
Cascading Style Sheets

Stanford University Continuing Studies Website Design

Cascading Style Sheets	1
What are Cascading Style Sheets?	2
The Cascading Style Sheets (CSS) Language	4
Comments	5
Inheritance	6
Using <code>!important</code>	7
Unit Measurements	8
Classes and IDs	9
Span and Div	11
CSS Locations	12
CSS: External location	12
CSS: Internal location	14
CSS: Inline location	15
Grouping Styles and Selectors	16
Using CSS to Change Text Properties	17
CSS and Fonts	19
Colors and Backgrounds	21
Pseudo-elements, Pseudo-classes, and Generated Content	22
Specificity – What happens if two rules conflict?	24
The Box Model – height, width, padding, border, margin, outline	25
Using CSS for Visual Effects	31
Display	31
Using <code>display: block;</code>	32
Using <code>display: inline;</code>	33
Visibility	34
<code>display: none;</code> vs. <code>visibility: hidden;</code>	34
Positioning	36
Using <code>position: absolute;</code>	37
Using <code>position: relative;</code>	38
Using <code>position: fixed;</code>	39
Layers and the Bounding Box	41
Layering Example 1	42
Layering Example 2	43
Clip (only for <code>position: absolute</code>)	44
Float / Clear	46
Modifying List Elements	48
Tables and CSS	50
Changing the Cursor's Appearance	52
Printing and CSS	53

What are Cascading Style Sheets?

The Cascading Style Sheets (CSS) language is:

- Used to **describe the presentation** (the look and feel) of a document written in a markup language.
- Primarily used to **style documents** written in HTML/XHTML (web pages), but it can also be used to style XML and other markup languages.
- **Made up of rules.** Web browsers interpret these rules in a specific, top-to-bottom, or *cascading* manner (that is, if two rules are in conflict with one another, the last rule listed overrides the previous rule). The CSS rules control the look, or *style* of the content of these markup languages.
- **Placed in a central location** (*sheet*) for the browser to interpret (generally located in either in the `<head>` of the document that is being styled, or in a separate linked file).

All matching rules for a particular selector will be applied, except where they conflict with each other. If rules are in conflict, the last rule to be declared is applied. In the following example, `<h2>` tags would be displayed in red and italics (but not blue):

```
h2 {font-style: italic;}  
  
h2 {color: darkblue;}  
  
h2 {color: red;}
```

Note:

Web browsers do not necessarily interpret CSS rules in exactly the same way. You will want to test your web pages in multiple browsers to ensure the pages end up having the desired look. In particular, *Internet Explorer* is known to follow slightly different interpretations of various CSS rules.

CSS-aware browsers apply their own stylesheet for every HTML element as the first set of rules in the cascade. This set of rules forms the default display for every element. For example, most browsers treat the `<p>` tag as a block element, as though there were the explicit declaration `p {display: block;}`. By using CSS, you override these implicit styles with an explicit declaration. For more on the block display, see *Inline vs. Block display*, page 31.

By using CSS, you can also:

- Control text formatting and location on the page.
- Eliminate the need for tables as a layout tool.
- Create logos merely using text, instead of having to rely on graphics.

These changes make pages more accessible to a wider audience.

The World Wide Web Consortium (W3C) is the organization that establishes the CSS rules and specifications. The latest specifications for CSS can be found at the following sites:

- CSS 1: <http://www.w3.org/TR/CSS1/>
- CSS 2: <http://www.w3.org/TR/CSS2/>
- CSS 2.1: <http://www.w3.org/TR/CSS21/>
- CSS 3 (still under development):
<http://www.w3.org/Style/CSS/current-work>

Detailed technical explanations of the differences between CSS 1, CSS 2, and CSS 2.1:

- Between CSS 1 and CSS 2: <http://www.w3.org/TR/CSS2/changes.html>
- Between CSS 2 and CSS 2.1: <http://www.w3.org/TR/CSS21/changes.html>

The Cascading Style Sheets (CSS) Language

CSS contains *rules*. Each rule consists of a *selector* and a *declaration* (which in turn is made up of a *property* and a *value*).

In HTML, to create a web page with `<h2>` tags that have not only the standard features of a Header tag (that is, their own paragraph, bold, with a size change) but also are dark blue, you might use the now-deprecated tag ``:

```
<h2><font color="darkblue">This is a darkblue H2 tag</font></h2>
```

That's a lot of information to type every time you want to use a dark blue `<h2>` tag. If you styled the `<h2>` tag using CSS instead, you could use just the regular `<h2>` tag in your HTML. The style information will be included in the Style Sheet as follows:

```
h2 { color: darkblue;}
```

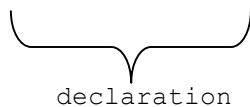
To change the color of all `<h2>` tags from darkblue to green, in the CSS, simply change the declaration from `color: darkblue;` to `color: green;`. The next time anyone visited the site, all the `<h2>` tags on all the pages would display as green instead of darkblue.

In the example below, `h2` is the selector, `color` is the property, and `darkblue` is the value. When used with web pages, selectors are usually HTML tags.

```
h2 { color: darkblue;}
```

Syntax for a CSS rule:

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



Comments

Adding comments in your CSS will help you (and future developers) understand the exact reasoning and methodology you utilized when writing a particular CSS rule.

To add a comment, separate the text of the comment between `/*` and `*/`

For example:

```
/*
Color Stylesheet: Provides colors from the University palette as
classes
*/

/* Text colors */
.white { color: #fff }
.red { color: #820000 }
.black { color: #000 }
.warmgray { color: #3f3c30 }

/* Background colors */
.white_bg { background-color: #fff }
.red_bg { background-color: #820000 }
.black_bg { background-color: #000 }
.warmgray_bg { background-color: #3f3c30 }
```

Note:

Comments in HTML are created differently – in HTML, a comment is created by surrounding the comment with `<!--` and `-->`.

An HTML comment example:

```
<!-- enter comment here -->
```

Inheritance

Under both HTML and CSS rules, elements that are inside of other elements (this is known as a child element) inherit the style of the parent.

For example, suppose you have a web page with the following HTML code:

```
<h1>The <em>most important</em> headline information</h1>
```

And the web page is styled with the following CSS code:

```
h1 { color: green; font-size: 36pt; font-weight: bold; }
```

The emphasized phrase “most important” in this example would display as italicized, bolded, green, 36 point font.

The `em` inherits the styles (green, 36 point, bold) from its parent – `h1`.

Using !important

Normally, the last rule listed in the cascade will take precedence over previous rules.

In this example, the body font will be Verdana, not Times:

```
body {font-family: Times;
      font-family: Verdana;}
```

However, by entering !important in a rule, that rule will take precedence, regardless of its location.

In this example, the body font will be Times, not Verdana:

```
body {font-family: Times !important;
      font-family: Verdana;}
```

Note: !important does not work with all properties in *Internet Explorer*.

Unit Measurements

In CSS, you can measure units either in absolute values or in relative values.

Absolute values are fixed, specific values. Since they are exact measurements, they allow the designer complete control over the display of the web pages.

Absolute values include:

mm, cm, in, px, pt, pc, xx-small, x-small, small, medium, large, x-large, xx-large

Relative values have no fixed, specific values, and are calculated in comparison to something else (usually the size of the default font or line size). Relative values tend to be a better choice, because different computers have different video cards and screen sizes, and users have differing eyesight abilities. They give the designer less absolute control but often create a better experience for the visitor.

Relative values include:

em, ex, larger, smaller, num%

Examples of unit measurements in a CSS file:

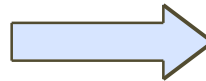
```
body { font-size: 1.2em; }
h1, h2, h3 { line-height: 200%;}
```

Warning:



If you use a relative value and nest one element inside another, the result will be cumulative.

```
<head>
<title>Untitled Document</title>
<style type="text/css">
ul {font-size: 70%}
</style>
</head>
<body>
A list:
<ul>
  <li>Item
    <ul>
      <li>Sub item</li>
    </ul>
  </li>
</ul>
</body>
```



A list:

- Item
 - Sub item

Classes and IDs

HTML has two attributes that make CSS even more useful: *class* and *ID*. They make it easy to apply style to just about any tag.

Classes can describe a generic style that can be applied to any HTML element, or can be created for specific elements. When defining a style for elements with a particular class attribute in the Style Sheet, declare a rule using a dot (.) followed by the class name. To limit the style to a particular element with that class attribute, use a selector combining the tag name with a dot followed immediately by the class name.

The following rule would apply to any element with the attribute **class="shade"**:

```
.shade { background: yellow; }
```

The following rule would apply only to paragraph tags with the class **shade**

```
(<p class="shade">):  
p.shade { background: red; }
```

IDs are similar to classes, but IDs are unique. They can only be used with one instance of an element within a document. When defining a CSS rule using an ID-based selector, use a number sign (#) followed by the style name. To limit the style to a particular element with that ID attribute, use a selector combining the tag name with a # and then the ID name.

The following rule would apply to any element with the attribute **id="intro"**:

```
#intro { font-size: 2em; }
```

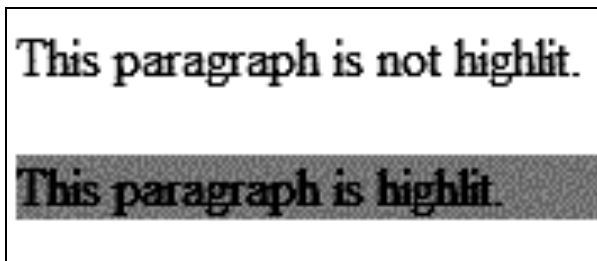
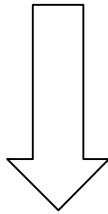
The following rule would apply only to heading 1 tags with the ID **intro**

```
(<h1 id="intro">):  
h1#intro { color: green; }
```

EXAMPLE: A web page with an internal CSS style with a class called "highlight"

```
<head>
<style type="text/css">
<!--
.highlight { background-color: #ccc;}
-->
</style>
</head>

<body>
<p>This paragraph is not highlit.</p>
<p class="highlight">This paragraph is highlit.</p>
</body>
</html>
```



Span and Div

There are two tags that are particularly useful when using CSS: `` and `<div>`. They are both container tags that have minimal formatting associated with them.

Using `` and `<div>` tags in conjunction with classes and IDs allows for great flexibility in creating pages.

The `` tag is an empty inline element that can be used to hold text. It is a generic wrapper for content that, by itself, does not do or represent anything.

The `<div>` tag is an empty block element designed to hold a division of text. Like the `` tag, it is a generic container for content that by itself does not do or represent anything (except to place the content onto its own line).

For example, to apply a font to a specific section of text, create a class, and use the span tag with that class:

In the CSS:

```
.neatstuff {font-family: Comic Sans MS;}
```

In the HTML:

```
<span class="neatstuff">This is in Comic Sans</span>
```

CSS Locations

Style information can be located:

- External to the pages in a site (the CSS rules are listed in a separate file and linked to the HTML files).
- Internal to each page.
- Inline with individual tags.

Generally, creating an external style sheet file is the preferred method. To take full advantage of CSS, the Style Sheet for a site should be in an external file, so that any changes made there will apply throughout the site. This also means that only one style document has to be downloaded for a single site, making the web pages load faster.

CSS: External location

The most common place to put style information is in an external document that each page of a web site points to directly. Any changes made to this single document will then be applied throughout the entire web site as each page is accessed by users. External Style Sheets have a .css extension.

When linking to an external style sheet, you can also specify separate style sheets by media type:

Media Type	Intended for Use With
all	All devices
aural	Speech synthesizers
braille	Braille tactile feedback devices
embossed	Paged braille printers
handheld	Handheld devices (typically small screen, monochrome, limited bandwidth)
print	Paged, opaque material and for documents viewed on the screen in print preview mode
projection	Projected presentations
screen	Computer screens, primarily
tty	Media using a fixed pitch character grid, such as teletypes, terminals, or portable devices with limited display capabilities
tv	Television-type devices

There are two methods you can use to link web pages to an external style sheet: `<link>` and `@import`.

The `<link>` method was introduced with CSS 1, and specifically links an HTML document to a CSS file. With the `<link>` method, you can specify the media type using the `media="type"` attribute. (See below for an example.)

The `@import` method was introduced with CSS 2, and is used to import style sheets into web pages, XML files, or even other style sheets. With the `@import` method, you can specify the media type by adding an `@media` rule. (See below for an example.)

If you want to attach a style sheet to a web page, either method will work.

EXAMPLE: A web site with two web pages, using two external style sheets (4 separate files – when viewed on-screen, the *basic.css* rules are used; when printed, the *print.css* rules are used)

```
Text that appears in the basic.css style sheet document:  
h2 {font-family: Arial, sans-serif; font-style: italic;}  
p {font-family: Courier, monotype; font-style: bold; }
```

```
Text that appears in the print.css style sheet document:  
h2 {font-family: Book Antiqua, serif; font-style: italic; }  
p {font-family: Courier, monotype; font-style: bold; }
```

```
HTML document 1, using the <link> tag method:  
<head>  
<link rel="stylesheet" type="text/css" href="basic.css" media="all">  
<link rel="stylesheet" type="text/css" href="print.css" media="print">  
</head>
```

```
HTML document 2, using the @import and media method:  
<head>  
<style type="text/css">  
<!--  
@import url("basic.css") all;  
@import url("print.css") print;  
-->  
</style>  
</head>
```

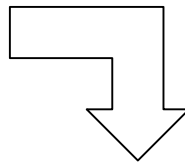
CSS: Internal location

Style information can also be included in the <head> section of an individual web page. This tends to work best when a single page needs to have a slightly different look than the rest of the site. If you plan to apply these styles to more than one web page, it is more useful to create an external style sheet and link the web pages to that style sheet.

EXAMPLE: A web page using internal styles

```
<head>
<style type="text/css">
<!--
h1 { font-family: Arial; font-style: italic; color: green;}
-->
</style>
</head>

<body>
<h1>This is soooooooooooooo cool!</h1>
<p>Nothing cool here.</p>
<h1>This one's cool, too!</h1>
</body>
```



This is soooooooooooooo cool!

Nothing cool here.

This one's cool, too!

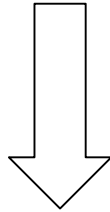
CSS: Inline location

For extreme control, style information can be included in an individual tag. The style effects only that tag and no others in the document. This option is most useful for those rare occasions when a single tag needs to have a slightly different style.

EXAMPLE: A web page using inline styles

```
<body>
<h1 style="font-family: Arial; font-style: italic; color: green;">This
is soooooooooooooo cool!</h1>
<p>Nothing cool here.</p>
<h1>This isn't cool, either.</h1>

</body>
</html>
```



This is soooooooooooooo cool!

Nothing cool here.

This isn't cool, either.

Grouping Styles and Selectors

- **Multiple styles** – Each rule can include *multiple styles*, using semicolons to separate them.

```
h2 {color: darkblue; font-style: italic;}
```

- **Multiple selectors.** Additionally, *multiple selectors* that have the same styles can be grouped, using commas to separate them.

```
h1, h2, h3 {font-style: italic;}
```

- **Contextual selectors.** *Contextual selectors* allow you to specify that something will occur only when it is used in conjunction with something else. In the style below, *em* will display in red, but only when it occurs within *li* within *ul*:

```
ul li em {color: red;}
```

Elements being modified by contextual selectors need not appear immediately inside one another. For example, using the style above with the HTML below, **blah** would still be red text:

```
<ul><li><strong><em> blah </em></strong></li></ul>
```

- **Direct child selectors** . *Direct child selectors* allow you to specify that something will change, but only when immediately inside another element. With the style below, only those *strong* elements that are directly inside *h1* will be purple; no *strong* tags deeper within the sheet will be purple:

```
h1 > strong {color: purple;}
```

- **Adjacent selectors** . *Adjacent selectors* allow you to specify that something will change only when preceded by something else. In the style below, only those links that are preceded by an *h2* will be green:

```
h2 + a {color: green;}
```

Elements being modified by adjacent selectors appear immediately after one another.

Using the style above, this link would be green:

```
<h2>Visit Stanford!</h2><a href="http://www.stanford.edu">click here</a>.
```

But this link would not:

```
<h2>Visit Stanford! <a href="http://www.stanford.edu">click here</a></h2>.
```

- **By attribute.** You can also group selectors *by attribute*. In the example below, text that is inside centered *h2* tags (`<h2 align="center">`) will be surrounded by a dotted border:

```
h2[align="center"] {border: dotted;}
```

Using CSS to Change Text Properties

With CSS, you can define how text appears when the browser renders it.

The textual properties are:

- **word-spacing.** This property defines the spacing between words. See Unit Measurements on page 8 for details on the units of measurement you can use.
`h1 { word-spacing: 1.2 em; }`
- **letter-spacing.** This property defines the spacing between letters. See Unit Measurements on page 8 for details on the units of measurement you can use.
`address { letter-spacing: 1px; }`
- **text-decoration.** This property defines the decorations that are to be added to the element (underline, overline, line-through, blink, none).
`a:link { text-decoration: underline; }`
`a:hover { text-decoration: none; }`
- **vertical-align.** This property defines the vertical alignment of the element. The vertical alignment choices are:
 - `baseline` – align the baseline (or bottom) of the element
 - `middle` – align the vertical midpoint of the element
 - `sub` – subscript the element
 - `super` – superscript the element
 - `text-top` – align the top of the element with the top of the font
 - `text-bottom` – align the bottom of the element with the bottom of the font
 - `top` – align the top of the element with the tallest element on the line
 - `bottom` – align the bottom of the element with the lowest element on the line
`.imagetext { vertical-align: middle; }`
- **text-transform.** This property defines the case of the font (uppercase, lowercase, capitalize, none)
`h2 { text-transform: uppercase; }`
- **text-align.** This property defines the alignment of the text within the element (left, right, center, justify)
`h1 { text-align: center; }`

- `text-indent`. This property defines the amount of space to indent the text. See Unit Measurements on page 8 for details on the units of measurement you can use.

```
p { text-indent: 20px;}
```

- `line-height`. This property defines the height of a line. See Unit Measurements on page 8 for details on the units of measurement you can use.

```
h3 {line-height: 1.2ex;}
```

- `white-space`. This property defines specifies how white-space inside an element is handled (normal, nowrap, pre, pre-line, pre-wrap, inherit)

```
p { white-space: nowrap;}
```

CSS and Fonts

When choosing a font, there are several things to keep in mind.

- Not everyone has the same set of fonts.
- If you use a font that the visitor doesn't have, the page will display in the default font (usually Times), unless you provide more choices. To do this, add more than one font in your declaration, and always end with the font family (serif, sans-serif, or monospace):
`font-family: Verdana, Arial, Helvetica, sans-serif`
- Printed documents tend to look better in Serif fonts (Times New Roman, Georgia, Book Antiqua, etc.).
- Documents to be viewed on-screen tend to look better in Sans-serif fonts (Verdana, Arial, Helvetica, etc.).

To apply a font to the entire web page, modify the `<body>` tag in the CSS. For example, this CSS rule changes the default font for the web page to Verdana:

```
body {font-family: Verdana;}
```

To apply a font to a specific letter, word, or series of words, see Span and Div on page 11.

In CSS 1, the following font properties can be defined:

- `font-family` – used to define the font family or generic family names
`p { font-family: Verdana, Arial, sans-serif; }`
- `font-style` – used to define the style as either normal (upright), italic (slanted but using a different glyph than normal), or oblique (slanted but using the same glyph as normal – oblique looks like italic font distorted).
`address { font-style: oblique; }`
- `font-variant` – used to display the font in small caps or normal.
`.important { font-variant: small-caps; }`
- `font-weight` – used to define how bold the text will be displayed. The choices are: normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900. 'normal' is equivalent to 400; 'bold' is equivalent to 700; bolder and lighter will display the font one level bolder or lighter than the weight of the parent element.
`.veryimportant { font-weight: 900; }`
- `font-size` – used to define the size of the font. See Unit Measurements on page 8 for details on the units of measurement you can use.
`body { font-size: medium; }`
- `font` – used to combine the various font properties into a single rule.
`body { font: Verdana, Arial, sans-serif normal small-caps 400 medium; }`

In CSS 2, the following font properties have been added:

- `direction` – used to specify the text direction/writing direction
`body { direction: rtl; }`
- `text-shadow` – used to specify the shadow effect added to text. There are four values:
 1. The color of the text-shadow
 2. The X-coordinate of the text-shadow, relative to the text
 3. The Y-coordinate of the text-shadow, relative to the text
 4. The blur radius (amount of space that the shadow text will be "stretched") of the text-shadow.
`h1 { text-shadow: #999 1px 1px 5px; }`

Colors and Backgrounds

In CSS, **colors** can be defined:

- Using the **6-digit hexadecimal code** (the first 2 numbers refer to the amount of Red, the next 2 refer to the amount of Green, and the last two refer to the amount of Blue). Most designers use this method to define color. In hexadecimal code, 0 is the smallest number, and F is the largest. So, for example, "green" is defined as #00FF00 (no Red, all Green, no Blue). And "white" is defined as #FFFFFF (all Red, all Green, all Blue), while "black" is defined as #000000 (no Red, no Green, no Blue).
- Using the **3-digit hexadecimal code** (the first number refers to the amount of Red, the second the amount of Green, and the last the amount of Blue). A 3-digit hexadecimal color code is equivalent to the 6-digit hexadecimal code with each number repeated once. For example, #039 is the same as #003399.
- Using the **integer RGB color model**. The first number refers to the amount of Red, the second the amount of Green, and the last the amount of Blue. Each number is between 0 and 255. For example, `rgb(255, 0, 0)` is Red.
- Using the **percentage RGB color model**. The first number refers to the percentage of Red, the second the percentage of Green, and the last the percentage of Blue. For example, `rgb(0%, 0%, 100%)` is Blue.
- Using the **keyword color names**. CSS defines these names using the Windows VGA model: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow.

Backgrounds can be defined as either a color or as a URL to a picture. If it is a picture, The format of a URL value is `url(` followed by optional white space followed by an optional single quote ' or double quote " character followed by the URL itself followed by an optional single quote (') or double quote (") character followed by optional whitespace followed by `)`. Quote characters that are not part of the URL itself must be balanced. Parentheses, commas, whitespace characters, single quotes (') and double quotes (") appearing in a URL must be escaped with a backslash: `'\('`, `'\)'`, `'\,'`, `'\"`, `'\"'`.

To define the color of a font, use the property `color`:

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

To define the background, use the property `background`:

```
body {background: yellow;}
#sidebar { background: url('http://foo.stanford.edu/images/foo.gif');
```

Pseudo-elements, Pseudo-classes, and Generated Content

Pseudo-elements and *pseudo-classes* are special, predefined groupings used in CSS to deal with special situations that do not exist in HTML.

Pseudo-elements and Generated Content

In CSS 1, the pseudo-element `:first-letter` allows you to specify a style that affects the first letter, and the pseudo-element `:first-line` allows you to specify a style that affects the first line.

In CSS 2, the following pseudo-elements have been added:

- `:before` - inserts content in front of the selected element

```
div.important:before {content: "text text text";}
```
- `:after` - inserts content after the selected element

```
div.important:after {content: "text text text";}
```
- `:after` - inserts content after the selected element

```
div.important:after {content: "text text text";}
```

When inserting content using the `:before` or `:after` pseudo-element, the content, counter-increment, and counter reset properties can be modified. Counter-increment and counter-reset do not work in Internet Explorer 7 or earlier; in Internet Explorer 8 and newer, a declared `!doctype` is required.

- `content` - used with the `:before` and `:after` pseudo-elements to insert generated content. Allowed values are: `none`, `normal`, `counter`, `attr(attribute)`, `open-quote`, `close-quote`, `no-open-quote`, `no-close-quote`, `url(url)`

```
div.important:after {content: url(arrow.gif);}  
blockquote:before {content: open-quote;}  
blockquote:after {content: close-quote;}
```
- `counter-increment` - counter-increment property increments one or more counter values. Allowed values are: `none`, `inherit`, and *identifying name* (e.g., "section")

```
h2:before { counter-increment: section;  
            content: counter(section) ". ";}
```
- `counter-reset` - The counter-reset property creates or resets one or more counters.. Allowed values are: `none`, `inherit`, and *identifying name*

```
h1 { counter-reset: subsection; }  
h1:before { counter-increment: section;  
            content: "Section " counter(section) ". "; }
```

Pseudo-classes

Pseudo-classes found in CSS 1:

- `:link` – styles unvisited links
`a:link { color: blue; text-decoration: underline;}`
- `:visited` – styles visited links
`a:visited { color: purple; text-decoration: underline;}`
- `:active` – styles active links
`a:active { color: red; text-decoration: underline;}`
- `:hover` – styles links as the mouse hovers over the link
`a:hover { color: red; text-decoration: none;}`

Pseudo-classes added in CSS2:

- `:focus` – styles the element that is in focus (actively being used – for example, a form field is in focus when you are entering data)
`input:focus { background: yellow;}`
- `:first-child` – styles the element that is the first child of another element – for example, only the first emphasized text within a paragraph.
`p em:first-child { background: yellow;}`
- `:lang` – styles an element with a specific lang attribute
`:lang(en) { quotes: '""' '""' "''" "''"; }`
`:lang(fr) { quotes: "<<" ">>" "<" ">"; }`

Specificity – What happens if two rules conflict?

When two (or more) CSS rules refer to the same element, the browser determines which rule relates more to that element (is more *specific*) and applies that rule. To calculate the specificity of a rule, browsers give IDs a score of 100, classes a score of 10, and HTML selectors a score of 1. The rule with the highest score is invoked.

Suppose you have the following CSS:

```
div#cool p { color: red; }  
.nifty { color: green; }  
#cool { color: purple; }  
p { color: blue; }
```

And you also have the following HTML, and that the above CSS has been invoked:

```
<div id="cool"><p class="nifty">What color am I?</p></div>
```

You might think the color of the text in the paragraph would be blue, since `p { color: blue; }` was the last rule to be written. You might think the color would be purple, since the ID `#cool` defines color text to be purple. Or, you might think the color would be green, since the class `.nifty` defines color text to be green.

In actuality, the browser displays the paragraph in red text. The more specific a selector, the more preference it is given in regards to styles that are otherwise in conflict (and `div#cool p` is more specific than `p`, `.nifty`, or `#cool`).

So, in the example above, `p` has a specificity of 1 (1 HTML selector); `div p` has a specificity of 2 (2 HTML selectors); `.nifty` has a specificity of 10 (1 class selector); `div p.nifty` has a specificity of 12 (2 HTML selectors plus 1 class selector); and `#cool` has a specificity of 100 (1 ID selector).

Even though `div#cool p` is the first rule listed, it is the one invoked, since it has the highest score (102) in this group of styles:

Specificity Score	CSS rule
102	<code>div#cool p { color: red; }</code>
100	<code>#cool { color: purple; }</code>
10	<code>.nifty { color: green; }</code>
1	<code>p { color: blue; }</code>

The Box Model – height, width, padding, border, margin, outline

When a browser draws an object on a page, it places it into an invisible rectangular space called a “bounding box.”

You can specify the size, look, and feel of the *margins*, the *padding*, the *border*, and the content of that bounding box.

In CSS, the `width` property is defined as the distance between the left and right edges of the bounding box that surrounds the element’s content.

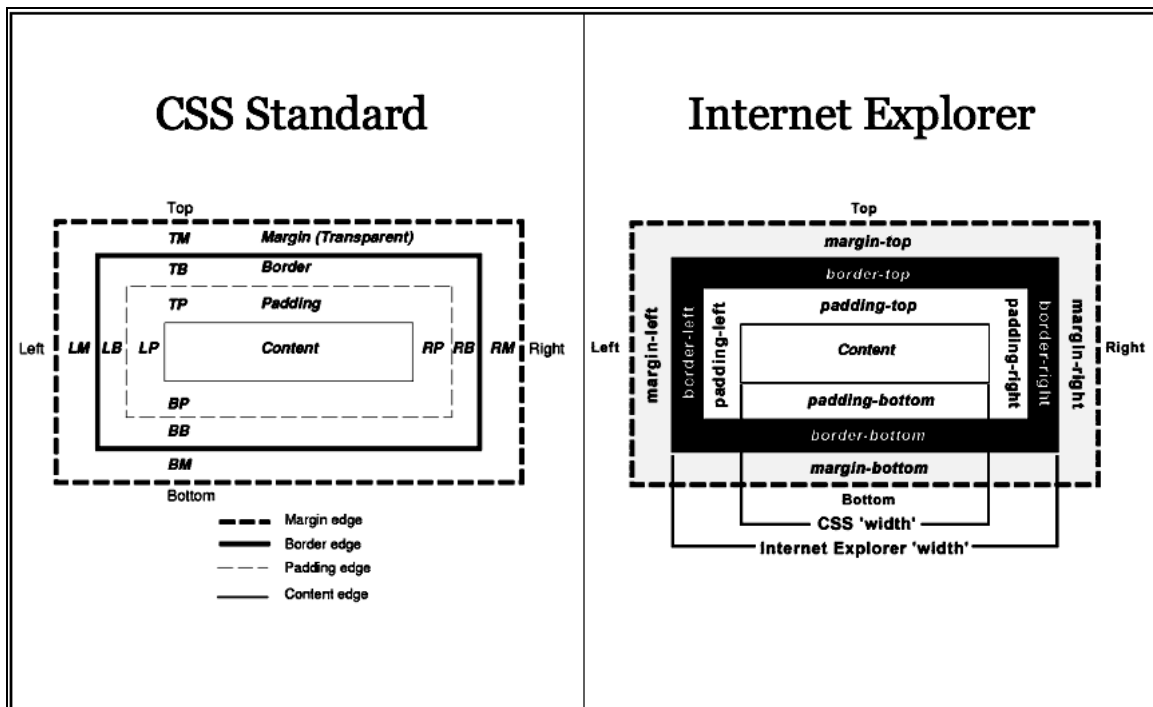
Likewise, the `height` property is defined in CSS as the distance between the top and bottom edges of the bounding box.

If no `!doctype` is defined in the HTML, *Internet Explorer* goes into “quirks mode” and interprets CSS box styles differently than most other web browsers. In this mode, *Internet Explorer* interprets the `width` and `height` properties to also include the border and padding belts that surround the element’s bounding box.

Internet Explorer will interpret the box model using the CSS standard if the HTML is written using a declared `!doctype`.

For example, this web page was created using the XHTML 1.1 doctype:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```



The CSS box model properties are:

- `padding`. The padding properties define the space between the edge of the content and the border. The space can be defined using a specific length, a percentage, or "auto".
 - `padding-top`. This property is used to set the amount of padding on the top.
 - `padding-right`. This property sets the amount of padding on the right.
 - `padding-bottom`. This property sets the amount of padding on the bottom.
 - `padding-left`. This property sets the amount of padding on the left.
 - `padding`. This property is used to combine the various padding properties into a single rule. The first value refers to the top, the second value the right, the third value the bottom, and the last value the left. If only one value is used, the top, right, bottom, and left values are the same.

```
#footer { padding: 0px auto 0px auto; }  
#header { padding: 5px;}
```

- `border`. The border properties define the size and style of the border (located between the padding and the margin).
 - `border-width`. This property sets the width of the border. Border widths can be defined as a specific width, a percentage, or thin, medium, or thick.
 - `border-color`. This property sets the color of the border. See Colors and Backgrounds on page 21 for color choices.
 - `border-style`. This property sets the style of the border. Style values are:
 - `none`. No border is drawn.
 - `dotted`. The border is a dotted line.
 - `dashed`. The border is a dashed line.
 - `solid`. The border is a solid line.
 - `double`. The border is a double-line.
 - `groove`. The border is a 3-dimensioned groove drawn using the colors defined in the border-color.
 - `ridge`. The border is a 3-dimensioned ridge drawn using the colors defined in the border-color.
 - `inset`. The border is a 3-dimensioned inset drawn using the colors defined in the border-color.
 - `outset`. The border is a 3-dimensioned outset drawn using the colors defined in the border-color.
 - `border-top`. This property is used to combine the various border properties into a single rule for the top portion of the border.
 - `border-right`. This property is used to combine the various border properties into a single rule for the right portion of the border.
 - `border-bottom`. This property is used to combine the various border properties into a single rule for the bottom portion of the border.
 - `border-left`. This property is used to combine the various border properties into a single rule for the left portion of the border.
 - `border`. This property is used to combine the various border properties into a single rule.

```
.important { border: thick red double;}
a:hover { border-bottom: thin #333 dotted;}
```

- `margin`. The margin properties define the space between the edge of the border and adjacent elements.

- `margin-top`. This property sets the amount of space between the edge of the top border and the adjacent elements on the top.
- `margin-right`. This property sets the amount of space between the edge of the right border and the adjacent elements on the right. If the right and left margins are set to `auto`, and a width is given to the element, the element will be centered.
- `margin-bottom`. This property sets the amount of space between the edge of the bottom border and the adjacent elements on the bottom.
- `margin-left`. This sets the amount of space between the edge of the left border and the adjacent elements on the left. If the right and left margins are set to `auto`, and a width is given to the element, the element will be centered.
- `margin`. This property is used to combine the various margin properties into a single rule. The first value refers to the top, the second value the right, the third value the bottom, and the last value the left. If only one value is used, the top, right, bottom, and left values are the same.

```
.important { margin: 0px auto 0px auto; }
```

- `width`. This property defines the width of the element. In *Internet Explorer* “quirks” mode, in addition to the width of the content, the width property also includes the border and the padding.


```
.landscape_image { width: 200px;}
```
- `height`. This property defines the height of the element. In *Internet Explorer* “quirks” mode, in addition to the height of the content, the height property also includes the border and the padding.


```
.portrait_image { height: 200px;}
```

In CSS2, the following properties were added to the box model:

- `outline`. This property sets the outline surrounding an element (after the border, but before the margin)
 - `outline-width`. This property sets the width of the outline. Outline widths can be defined as a specific width, a percentage, or thin, medium, or thick.
 - `outline-color`. This property sets the color of the outline. See Colors and Backgrounds on page 21 for color choices.
 - `outline-style`. This property sets the style of the outline. Style values are:
 - `none`. No outline is drawn.
 - `dotted`. The outline is a dotted line.
 - `dashed`. The outline is a dashed line.
 - `solid`. The outline is a solid line.
 - `double`. The outline is a double-line.
 - `groove`. The outline is a 3-dimensioned groove drawn using the colors defined in the `outline-color`.
 - `ridge`. The outline is a 3-dimensioned ridge drawn using the colors defined in the `outline-color`.
 - `inset`. The outline is a 3-dimensioned inset drawn using the colors defined in the `outline-color`.
 - `outset`. The outline is a 3-dimensioned outset drawn using the colors defined in the `outline-color`.
 - `outline`. This property is used to combine the various outline properties into a single rule.

```
.important { border: 3px red double;
              outline: thick green solid; }
```

```
a:hover { border-bottom: thin #333 dotted;
           outline: thin yellow dotted; }
```

- `min-height`. This property sets the minimum height of an element.
`#header { min-height: 150px; }`
- `max-height`. This property sets the maximum height of an element.
`#header { max-height: 500px; }`
- `min-width`. This property sets the minimum width of an element.
`#header { min-width: 800px; }`
- `max-width`. This property sets the maximum width of an element.
`#header { max-width: 950px; }`
- `overflow`. This property specifies what happens if content overflows an element's box (visible, hidden, scroll, auto):
 - `visible`. The overflow is not clipped. It renders outside the element's box. This is default
 - `hidden`. The overflow is clipped, and the rest of the content will be invisible
 - `scroll`. The overflow is clipped, but a scroll-bar is added to see the rest of the content
 - `auto`. If overflow is clipped, a scroll-bar should be added to see the rest of the content
 - `inherit`. Specifies that the value of the overflow property should be inherited from the parent element

```
#sidebar {overflow: scroll;}
```

Using CSS for Visual Effects

You can use Cascading Style Sheets to affect how elements appear. For example, consider a web site that uses a drop-down menu. To create this drop-down menu effect, the designer hides from view certain list items until the visitor's mouse hovers over the top list item.

To change the way in which elements appear, there are two CSS properties that you can use – `display` and `visibility`.

Display

All HTML elements (tags) are assigned a display property value of either *inline* or *block*.

Inline elements display in browsers horizontally.

```
[INLINE ELEMENT 1] [INLINE ELEMENT 2] [INLINE ELEMENT 3]
```

Examples of inline elements:

```
<a> <img> <strong> <em> <span>
```

Block elements display in browsers vertically (stacked one on top of the other).

```
[BLOCK ELEMENT 1]  
[BLOCK ELEMENT 2]  
[BLOCK ELEMENT 3]
```

Examples of block elements:

```
<p> <h1-h6> <div> <hr> <table> <ul> <ol>
```

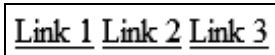
Using CSS, you can change the inherent display property to:

- Force a block display, by using the declaration `display: block;`
- Force an inline display, by using the declaration `display: inline;`
- Force a list, by using the declaration `display: list-item;`
- Hide elements, by using the declaration `display: none;`

Using display: block;

Normally, <a> tags display inline.

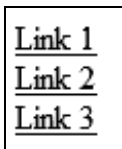
```
<body>
<a href="link1.html">Link 1</a> <a href="link2.html">Link 2</a> <a
href="link3.html">Link 3</a>
</body>
```



Link 1 Link 2 Link 3

But, if you add the style `a {display: block;}`, they will display as a vertical navigation menu:

```
<style type="text/css">
<!--
a { display: block;}
-->
</style>
</head>
<body>
<a href="link1.html">Link 1</a> <a href="link2.html">Link 2</a> <a
href="link3.html">Link 3</a>
</body>
```



Link 1
Link 2
Link 3

Using `display: inline;`

Normally, the heading tags display in block format.

```
<body>
<h1>Heading 1</h1><h2>Heading 2</h2><h3>Heading 3</h3>
</body>
```



Heading 1
Heading 2
Heading 3

But, to have them display inline, add the style `h1, h2, h3 {display: inline;}`.

```
<style type="text/css">
<!--
h1, h2, h3 {display: inline;}
-->
</style>
</head>
<body>
<h1>Heading 1</h1><h2>Heading 2</h2><h3>Heading 3</h3>
</body>
```



Heading 1Heading 2Heading 3

Visibility

All HTML elements (tags) are assigned a visibility property value of either *visible* or *hidden*.

- *visible*. This value makes the element display (this is the default setting).
`#main { visibility: visible;}`
- *hidden*. This value makes the element disappear.
`#secret {visibility: hidden;}`

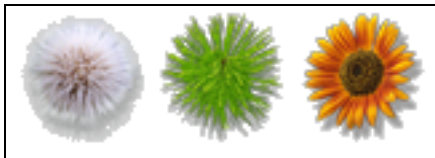
display: none; VS. visibility: hidden;

With the `display: none;` declaration, elements are removed from view and take up no space. When you use this declaration, it is as if the elements were never placed on the web page.

With the `visibility: hidden;` declaration, elements are merely hidden from view. They still occupy the same place that they originally would have taken up.

Examples:

Neither visibility: hidden nor display: none;



```
<!DOCTYPE html>
<html>
<head>
<title>Flowers</title>
<style type="text/css">
<!--
.hidden { visibility: hidden; }
.nodisplay { display: none; }
-->
</style>
</head>

<body>



</body>
</html>
```

Display: none;



```
<!DOCTYPE html>
<html>
<head>
<title>Flowers</title>
<style type="text/css">
<!--
.hidden { visibility: hidden; }
.nodisplay { display: none; }

-->
</style>
</head>

<body>



</body>
</html>
```

Visibility: hidden;



```
<!DOCTYPE html>
<html>
<head>
<title>Flowers</title>
<style type="text/css">
<!--
.hidden { visibility: hidden; }
.nodisplay { display: none; }

-->
</style>
</head>

<body>



</body>
</html>
```

Positioning

Using CSS, you can place elements exactly on a page using a technique called "positioning." Positioning is determined by an X axis and Y axis. To specify a point on the screen, you can use the X and Y coordinates for that point.

There are several ways to specify position in CSS: *absolute*, *relative*, *fixed*, *inherit*, and *static*.

The three most often used are *absolute*, *relative*, and *fixed*.

- *Absolute positioning* defines the position of a given bounding box from the top and left side margins of the web page. This not only allows objects to be placed in an exact location, it also allows objects to be placed one on top of another. Absolutely positioned elements do not follow the flow of the document.
- *Relative positioning* defines positioning in such a way that elements are offset from the previous element in the HTML code. This allows objects to be placed in relation to one another.

Note:

You can use *position: absolute* within a containing div of *position: relative*. The absolute position of the inner div will display relative to the containing div.

- *Fixed positioning* defines the position of a given box relative to the window and remains in its specified location even as the content scrolls underneath it. This value does not work in *Internet Explorer 6* or earlier. In *Internet Explorer 7* and newer, the browser must be in "standards-compliance mode."

To force standards-compliance mode, make sure your web pages are created using a declared `!doctype`.

For example, this web page was created using the XHTML 1.1 doctype:

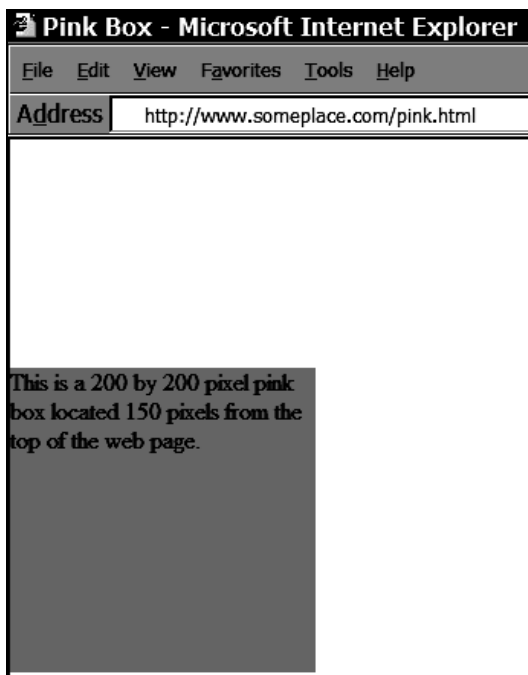
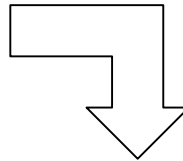
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

- *Inherit positioning* explicitly sets the value to that of the parent. If the parent is `position: absolute`, the child will be `position: absolute`. If the parent is `position: fixed`, the child will be `position: fixed`.
- *Static positioning* is the default. It essentially defines the position of a given box as an unpositioned element. It flows in the normal rendering sequence of the web page.

Using position: absolute;

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Pink Box</title>
<style type="text/css">
<!--
#pinkbox {
    position: absolute;
    width: 200px;
    background-color: pink;
    left: 0px;
    top: 150px;
    height: 200px;
}
-->
</style>
</head>

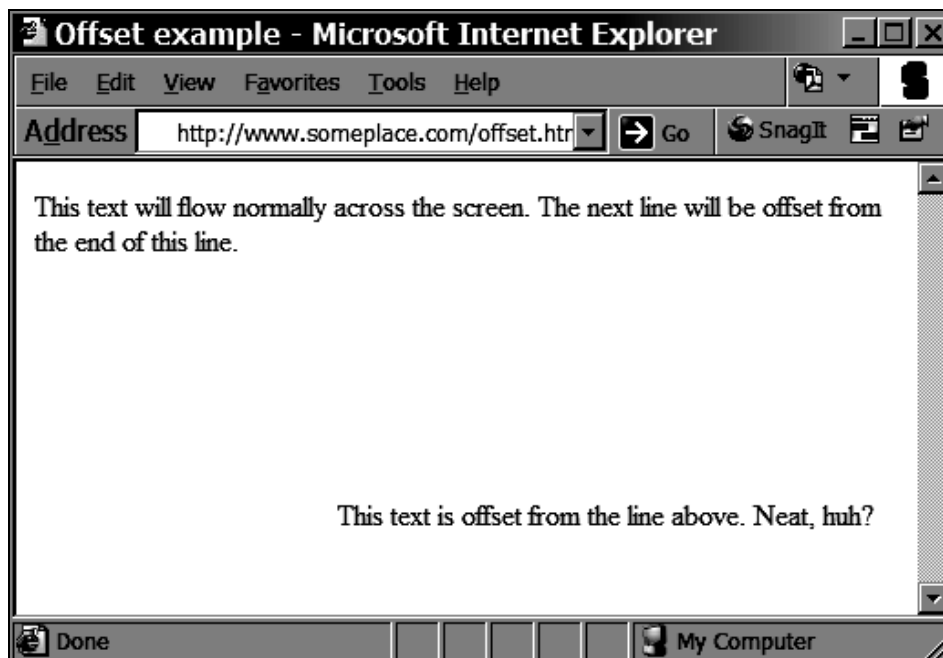
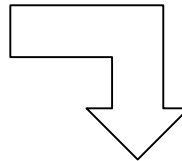
<body>
<div id="pinkbox">This is a 200
by 200 pixel pink box located
150 pixels from the top of the
web page.
</div>
</body>
</html>
```



Using position: relative;

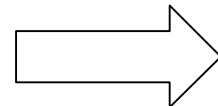
```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Offset example</title>
<style type="text/css">
<!--
.offset {
  position: relative;
  top: 150px;
  left: 50px;
}
-->
</style>
</head>

<body>
<p>This text will flow normally across
the screen. The next line will be offset
from the end of this line.
<span class="offset">This text is offset
from the line above. Neat, huh?</span>
</p>
</body>
</html>
```



Using position: fixed;

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Untitled Document</title>
<style type="text/css">
<!--
#links {
  position:fixed;
  border:dotted;
  border-color:#000000;
  width:20%;
  height:100%;
  z-index:1;
  left: 0px;
  top: 0px;
  background-color: #FFFFCC;
}
#main {
  position:absolute;
  left:25%;
  top:0px;
  width:70%;
}
-->
</style>
</head>
<body>
<div id="main">
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque ultrices, nibh ac rhoncus fermentum, orci sem
dapibus nisi, sed tincidunt lectus lectus at augue. In consectetur vehicula enim. In hac habitasse platea dictumst.
Donec a nisl vitae tortor tristique viverra. Sed at lorem a ante lobortis molestie. Nulla ullamcorper urna accumsan
diam. Aliquam non eros. Pellentesque egestas ultricies enim. Aenean lobortis. Nulla interdum commodo turpis. Sed ut mi
id elit vehicula sollicitudin. Sed lobortis, ligula sit amet euismod egestas, mi ante iaculis nunc, ut rhoncus magna
lectus ac arcu. In hac habitasse platea dictumst. Proin quis ligula vitae quam pharetra adipiscing. Pellentesque
tincidunt suscipit nibh. Ut fermentum suscipit justo. </p>
  <p>Fusce purus lectus, ultricies nec, aliquam at, facilisis id, arcu. Vestibulum quis mi vel massa porta hendrerit.
Nulla ullamcorper ligula nec lectus. Quisque tempor, augue in molestie gravida, eros arcu luctus tortor, eu dignissim
diam urna sed urna. Ut dictum ultrices lacus. In hac habitasse platea dictumst. Suspendisse sed purus blandit metus
ultricies suscipit. Proin diam justo, rhoncus eget, facilisis ut, lacus. Vivamus dignissim dui in justo.
Suspendisse elit. Nam nulla tortor, fringilla sed, faucibus quis, ullamcorper a, leo. Fusce blandit condimentum
turpis. Pellentesque vel odio et odio suscipit egestas. Nullam ullamcorper sagittis ipsum. Maecenas fringilla malesuada
pede. Duis ut quam. </p>
</div>
<div id="links">
  <p>This area is fixed and will never move. It's good for things like navigation bars.</p>
  <ul>
    <li><a href="page1.html">Page 1</a></li>
    <li><a href="page2.html">Page 2</a></li>
    <li><a href="page3.html">Page 3</a></li>
    <li><a href="page4.html">Page 4</a></li>
    <li><a href="page5.html">Page 5</a></li>
  </ul>
</div></body></html>
```



<p>This area is fixed and will never move. It's good for things like navigation bars.</p> <ul style="list-style-type: none">◆ Page 1◆ Page 2◆ Page 3◆ Page 4◆ Page 5	<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque ultrices, nibh ac rhoncus fermentum, orci sem dapibus nisi, sed tincidunt lectus lectus at augue. In consectetur vehicula enim. In hac habitasse platea dictumst. Donec a nisl vitae tortor tristique viverra. Sed at lorem a ante lobortis molestie. Nulla ullamcorper urna accumsan diam. Aliquam non eros. Pellentesque egestas ultricies enim. Aenean lobortis. Nulla interdum commodo turpis. Sed ut mi id elit vehicula sollicitudin. Sed lobortis, ligula sit amet euismod egestas, mi ante iaculis nunc, ut rhoncus magna lectus ac arcu. In hac habitasse platea dictumst. Proin quis ligula vitae quam pharetra adipiscing. Pellentesque tincidunt suscipit nibh. Ut fermentum suscipit justo.</p> <p>Fusce purus lectus, ultricies nec, aliquam at, facilisis id, arcu. Vestibulum quis mi</p>
Done	

<p>This area is fixed and will never move. It's good for things like navigation bars.</p> <ul style="list-style-type: none">◆ Page 1◆ Page 2◆ Page 3◆ Page 4◆ Page 5	<p>urna. Ut dictum ultrices lacus. In hac habitasse platea dictumst. Suspendisse sed purus blandit metus ultricies suscipit. Proin diam justo, consequat id, rhoncus eget, facilisis ut, lacus. Vivamus dignissim dui in justo. Suspendisse elit. Nam nulla tortor, fringilla sed, faucibus quis, ullamcorper a, leo. Fusce blandit condimentum turpis. Pellentesque vel odio et odio suscipit egestas. Nullam ullamcorper sagittis ipsum. Maecenas fringilla malesuada pede. Duis ut quam.</p> <p>Donec erat. Mauris enim nibh, ornare quis, ullamcorper nec, consectetur ut, urna. Fusce id nulla in orci faucibus dictum. Praesent aliquet tempus purus. In mi ligula, facilisis ac, mollis sed, sodales at, magna. Nullam pharetra. Nullam ac lorem sit amet augue porta consequat. Donec justo erat, rhoncus congue, consequat a, malesuada ac, leo. Suspendisse potenti. Aliquam tempus elementum insum. Praesent eu felis. In adipiscing</p>
Done	

Layers and the Bounding Box

The Box Model

When the browser draws an object on a page, it places it into an invisible, rectangular space called a bounding box. You can set the box's exact location on the page or offset it from other objects on the page. As mentioned in the previous pages, you can also specify the size of the box. For more information, see the previous section on the box model (page 49).

With CSS, these boxes can be stacked one on top of another as layers. Horizontal and vertical positioning happen along the X and Y axes, and the layered positioning happens along the Z axis.

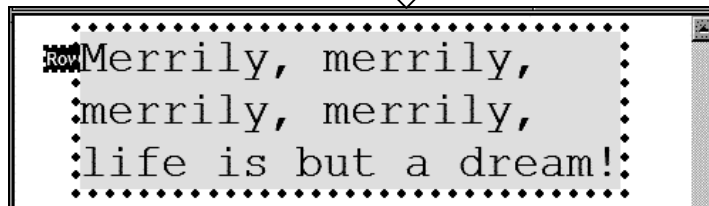
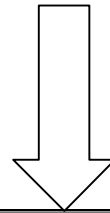
The z-index Style

The Z axis is set using the CSS style `z-index`, which allows you to specify which layer appears on top of the others. By setting the `z-index` higher or lower, an object can move up and down a stack. The higher the `z-index`, the more "on top" it is.

Layering Example 1

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Untitled Document</title>
<style type="text/css">
<!--
.over {
    font-family: "Courier New", Courier, mono;
    font-size: 200%;
    color: #000000;
    background-color: #FFFF00;
    border: thick dotted #990000;
    position: absolute;
    left: 40px;
    top: 5px;
    z-index: 2;
}
.under {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 75%;
    font-weight: bold;
    color: #FFFFFF;
    background-color: #009900;
    border: thin dashed #0000FF;
    position: absolute;
    left: 20px;
    top: 20px;
    z-index: 1;
}
-->
</style>
</head>

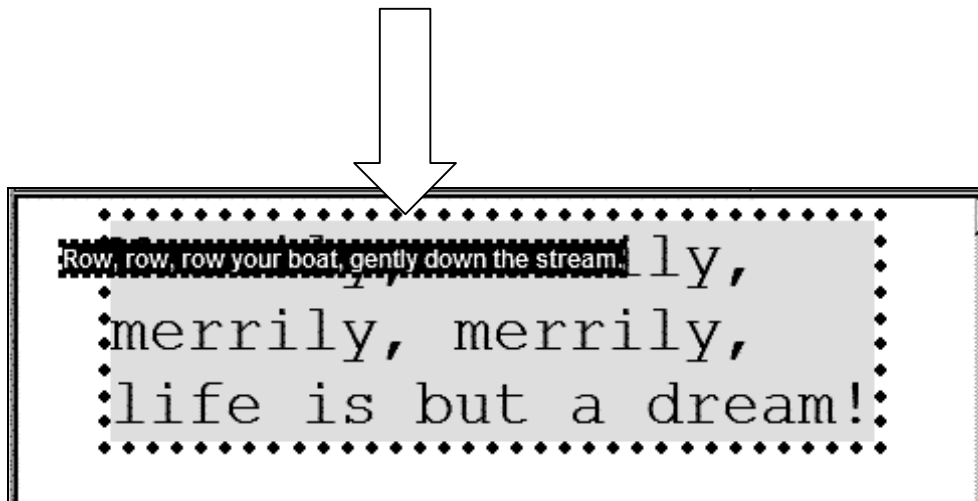
<body>
<p class="under">Row, row, row your boat, gently down the stream.</p>
<p class="over">Merrily, merrily, merrily, merrily, life is but a dream!</p>
</body>
```



Layering Example 2

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Untitled Document</title>
<style type="text/css">
<!--
.over {
    font-family: "Courier New", Courier, mono;
    font-size: 200%;
    color: #000000;
    background-color: #FFFF00;
    border: thick dotted #990000;
    position: absolute;
    left: 40px;
    top: 5px;
    z-index: 1;
}
.under {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 75%;
    font-weight: bold;
    color: #FFFFFF;
    background-color: #009900;
    border: thin dashed #0000FF;
    position: absolute;
    left: 20px;
    top: 20px;
    z-index: 2;
}
-->
</style>
</head>

<body>
<p class="under">Row, row, row your boat, gently down the stream.</p>
<p class="over">Merrily, merrily, merrily, merrily, life is but a dream!</p>
</body>
```



Clip (only for position: absolute)

The clip property lets you specify the dimensions of an absolutely positioned element that should be visible. The element will be clipped into the defined rectangle shape.

Clip properties:

- `rect`. This property clips an element. The *rect* value's sizes can be any of the unit measurements allowed in CSS (see *Unit Measurements* on page 8).
`rect ([top size] [right size] [bottom size] [left size])`
- `auto`. No clipping will be applied. This is default.
- `inherit`. Specifies that the value of the clip property should be inherited from the parent element

```
img { position:absolute;  
      clip:rect(0px 60px 200px 0px);  
}
```

One use of the *clip* property is to use one image to display different things at different places on the web page. For example, suppose this is the image *3flowers.png*:



Using the clip property, a web designer could display just the flower desired without needing to load 3 separate photos (the *3flowers.png* file would be loaded and stored in the browser's cache):

```
<!DOCTYPE html>
<html>
<head>
<title>Flowers</title>
<style type="text/css">
<!--
.dandelion { position: absolute; clip:rect(0,55px,60px,0px); }
.spiked { position: absolute; clip:rect(0,110px,60px,55px); }
.sunflower { position: absolute; clip:rect(0,157px,60px,110px); }
p {margin-bottom: 100px;}
-->
</style>
</head>
<body>
<p>Dandelion only: </p>
<p>Spiked only: </p>
<p>Sunflower only: </p>
</body>
</html>
```

Dandelion only:



Spiked only:



Sunflower only:



Float / Clear

If you want to wrap content around other content (such as text around a picture), you can use the `float` property.

The `float` property (`left`, `right`) determines on which side of the bounding box the element aligns, so that the other content wraps around it.

The `clear` property (`left`, `right`, `both`, `none`, `inherit`) forces an element to start below a floated element.

Float/Clear Example

In *Example 1*, the three flowers float to the left of the paragraph.

In *Example 2*, the three flowers float to the right of the paragraph.

In *Example 3*, although the three flowers are floating to the left, the paragraph is forced onto its own line.

In *Example 4*, although the three flowers are floating to the right, the paragraph is forced onto its own line.

```
<!DOCTYPE html>
<html>
<head>
<title>Flowers</title>
<style type="text/css">
<!--
.floatleft { float: left; }
.floatright { float: right; }
.nofloat { clear: both; }
-->
</style>
</head>

<body>
<p class="floatleft"></p>
<p>Example 1: These are three flowers that are floating to the left of the content that is present
on this web page. Note that the content floats to the right but the flowers float to the left.
</p>

<p class="floatright"></p>
<p>Example 2: These are three flowers that are floating to the right of the content that is present
on this web page. Note that the content floats to the left but the flowers float to the right.
</p>

<p class="floatleft"></p>
<p class="nofloat">Example 3: Although the flowers are floating to the left, because this paragraph
has been cleared, the flowers are on one line and the text is on another.</p>

<p class="floatright"></p>
<p class="nofloat">Example 4: Although the flowers are floating to the right, because this paragraph
has been cleared, the flowers are on one line and the text is on another.</p>

</body>
</html>
```



Example 1: These are three flowers that are floating to the left of the content that is present on this web page. Note that the content floats to the right but the flowers float to the left.

Example 2: These are three flowers that are floating to the right of the content that is present on this web page. Note that the content floats to the left but the flowers float to the right.



Example 3: Although the flowers are floating to the left, because this paragraph has been cleared, the flowers are on one line and the text is on another.



Example 4: Although the flowers are floating to the right, because this paragraph has been cleared, the flowers are on one line and the text is on another.

Modifying List Elements

In HTML, by default, unordered lists () appear as bullets and ordered lists () appear as numbers. Using CSS, you can modify how list items appear.

Note: *Internet Explorer* only recognizes the **bolded** values listed below.

- Properties:
list-style, list-style-type, list-style-image, list-style-position
- Values:
disc, circle, square, decimal, decimal-leading-zero, lower-roman, upper-roman, lower-alpha, upper-alpha, lower-greek, lower-latin, upper-latin, hebrew, armenian, georgian, cjk-ideographic, hiragana, katakana, hiragana-iroha, katakana-iroha, none, url("graphic.gif"), inside, outside

Examples:

```
ul { list-style: disc; }
ol { list-style: upper-roman;}
li { list-style: url("blackball.gif");}
ul li { list-style-position: inside;}
```

The following page gives the code for all list style types and then shows how the codes are rendered in *Firefox* and in *Internet Explorer*.

HTML / CSS Code

```

<head>
<title>Lists</title>
<style type="text/css">
<!--
.a {list-style:armenian;}
.b {list-style:circle;}
.c {list-style:cjk-ideographic;}
.d {list-style:decimal;}
.e {list-style:decimal-leading-zero;}
.f {list-style:disc;}
.g {list-style:georgian;}
.h {list-style:hebrew;}
.i {list-style:hiragana;}
.j {list-style:hiragana-iroha;}
.k {list-style:katakana;}
.l {list-style:katakana-iroha;}
.m {list-style:lower-alpha;}
.n {list-style:lower-greek;}
.o {list-style:lower-latin;}
.p {list-style:lower-roman;}
.q {list-style:none;}
.r {list-style:square;}
.s {list-style:upper-alpha;}
.t {list-style:upper-latin;}
.u {list-style:upper-roman;}
.v {list-style:url(blackball.gif);}
.w {list-style:inside;}
.x {list-style:outside;}
-->
</style>
</head>
<body>
<ul>
<li class="a">armenian</li>
<li class="b">circle</li>
<li class="c">cjk-ideographic</li>
<li class="d">decimal</li>
<li class="e">decimal-leading-zero</li>
<li class="f">disc</li>
<li class="g">georgian</li>
<li class="h">hebrew</li>
<li class="i">hiragana</li>
<li class="j">hiragana-iroha</li>
<li class="k">katakana</li>
<li class="l">katakana-iroha</li>
<li class="m">lower-alpha</li>
<li class="n">lower-greek</li>
<li class="o">lower-latin</li>
<li class="p">lower-roman</li>
<li class="q">none</li>
<li class="r">square</li>
<li class="s">upper-alpha</li>
<li class="t">upper-latin</li>
<li class="u">upper-roman</li>
<li class="v">url(blackball.gif)</li>
<li class="w">inside</li>
<li class="x">outside</li>
</ul>
</body>

```

Firefox

- U. armenian
 - ◊ circle
- 三. cjk-ideographic
- 4. decimal
- 05. decimal-leading-zero
 - ◆ disc
- Ⴂ. georgian
- ⱦ. hebrew
- ㇀. hiragana
- ㇀. hiragana-iroha
- ㇀. katakana
- ㇀. katakana-iroha
- m. lower-alpha
- ξ. lower-greek
- o. lower-latin
- xvi. lower-roman
- none
 - square
- S. upper-alpha
- T. upper-latin
- XXI. upper-roman
 - ◆ url(blackball.gif)
 - ◆ inside
 - ◆ outside

Internet Explorer

- armenian
- ◊ circle
- cjk-ideographic
- 4. decimal
 - decimal-leading-zero
 - disc
 - georgian
 - hebrew
 - hiragana
 - hiragana-iroha
 - katakana
 - katakana-iroha
- m. lower-alpha
 - lower-greek
 - lower-latin
- xvi. lower-roman
- none
 - square
- S. upper-alpha
- upper-latin
- XXI. upper-roman
 - url(blackball.gif)
 - inside
 - outside

Tables and CSS

Although you should never use tables to control page layout, tables can and should be used to list data in tabular format. You can use CSS to greatly improve the look and feel of your tables.

The CSS styles you can apply to a table include:

- **Borders**
 - **border.** As mentioned earlier, used to define a border.
`table, th, td { border: solid thin pink;}`
 - **border-collapse.** This property (collapse, separate, inherit) sets whether the table borders are collapsed into a single border or detached as in standard HTML.
`table { border-collapse: collapse;}`
 - **border-style.** As mentioned earlier, this property (dotted, dashed, solid, double, groove, ridge, inset, outset) sets the style of borders.
`table { border-style: dotted;}`
 - **border-color.** As mentioned earlier, this property sets the color of the border (color name, hexadecimal code, or rgb value)
`table { border-color: rgb(100%,0,0);}`
 - **border-spacing.** This property (width and height) sets the distance between the borders of adjacent cells. Only affects tables whose borders are detached (`border-collapse: separate;`).
`table { border-collapse: separate; border-spacing: 5px 10px;}`
- **Other table properties:**
 - **width.** This property sets the width of the element (in pixels, percentage, or em).
`table { width: 85%;}`
`td, th { width: 150px;}`
`tr { width: 1.5em;}`
 - **height.** This property sets the height of the element (in pixels, percentage, or ex).
`table { height: 85%;}`
`td, th { height: 150px;}`
`tr { height: 1.5ex;}`
 - **padding.** This sets the amount of space between the edge of the content and the border.
`td { padding: 15px;}`

- **background.** This property sets the color or picture that becomes the **background of the element.**
`table { background: #eee url(graygradient.gif);}`
- **empty-cells.** This property (**show, hide, inherit**) defines how borders and backgrounds on empty cells are displayed in a table. Only affects tables whose borders are detached (`border-collapse: separate;`).
`table { border-collapse: separate; background: yellow; empty-cells: hide;}`
- **text-align.** This property (**center, left, right, justify, inherit**) sets the horizontal alignment of text in an element.
`td, th { text-align: center;}`

Changing the Cursor's Appearance

For dramatic or artistic effect, web designers can now change the appearance of the cursor of the visitor's mouse, using the `cursor` property.

The allowed values of the cursor property are:

- `URL`. This value allows you to specify a comma separated of URLs to custom cursors.
- `auto`. This value will display the cursor that the browser sets as its default.
- `crosshair`. This value will display a crosshair icon as a cursor.
- `default`. This value will display the default cursor.
- `e-resize`. This value will display a cursor that indicates that an edge of a box is to be moved right (east).
- `help`. This value will display a cursor indicating that help is available.
- `move`. This value will display a cursor indicating that something that should be moved.
- `n-resize`. This value will display a cursor indicating that an edge of a box is to be moved up (north).
- `ne-resize`. This value will display a cursor indicating that an edge of a box is to be moved up and right (north/east).
- `nw-resize`. This value will display a cursor indicating that an edge of a box is to be moved up and left (north/west).
- `pointer`. This value will display a pointer icon as the cursor.
- `progress`. This value will display a cursor indicating that the web site is in progress (busy).
- `s-resize`. This value will display a cursor indicating that an edge of a box is to be moved down (south).
- `se-resize`. This value will display a cursor indicating that an edge of a box is to be moved down and right (south/east).
- `sw-resize`. This value will display a cursor indicating that an edge of a box is to be moved down and left (south/west).
- `text`. This value will display a cursor indicating text.
- `w-resize`. This value will display a cursor indicating that an edge of a box is to be moved left (west).
- `wait`. This value will display a cursor indicating that the web site is busy.
- `inherit`. This value will display a cursor that is the same value of the its parent element.

Printing and CSS

As mentioned earlier (see *CSS: External location* on page 12), you can specify a different style sheet to be deployed when the visitor prints a page.

Using the `<link>` method, use the following HTML:

```
<link rel="stylesheet" type="text/css" href="print.css" media="print">
```

Using the `@import/media` method, use the following HTML:

```
@media url("print.css") print;
```

In addition, the following properties are available:

- `page-break-after`. This property sets a page-break (*auto, always, avoid, left, right, inherit*) after an element.
`table { page-break-after: always; }`
- `page-break-before`. This property sets a page-break (*auto, always, avoid, left, right, inherit*) before an element.
`table { page-break-before: always; }`
- `page-break-inside`. This property sets a page-break inside (*auto, always, avoid, left, right, inherit*) an element.
Note: This property is currently only supported in Opera and Internet Explorer 8 and newer (not currently supported in Firefox, Safari, or Chrome).
`p { page-break-inside: inherit; }`
- `orphans`. This property sets the limit on the number of orphaned lines at the bottom of a page when a page break occurs inside of an element.
Note: This property is currently only supported in Opera and Internet Explorer 8 and newer (not currently supported in Firefox, Safari, or Chrome).
`p { orphans: 2; }`
- `widows`. This property sets the minimum number of widowed lines at the top of a page when a page break occurs inside of an element.
Note: This property is currently only supported in Opera and Internet Explorer 8 and newer (not currently supported in Firefox, Safari, or Chrome).
`p { widows: 2; }`

