

# **CSS**

## ***Cascading Style Sheets***

### **Module 2** *(CSS Models)*

*Applying Colors*  
*Background Techniques*  
*Formatting with the Box Model*  
*The Border Model*  
*Floating Elements*  
*Styling Lists*  
*Positioning Elements*  
*Layering Elements (the Z-Index Property)*  
*Applying CSS Wisely*  
*Dynamically Linked Style Sheets*  
*Contextual Selectors*  
*Validating and the Future of CSS*

<b>1.) Applying Colors</b>	4
<b>Color Names</b>	4
<b>The Color and Background-Color Properties</b>	4
<b>Hexadecimal Values</b>	4
<b>The 216-Color Browser-Safe Palette</b>	5
<i>Shorthand Hex Values</i>	5
<b>The Full Color Palette</b>	5
<b>RGB Values</b>	5
<i>Support for RGB Notation</i>	6
<b>2.) Background Techniques</b>	6
<b>The Background-Image Property</b>	6
<b>The Background-Repeat Property</b>	6
<b>Background Images for Elements Other Than &lt;body&gt;</b>	6
<i>Height and Width</i>	6
<b>The Background-Attachment Property</b>	7
<b>The Background-Position Property</b>	7
<b>The Background Property</b>	8
<b>3.) Formatting with the Box Model</b>	8
<b>Replaced Elements</b>	8
<b>Box Properties and Inheritance</b>	9
<b>Margins</b>	9
<i>Length Units</i>	9
<i>Negative Margins</i>	9
<b>The Margin Property</b>	9
<i>Hanging Indents</i>	10
<i>Collapsing Margins</i>	10
<b>4.) The Border Model</b>	11
<b>The Border Property</b>	11
<b>Applying Borders to Page Elements</b>	11
<i>Border Properties for Internet Explorer</i>	11
<i>Border Properties for Netscape Navigator 4.x</i>	11
<b>Padding</b>	11
<b>5.) Floating Elements</b>	12
<b>Drop-Caps: How It's Supposed to Work</b>	12
<b>The Clear Property</b>	12
<b>6.) Styling Lists</b>	13
<b>List-Style-Type</b>	13
<b>The List-Style-Position Property</b>	13
<b>The List-Style-Image Property</b>	13
<b>The List-Style Property</b>	13
<b>7.) Positioning Elements</b>	14
<b>The Position Property</b>	14
<b>The Top and Left Properties</b>	14
<b>Absolute Positioning</b>	14
<i>Reference Points</i>	14
<b>The Width Property</b>	15
<b>Relative Positioning</b>	15
<b>The Overflow Property</b>	15
<i>Browser Issues to Consider</i>	16
<b>The Display Property</b>	16
<b>The Visibility Property</b>	16
<b>8.) Layering Elements (the Z-Index Property)</b>	17
<b>Dynamic Layers</b>	17
<b>9.) Applying CSS Wisely</b>	17
<i>Simplify, Simplify</i>	17
<i>Make Complete Testing a Routine</i>	18

<b>Styles That Break Down Gracefully</b>	18
<i>The Hover Pseudo-Class</i>	18
<i>Cursor Styles</i>	19
<b>CSS Filters</b>	20
<i>Filters: Why They're Cool, Yet Unpopular</i>	20
<i>The Ubiquitous Caveat</i>	21
<i>Getting Started: The Filter Property and Syntax</i>	21
10.) <b>Dynamically Linked Style Sheets</b>	21
11.) <b>Contextual Selectors</b>	22
<i>The Advantage of Contextual Styles</i>	22
<i>Contextual Selectors and Specificity</i>	22
12.) <b>Validating and the Future of CSS</b>	23
<i>The Future of CSS</i>	23
<i>A Final Note on CSS</i>	23

---

## 1.) Applying Colors

With CSS, applying colors to web documents is as easy as applying colors with HTML attributes, but far more powerful and flexible. You can specify colors in three ways: with color names, with hexadecimal values, and with RGB (Red, Green, Blue) values.

### Color Names

There are 16 predefined color names that are considered "safe colors" because they display exactly the same on all browsers and platforms. These colors are *aqua, black, blue, silver, gray, white, maroon, red, purple, fuchsia, teal, green, lime, olive, yellow, and navy*. There are hundreds of additional color names that the 4.x and later browsers support, but their appearance might not be the same across all platforms. Older browsers that don't support a particular color name might display a default color instead, or no color at all. Beyond the 16 safe colors, you should always test your color names in a variety of browsers and platforms to ensure conformity.

### The Color and Background-Color Properties

The *color* property applies color to an element's foreground (text), while the *background-color* property applies color to an element's background. You can declare a color for any text element. Most elements can have a background color, including paragraphs, lists, and tables. Specifying color and background color is a piece of cake – to give a level-one heading blue text and a silver background, for example, your rule would look as follows:

```
H1 { color: blue;
      background-color: silver; }
```

With the *background-color* property, you can also specify the value *transparent*, which is the default value. You can use this value on a child element if you don't want it to inherit the background color of its parent. Unfortunately though, Internet Explorer for Windows, and some early versions of Navigator, don't support it.

### Hexadecimal Values

Using hexadecimal values to apply color to Web documents is a more powerful tool than applying color names, because there are more "browser-safe" colors available in the hexadecimal palette: 216 colors display identically on all platforms. (40 colors in the standard system palette of 256 colors render differently depending on the platform – Mac, Windows, Windows 95, and so forth.)

Hexadecimal values are equivalent to RGB (Red, Green, Blue) values, which specify exact gradations of red, green and blue that combine to make a color. Hexadecimal notation consists of a pound-sign (#) followed by any of the following values: 0-9, and A-F. Each of the three colors is given two decimal values (for a total of 6 decimals – thus the name *hexa(6)* – decimal). A value of 00 gives no color value, while FF gives a full red, green, or blue value, and values in between give varying degrees of each color. For example, the following rule gives the page background no value for red, green, or blue, so the page background will be rendered black:

**R G B**

```
body { background-color: #000000 }
```

The first two zeroes represent the red values, the second two represent the green values and the last two represent the blue values. The next example gives the page background full red, green, and blue value, so the page background will be rendered white:

```
body { background-color: #FFFFFF }
```

Color values are not case sensitive so you can use upper or lower case characters for the letter values to define your colors. That is to say *#CCCCCC*, *#ccccc* and *#ccCcc* will all produce the same results.

### The 216-Color Browser-Safe Palette

The “browser-safe” colors in the hexadecimal palette are those colors that are represented by combinations of the following six values: 00,33,66,99,CC, and FF. So, since there are six possible values for each of the three RGB variables, six times six times six equals 216 possible combinations. You might find 216 colors a bit limiting, and considering there are millions more out there, you’d be right. But take some time and view the source of some of your favorite sites – chances are they use colors within the browser-safe palette.

### *Shorthand Hex Values*

You can use a shorthand notation to declare a color by using single characters to represent two-character color values. For example, the following two rules specify the same color.

Standard hexadecimal notation:

```
h1 { color: #3366CC }
```

Shorthand hexadecimal notation:

```
h1 { color: #36C }
```

This means, however, that the shorthand method can’t access all colors in the palette, because you have to use a color with RGB pairs, such as #66CCFF. Instead of typing all six characters, you use a shorthand for each pair. In other words, you couldn’t use the shorthand notation to declare the hex color #AC3FDE.

### **The Full Color Palette**

Of course, you don’t have to stick to the 216 “browser-safe” colors. Actually, there are over 16 million colors available in the hexadecimal RGB palette. There are 256 possible gradations (0-255) for each red, green, and blue value. So, 256 times 256 times 256 equals 16,777,216 colors. Not too shabby.

Specifying a hexadecimal color value outside the “safe” 216 means that one or more of the color values won’t consist of 00,33,66,99,CC, or FF. (And it likely won’t produce a consistent color across all platforms.) An “unsafe” hex value might look like this:

```
h1 { background-color: #3C6F96 }
```

Or even this:

```
h1 { background-color: #ABCDEF }
```

### **RGB Values**

Sure, hex values are cool, but CSS introduces a more intuitive method for specifying color: RGB format. Two examples of the syntax for applying colors in RGB format are as follows:

```
div { background-color: rgb(100,50,55) }  
td { color: rgb(40%, 12%, 67%) }
```

The value of the color or background-color property must begin with rgb, followed by parentheses that enclose the red, green, and blue values. You can specify values by integer (0-255) or percentage (0-100%). You can also combine the two methods, as shown in the following example:

```
p { color: rgb(53%,114,21%) }
```

The preceding rule states that paragraph text will be displayed in a color consisting of 53% red, 114 (of 255) green, and 21% blue, which is a brownish-green color, incidentally. A greater percentage value or number will result in greater intensity of that color. You can create shades of gray by specifying equal

values of red, green, and blue in either hexadecimal or RGB notation. For example, `rgb (190,190,190)` is a light shade of gray. The exception to this, of course, is black (all values 255) and white (all values 0).

### *Support for RGB Notation*

Internet Explorer 4.x supports RGB notation, but early versions of Navigator 4 don't (4.0, 4.01, 4.02, 4.03). If you're using Navigator 4.5 or later you're all set. As less and less users out there are using older browser versions it becomes easier to use RGB notation with confidence.

---

## 2.) Background Techniques

Using images as backgrounds is a common HTML technique. Browsers force background images to repeat horizontally and vertically, or "tile" so that the image can remain small in size while still filling up the entire page. With straight HTML, there's no method to control how the background image tiles, or if it should even tile at all. With CSS, you can apply background images and control how the image is repeated on the page.

### ***The Background-Image Property***

With the *background-image* property, you can use any image as a background for the entire document, and you'll get the same results that you would using the HTML `<body background="image.gif">`. The syntax is relatively simple; the *background-image* property is used, together with "url", followed immediately by parentheses that contain the filename or location of the image, as follows:

```
body { background-image: url(background.gif) }
```

### ***The Background-Repeat Property***

```
repeat (default) | repeat-x | repeat-y | no-repeat
```

With the *background-repeat* property, you can tell the browser to repeat the background image horizontally across the browser window using the *repeat-x* value, or vertically using the *repeat-y* value. The default value of the *background-repeat* property is *repeat*, which tiles the background image horizontally and vertically (x = horizontal, y = vertical). You can also use *no-repeat* to prevent the background image from tiling in either direction.

### ***Background Images for Elements Other Than <body>***

You can also attach a background image to a particular element on the page. Attaching background images to page elements other than `<body>` can only be done with CSS. The image you insert as a background will fill up the size of only the element's containing area, regardless of how large the image is. If, for example, you have an image that is 3 inches wide by 4 inches high, and you use the image as a background for a paragraph element that is only one sentence long in a normal font size, you'll only see the top of the image. The area of the paragraph will not stretch to accommodate the image. And that's where the *height* and *width* properties are handy.

### *Height and Width*

Without specifying a height or width, HTML elements automatically size according to their content. For example, a heading made up of the words "Read me first, I'm a heading!" will have a height and width dependent upon the font size used for the heading. With the *width* and *height* properties, you can control

the size of an element, regardless of the size of the element's contents. Older browsers (4.x) have only partial support for the height and width properties.

### The Background-Attachment Property

```
scroll (default) | fixed
```

By default, background images are attached to page elements; they scroll together with page content. With the *background-attachment* property, however, you can secure the background so that the image does not scroll with page content. Page elements appear to float on top of the image. This feature is unique to CSS and is particularly useful if the background image is a watermark that you want to remain in position when a user scrolls through the content.

There are two possible values to the background-attachment property; *scroll*, which is the default, and *fixed*, which holds the background image in place. Navigator (Mac and Windows) doesn't support background-attachment, but Internet Explorer does, and depending on how you use it, it's a style that breaks down gracefully.

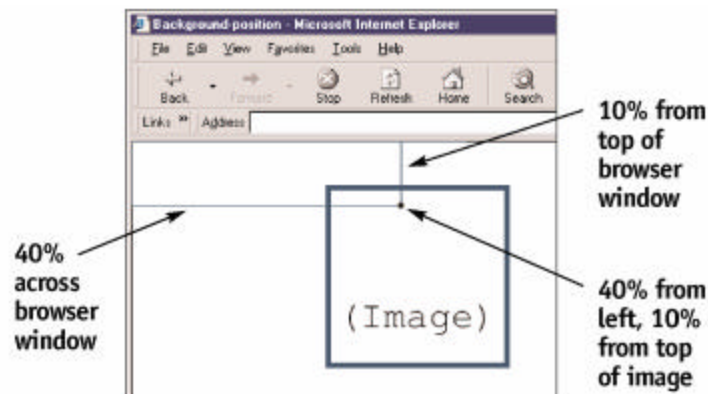
### The Background-Position Property

```
(length or %) | top | center | bottom | left | right
```

With the *background-position* property, you can determine where a background image is positioned, on the canvas (*body*), or behind a particular element on the page. Its position is specified in either percentages, length values, or keywords (*top*, *left*, *right*, *bottom*, *center*). Values are specified in pairs representing the horizontal and vertical positions, as shown below:

```
body { background-position: 40% 10% }
```

The first value specifies the horizontal position from the top left edge of the element (in this case, *<body>* – the browser window), while the second value specifies its vertical position. The preceding rule states that the point in the image that's 40% across and 10% down is positioned at those same coordinates in the browser window (see figure below).



It works differently using absolute length values, however. For example, in Internet Explorer, specifying background-position by length values, such as 2 inches by 1 inch, will use the top left edge of the image as its reference point. In other words the top left corner of the image will be positioned 2 inches from the left and 1 inch from the top of the browser window.

With all value types, if the horizontal position is the same as the vertical position, you can simply use one value, like this:

```
body { background-position: 50% }
```

This rule will position the background in the center of the page, 50% across the *<body>* element and 50% down. Using the keyword *center* would have precisely the same effect. If you choose the keyword *bottom*,

the bottom of the image is positioned at the bottom of the page; likewise for *top*. Percentages are probably the best method, provided that other content will scale according to the size of the browser window. If you want the background image to appear in a fixed position no matter the size of the user's browser, you can use absolute length values, like inches or pixels.

### **The Background Property**

You can apply all the background properties at once using the *background* shorthand property. All five of the background properties are implicit in the shorthand. You can specify values for one property, or all five, and in any order. For any value left out, that property's default, or initial value, is assumed. In other words, you could have a rule that looked like this:

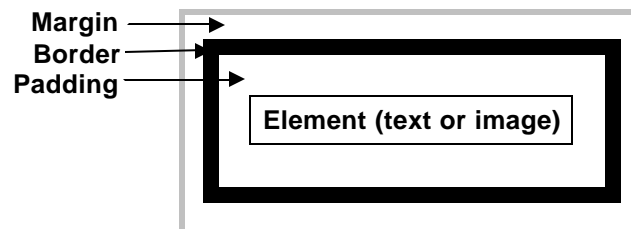
```
body { background: url(this.gif) no-repeat fixed 10% 20% }
```

This rule would apply the *background-image*, *background-repeat*, *background-attachment*, and *background-position* properties all at once. Each value implies the property they're associated with. This is really great, but be careful of the results in 4.x browsers.

---

## 3.) Formatting with the Box Model

Block-level elements like headings, paragraphs, divisions, and lists are defined according to the CSS box model. The *box model* describes the basic structure of page elements. A box consists of the element itself (either text or a replaced element), and several options: the *padding* that surrounds the element; the *border* around the padding; and the *margin* around the border (see diagram below). The element's box, therefore, includes all of these components. Each component is controlled with its own properties, and they're all optional – that is, you don't have to specify an element's padding, border, or margin.



The box model is perhaps the most important fundamental of CSS. It's also where the older (4.x) browsers demonstrate the majority of their shortcomings, especially Navigator 4.x. The box model is the formatting model for CSS, based on concentric rectangles. The box properties dictate how an element's content box is formatted. There are two types of box; *block-level* boxes and *inline* boxes, but you'll mainly deal with block-level boxes. Every element creates a box. So, a block-level element like a paragraph `<p>` or a division `<div>` creates a block-level box when displayed.

### **Replaced Elements**

Boxes apply to block-level text elements and *replaced elements*, which are simply placeholders that are replaced by an element outside of the document (`<img>` or `<object>` elements, for example). Images are the most common replaced element. Form elements like `<input>`, `<select>`, and `<textarea>` are also considered replaced elements.

### **Box Properties and Inheritance**

Unlike text and font properties, box properties are not inherited from parent to child; a paragraph, for example, will not inherit the margin of its parent element the same way it would inherit properties such as font and color. Elements may *appear* to inherit certain box properties, however, because an element's margin influences the placement of any child element's box.

In the same way that elements contain other elements, boxes can contain other boxes. If, for example, a paragraph `<p>` is contained by a division `<div>`, then the paragraph's box is displayed within the division's box. Let's say the `<div>` has a left margin of one inch. As a child of the `<div>`, the `<p>` element is confined within the margin of its parent box, but the element itself does not have a one-inch left margin. If it did, it would appear one inch from the left of its parent `<div>`, as shown in Figure **Error! Reference source not found.**-2. You could, however, force the `<p>` element outside of its parent margin by using a negative margin.

The default, or initial values for an element's border, margin, and padding is 0, which means none of these element features will display without setting values greater than 0. (Browsers do add their own default padding, however.)

### **Margins**

There are four properties that control margins; *margin-top*, *margin-bottom*, *margin-left*, and *margin-right*. You can declare any length value, or a percentage value. You can set the margin for an entire page by applying margin styles to the `<body>` element, or you can set margins element-by-element.

Let's say you want to create a four centimeter left margin for an entire page, rather than for a specific element or class of elements. You simply apply the *margin-left* property with a value of *4cm* to the `<body>` element, and voila: the margin of all elements on the page start four centimeters from the left of the browser window. Remember, these elements didn't inherit the left margin from the `<body>` element, per se – they only appear to. The position of a box is relative to its parent box. In this case, all elements contained by the `<body>` element have a margin of 0. If you then give, say, a paragraph a left margin of one centimeter, it will display one inch from the left of the existing page margin, a total of five centimeters.

### *Length Units*

As with other CSS properties, you can use any length unit to specify an element's margin. These units include *mm* (millimeters), *cm* (centimeters), *in* (inches), *pt* (points), *pc* (picas), *em* (ems), *ex* (ex-height), and *px* (pixels). When you specify a margin by *percentage* (%), the size of the margin is calculated according to the margin of its parent element. If no margin is set for a parent element, the default parent element is `<body>` – the browser window.

### *Negative Margins*

If you want to force an element outside of its container's margin, you can use a negative margin value, as shown below.

```
p { margin-left: -4cm; }
```

Provided that a 4 centimeter page margin exists, this rule will, in effect, remove the four centimeter margin for all paragraphs. Be careful with negative margins, however. If there's no 4 centimeter page margin to offset this negative value, all paragraphs will start outside of the viewable area. Not necessarily a good idea.

### **The Margin Property**

The *margin* shorthand property enables you to set all four margins at once. You can set the top, bottom, left and right margin simultaneously by specifying one value for the margin property, as follows:

```
body { margin: 1in }
```

This rule will set all four margins to one inch. And guess what – it works in both Internet Explorer and Netscape Navigator. If you don't want all sides to have the same margin, you can set each value separately, as follows:

```
body { margin: 5% 10% 5% 8% }
```

When four margin values are specified in this way, they are set in the order **top – right – bottom – left**. In other words, the above rule will set the top margin to 5%, the right margin to 10%, the bottom margin to 5%, and the left margin to 8%. Pretty cool indeed, and it too works in both Internet Explorer and Netscape Navigator.

Another technique that works in both browsers is to use the single value for the shorthand margin property to set all four margins, then set one of the margins more explicitly, as follows:

```
body { margin: 1in;
      margin-right: 2in }
```

This way, if you only need one unique margin, you don't need to specify the same value for the remaining three margins. The specifically declared margin-right property overrides the right margin inherent in the shorthand property.

### *Hanging Indents*

While we're on the topic of margins, there's an easy way to create hanging indents when you've got a left margin. By specifying a negative value with the *text-indent* property, you can offset the first line of a text element from its margin.

```
p { margin-left: 30px;
   text-indent: -10px }
```

### *Collapsing Margins*

Now that we understand CSS margins, what happens if an element has, say, a bottom margin of one centimeter and an element directly below it has a top margin of one-half centimeter? Is the distance one and one-half centimeters? Sometimes. That's right – it depends on the browser. CSS margins are supposed to collapse, meaning that in this case, the browser will render the space between our two imaginary elements as one centimeter. Instead of adding the bottom margin of one element to the top margin of the next, the greater of the two margins is used, as shown below. This works in Internet Explorer, but Navigator may have a bit of trouble with it. Let's have a look.

**This heading has a 50 pixel bottom margin.**



This paragraph has a 10 pixel top margin. The total margin between the heading and this paragraph should be 50 pixels, as the 10 pixel margin is collapsed into the larger 50 pixel margin.

## 4.) The Border Model

You can add a border around an entire document, or apply a border to a specific element. This border can be a full four-sided border, or you can declare border styles for any side of an element's box. (An element's border is displayed between its margin and its padding.) Be careful about the use of borders with older browsers (4.x) as rendering differences abound. Later versions of both IE and Navigator handle borders more consistently. There are three essential properties that can be used to create borders:

property	value
border-width	(length unit), thin, medium, thick
border-style	none, dotted, dashed, solid, double, groove, ridge, inset, outset
border-color	color name, hexadecimal or rgb value

The dotted and dashed values aren't yet supported by either browser. Both browsers apply a solid style when dotted or dashed is specified. Avoid using border length names such as thick and thin, as different browsers don't necessarily agree on how thick "thick" or "thin" should be displayed. It's better to specify border lengths with absolute units, such as pixels, points, or millimeters.

### The Border Property

There's a far better way to apply borders, and it works in both browsers. With the *border* shorthand property, you can specify all three aspects of a border, as shown in the following example:

```
blockquote { border: 10px double green; }
```

This shorthand property covers the border-width, border-style, and border-color properties all at once. It doesn't matter which order they appear in the rule; the properties are implicit in the values specified.

### Applying Borders to Page Elements

If you want full borders – that is, four-sided borders – the border shorthand property is the best way to make them work in both browsers. However, if you only want a border on one side of an element, or two or three sides, then you'll have to deal with using separate properties for the two browsers.

#### Border Properties for Internet Explorer

Internet Explorer 4.x and later support the following properties: *border-left*, *border-right*, *border-top*, *border-bottom*, and of course the shorthand property, *border*. Of these, Netscape Navigator 4.x supports only the border property, which is fine if you're looking for a complete four-sided border.

#### Border Properties for Netscape Navigator 4.x

To achieve specific sides of a border in Navigator 4.x, you have to use the following properties: *border-left-width*, *border-right-width*, *border-top-width*, and *border-bottom-width*. If you want to add color and border type, you'll also have to add *border-color* and *border-style* to your rules. Internet Explorer ignores all of these properties except *border-style*. So, by specifying both families of border properties, you can make them work in both browsers. Keep in mind that browser versions beyond 4.x have better and more consistent support for borders but still display some rendering differences. Be sure to check your borders in different generations of the various browsers to ensure uniformity.

### Padding

An element's padding is the space between the element itself and its border. You can apply padding to any HTML element, although the 4.x browsers have some trouble with padding applied to tables. There are five properties associated with padding; *padding-left*, *padding-top*, *padding-right*, *padding-bottom*, and the shorthand property *padding*. Like *border*, the *padding* shorthand property applies padding to all sides of the element simultaneously. As a card-carrying member of the *box* properties, padding cannot be inherited.

---

## 5.) Floating Elements

The *float* property is to CSS what the *align* attribute is to the `<img>` tag in HTML. Both are used to move an element from its normal flow in the HTML and align it with another element. The float property forces an element to the left or right of its parent. There are three possible values to the float property; *left*, *right*, and *none*. For now, however, you're probably better off using that old *align* attribute, since the float property is bound to give varying results depending on the browser, version, and platform.

### **Drop-Caps: How It's Supposed to Work**

*Drop-caps* are a layout technique common in print publishing. You can create drop-caps using *float: left*, but it's not a recommended method. Normally, you wouldn't have to use float to create drop-caps. The CSS-1 specification has this great feature called the *first-letter* pseudo-class that creates drop-caps with far better results in 5.x browser versions but is not supported in earlier versions.

Other ways to create the same result are to either use an image of a letter, as most designers do today, or you can float a letter to the left and apply other styles to it, most commonly a larger font size, and perhaps a different color, or even background color. To do this, you can create a class style with *float: left* and apply it to the first letter of a paragraph or other text element by using the `<span>` tag. But you'll get inconsistent results.

**Y**ou can try and try, but creating drop-caps with CSS won't look good in all browsers and platforms. If you really need a drop-cap, you're better off using an image of a first letter, and wrapping text around it. Someday, that brilliant first-letter pseudo-class will be supported, making our lives easier. Until then, do it the old fashioned way.

### **The Clear Property**

left   right   both   none (default)
--------------------------------------

The *clear* property is used along with the float property to prevent text or other elements from wrapping on either side of a floated element. For example, if you have two elements (a heading and a paragraph) wrapped around an image that is floated left, but you only want the heading to wrap around the image, you can use clear to prevent the paragraph from wrapping to one side of the image.

There are four possible values to the clear property; *left*, *right*, *both*, and *none*. *None* is the default, or initial value, and allows an element to float on either side of a floated element. *Both* ensures that the element won't wrap on either side. *Left* and *right* are more self-explanatory.

## 6.) Styling Lists

Lists. One of the earliest formatting tools in HTML, and still the best way to present things like indexes, definition lists, tables of content, and that sort of thing. When it comes to styles, however, lists have a bit of trouble, especially in Navigator 4.x. Many styles won't work at all if applied to a list, while others render differently according to the browser. For the most part, list styles break down gracefully in those browsers that don't support them.

### ***List-Style-Type***

```
disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha |
upper-alpha | none
```

With the *list-style-type* property, you can specify the type of label, or bullet, you want your ordered or unordered list to have. The available values for this property are: *disc* (the default value), *circle*, *square*, *decimal*, *lower-roman*, *upper-roman*, *lower-alpha* and *upper-alpha*, and *none*. If you don't want your list items to have any label, *none* is your friend. You can style an unordered list with an ordered list type, or vice-versa, but it's still a better idea to use `<ol>` for an ordered list and `<ul>` for a bulleted list. Just because you can, doesn't mean you should. Again, support varies in different versions of the browsers, so be careful and preview your pages for consistency.

### ***The List-Style-Position Property***

```
inside | outside
```

This property has only two possible values, *inside* and *outside*. *Outside* is the default value. It's really very simple: each list item has its own box. Normally, a list item's label is rendered just outside of this box, to the left. If you want your label to exist inside of the list item's box, you can use the *inside* value. In a list item positioned inside, any text that runs to a new line will start at the same point that the bullet starts, which arguably doesn't look as nice. The *list-style-position* property is supported well in IE and newer versions of Navigator but is not supported in Navigator 4.x.

### ***The List-Style-Image Property***

With the *list-style-image* property, you can apply images as list item bullets. The syntax for *list-style-image* values is the same as other image-embedding properties in CSS; the property, followed by URL, followed immediately by parentheses containing the location or filename of the image. No quotation marks are necessary.

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

This property is supported well by Internet Explorer 4.x and later for Windows and Mac and Navigator 6.x and newer as well. The *list-style-image* is not supported by Navigator 4.x but it breaks down gracefully: instead of images, you'll get the default "disc" bullets, (unless you've applied another list style type) and that won't hurt too much.

### ***The List-Style Property***

*List-style* is another shorthand property, combining *list-style-type*, *list-style-position* and *list-style image*, all in one. Since Navigator 4.x only supports the *list-style-type* property, it simply ignores the other values in the rule (*list-style-image* and *list-style-position*). As with other shorthand properties, the values of the *list-style* shorthand property are separated by white space, as shown in the following example:

```
ol { list-style: url(image.gif) lower-roman outside }
```

## 7.) Positioning Elements

CSS positioning (CSS-P) is an extension of CSS-1. The W3C presented CSS-P in its own specification, titled *Positioning HTML Elements With Cascading Style Sheets*. CSS-P introduces a whole new way to construct web documents. Rather than mixing structure and presentation by setting pages into HTML tables, which is tedious and can cause slow page load time, you can precisely declare where you want certain elements to appear on the page, without having to clog up your HTML with table tags and other formatting hacks.

Their value aside, it's not a good idea to use positioning as the backbone of your page constructs until the 4.x browsers are phased out. As browsers continue to add support for CSS specifications, positioning will inevitably be the layout tool of choice.

### **The Position Property**

```
absolute | relative | static
```

The *position* property enables you to precisely control the vertical and horizontal placement of page elements, independent of other elements or relative to them. There are three possible values to the position property: *absolute*, *relative*, and *static*. *Static* is the default value, and the least likely to be used, since it does essentially nothing. (You can use it if you want to prevent an element from being positioned, or to make any child elements flow naturally.) To declare an element's vertical and horizontal position, two properties are used in conjunction with the position property, *top* and *left*.

### **The Top and Left Properties**

The coordinates of a positioned element are controlled by two properties: *top* and *left*. The *left* property specifies an element's horizontal position on the page, and the *top* property specifies its vertical position. In the following example, the element named footnote will be positioned one inch from the left of the browser window (assuming it has no other relatively or absolutely positioned parent elements) and half an inch from the top:

```
.footnote { position: absolute;
            left: 1in;
            top: .5in; }
```

The CSS-P specification states that any HTML element can be positioned, but most of the browsers in use today only partially support that specification. For example, Internet Explorer 4.x requires that you use the <div> tag to position some elements. As with many other CSS properties, how positioning is supposed to work and how it actually works is often a different story. Hopefully consistency for positioning support will improve but, for now, you can use CSS-P techniques provided you thoroughly test your pages in multiple browsers.

### **Absolute Positioning**

Absolutely positioned elements do not follow the natural flow of the HTML document; rather, they are independent of other elements. You can specify the precise coordinates at which an element is displayed on a page; if the positioned element has no absolutely or relatively positioned parent element, then the left and top properties are calculated according to the upper-left corner of the browser window. If an element is positioned absolutely, but it has an absolutely or relatively positioned parent element, then the left and top properties are calculated according to the upper-left corner of its most direct parent.

### **Reference Points**

With absolute positioning, the top and left properties refer to the upper-left corner of the browser window, starting at *top: 0* and *left: 0*. Browsers have their own default margins, usually about 10 pixels, more or less. Specifying top and left values bypasses the browser's default margins, so declaring *left: 5px* will position an element 5 pixels from the leftmost edge of the browser window, rather than add it to the default margin.

If you were to give the <body> element a left value of zero and a top value of zero, then all elements on the page will be positioned at the leftmost edge of the browser window, and the element at the top of the document will be positioned at the topmost edge of the browser window.

Because an absolutely positioned element is independent of other elements (removed from the normal flow of the HTML), it will not re-flow, or change position, if the browser is resized.

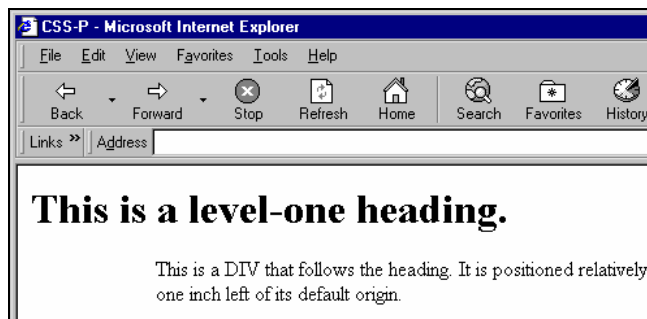
### **The Width Property**

You can set a positioned element's width to establish its horizontal length. This is useful for controlling how an element's text wraps. In the following example, the paragraph text is constrained by a width of 100 pixels. Once the text reaches the end of 100 pixels, it wraps to the next line.

```
p { position: absolute;
    top: 30px;
    left: 120px;
    width: 100px; }
```

### **Relative Positioning**

With relative positioning, you can control where elements are displayed relative to their parent elements, or more precisely, according to their default position in the HTML. Relatively positioned elements follow the natural flow of the HTML document; they are dependent upon a parent element for their position. In other words, if an element that's nested inside another element is positioned relatively, the coordinates are calculated from the point that the element would normally begin (see illustration below).



The same is true for an element that shares a common parent; the left and top properties of its relative position are calculated according to the element's default position in the normal flow of the document. Because of their dependence on other elements, relatively positioned elements re-flow, or are likely to change position, if the browser is resized.

### **The Overflow Property**

```
clip | scroll | none (default)
```

If you have a section of a web page that you want to occupy a precise amount of space, say, 80 pixels wide and 50 pixels high, you can use the overflow property to ensure that whatever content exists in this space, it won't exceed the boundaries you've set. There are three values to the overflow property: *clip*, *scroll*, and *none*.

For example, if you have a division with a width of 80 pixels and a height of 50 pixels, and the content exceeds these dimensions, the content will flow outside these set boundaries if no overflow property is present. If you use the overflow property with the value *clip*, the content will cut off horizontally when it reaches 80 pixels and vertically once 50 pixels is reached. If the overflow property is set to *scroll*, the browser will display a scrollable area in which all the content can be accessed. *None*, the default value, allows the content to flow outside of your set boundaries, which is how straight HTML handles the situation.

### *Browser Issues to Consider*

As if you haven't considered enough. Navigator 4.x supports the overflow property only for positioned elements. Internet Explorer 4.x and 5 aren't as picky. The other and more annoying reality is that Navigator 4.x supports clip but not scroll, and Internet Explorer 4.x and 5 for Windows support scroll but not clip. Meanwhile, Internet Explorer 4.5 for the Macintosh supports clip but not scroll. The later versions of the browsers have more stable support. The gist of all this is, once again, test your results in multiple browsers/versions.

### **The Display Property**

block   inline   list-item   none
-----------------------------------

The *display* property is actually a classification property, grouped with the family of list-style properties. With it, you can specify if an element is *block-level*, *inline*, a *list-item*, or not displayed at all (*none*). So, if you wanted to use `<div>`, or some other block-level element, but you want it to display as an inline element, you can use the display property to change its native status.

Setting an element's display to *block* will start the element on a new line, as block level elements do. As mentioned above, declaring an element as inline will prevent it from creating a line break, regardless of the element used. *List-item* will make an element display as a block-level element with a small left margin, as standard HTML list items do. The utility of this property is limited unless used together with scripts to create dynamic page elements.

Of its four possible values, *block*, *inline*, *list-item*, and *none*, the value *none* is most commonly used in DHTML techniques. When an element's display is set to *none*, it's no longer rendered by the browser and neither is the space it occupies. It's as if it doesn't exist in the document. With a script, you can make elements appear and disappear depending on the actions of the user. Both Navigator and Internet Explorer *display: none*.

It all sounds nice and powerful, but unfortunately the most we get is partial support from any current browser version. For example, Navigator 4.x supports *block* and *list-item*, but not *inline* for block-level elements. Internet Explorer 4.x and later supports *block* and *inline*, but not *list-item*. But that's okay. If you need a list item, you should probably use standard HTML anyway. As mentioned earlier, both browsers support *display: none*, which is the most effective use of the display property.

### **The Visibility Property**

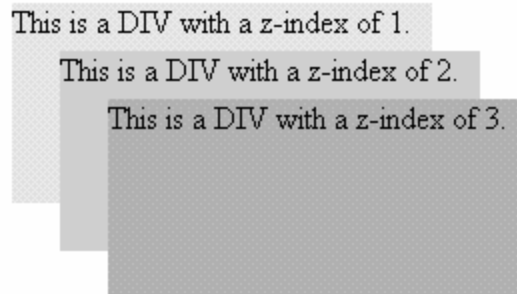
inherit (default)   visible   hidden
--------------------------------------

The *visibility* property works similarly to the display property, but whether an element is visible or not, the space it occupies remains a part of the document flow. In other words, the visibility property, unlike display, does not affect layout. You might be thinking, "Why would I want an element to be invisible?" On its own, you probably won't want to use the visibility property, but with JavaScript, you could create dynamic displays that react to the actions of the user. And whether you'd want the dynamically displayed element to affect the layout of the page is the difference between using display or visibility.

## 8.) Layering Elements (the Z-Index Property)

If one or more positioned elements on a page overlap each other, you can control the order in which the elements are stacked on top of each other. The default order for overlapping elements is bottom-to-top (back to front), meaning that if two elements occupy the same space, the last element to appear in the document flow will display above the previous element. With the *z-index* property, you can override the default and control the layering of overlapping elements. The *z-index* property works only for positioned elements; elements that overlap due to margin properties cannot be controlled with z-indices.

The *z-index* property is applied with integers; the higher the number, the higher the element appears in the stacking order – in front of other elements. For example, an element with a z-index of 3 will display on top of an element with a z-index of 2, 1, or 0, as shown in the illustration below. (The three <div>'s are staggered to clearly demonstrate the layering.)



### *Dynamic Layers*

The z-index property is particularly useful if applied dynamically. With JavaScript, you can dynamically control the z-index of overlapping elements, which can create some useful effects and economize page real estate. A script can alternate the z-indices of each layer automatically, or in response to the actions of the user. For example, a series of images or articles can be displayed in the same space on a given page as the user clicks on the 'next' button. The JavaScript would reset the z-indices in succession each time the button was clicked to display a different element (image or text) each time.

---

## 9.) Applying CSS Wisely

You've probably gathered that CSS is a powerful style language. You've probably also gathered that designing with CSS can be daunting due to inconsistent browser support. But don't let that scare you away from CSS altogether. CSS continues to grow in preponderance on the web. As older browsers become phased out, and newer browsers continue to add support for the CSS specifications, you'll find that designing for the web without CSS will limit your possibilities. Designing with cascading style sheets is becoming the developer's standard. The key to successful CSS authoring is to use it wisely.

### *Simplify, Simplify*

Until you are confident in the browser support for the CSS you write, you should keep your styles simple. The color and font properties are safe; use them freely while keeping in mind the issue of font names and sizes across platforms. Text properties are relatively safe, with some notable exceptions. If possible, avoid the over-use of margin, border, and especially positioning properties to design your pages until you are fairly sure your pages won't be showing up on 4.x level browsers. Likewise with box properties and positioning properties, unless you're developing for a specific browser or using a script to link pages to a browser-specific style sheet.

To start, keep your focus on creating pages that work well in all browsers – even if this means reverting to some of the old HTML tricks to obtain the desired results. Soon you will find that as your knowledge of CSS and browser support improves you can then enhance those pages and use CSS to design to your heart's content.

### *Make Complete Testing a Routine*

When you decide to apply a particular CSS property to a Web document, you should first be certain that the style doesn't in any way ruin your intended design when viewed in a browser with partial or no support for that property. You should use a variety of browsers, various versions of each browser, and every platform-specific version of each browser, to test all the CSS-styled pages you create. Otherwise, you run the risk of losing a percentage of your audience.

Realistically, you probably don't have access to multiple platforms and the various browsers and versions for those platforms. If that's the case, there are several sites on the Web that offer quick reference charts with all the CSS properties and their levels of support per browser, version, and platform. Some of these charts might not be up-to-date, but they'll undoubtedly help. Just use your favorite search engine to search for something like "CSS browser support" and you will find a number of sites to reference.

### **Styles That Break Down Gracefully**

There are several CSS properties that work well in one browser but are ignored in others. As long as other browsers ignore the property, chances are that using it won't compromise your intended presentation too much. Applying styles that break down gracefully is one method of getting the most out of CSS. The following table describes some of the browser-specific styles that break down gracefully in browsers that don't support them:

<b>Style</b>	<b>Description</b>
Dynamic link effects (the hover pseudo-class)	With Internet Explorer 4.x and later, you can control the color, font, size, and underlining (among other styles) of hyperlink text when the user's mouse passes over a link. Navigator 4.x and other browsers ignore the style.
Colored form input controls	With Internet Explorer 4.x and later, you can add color to form input controls. Navigator 4.x and other browsers ignore most form styles. Navigator does support the font-size, font-family, and font-style property with forms.
Colored horizontal rules and table borders	Internet Explorer 4.x and later supports colors for horizontal rules and table elements. Navigator 4.x and some other browsers ignore these styles.
Cursor styles	Internet Explorer 4.x and later supports a variety of cursor styles to customize the user interface. In other browsers, the styles are ignored.

### *The Hover Pseudo-Class*

One example of a style that breaks down gracefully is the hover pseudo-class. Just as pseudo-classes like *link*, *active*, and *visited* add style to customize the appearance of hyperlinks, you can use the hover pseudo-class to create a MouseOver effect. When a user positions a mouse pointer over a link the text changes to call extra attention to it and to help verify that it is the selected link. This allows you to simulate the effect of a rollover image in a navigation bar without having to use images – just text.

You can apply several different styles to the hover pseudo-class, including changing its font, font-size, color, and background. The syntax is the same as the other pseudo-classes; the selector must begin with the <a>(anchor) element, followed by a colon ( : ), and then *hover*, followed by the style declaration, as shown in the following example:

```
a:hover { background-color: red;
          font-weight: bold; }
```







This will change links from their default style to the new style when a user positions the mouse pointer over a link. This example will apply the same style to all links; however, you can create a variety of different pseudo-class styles by applying classes to the pseudo-class, as follows:

```
a.search:hover { background-color: gray;
                color: red; }
```

Browsers that do not support the hover pseudo-class ignore the style completely so the style doesn't affect an element's placement, making it safe to use.

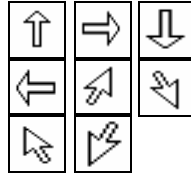
### Cursor Styles

Cursor styles are neat and all, but don't get carried away. Cursor icons like the hand and hourglass are well-established user interface conventions that give the user important feedback. Messing with those conventions is not exactly a service to your audience. You wouldn't want a line of text to have a hand cursor style attached to it if the text is not a hyperlink. You'll end up with users who click all over it, wondering what the problem is. Likewise, it would be foolish to use the hourglass cursor style unless you really, really need to communicate that there's something the user is, or should be, waiting for. That said, there are a bunch of different cursor styles to choose from:

Property value	Cursor style	Description
cursor: auto	(determined by the browser)	Gives the browser the control over what cursor is displayed. Most browsers have the same default cursor styles.
cursor: default	(default cursor is used)	Default will change nothing; mousing over text will show the standard I-beam cursor, non-text elements will show the standard pointer, and links will show the hand cursor.
cursor: crosshair		Usually a precise aiming tool. Its value on the Web is in the eyes of the beholder, I suppose.
cursor: move		Can be useful with dynamic pages; if items on a page can be dragged/dropped, moved, and so on.
cursor: hand		Probably not a good idea to use this one frivolously, as it's a firmly established convention indicating a hyperlink or some other object you can click on.
cursor: text		Another standard for text elements. Most browsers have this as a default cursor for text elements already.
cursor: wait		Another one to use sparingly, if at all. The hourglass cursor is a source of aggravation for some people, as it commonly asks for patience while something loads. Using it for no good reason might send the wrong message.
cursor: help		This can be useful if you want to direct a user to a Help page, or item that offers some sort of assistance.

-- table continued on next page --

cursor: x-resize



(Where X equals one of eight directional variables, N (north), E (east), S (south), W (west), and the compass points in between each, such as NE, or SW.) Can be useful if you want to direct a user to a particular item or section of a page.

Cursor styles aren't part of the official CSS-1 specification; they're an extension that only Internet Explorer 4.x and later supports. Using them (wisely) won't hurt in browsers that don't support the cursor styles; they'll simply be ignored.

## CSS Filters

Filters are only supported by Internet Explorer 4.x and later, and whether or not these styles break down gracefully depends on how they're applied. You can apply filters to text elements or images (several filters are designed specifically for images). Generally, filters are used to create dynamic effects, which requires scripting to control their behavior. Since scripting is beyond the scope of this course, we'll only discuss the fundamentals; the available properties, values, and a few effects.

### *Filters: Why They're Cool, Yet Unpopular*

Filters offer the ability to control display effects like no other CSS property. They add quite a lot to the web designer's palette, especially for dynamic HTML effects. Scripting aside, CSS filters alone can do some valuable stuff. First of all, the Web is bloated with GIF images created in graphics programs, for the express purpose of presenting text in ways that HTML can't. Even with CSS, some textual designs are difficult, if not impossible, to achieve.

Besides the obvious bandwidth issue, the problem with using GIFs to present text is that the image doesn't exist as text on the page, so it can't be found by search engines, and won't display at all if a user turns images off in the browser preferences.

Filters give added design capability to text and images, offering the better of both worlds: you can create eye-catching text effects like drop-shadows, waves and glow, without having to resort to a GIF or bulky animated GIF. There are several filters available, and most of them are meant to be used with scripts to control transitions, blending, and other animation effects. The following table contains some of the filters available for use with Internet Explorer. Some filters not addressed here are convoluted and beyond the scope of this lesson.

<b>Filter</b>	<b>Properties and Values</b>	<b>Description</b>
shadow	color (colorname or hex value) direction (0-360)	The color property will accept a color name or hexadecimal value. The direction property specifies the origin of the shadow in 360 degrees clockwise.
dropshadow	color (colorname or hex value) offx (value in pixels) offy (value in pixels) positive (0 or 1)	The color property will accept a color name or hexadecimal value. The offx property specifies the number of pixels horizontally, and offy the number of pixels vertically. The positive property is either a value of 0, which means transparent pixels get the shadow, or 1, which means non-transparent pixels get the shadow.
Glow	color (colorname or hex value) strength (number)	The color property will accept a color name or hexadecimal value, and strength determines the strength of the glow.
<b>-- table continued on next page --</b>		
blur	add (1 or 0)	Simulates an object in motion by giving it a

direction (0-360) strength (number)	directional blur. The add property specifies if the element itself is blurred (1=yes) or not (0=no). The direction property specifies the origin of the blur in 360 degrees clockwise, and the strength property specifies the distance the blur travels from the element.
flipV, flipH	flipV and flipH don't require anything but themselves. They flip an element vertically (flipV) or horizontally (flipH).
Gray	Gray doesn't require any special properties or values. Using filter: gray will convert all colors into grayscale.
Invert	Another one that has no special properties or values. Invert is used with images; it converts a photograph to its negative.

### *The Ubiquitous Caveat*

Filters are not part of the CSS-1 specification. Microsoft proposed the filter properties, which may or may not be adopted into a future CSS specification. These proprietary CSS properties work only in the Windows versions of Internet Explorer 4.x and later, so you can draw your own conclusions about their benefits in any real-world application – at least for the time being.

Some designs on the web can be a bit over-the-top. They might look nice, but sometimes functionality suffers and information is hard to find. Likewise, just because you can doesn't mean you should use filters. It's impossible to control every user's experience with so many platforms, browsers, color settings, and resolutions at play. "All things in moderation" is a good rule to design by. Filters might grow in support eventually, but it's not time quite yet to abandon GIF text.

### *Getting Started: The Filter Property and Syntax*

For whatever reason, you have to apply filters inline using the `<span>` element. You also have to specify either a width or height, or both, for the element; otherwise, you'll get nothing. Also, your height and width properties must be in the inline rule as well. You can even specify widths and heights of 0; it just wants the company. It's odd, but there you have it. The *filter* property is followed by the type of filter you want to apply (*blur*, *shadow*, and so on) followed by parentheses that contain the filter variables, as shown below:

```
<p><span style="width: 100px; filter:glow(color=silver, strength=4)">
To Glow Or Not to Glow?
</span></p>
```

---

## 10.) Dynamically Linked Style Shets

One of the most important principles of Web design is to create pages that display the same in all browsers and on all platforms. Designing with CSS, like designing with HTML, has its challenges in keeping with this goal, but it's not impossible – if you use CSS wisely. Keeping your styles simple and testing them to make sure your pages work in a variety of browsers are critical principles to adhere to when authoring with CSS. But it's not the only way to handle the inconsistencies in browsers. Another method is to use JavaScript to identify a user's browser, and then link the page dynamically to a style sheet created specifically for that particular browser, its version, and platform. This means you have to create several style sheets, each one customized for a specific browser environment.

There are many sites on the web that offer free JavaScripts that you can simply copy and paste into your documents, then plug in your specific information in the appropriate places. Many of these sites also provide instruction on how to customize the scripts they make available.

Scripts that are meant to run when a page is loaded are inserted into the <head> section of the document. When the page loads, the browser runs the script, which identifies the browser and links the page to an appropriate style sheet. It's up to you how many different browsers, versions, and platforms you're willing to support with this technique.

## 11.) Contextual Selectors

With contextual selectors, you can give styles to elements that exist in a certain context. Contextual selectors are simply two or more selectors separated by white space. The left-most selector is the highest level parent element. The last selector is the element the styles apply to, as shown in the following example:

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

This rule says that all bold (strong) text contained by a <p> element will be rendered with a yellow background. Bold text that is not a direct child of a <p> element will remain unaffected.

### *The Advantage of Contextual Styles*

Creating contextual styles has distinct advantages. Larger documents that require extensive styling with classes also require numerous *class* attributes scattered throughout the HTML. If your documents contain commonly used structures, such as sidebars, navigation footers, or running heads, and these structures often contain elements of their own, you can avoid having to create additional classes for these nested elements by applying their style in the context of their parent structure. In the following example, a section of text is bolded within a division named footer:

```
<!-- footer -->
<div class="footer">
These pages are <strong>Copyright 1999</strong> by Noone N. Particular. If
you wish to use any of this material, ask Mr. Particular himself.
</div>
```

Rather than having to spend the <strong> as a selector (it might be needed in other contexts), or creating a class of *.strong* (which would require a *class* attribute in the HTML), you can create a contextual style for only those <strong> elements that are nested within a footer. The rule might look like this:

```
.footer strong { color: darkred }
```

There's no limit to how many parent elements are included in a contextual style, and you can mix selector types. For example, you could create a rule that looks like this:

```
div.content div blockquote i { text-decoration: underline }
```

### *Contextual Selectors and Specificity*

When it comes to calculating specificity for purposes of the cascade, contextual selectors are counted normally. So, for every element selector, count 1. For every class, count 10. For every ID, count 100. For example, the following rule has a specificity of 113:

```
.home #footer div div strong { text-decoration: underline }
```

## 12.) Validating and the Future of CSS

If you're having trouble getting your style sheets to work as you envision, chances are it's not your fault; as you've seen, browsers have a long way to go toward full, consistent implementation of the CSS-1 specification. Another possibility is that your rules aren't constructed properly; perhaps you have a missing semicolon that interrupts a string of properties, or you left out a closing brace. If your style sheets are long and you'd rather not parse the entire thing looking for a problem, you can use a *CSS validator*.

The World Wide Web Consortium has a nice validation service that you can either download or use directly off their site, free of charge. It's a fast, easy way to ensure that you're on the right track in your CSS development. There are also a number of other websites which offer CSS validating tools. Use your favorite search engine to search for something like "CSS validator" or "CSS validating tools."

### ***The Future of CSS***

There is little doubt that CSS will continue to grow in use on the Web. It's the easiest, most sensible way to design, and browsers continue to add support for the specification. Older browsers continue to be phased out, opening the door to a better web. CSS-2 broadens the designer's palette even further, adding a variety of new selector types, properties, and design capabilities. It introduces style sheets for specific media, including printers and palm devices, as well as speech and braille instruments. CSS-2 also solidifies positioning, and adds downloadable fonts, new table properties, and several other valuable features that will continue to add power, compatibility, and flexibility to the web.

### ***A Final Note on CSS***

If you've ever modified a web site made with straight HTML, chances are you've had to search the code jungle for those awful <font> tags and their evil formatting attributes, a task that adds extra weight to tedium. If you don't already, start using CSS, at least for the fundamental design stuff, like colors and fonts. You'll make your life a whole lot easier. There are essentially two schools of thought on developing with CSS; you can write to the lowest common denominator, and use the old HTML hacks and cumbersome tables to set up your pages so that users with 3.x and (gulp) even older browsers can view your site, or you can use the 4.x browsers as your lowest common denominator, which arguably encourages users to upgrade and hence keeps real-world web development capabilities up to speed with the available standards. As we speak, even 4.x browsers have all but fallen to the wayside so we are well on our way to having solid, fully compatible support for CSS. So avoid the hacks, separate your structure from your presentation, and remember – without style, it's nothing but sheet.

---