none

`list-style-type` can be applied to the list container element (`ul` or `ol` in HTML) as well as to the list item element (`li` in HTML).

CSS rules:

```
ul {  
  list-style-type:disc  
}
```

CSS

Sample HTML:

```
<ul>
<li>Item A</li>
<li>Item B
<ul>
<li>Item B1</li>
<li>Item B2
<ul>
<li>Item B2a</li>
<li>Item B2b</li>
</ul>
</li>
</ul>
</li>
</ul>
```

Example rendering:

- Item A
- Item B
 - Item B1
 - Item B2
 - Item B2a
 - Item B2b

Notice that all the bullets in the sub-lists are discs. If you want sub-lists to have different types you need extra rules, e.g.

```
ul {
list-style-type:disc
}
```

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

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

Example rendering:

- Item A
- Item B
 - Item B1
 - Item B2
 - Item B2a
 - Item B2b

list-style-image

W3C Specification [CSS1](#) [CSS2.1](#)

The list-style-image property allows you to specify an image to use as the bullet for list items. The value is either a URI or none. When the value is none the bullet will be set by the list-style-type property. When it is a URI the web browser will attempt to load the image from the given URI. If it succeeds it will use the image as the bullet. If it fails it will use the bullet specified by the list-style-type property.

The example below shows a list with alternating red and green bullets.

CSS rules:

```
li.odd{
  list-style-image: url(green_bullet.gif)
}

li.even {
  list-style-image: url(red_bullet.gif)
}
```

Sample HTML:

```
<ul>
  <li class="odd">Item 1</li>
  <li class="even">Item 2</li>
  <li class="odd">Item 3</li>
  <li class="even">Item 4</li>
</ul>
```

Example rendering:

- Item 1
- Item 2
- Item 3
- Item 4

list-style-position

W3C Specification [CSS1](#) [CSS2.1](#)

The list-style-position property determines whether the bullet for a list item is considered part of the content of the list item (inside) or is placed outside the block (outside). Most web browsers place the bullet outside for (X)HTML lists.

CSS rules:

```
li {
  border: 1px solid red
}

ul#inner {
  list-style-position: inside
}
```

CSS

```
}  
  
ul#outer {  
  list-style-position: outside  
}
```

Sample HTML:

```
<p>The first list has the bullets on the inside.</p>  
<ul id="inner">  
  <li>This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
  </li>  
  <li>This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
  </li>  
  <li>This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
  </li>  
</ul>  
<p>The second list has the bullets on the outside. This is the default.</p>  
<ul id="outer">  
  <li>This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
  </li>  
  <li>This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
  </li>  
  <li>This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
    This text needs to be long enough to wrap on to the next line.  
  </li>  
</ul>
```

Example rendering – notice where the text wraps to relative to the bullet and the border.

The first list has the bullets on the inside.

- This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line.
- This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line.
- This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line.

CSS

The second list has the bullets on the outside. This is the default.

- This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line.
- This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line.
- This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line. This text needs to be long enough to wrap on to the next line.

The next example shows how altering the margin or padding affects the position of the bullet when the bullet is inside. Since the bullet is inside the content the padding appears between the border and the bullet.

- Margin 0
- Margin 1em
- Margin 2em
- Padding 0
- Padding 1em
- Padding 2em

The next example shows how altering the margin or padding affects the position of the bullet when the bullet is outside. Since it is outside the content it remains in the same place relative to the border.

- Margin 0
- Margin 1em
- Margin 2em
- Padding 0
- Padding 1em
- Padding 2em

Shorthand property

W3C Specification [CSS1](#) [CSS2.1](#)

The list-style shorthand property can be used to set all three individual list style properties in a single declaration.

The rule

```
ul {  
  list-style:circle url(wavy_circle.png) inside  
}
```

is equivalent to

CSS

```
ul {  
  list-style-type:circle;  
  list-style-image:url(wavy_circle.png);  
  list-style-position:inside  
}
```

The individual properties may be given in any order, so the following declarations are all equivalent.

```
list-style:circle url(wavy_circle.png) inside;  
list-style:url(wavy_circle.png) circle inside;  
list-style:inside circle url(wavy_circle.png);
```

If any of the individual properties are omitted they are set to their initial values. For example

```
ul {  
  list-style:url(wavy_circle.png)  
}
```

is equivalent to

```
ul {  
  list-style-image:url(wavy_circle.png);  
  list-style-type:disc; /* initial value */  
  list-style-position:outside /* initial value */  
}
```

Box Model

Block-level and inline elements – the display property

W3C Specification [CSS1](#) [CSS2.1](#)

CSS has no understanding of the semantics of the underlying document. In particular it has no idea that the p elements in the HTML fragment below should start on a new line whilst the em element should not.

```
<p>This paragraph contains an <em>emphasised</em> word.</p><p>This paragraph does not.</p>
```

The display property provides this information.

CSS1 defines four values for display. CSS2.1 adds two more general purpose values plus a number of values for creating tables. The table values are described in [Displaying Tables](#).

- block – this causes the element to start a new line.
- inline
- list-item
- none
- inline-block (CSS2.1)

CSS

- run-in (CSS2.1)

The values are explained in detail in later sections.

Note: in CSS1 the initial value for display is block. Under CSS2.1 this changes to inline. This property should be explicitly set for all element types that can occur in a document to ensure consistency between different implementations. Most web browsers will set appropriate values for elements in (X)HTML documents. (See [Appendix D](#) of the CSS 2.1 Specification for the suggested values.) In XML documents use a rule such as

```
* {  
  display: block  
}
```

to set the display property for all element types. Then use more specific rules to set the value for individual element types.

The display value affects the set of other properties that are valid for an element. For example the list properties are only valid if display is list-item. In the example below no bullets appear on the list items because display has been set to inline.

CSS rule:

```
li {  
  display:inline;  
  list-style-type:circle  
}
```

Sample HTML:

```
<ul>  
  <li>Parsley, </li>  
  <li>sage, </li>  
  <li>rosemary and </li>  
  <li>thyme.</li>  
</ul>
```

Example rendering:

Parsley, sage, rosemary and thyme.

block

An element with display set to block starts a new line. It also acts as a container for the content of its child elements.

inline

Elements with display set to inline flow with other content within the containing box of their nearest block ancestor. They do not start new lines.

The width and height properties are not valid on non-replaced inline elements. In (X)HTML documents replaced elements are embedded content such as images.

CSS

The alignment of inline elements is controlled by the vertical-align property. The initial value for this property is baseline. Since (X)HTML images are normally inline elements this means that the bottom edge of images are aligned with the baseline of the text. If there is no text on the line the web browser must still calculate where the baseline would be and leave the appropriate amount of room for it. This can cause problems when trying to align images.

list-item

list-item is similar to block except that it enables the list properties list-style-type, list-style-image, list-style-position and list-style.

none

When display is none the web browser must act as though the element and all its child elements do not exist. This can't be overridden by setting the display property on the child elements because they don't exist to have properties applied to them. The web browser must not leave any space on the display or print out or speak any of the content or give any other hint of the existence of the element.

Hiding elements with this value can be very useful in conjunction with CSS2.1 [media types](#). For example you can specify that the navigation bar is removed from the printed page by setting its display value to none for print media.

If you want to simply hide the element whilst leaving a visible gap set the CSS2.1 property visibility to hidden.

inline-block

An inline-block element flows in the same manner as an inline element but it acts as a block container for its own children.

run-in

run-in allows a single element to be merged into the start of a block element that follows it immediately. The run-in element will only be merged if it doesn't contain any block elements. If a run-in element can't be merged it acts as though it was a block element.

CSS rule:

```
h1, h2 {
  display:run-in;
  margin-right:0.5em
}
```

```
h1 {
  font-size:120%
}
```

```
h2 {
  font-size:100%
}
```

CSS

Sample HTML:

```
<h1>Main heading</h1>
<h2>Sub-heading</h2>
<p>Only the sub-heading is allowed to run into this paragraph.
  The heading can't be merged because only one <code>run-in</code> element is allowed.
</p>
```

Example rendering:

Heading

Sub-heading Only the sub-heading is allowed to run into this paragraph. The heading can't be merged because only one run-in element is allowed.

Interaction with float

If the float property for an element is set to anything other than none the element behaves as though display is set to block regardless of its actual value.

The Box Model

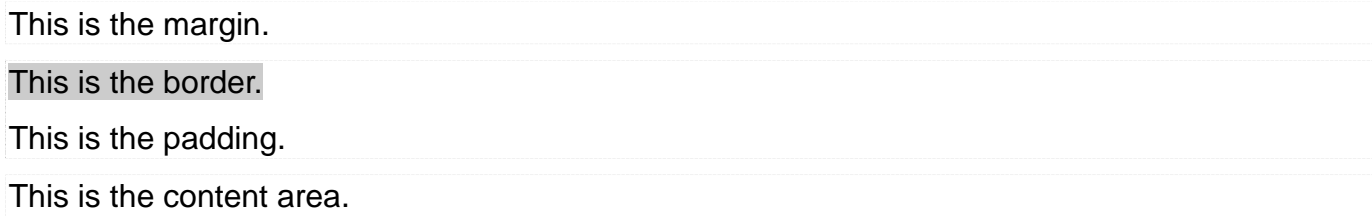
The CSS box model is fundamental to laying out documents with CSS. The content of every element is rendered within one or more boxes. A CSS box consists of:

- the content area,
- surrounded by padding (optional),
- surrounded by borders (optional),
- surrounded by margins (optional).

If all three optional parts of the box are present then crossing the box from one outer margin edge to the content the following boundaries are encountered:

- outer margin edge
- outer border edge = inner margin edge
- outer padding edge = inner border edge
- content edge = inner padding edge

as illustrated by the following diagram.



Height and Width

- height W3C Specification [CSS1](#) [CSS2.1](#)

CSS

- width W3C Specification [CSS1](#) [CSS2.1](#)

width is ignored by non-replaced inline elements, table columns and table column groups.

height is ignored by non-replaced inline elements, table rows and table row groups.

The width and height properties specify the dimensions of the content area (but see the section on quirks mode below). This is the distance from the inner edge of the padding on one side to the inner edge of the padding on the other side. The values may be given as non-negative [lengths](#) or percentages. If width is given as a percentage the value is based on the width of the containing block. If height is given as a percentage the value is based on the height of the containing block. If the height of the containing block is not an absolute value percentages may not work as expected. Refer to the W3C Specification for details.

It is usual to give the size of elements containing text in ems or percentages. Images and other replaced elements are normally sized in pixels (px).

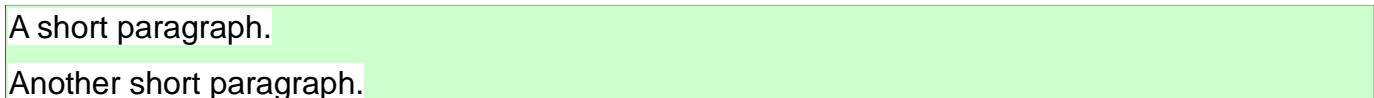
CSS rule:

```
p {
  width:50%;
  height:3em;
  background:#cfc /* Set a background so the size of the content area is obvious. */
}
```

Sample HTML:

```
<p>A short paragraph.</p>
<p>Another short paragraph.</p>
```

Example rendering:



Notice that the left edge of the text (the contents) touches the left edge of the background (the content area).

If the content area is too small the contents will overflow the box (except in Internet Explorer).

CSS rule:

```
p {
  width:8em;
  height:3em;
  background:#cfc /* Set a background so the size is of the content area is obvious. */
}

#para2 {
  color:red /* Make the text of the second paragraph stand out. */
}
```

Sample HTML:

```
<p>A paragraph that is far too long for the tiny box it is supposed to fit inside.</p>
<p id="para2">A short paragraph.</p>
```

CSS

Example rendering (this example does not work in Internet Explorer):

A paragraph that is far too long for the tiny box it is supposed to fit inside.

A short paragraph.

CSS2.1 minimum and maximum width and height

- min-height W3C Specification [CSS2.1](#)
- max-height W3C Specification [CSS2.1](#)
- min-width W3C Specification [CSS2.1](#)
- max-width W3C Specification [CSS2.1](#)

CSS2.1 introduces four extra properties min-width, max-width, min-height and max-height. For example, suppose you are creating a layout with two columns side by side on the page. You want the column for the main content to be 27em wide but no more than 75% of the page width. The navigation column should be 9em or 25%. The appropriate rules would be:

```
#content {  
  width:27em;  
  max-width:75%  
}
```

```
#navigation {  
  width:9em;  
  max-width:25%  
}
```

The min-width property is often used with a length in pixels to ensure that a column does not become too narrow for the images it contains.

Internet Explorer does not support these CSS2.1 properties.

Internet Explorer Quirks Mode

Internet Explorer versions 5.0 and 5.5 allow width and height on non-replaced inline elements. Internet Explorer version 6 also applies these properties if it is in [quirks mode](#). In [standards mode](#) it correctly ignores them on non-replaced inline elements.

In quirks mode width and height incorrectly set the distance between the outer edges of the borders not the edges of the content area.

CSS 3 Box Models

The current draft of [CSS3](#) introduces a new property, box-sizing. This property takes one of two values, content-box or border-box.

When the value is set to content-box the normal CSS1/CSS2.1 box model is used so width and height define the size of the content area.

When the value is set to border-box the width and height properties set the distance between the outer edges of the borders, i.e. the box behaves like a quirks mode box.

CSS

Mozilla Firefox does not support box-sizing but instead uses `-moz-box-sizing` for the same purpose.

Padding

Padding is blank space inserted between the content of the element and its border. The element's background is seen through the padding. The padding is set by four properties:

- `padding-top` W3C Specification [CSS1](#) [CSS2.1](#)
- `padding-right` W3C Specification [CSS1](#) [CSS2.1](#)
- `padding-bottom` W3C Specification [CSS1](#) [CSS2.1](#)
- `padding-left` W3C Specification [CSS1](#) [CSS2.1](#)

The padding can either be set to a length, e.g. 1em, or a percentage, e.g. 5%. Values must not be negative. Percentages are relative to the *width* of the containing block even for the top and bottom padding.

CSS rules:

```
p {
padding-top:1em;
padding-right:2em;
padding-bottom:3em;
padding-left:4em;
background-color:#fc9;
width:10em;
text-align:justify
}
```

Sample HTML:

```
<p>This text has padding around it. The orange background
  appears behind both the padding and the content.
</p>
```

Example rendering:

This text has padding around it. The orange background appears behind both the padding and the content.

Shorthand property

W3C Specification [CSS1](#) [CSS2.1](#)

The padding on all four sides can be set at once using the padding shorthand property. This takes a list of whitespace separated values. The first value is the top padding. The other values work clockwise round the sides, so right is second, bottom is third and left is last. The previous example could be shortened to:

```
p {
padding:1em 2em 3em 4em;
background-color:#fc9;
width:10em;
```

CSS

```
text-align:justify
}
```

You can specify fewer than four values in the list for padding. If the value for the left padding is missing it is set equal to the right padding. If the value for the bottom padding is missing it is set equal to the top padding. If only the top padding is given all the sides are given the same padding.

```
p {
padding:1em 2em 3em 4em; /* top = 1em, right = 2em, bottom = 3em, left = 4em */
}
```

```
p {
padding:1em 2em 3em; /* top = 1em, right = 2em, bottom = 3em, left = right = 2em */
}
```

```
p {
padding:1em 2em; /* top = 1em, right = 2em, bottom = top = 1em, left = right = 2em */
}
```

```
p {
padding:1em; /* top = 1em, right = top = 1em, bottom = top = 1em, left = top = 1em */
}
```

padding always sets the padding on all four sides. If you want to alter the padding on only some of the sides you must use the individual properties.

```
p {
padding:0.5em
}
```

```
p.narrow {
padding-left:1.5em;
padding-right:1.5em
}
```

Margins

Margin is the outermost layer in the CSS box model, located outside of border.

CSS properties that enable setting of the width of the margin:

- margin-top W3C Specification [CSS1](#) [CSS2.1](#)
- margin-right W3C Specification [CSS1](#) [CSS2.1](#)
- margin-bottom W3C Specification [CSS1](#) [CSS2.1](#)
- margin-left W3C Specification [CSS1](#) [CSS2.1](#)

Shorthand property

W3C Specification [CSS1](#) [CSS2.1](#)

Box shadow

Box shadow is a part of CSS3. It creates a [drop shadow](#) that follows the shape of the box. It is currently supported by WebKit and Gecko. Box shadow takes 4 arguments and supports multiple shadows.

CSS3	WebKit	Gecko
------	--------	-------

box-shadow	-webkit-box-shadow	-moz-box-shadow
------------	--------------------	-----------------

- box-shadow: X-offset Y-offset blur/diffusion color;

This box should have a sharp rectangular shadow to the lower right.

```
div {  
  border: 1px solid;  
  box-shadow: 10px 10px 0px black;  
  -webkit-box-shadow: 10px 10px 0px black;  
  -moz-box-shadow: 10px 10px 0px black;  
  padding: 0.5em;  
}
```

This box should have a sharp round shadow to the lower right.

```
div {  
  border: 1px solid;  
  box-shadow: 10px 10px 0px black;  
  -webkit-box-shadow: 10px 10px 0px black;  
  -moz-box-shadow: 10px 10px 0px black;  
  padding: 0.5em;  
  border-radius: 10px;  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
}
```

This box should have a red diffuse round shadow to the upper left.

```
div {  
  border: 1px solid;  
  box-shadow: -10px -10px 5px red;  
  -webkit-box-shadow: -10px -10px 5px red;  
  -moz-box-shadow: -10px -10px 5px red;  
  padding: 0.5em;  
  border-radius: 10px;  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
}
```

This box should have a red, green and blue diffuse round shadows to the lower right.

```
div {  
  border: 1px solid;  
  box-shadow: 10px 10px 5px red, 15px 15px 5px green, 20px 20px 5px blue;  
  -webkit-box-shadow: 10px 10px 5px red, 15px 15px 5px green, 20px 20px 5px blue;  
  -moz-box-shadow: 10px 10px 5px red, 15px 15px 5px green, 20px 20px 5px blue;  
  padding: 0.5em;  
  border-radius: 10px;  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
}
```

Box Sizing

[Box-sizing](#) is a CSS3 property to make certain layouts simpler. Box-sizing is supported by WebKit, Gecko and Presto.

box-sizing take 1 argument that is any of the following:

- content-box
boxes inside arrange normally.
- border-box
boxes inside arrange around the border.
- inherit
inherit the parent behavior.

W3C	WebKit	Gecko
box-sizing	-webkit-box-sizing	-moz-box-sizing

CSS fragment:

```
div.bigbox {  
  width: 40em;  
  border: 1em solid red;  
  min-height: 5em;  
}
```

```
div.littlebox {  
  width: 50%;  
  border: 1em solid;  
  box-sizing: border-box; /* this will set the boxes to flow along the border of the containing box */  
  -webkit-box-sizing: border-box;  
  -moz-box-sizing: border-box;  
  float: left;  
}
```


CSS

HTML fragment:

```
<div class="bigbox">  
  <div class="littlebox" style="border-color: green;">This should be on the left.</div>  
  <div class="littlebox" style="border-color: blue;">This should be on the right.</div>  
</div>
```

This should be on the left.
This should be on the right.

The two boxes should be above this text side by side if your browser supports box-sizing.

Background

Background properties

There are five properties that control the background of an element:

`background-color` W3C Specification [CSS1](#) [CSS2.1](#)

sets the background color, this will be seen if no background image is specified or the image does not cover the entire element. It will also show through any transparent areas in the image. The color may be set to any [CSS color](#) or the keyword transparent.

`background-image` W3C Specification [CSS1](#) [CSS2.1](#)

the image to use as a background.

`background-repeat` W3C Specification [CSS1](#) [CSS2.1](#)

controls the tiling of the background image (possible values: repeat | repeat-x | repeat-y | no-repeat | inherit)

`background-attachment` W3C Specification [CSS1](#) [CSS2.1](#)

does the background scroll with the page (the default) or is it fixed within the viewport?

`background-position` W3C Specification [CSS1](#) [CSS2.1](#)

places the image relative to the element's bounding box.

You should consider supplying a background color as well as a background image in case the user has disabled images.

Examples:

```
p {  
  background-color: #ccc  
}
```

```
body {  
  background-color: #8cf;  
  background-image: url(sea.jpg);  
  background-repeat: no-repeat;  
  background-attachment: fixed;
```

CSS

```
background-position: bottom
}
```

Shorthand property

W3C Specification [CSS1](#) [CSS2.1](#)

The background shorthand property can be used to set all five individual background properties in a single declaration, e.g.

```
p {
  background-color: #ccc
}
```

```
body {
  background: #8cf url(sea.jpg) no-repeat fixed bottom
}
```

is equivalent to the previous example.

Note that if some values are omitted from the background property the individual properties for the missing values are set to their initial values. This is important to remember if more than one rule is setting the background properties on an element. For example

```
div#navigation {
  background: #933
}
```

is equivalent to

```
div#navigation {
  background-color: #933;
  background-image: none; /* initial value */
  background-repeat: repeat; /* initial value */
  background-attachment: scroll; /* initial value */
  background-position: 0% 0% /* initial value */
}
```

Advanced Opacity

Opacity is a block level property that effects all the children. To use colors with alpha channels, see [RGBA](#) and [HSLA](#).

Opacity in the background property is still a new concept and is not compliant across all web browsers however here is a tiny bit about this function

```
filter: alpha(opacity=40); -moz-opacity:.40; opacity:.40;
```

CSS

adding the alpha(opacity=40) will add a 40% alpha transparency however as mentioned this still does not work on all browsers, for this to work on Firefox for example the -moz hack has to be used . So -moz-opacity:.40 will obviously do the same thing but this code is only relevant for Firefox users.

transparency W3C Specification [CSS3](#)

External links

[A nice demonstration of the effects that can be achieved with backgrounds on css/edge](#)

[A nice tutorial on the use of opacity in the background.](#)

CSS Shorthand Properties

[Cascading Style Sheets/Shorthand](#)

Positioning Elements

Position Schemes

Instead of using table(s) to layout a page you can use CSS positioning. By using the CSS positioning you can make a page more dynamic. Positioning is not compatible with all browsers, so when coding it is necessary to know your audience.

Types of positioning:

- Normal Flow/Static Positioning
- Relative Positioning
- Absolute Positioning
- Fixed Positioning

The examples below are all using basically the same XHTML document. Only the CSS positioning changes. To make it easy to see what is happening with the CSS there are colored borders around the two blocks. The outer block has a blue border (#0000FF, abbreviated #00F).The inner block has a red border (#FF0000, abbreviated #F00). Only the rule for the inner block changes in these examples.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Standard XHTML</title>
<style type="text/css">
#main {
```

CSS

```
border: 1px solid #00F;
}

#content { /* This rule changes in the example below. */
border: 1px solid #F00;
}
</style>
</head>
<body>
<div id="main">
<p>Lorem ipsum pro ne apeirian cotidieque, eu per nihil everti ornatus.
At nam iudico dolore suavitate. Harum quaeque consetetur ei usu, ius et
impetus aliquid consequat, at pro nullam oporteat partiendo. Sed ea nonummy
suscipiantur. Nec libris erroribus scriptorem ut.</p>
<div id="content">
<p>His ne albucius liberavisse definitionem. His eu kasd eligendi
invidunt, et prima legimus adolescens mea. Nonummy aliquid pri et, qui
ex dicant nostrum moderatius. Eam in malorum efficiantur, falli eleifend
cotidieque qui ne. At modus vituperata dissentiet has. Mel postea aeterno
diceret eu, eu postea laoreet sea, nam te meliore platonem necessitatibus.</p>
</div>
<p>Vix in causac adipisci, dicit facete vulputate te mel. Et vis noster
admodum mediocritatem, quaeque mnesarchum sea id. Quas vocibus id qui. Ne
delenti iracundia conitutum sed, erudin invenire consulanu usu in. Vero
legimus duo ex, his no suscipit vituperata delicatissimi.</p>
</div>
</body>
</html>
```

Normal Flow

Normal Flow is the default behavior of a web browser. You do not specify this in your style sheet since it is the default. With normal flow boxes will show up in the order that you placed them in your code, and each box level element is stacked on the next.

```
<style type="text/css">
#main {
border: 1px solid #00F;
}

#content {
border: 1px solid #F00;
}
</style>
```

Static Positioning

Static positioning is applied by the declaration `position: static`. This places the element in the normal flow. Since normal flow is the default it is not normally necessary to explicitly use this.

Where it is useful is over-riding another rule of lower specificity, e.g.

```
div { position: absolute; }
```

CSS

```
#notMe { position:static; }
```

would absolutely position all div elements except the one whose id attribute has the value *notMe*.

The left, top, right and bottom properties are not needed since they don't influence static positioning. They are used below to show they have no influence.

```
<style type="text/css">
#main {
border: 1px solid #00F;
}

#content {
border: 1px solid #F00;
position: static;
left: 100px;
top: 125px;
right: 50px;
bottom: 30px;
}
</style>
```

Relative Positioning

The browser first lays out the element as though it was in the normal flow. The element is then displaced by the amount specified by the left or right properties and the top or bottom properties. A gap is left in the normal flow at the point the element should have appeared. Relative positioning does not allow an element to change size. If both left and right are specified right will be ignored in languages which are written left to right such as English. bottom is ignored if top is specified.

```
<style type="text/css">
#main {
border: 1px solid #00F;
}

#content {
border: 1px solid #F00;
position: relative;
left: 100px;
top: 125px;
right: 50px;
bottom: 30px;
}
</style>
```

Absolute Positioning

This positions a box relative to its containing block. However, unlike relative positioning the gap in the normal flow left by removing the element closes up. The containing block is the nearest ancestor with a 'position' of 'absolute', 'relative' or 'fixed'.

CSS

You can use any one or combination of left, top, right, and bottom properties to position the box. The co-ordinates for absolute position have (0,0) at the top left of the containing block. Increasing the value of top moves the element down the page.

Since absolutely positioned boxes are taken out of the normal flow they can be positioned anywhere on the page regardless of their position in the document's source order.

```
<style type="text/css">
#main {
border: 1px solid #00F;
}

#content {
border: 1px solid #F00;
position: absolute;
left: 100px;
top: 125px;
right: 50px;
bottom: 30px;
}
</style>
```

Fixed Positioning

Fixed positioning is a subcategory of absolute positioning. The only difference is that for a fixed positioned box, the containing block is established by the browser window size. A fixed element does not move when a web page is scrolled as all other elements do. It is calculated in the same way as absolute positioning with respect to containing blocks in that it pulls the positioned box out of the normal flow.

```
<style type="text/css">
#main {
border: 1px solid #00F;
}

#content {
border: 1px solid #F00;
position: fixed;
left: 100px;
top: 125px;
right: 50px;
bottom: 30px;
}
</style>
```

Floating Elements

Elements can be made to float within the normal flow. Boxes are moved left or right as far as they can go. Elements after the float box will move up to fill any gap left behind thus flowing around the box with the float position.

Notice that float is not a position property, but it acts like one. Float is applied with the float

CSS

property not the position property. The float is positioned to the left in the example, but you could easily make it positioned to the right. A width was added to the content block so you can see how elements after the float block move up and wrap around the area the content block does not occupy.

You must set the width property when floating block-level elements otherwise they will expand to fill the entire width of their container.

```
<style type="text/css">
#main {
  border: 1px solid #00F;
}

#content {
  border: 1px solid #F00;
  left: 100px;
  top: 125px;
  right: 50px;
  bottom: 30px;
  float: left;
  width: 425px;
}
</style>
```

- -
- Float Positioning

Tables

Displaying Tables

W3C Specification [CSS2.1](#)

The display property is extended with the following values by CSS2.1:

- table
- inline-table
- table-row-group
- table-header-group
- table-footer-group
- table-row
- table-column-group
- table-column
- table-cell
- table-caption

Adjacent borders of cells can be made one border thin using the border-collapse property, like "border-collapse:collapse;".

The spacing between the borders of cells can be set using the border-spacing property, since CSS2. The setting of the property has no effect if borders have been made look one border thin

using "border-collapse:collapse;".

Floating Elements

IE6 and IE7 will place a floated element below an inline box that is its previous sibling

IE6 and IE7 will place an inline element below a floated element that is its previous sibling if it is floated to the right of that element

IE6 and IE7 will shrinkwrap an element styled float:left to the width of its containing block if the container has a right-floated child and an undefined width

IE6 and IE7 cleared elements will clear other nested floats even if they don't share a containing block

IE6 and IE7 cleared elements will have twice their declared top padding when displayed

A floated box can overlap block-level boxes adjacent to it in the normal flow, so use the clear property to reposition static boxes in the flow

The clear property can be used on a floated element to force it below adjacent floats, which is useful for aligning multiple blocks floating in the same direction

The clear property specifies if an element can have elements floated to its left, to its right, or not at all. When an element does not allow floating on a side, the element appears below the floated element.

Internet Explorer versions up to and including 6 add three pixels of padding (in the floated direction) to adjacent line boxes.

In Internet Explorer versions up to and including 6, the left or right margins are doubled on floated elements that touch their parents' side edges. The margin value is doubled on the side that touches the parent. A simple fix for this problem is to set display to inline for the floated element.

Internet Explorer for Windows versions up to and including 7: will place a floated box below an immediately preceding line box will expand a left-floated box to the width of the containing block if it has a right-floated child and a computed width of auto will apply a layout to a floated element don't support the value inherit for any properties except visibility and direction

In Firefox versions up to and including 2.0, a floated box appears below an immediately preceding line box. A left-floated box with a right-floated child and a computed width of auto expands to the width of the containing block.

In Opera up to and including version 9.2, if the computed width of the floated box is auto and it has floated children, its width is computed as if the floats don't wrap and aren't cleared.

Use overflow: auto; display: block; float: left; margin: 0 auto; to position block elements onto the same line.

Floated elements are shrinkwrapped by surrounding block content unless overflow is set to auto or hidden on the surrounding blocks, in which case they are treated as inline blocks.

CSS

Floated elements following an absolutely positioned element will cause it to disappear in IE6 and IE7 unless the absolutely positioned element is set to clear:both

Instead of applying top margins to cleared static content apply bottom margins to floats for cross-browser consistency.

Floated elements do not support margin collapsing

Inline elements do not support clearing

Absolutely positioned elements do not support floating

Floating elements do not support inline-block or inline-table, each is rendered as block or table respectively

Floated elements cause adjacent inline elements to reflow

Example 1

CSS rule:

```
.sidenote {  
  float:left;  
  width:10em;  
  padding:0.5em;  
  border:1px solid green;  
  margin:0.5em 1em 0.2em 0  
}
```

Sample HTML:

```
<p class="sidenote">This sentence is a note inserted to one side of the main text.</p>  
<p>  
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod  
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,  
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo  
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse  
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat  
non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.  
</p>
```

Example rendering:

This sentence is a note inserted to one side of the main text.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

CSS

Example 2

Multiple objects can be floated within the same block

CSS rule:

```
.outerBlock {  
  float:left;  
  width:100%;  
  background-color:yellow  
}
```

```
.shortBlock {  
  float:left;  
  width:30%;  
  height:1em;  
  margin:2px;  
  background-color:red  
}
```

```
.tallBlock {  
  float:left;  
  width:30%;  
  height:2em;  
  margin:2px;  
  background-color:blue  
}
```

Sample HTML:

```
<div class="outerBlock">  
<div class="shortBlock"></div>  
<div class="tallBlock"></div>  
<div class="shortBlock"></div>  
<div class="tallBlock"></div>  
<div class="shortBlock"></div>  
<div class="tallBlock"></div>  
<div class="shortBlock"></div>  
<div class="tallBlock"></div>  
</div>
```

Example rendering:



Property clear

The property *clear* determines which sides of the floating element can be occupied by any of the *previous* floating elements. The property has no effect on *next* floating elements.

Possible values:

- left
- right
- both

CSS

- none

To understand the effect, compare the following examples featuring one non-floating element and two floating elements.

In the **first example**, the property *clear* is not used.

```
<p style="float: left; border:1px solid green;">Floating box 1.</p>
<p style="float: left; border:1px solid green;">Floating box 2.</p>
<p>
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur.
</p>
```

Floating box 1.
Floating box 2.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

In the **second, example**, the property *clear* is used in the second element to make sure no previous floating element comes to the left of it.

```
<p style="float: left; border:1px solid green;">Floating box 1.</p>
<p style="float: left; clear: left; border:1px solid green;">Floating box 2.</p>
<p>
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur.
</p>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.	
Floating box 1.	Floating box 2.

In the **third example**, the float property has no effect, as it only affects *previous* floating elements.

```
<p style="float: left; clear: both; border:1px solid green;">Floating box 1.</p>
<p style="float: left; border:1px solid green;">Floating box 2.</p>
<p>
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
```

CSS

cillum dolore eu fugiat nulla pariatur.
</p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.	
Floating box 1.	Floating box 2.

Links

A few tips:

- [a short discussion of "legacy" vs. "XML" styles of "clear" in CSS.](#)
- ["Clearing space beneath floated elements"](#)
- [If you want to add, say, a border around all floats ...](#)
- [W3.org: "Controlling flow next to floats: the 'clear' property"](#)

Editor's note

- Floating is also covered at [Cascading Style Sheets/Positioning](#)

Media Types

To do:

Format; flesh out.

Individual CSS rules and whole style sheets can be made apply selectively only to certain media types, such as screen, projection, print or handheld.

Specification of media types:

- In the LINK tag
 - `<LINK rel="stylesheet" type="text/css" media="screen, projection" href="stylesheet.css">`
- Within the style sheet
 - `@media screen { body { font-family: Verdana; width: 800px; } }`
- Within the style attribute

Media types

Media types by which the application of CSS can be restricted:

- all
- aural – is generally used for rules for the hearing impaired. In CSS3 it is being deprecated in favor of the speech media type.
- braille
- embossed
- handheld
- print
- projection
- screen
- speech
- tty
- tv

Media Queries

Media queries are a new rule format under CSS3 that will allow for behaviors based on properties of the user agent, such as the screen resolution of mobile devices. This feature is only partially implemented in the latest builds of WebKit, Presto and Gecko.

To try this out copy this code to your snippet editor or a new HTML file:

```
@media all {
  div.mediaqueryexample {
    height: 10em;
    width: 10em;
    background-color: red;
  }
}

@media all and (min-width: 640px) {
  div#mediaqueryexample1 {
    background-color: green;
  }
}
```

And then put this in the body:

```
<div class="mediaqueryexample" id="mediaqueryexample1">min-width: 640px</div>
```

After loading the page in your browser, resize the window to see the background color change when the condition is met.

Standards Mode and Quirks Mode

Use the HTML 4.01 Transitional doctype without a URL to force quirks mode on all browsers

Use the HTML 4.01 Strict doctype to force strict on all browser except IE6 and IE7, which use almost standards mode

In the W3C box model, element width excludes padding and borders

In the IE quirks mode box model, element width includes padding and borders, giving extra spacing between containers, but IE6 and later use the W3C box model when in standards compliant mode

Design markup so that fixed width elements that need padding or borders have width set on the container, and borders and padding set on the content element to avoid issues in IE quirks mode with the border box model

References

[CSS Differences between Strict and Quirks Mode](#)

[Overview of Quirks Mode](#)

Browser Compatibility

Test in at least four browser engines (Gecko, Webkit, Trident, Presto) each time you make a change to avoid massive debugging efforts at the "end" of a layout styling task

Only use classes for a collection of elements that may appear in multiple sections of a single page, otherwise just reference the collection using the ID of their container

Validate the HTML and CSS code to make sure the markup is not breaking the layout and the style rules do not contain any syntax errors

Make sure that margins and paddings have been reset if there is mysterious whitespace between elements

Make sure that there are no line breaks or whitespace between an anchor tag and a nested image or span if the layout is mysteriously broken

Make sure that all four corners of each element is visible using the a browser-based box model tool (Web Developer Toolbar, IE Developer Toolbar, Dragonfly, Display Element Info Bookmarklet) to identify overflow related issues. If the width is too large and the right border is missing, reduced the fixed width of the element. If the height is too large and the bottom border is missing, turn off overflow if the content is not truncated, otherwise reduce the height or reposition any neighboring elements with absolute positioning, relative positioning, or negative margins that are obscuring the content

Use applied styles context menu in the IE Developer Toolbar to see which rules in the cascade are overwriting the ones you want

CSS

Make sure the parent element used to increase specificity is available in all templates

Copy and paste generated source in HTML validator for pages where the validator doesn't work on the URL

Make sure that padding for any child elements is not forcing the container to expand outside of its defined dimensions, and reduce the dimensions accordingly if you cannot avoid setting both padding and width or height on the same element

Make sure to test the layout with and without toolbars in the browser chrome to make sure no min-height bugs are triggered by adding or removing menus from the browser

Make sure that styles you have defined are not being overwritten by existing styles with more specificity or those defined later in the CSS code by increasing the selector weight using references to parent classes or IDs

Make sure to use Firebug or Web Inspector to verify mini-resets of localized CSS rules and increase the selector weight if necessary to override cascading that may inadvertently cause mysterious layout issues

Make sure to increase the selector weight of IE specific CSS rules when doing so for universal styles

Make sure that the width and height dimensions of the container allow enough space to provide the vertical or horizontal positioning declared in the CSS code

Make sure that the width and height dimensions of all floated elements fit within the constraints of the container

Make sure that the width of right aligned content such as label copy is only as large as needed to avoid using left margins or padding for alignment since it may not be consistent across browsers

Make sure that floated elements with wrapping content have a defined width and proper text alignment

Make sure that any containers that expand on :hover have the a default width and height which is greater than or equal to the size of any nested elements

Make sure that fixed-width containers use overflow:hidden so shrinkwrapped nested blocks don't cause margin collapsing in older versions of Firefox

Make sure that all inline elements are absolutely positioned or contained in block level elements

Change the position attributes of the container to make sure it allows absolute or relative placement of nested elements

Create a copy of the template and remove all divs except the misaligned one and its adjacent divs to debug positioning issues

Put a container div around block level elements that don't respond to display:inline or display:inline-block declarations

Remove margins from inner divs when debugging ignored positioning values in absolutely positioned divs

CSS

Remove all inline text and start with a non-breaking space as the placeholder content to debug width and height issues

Set margin, padding, font-size, and border size to zero for debugging unseen inheritance issues

Use display:block with a fixed height if margins are inconsistent for inline text in situations such as creating a header using markup and one-pixel high element with margin offsets to create rounded corners

Use a unique fixed width for each row instead of margin offsets to create rounded corners inside of inline-block containers for IE6 and IE7

Use a unique background-position for each row to create rounded corners with a background image covering half of the container but not the other

Use IE specific styles when absolutely positioned elements or margins are inconsistent with other browsers

Use a fixed width container when absolutely positioned elements do not appear within the correct relatively positioned parent in IE

Use position:relative on a container with overflow:scroll or auto if content with position:absolute or relative behaves like position:fixed upon scrolling in IE

Use obvious background colors to highlight misplaced or misaligned nested elements when debugging inner divs

Use inline styles and remove ID and class references to display elements that are invisible due to an unknown CSS error

Use word-wrap:break-word(CSS3 Property) or wbr(and its alternatives) to break lines within long words to prevent overflow of an unbreakable string from causing horizontal scrollbars or inadvertent wrapping of blocks to a new line

Use table specific font rules in IE5 set since those applied to html and body tags do not cascade to table elements

Use divs instead of paragraphs when margins don't take effect

Use divs instead of paragraphs to avoid the inherit vertical margins between paragraphs

Use display:inline on an absolutely positioned elements to avoid hasLayout issues

Use margins to move absolutely positioned elements instead of directional shifts

Use border-collapse: collapse; and border-spacing:0 to remove gaps between table columns

Use border styling on the td element instead of the tr element after doing the above to create grid lines on a table

Use empty-cells: hide to remove borders from empty table cells

Use position:relative on inline text and position:absolute on block-level containers and media

Use inline-block to give hasLayout to a relatively positioned container

Make sure class selectors aren't inadvertently overridden by default values of ID selectors, especially background-position changes for sprites, by dividing the properties of ID selectors

CSS

using longhand notation

Use vertical-align:middle to line up inline-block elements in IE; use vertical-align:top for form elements

Use overflow:visible on parent elements if negative margins don't reveal overflow content

Create more classes with specific names instead of creating more complex parent=>child references

Make sure there's not a more specific CSS rule or one that comes later in the style sheet which is overriding the desired effect

Make sure there's not a number at the beginning of the ignored class or ID name

IE will only change the state of a hovered child element when something changes on the anchor tag, so use a redundant effect like a:hover{ visibility:visible; } if something like a:hover span { display:block; } doesn't work

IE will only change the state of a hovered child element that is referenced using a parent class or id if a property is set on the anchor tag that hasn't been declared in any other pseudo classes for that anchor, such as text-indent:0, float:none, direction:inherit, visibility:inherit, white-space:normal, etc ex. .class a:hover span{} needs .class a:hover{visibility:inherit;}

IE will increase the width of a container with an italicized font style so use overflow:hidden to avoid inconsistent wrapping if possible

If links on a page are clickable in IE only, look for an absolutely positioned element that overlaps the links and raise the z-index value of the link container or reposition the layout so the overlap does not happen

If the property value for a group of elements doesn't work, redefine that property value for one of those elements to see if a comma is missing or an unsupported selector nullified the entire rule

```
#business_photo, .sentence, .instruction, .list &gt;li { padding-bottom: 24px; }
```

```
/* Redefine the same padding value because the descendant selector nullifies the above line in IE6 */  
#business_photo { padding-bottom: 24px; }
```

If neighboring elements are mysteriously repositioned when an event or hover triggers the display or a submenu or other hidden content, make sure that content is not relatively positioned in external CSS, inline CSS, or generated JavaScript attributes.

If margins between a series of lists with headers are inconsistent due to special cases such as scrollbars for overflow in long lists or different margins for the first or last list, remove margins from the lists themselves and put them on the headers and use padding at the top and bottom of the container to handle spacing between the first or last list and the borders

If a border does not show up on around an element, make the color black or white to see if the color is too light or dark to distinguish from the background

If a div is being pushed beneath its specified position to a new line, use the mouse to select the content within the div to see if the dimensions of a block element within it are causing overflow outside the container. Make sure to set the width of any forms within the container in addition to validating the HTML to make sure no block tags are missing or inline tags have block tags

CSS

nested within them.

If a hover or visited state of a hyperlink is not working correctly, do the following checks: -Make sure the pseudo classes are defined in the correct order -Make sure the pseudo classes don't share the same definition as a less-specific element, like `a:hover{color: #ccc;}` and `.mylink:hover{color: #ccc;}` since the browser will group them together and parse the less-specific rules in the cascade before the more specific rule, such as `.mysite{color: #eee;}`. If this is the case, add a reference to the parent container to the pseudo class rules, for example `.mymenu .mylink:hover{color: #ccc;}` to increase the selector weight. -Make sure the pseudo classes are defined on the element selector instead of the class selector, for example `.nav a:hover` vs `nav a.submenu:hover`. This may not work in IE6. -Make sure the rules are defined in the proper order, which is the following: 1. `:link`, `:visited` 2. `:hover`, `:focus` 3. `:active`

Since a link can simultaneously be in the `:visited`, `:hover`, and `:active` states, and they have the same specificity, the last one listed wins.

Increase or decrease the text inside a container to make sure it wraps text and resizes properly

Increase or decrease the font size of the browser a few times to see how it affects your backgrounds and margins, especially for text used in headings and navigation

Apply classes to different elements to make sure each class is not dependent upon a specific tag

Include several block elements such as a few paragraphs and a large list inside a div to see if it breaks the flow of the container

Set `position:relative` on floated elements if they are not visible in IE6 or IE7

Set a fixed width or height for a data table cell if word wrapping causes any nested containers with relative dimensions to expand. If there is a div container with a percentage width inside any table cell, content that forces word wrapping in another cell causes the relative width of the div will grow in IE6 and IE7 to match the overflow width of the table cell if an explicit width or height is not set on the cell to contain the content

Reset the line-height of an element if it stretches larger than its defined height in IE

Use `clear` to break an element away from a floated sibling and wrap it to a new line

Separate HTML structure and CSS design by developing the naming conventions and organization of the container styles without thinking about the content

Create a reusable library of headers, footers, submenus, navigation bars, lists, forms, and grid layouts

Use consistent semantic style names for classes and IDs

Design modules to be transparent on the inside

Extend objects by applying multiple classes to an element

Use CSS reset styles, web-safe fonts, and grids on all projects

Avoid location dependent styles that break outside of the cascade

Avoid using a class name that describes the tag or property it is mapped to, such as `.hidediv`

CSS

or .floatleft

Avoid using IDs to style content within the main layout containers

Avoid using images to replace text

Avoid drop shadows and rounded corners over irregular backgrounds

Avoid nesting divs inside of spans or having spans adjacent to divs to prevent triggering side-effects caused by the behavior of anonymous block boxes

Using 1px dotted borders in IE6 is buggy and will display a dashed border. Use an IE6 specific style of 2px border-width and a lighter shade of the desired color to offset the larger pixel size.

Any vertical-align value besides top or bottom will be rendered inconsistently across browsers for form elements

Only use the !important declaration to create user style sheets or to override an inline style if the HTML markup is not editable. The !important declaration must follow the value of the property that it intends to give weight to. Since inline styles can be added to any HTML tag, using the style attribute selector is the only way to override those styles if a class or ID attribute is not defined for the element, as in the following example:

```
div[style]{ background-color: #f00 !important; }
```

When using abs pos elements, make them the last child element of the rel pos container in the HTML source order for IE6

Use an abs pos span nested inside an href to create dropdowns that show up in different places

Use the nested span trick to highlight the parent container of a nav list by setting its background color to the desired nav background and making it visible on hover. Make the ul the rel pos parent and give it overflow hidden, and make the empty span bottom:0; z-index:-1; height: 99em; and width equal to the nav

```
<ul id="nav"><li><a href="#"><span></span>Test 1</a></li></ul>
```

Use the nested span trick to create polygonal shaped links by offsetting their background positioning and placing them over the desired content

Use a:focus{width:0,height:0,outline:0} to remove dotted borders

Use abs/rel positioning to emulate outlines in IE6/7

Use resets to debug margin/padding issues

Use margins to remove padding and top/bottom positioning for abs positioned elements if spacing is off

Use :default to select all radio button sets

Use :active as :focus in IE6

Make sure parent div used for specificity is available in all templates

Use relative/absolute div with a background color to replace text with ellipsis using the following steps:

CSS

Create a content div with the original content and a height equal to the font-size * the number of rows you want visible

When a JavaScript event is triggered, reduce the height to the new number of rows you want visible by adding a class with that setting

Make sure class selectors aren't inadvertently overridden by default values of ID selectors, especially background-position changes for sprites, by dividing the properties of ID selectors into pieces

Use vertical-align middle to line up inline-block elements in IE; use vertical-align:top for form elements

Use change events to delegate input and select elements

Use active link states and relative positioning to respond to link clicks

Use the following alternatives for buffering offscreen content: -Negative margins instead of display:none -Relative positioning offsets instead of visibility:hidden for text -background-position instead of visibility:hidden for images -display:inline block instead of display:table -clip:rect instead of max-width/height

Absolutely positioned elements moved outside of their relatively positioned parents in Firefox 2 cause the containers to stretch

Use margins for the outermost elements

Use padding for elements that have a background color or image

- moz-only-whitespace matches an element that has no child nodes at all or empty text nodes or text nodes that have only white-space in them. Only when there are element nodes or text nodes with one or more characters inside the element, the element doesn't match this pseudo-class anymore.

- moz-selection (::selection) pseudo-element applies to the portion of a document that has been highlighted (e.g. selected with the mouse) by the user.

The :-moz-first-node pseudo-class matches an element that is the first child node of some other element. It differs from :first-child because it does not match a first child element with (non-whitespace) text before it.

The :-moz-last-node pseudo-class matches an element that is the last child node of some other element. It differs from :last-child because it does not match a last child element with (non-whitespace) text after it.

The ::-moz-list-bullet pseudo-class is used to edit the bullet of a list element.

The :default pseudo-class is used to identify a user interface element that is the default among a group of similar elements

Use -ms-text-align-last for orphan lines at the end of paragraphs

Use xsl:number to mimic generated content for counters in IE

Use outlines to style buttons with white accents

CSS

Use `q { quotes: "" "" }` to define quote characters

Use the following to trigger quote character display:

```
q:before { content: open-quote }  
q:after { content: close-quote }
```

Use fixed positioning with `@media print` to show header and footer containers on every page

Use `absolute:positioning` instead of fixed positioning to avoid showing header and footer containers on every page

CSS order of Rules

Use a reset block as the single location to style element selectors without classes or IDs to avoid cascading issues

Use a [CSS Validator](#) to ensure that typos and conflicting declarations aren't causing bugs on your page

Reference IDs and classes directly without prefixing them with parent tag or class names

Use shorthand syntax for classes and longhand syntax for IDs to avoid setting unnecessary defaults and simplify debugging of dynamic class changes through scripting or property changes through pseudo-classes

Don't use the same class to set both dimensions and other box model rules including margins, borders, and padding to avoid limiting the reuse of size or spacing rules among different elements

Use a simple class selector, such as `.new{ font:bold;}` instead of creating complex descendant selectors to define a single property, such as `.menu .title .new{ font:bold; }` to maximize to readability and reusability of the codebase

Don't use DOM property names (`length`, `item`, `namedItem`, `tags`, `urns`) as ID values since they will cause JavaScript conflicts in IE since it considers property names to be reserved words

Don't use attribute name values as ID values since they will cause JavaScript conflicts in IE. For example, using "description" as an ID value may conflict with the description meta tag.

Create unique new classes instead of parent dependent subclasses

Copy common properties, such as colors and font size, to new classes as a preventive measure so that an element is not dependent on multiple classes due to a single property of each one, since the class references in the HTML may end up looking like inline styling. This also helps avoid inheriting unneeded properties, such as padding and margins, and subsequently needed to reset these properties in new classes

Minimize usage of the descendant selector (e.g. `#parent .child`) and the child selectors when styling elements to avoid long lookups of the DOM tree

CSS

Define classes to style block level elements(.content) instead of using tag references by ID (#main p) to make them independent of their parent elements

Define state changes in CSS by prepending copies of existing selectors with semantic class names that map to JavaScript functions, then append the class name to the documentElement when the DOM event is triggered

Avoid semicolons on the last style declaration to avoid accidentally typing the end bracket before the semicolon and generating an error

Avoid naming conventions in which a class describes the property it invokes so that the style sheet does not become a one-to-one mapping of names to functionality which leads to HTML code which resembles inline styling with multiple class references for every element.

Use styles to set properties for tables instead of table tag attributes for cross browser consistency

Add subclasses for common needs, such as highlighting the first or last item in a list, hiding and showing comment text, emphasizing the current navigation selection, and switching star rating colors on and off

Use semantic ID names for standard layout components, when there is only one such component per page, such as banner, logo, badge, screenshot, thumbnail, breadcrumbs, navigation, sidebars, marquee, subnav, content, gutter, headmargin, footmargin, sidemargin, baseline, overlay, fold, ads, widgets, and audio/video players For example,

```

<p id="logo"><em>Happy Fun Ball</em></p>
<p>Normal text</p>
```

Use semantic class names for standard prose components that may need specific styling, such as sentence, phrase, clause, salutation, lead-in, suffix, prefix, anagram, homonym, contraction, gerund, idiom, interjection, onomatopoeia, timespan, etc.

Use semantic class names for standard article components that may need specific styling, such as headshot, caption, mugshot, masthead, folio, subhead, dateline, byline, topic, bio, ticker, alert, boilerplate, dropcap, slogan, keyword, tease, sample, snippet, sidebar, boxscore, ranking, standing, poll, addendum, soundbite, pullquote, plugs, followup, letterhead, etc.

Use semantic class names for essay components that may need specific styling, such as objective, excerpt, aside, citation, footnote, topic, annotation, summary, table of contents, etc. For example,

```
<p class="aside">(a parenthetical aside)</p>
<p>Normal text</p>
<p class="aside">(another parenthetical aside)</p>
```

Style fonts and colors for standard layout components first to avoid redundancy and take advantage of the cascade when defining nested elements

Reset styles at the beginning of the style sheet to overwrite inconsistent defaults across browsers

Remove redundant and obsolete styles when moving test code to production

CSS

Use classes instead of IDs for any code intended to work inside the sandbox of a third party API, since IDs may be stripped out to avoid conflicts

Use the following XSLT CSS preprocessor template to add CSS variables to your stylesheet using browser or server-side scripting:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="text" omit-xml-declaration="yes" media-type="text/css" />
  <xsl:param name="color" select="#fff"/>
  <xsl:template match="*">
    h1 { color: <xsl:value-of select="$color"/>; }
  </xsl:template>
</xsl:stylesheet>
```

Hacks and Filters

Browser Specific Filters

Opera 10+ CSS Filter

```
.dude:read-only { color: green; }
.dude:read-write /* form elements */
```

IE6/IE7 CSS Filter

```
@media, { .dude { color: silver; } }
```

IE7 CSS Filter

```
*:first-child+html #mytag to target IE7
```

IE8 CSS Filter

```
@media all { .dude { color: brown\0; } }
```

IE9 CSS Filter

```
@media all and (monochrome: 0) { .dude { color: pink\9; } }
```

Webkit CSS Filter

```
* > /**/ .dude, x:-webkit-any-link { color:red; } /* * &gt; /**/ hides from IE7; remove if unneeded */
```

Konqueror CSS Filter

```
* > /**/ .dude, x:-khtml-any-link { color:red; } /* * &gt; /**/ hides from IE7; remove if unneeded */
```

Safari 2 CSS Filter

```
html body:only-child .dude
```

Firefox CSS Filter

```
@-moz-document url-prefix() { .dude { color: green; } }
```

Firefox 3.6+ CSS Filter

```
@-moz-document url-prefix() { @media -moz-scrollbar-start-backward { .dude { color: green; } } }
```

Firefox 3.0+ CSS Filter

```
@-moz-document url-prefix() { .dude:-moz-system-metric(scrollbar-start-backward) { color: green; } }
```

Legacy Browser Hacks

All web browsers are not equal and different rendering of the same code, even if the code is validated, can destroy a web page's presentation. To work around this a large community of browser hacks has been developed. These hacks should be avoided at all cost because they may break one's code in other browsers or slow down performance. However sometimes it is impossible to avoid the use of all of these hacks.

Be aware that constant browser updates may cause hacks to not work in the future.

- [Caio Hack](#) — hides rules from Netscape 4, even 'inline' CSS

Browser Extensions

Each web browser's layout engine has its own extension for the features of CSS it supports, whether they be proprietary, experimental or purely internal. For instance, many browsers only have experimental CSS3 implementations, so the CSS property is often prefixed by the layout engine's name. The most famous example would probably be border-radius, which for the

CSS

longest time was only supported by WebKit and Gecko as `-webkit-border-radius` and `-moz-border-radius` respectively (`border-radius` is now supported by WebKit, Gecko, Presto & Trident without the vendor prefix).

- `-engine-property`: arguments;

Extensions by Layout Engine

Extension	Layout Engine	Browsers
<code>-webkit-</code>	WebKit	Safari, Shiira, OmniWeb, Flock (3.x+), Chrome
<code>-moz-</code>	Gecko	Firefox, Flock (up to 2.x), Iceweasel
<code>-o-</code>	Presto	Opera
<code>-ms-</code>	Trident	Internet Explorer

IE Specific Properties

-Use `(-ms-)writing-mode: bt-rl`; to vertically align text, with punctuation at the beginning of the string to display text that reads vertically in IE7+

-Use `(-ms-)text-align-last` for orphan lines at the end of paragraphs

-Use `text-align:justify`; `text-justify: newspaper`; `text-align-last:end` for newspaper layouts

(`-ms-` for IE8+)

Text is available under the [Creative Commons Attribution/Share-Alike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#).