ISTA 230

CSS 3
Selectors and Properties

CSS

CSS 3

The CSS 2.1 specification is nearly 500 pages. This makes it quite a feat to rework for today's web. Rather than trying to rewrite the entire specification, the W3C took a different approach with CSS 3. Instead of releasing a single gigantic specification, the CSS 3 specification is broken into modules. This allowed the W3C to move forward with some parts of the specification more quickly than others and also made an unwieldy task much more manageable.

There are three different "maturity levels" for these modules:

Working Draft (WD)
: A working draft is a document that has been published for the web development community outlining a proposed feature (or set of features) that could be implemented in browsers. This is basically a call for the community to review an idea. Someone says, *"What if browsers could do the following...?"* The community then discusses the idea (often, at length). When the proposal has been reviewed and met a few specific technical requirements, a "Last Call (LC)" announcement is made, basically saying that the proposal is going to mature to the next level in three or more weeks, unless significant discussions take place before then.

Candidate Recommendation (CR)
: Once a module has reached this maturity level, browser makers will begin implementing the module (if they haven't already done so) and will provide feedback to the working group about whether the specification is clear, complete, and relevant. During this phase, the W3C CSS Advisory Committee will begin to review the specification as well. There are three possible outcomes from this phase:

    Return to Working Draft
    : The specification needs more work. This isn't all that common, as modules with CR status are expected to be accepted as Recommendations. That said, sometimes things become more clear after implementation is attempted.

    Abandoned
    : While it doesn't happen often, sometimes ideas that looked good on paper have little traction in reality. When this occurs, the W3C will abandon a module,

    Proposed Recommendation (PR)
    : The module has been implemented successfully, proven to be clear, complete, and relevant in real-world applications, and approved by the director of the W3C. At this point, the W3C Advisory Group has the responsibility of reviewing the specification, though no substantive changes can be made.

Recommendation
: The final resting place for a CSS 3 module, this maturity level indicates that the module has been thoroughly reviewed and received endorsement of the W3C community and director. The W3C has basically signed off on large-scale implementation of this feature. If a module has reached this point, you can safely assume that most modern browsers either support it or are working to support it in the near future.

As with HTML 5, you should use CSS 3 properties keeping in mind our best practicies regarding progressive enhancement and graceful degradation. Users viewing your page with an older browser shouldn't be faced with a non-functional website (even though it may not be as nice as the view from a modern browser)!

CSS

CSS 3 Selectors

CSS 2.1 provided us with the following selectors to use in our style sheets:

- Class (.className)
- ID (#idName)
- Descendant (E F)
- Child (E > F)
- Siblings (E + F)
- Attribute (E[attr])

While this allowed us to do most of what we needed, there were some shortcomings (as you probably experienced during Assignment 2 in this course). The CSS 3 Selectors Level 3 module introduced a number of new CSS selectors, each of which are supported in the following browsers:

Browser Support:

Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: **>= IE 9 only**

CSS

CSS 3 Attribute Selectors

The selector below allows us to target all elements whose attribute value **begins exactly** with the value specified. For example, if we wanted to target every element whose class attribute started with the word 'user', we could do the following:

CSS

```
a[class^="user"] { background-color: #ff0; }
```

HTML

```
<a href="#" class="username">Name</a>
<a href="#" class="user_phone">Phone</a>
<a href="#" class="user-address">Address</a>
```

Browser

Name Phone Address

CSS

CSS 3 Attribute Selectors

The selector below allows us to target all elements whose attribute value **ends exactly** with the value specified. For example, if we wanted to target every element whose class attribute ended with the word 'user', we could do the following:

CSS

```
a[class$="user"] { background-color: #ff0; }
```

HTML

```
<a href="#" class="first_user">User 1</a>
<a href="#" class="secondUser">User 2</a>
<a href="#" class="third-user">User 3</a>
```

Browser

User 1 User 2 User 3

Note that User 2 was not highlighted, as the CSS 3 attribute selectors are case-sensitive.

CSS

CSS 3 Attribute Selectors

The selector below allows us to target all elements whose attribute value **contains** the substring specified. For example, if we wanted to target every element whose class attribute had the substring 'low' in it, we could do the following:

CSS

```
a[class*="low"] { background-color: #ff0; }
```

HTML

```
<div class="flower">Daisy</div>
<div class="low_res">dsy</div>
<div class="yellow">#ffffaa</div>
<div class="glowing">#ffff00</div>
```

Browser

Daisy
dsy
#ffffaa
#ffff00

CSS

CSS 3 Pseudo Class Selectors

Often times, we want to select a specific element in a sequence. The pseudo class below allows us to target an element which is the n^{th} child of its parent. For example, if we wanted to target the third child of a element, we could do the following:

CSS

```
a:nth-child(3) { background-color: #ff0; }
```

HTML

```
<div>
   <a href="#">Link 1</a>
   <a href="#">Link 2</a>
   <a href="#">Link 3</a>
   <a href="#">Link 4</a>
</div>
```

Browser

Link1 Link2 Link3 Link4

CSS

CSS 3 Pseudo Class Selectors

In addition to targeting a specific element using a numeric value, we can also target elements whose index is 'odd' or 'even'. In our previous example, we could target every other link by using the 'even' keyword:

CSS

```
a:nth-child(even) { background-color: #ff0; }
```

HTML

```
<div>
   <a href="#">Link 1</a>
   <a href="#">Link 2</a>
   <a href="#">Link 3</a>
   <a href="#">Link 4</a>
</div>
```

Browser

Link1 Link2 Link3 Link4

CSS

CSS 3 Pseudo Class Selectors

The n^{th} child selector is even powerful enough to handle complex equations:

CSS

```
a:nth-child(an+b) { background-color: #ff0; }
/* a - represents a cycle        */
/* n - a counter starting at 0   */
/* b - an optional offset value  */
```

Using the above syntax, we could duplicate the 'odd' keyword functionality with the following:

CSS

```
a:nth-child(2n+1) { background-color: #ff0; }
```

We could also target every fourth element:

CSS

```
a:nth-child(4n) { background-color: #ff0; }
```

We can even use this notation to target just the first three elements in a sequence:

CSS

```
a:nth-child(-n+3) { background-color: #ff0; }
```

This is probably the most useful of the new CSS 3 selectors. It also has a counterpart that works exactly the same **except that it starts counting backwards from the last child element.** For example, if we wanted to highlight just the last child element, we could use the following:

CSS

```
a:nth-last-child(1) { background-color: #ff0; }
```

CSS

CSS 3 Pseudo Class Selectors

Similarly, CSS 3 provides us with a pseudo-selector to target a specific type of element in a sequence. For example, if we wanted to target the second <a> element in a , we could use the following:

CSS

```
a:nth-of-type(2) { background-color: #ff0; }
```

HTML

```
<div>
   <a href="#">Link 1</a>
   <div>Content</div>
   <a href="#">Link 2</a>
</div>
```

Browser

Link1
Content
Link2

Like the n^{th} child selector, "nth-of-type" has a counterpart in "nth-last-of-type" which starts counting backwards from the last child element.

CSS

```
a:nth-last-of-type(2) { background-color: #ff0; }
```

HTML

```
<div>
   <a href="#">Link 1</a>
   <div>Content</div>
   <a href="#">Link 2</a>
</div>
```

Browser

Link1
Content
Link2

CSS

CSS 3 Pseudo Class Selectors

There are a few other selectors that introduced by CSS 3 that, while not the most commonly used, are still good to know about.

The :only-child pseudo class allows us to target an element that is the only child of its parent.

E:only-child - Targets an element E which is the only child of its parent

CSS

```
a:only-child { background-color: #ff0; }
```

Similarly, the :only-of type pseudo class allows us to target an element that is the only child of its parent of a specific type.

CSS

```
p:only-of-type { background-color: #ff0; }
```

CSS 3 even lets us target elements that don't have any children (including text).

CSS

```
div:empty { background-color: #ff0; }
```

CSS

CSS 3 Pseudo Class Selectors

CSS 3 also provided us with a negation pseudo class, allowing us to target elements that **don't** match a given selector. For example, if we wanted to style all elements that did **NOT** have an "id" attribute of "header", then we could use the following:

CSS

```
div:not(#header) { background-color: #ff0; }
```

It is worth noting that the negation pseudo class can have unintended side-effects due to it's far-reaching nature. For example, consider the following HTML:

HTML

```
<body>
   <p>Some text.</p>
   <p class='default'>Some more text.</p>
   <span>Even more text.</span>
   <div>
      <p class='default'>One last bit of text.</p>
   <div>
</body>
```

If we want to style every paragraph that didn't have the class 'default' with blue text, we could do so easily using a negation pseudo class. Additionally, we decide that we want to style all *non-paragraphical* elements with red text, again using a negation pseudo class for our selector:

CSS

```
p:not(.default) { color: blue; }
body :not(p) { color: red; }
```

To recap our goal, we want all paragraphs with the class 'default' to be displayed with black text (i.e., the default text color). All other paragraphs should have blue text. Any other elements (i.e., elements that aren't a paragraph element) should have red text. And the result...

Browser

Some text.

Some more text.

Even more text.

One last bit of text.

Alas, we have a red paragraph. The issue here is caused by inheritance and specificity. Pseudo-classes (like :not) have the same specificity value as a class name. Thus, our first rule as a specificity of 010 while our second has a specificity value of 011. Since text color is an inherited property, the browser will see both 'blue' and 'red' as color values, and will use specificity to determine which one to display.

CSS

CSS 3 - General sibling selector

With CSS 2.1, we had our adjacent sibling selector which would target an element **immediately preceeded** by another element. While useful, this selector was not without its limitations. One might end up using sets of painful CSS selectors similar to the following:

CSS

```
p+p,
p+p+p,
p+p+p+p
{
   color:#090;
}
```

Fortunately, CSS 3 introduced the general sibling selector, which targets an element preceeded by another element (immediate or not). For example, imagine that we wanted to higher any paragraphs that were preceeded by one or more paragraph siblings.

HTML

```
<article>
   <header>Header</header>
   <p>Paragraph 1</p>
   <p>Paragraph 2</p>
   <aside>Aside</aside>
   <p>Paragraph 3</p>
</article>
```

Without modifying the HTML, we'd have to use the following CSS selectors in CSS 2.1 to accomplish our task.

CSS - CSS 2.1

```
p + p,
p + aside + p
{
   background-color: #ff0;
}
```

Even the above isn't complete as it is pretty specific to our exact HTML. If we were to change our <aside> to a or add another element between paragraphs, we'd have to modify our CSS. Fortunately, CSS 3 introduced the general sibling selector, which targets an element preceeded by another element (immediate or not). We can now accomplish our task easily:

CSS - CSS 3

```
p ~ p { background-color: #ff0; }
```

CSS

CSS 3 Pseudo Classes

CSS 3 also introduced some pseudo classes that related to user actions. The :selection pseudo class allows us to style the way text should look when users "select" it (i.e., it has been highlighted using the mouse or keyboard). By default, browsers style text selections in varying ways. In Google Chrome, users might see something like the following if they had highlighted the phrase "brought forth on this continent a new nation".

Browser

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.

To provide users with a more consistent user experience across browsers, we could use CSS 3 to style this highlighted text.

CSS

```
body:selection
{
   background-color:#333;
   color: #fff;
}
```

Browser - With CSS 3 :selection styling

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.

CSS

CSS 3 Pseudo Classes

The :target pseudo class allows us to style an element whose id attribute matches the fragment identifier in the URL. For example, imagine if I were to visit the page http://somewhere.com/latestNews#article2.

Below is a snippet of the HTML found at that URL:

HTML

```
<div id='article1'>This is an article</div>
<div id='article2'>This is another article</div>
```

If we had 100 articles on the page, it might be useful to highlight the specific article that the user was looking for. Using CSS 3, we can do this!

CSS

```
div:target
{
   background-color: #ffa;
}
```

By adding a slight yellow background to the article, we've added a simple visual indicator that will help guide the user's eye to the specific article in the URL. Were they to click on another link which took them to another article on the page, that article would then be highlighted.

CSS

CSS 3 Pseudo Class Selectors

Using CSS 3 selectors, we also have the ability to target radio buttons and checkboxes that have been 'checked'. As we discussed during our lecture on HTML forms, styling checkboxes and radio buttons is a risky venture. They are inherently difficult to style and many browsers limit what you can change about their appearance. That said, this selector can provide us with a means to style elements around the checked input field. Consider the following HTML and CSS snippets:

HTML

```
<input type="checkbox" id="agree" value="1" />
<label for="agree">I agree to the terms of service</label>
```

CSS

```
label { color: #a00; }
input:checked + label { color: #0a0; }
```

Using the `:checked` selector, we are able to modify the display of our webpage based on the state of our checkbox. When unchecked, the label associated with our checkbox will be displayed as red text. However, when the user checks the box, we change the text color of the label to green. This approach opens up a number of opportunities for creating more interactive and dynamic web pages.

CSS

CSS 2.1 - Width/Height Calculation

Consider the following CSS rule:

CSS

```
div#header
{
   width:100%;
   border:10px solid black;
}
```

Given what we know about the box model, we can calculate the true width of an element by adding up the left/right margins, the left/right padding, the left/right borders, and the width of the content area. In the case above, the true width of the element would be 100% **plus 20px!** This would result in an overflow issue and would most likely cause horizontal scrolling. While we could set our width to something like 98%, that won't work on every screen size. Anything less than 1000px wide will have the same horizontal scrolling issues we ran into originally. While we could set the width even lower (say 93%), we would end up having an extra 60+ pixels on larger screens due to the reduced width...

Without knowing the exact size of our window, there's not a great CSS 2.1 solution. Fortunately, CSS 3 provides us with a better option:

CSS

```
div#header
{
   width: calc(100% - 20px);
   border:10px solid black;
}
```

Using the calc() function, we can ask the browser to perform calculations to determine the value for our CSS properties. While this is a great solution to our problem, it does present a challenge in that older browsers don't recognize the value. For the sake of graceful degradation, we'll use the following:

CSS

```
div#header
{
   width: 96%;
   width: calc(100% - 20px);
   border:10px solid black;
}
```

We include an initial width value using CSS 2.1 units. We then provide a second width value, using the calc() function. This second declaration will be used by modern browsers to give us the precision we want. Older browsers, however, will see this as an invalid CSS property value and will ignore it, falling back to the CSS 2.1 value. While this isn't ideal, the page will still be functional in older browsers.

CSS

CSS 3 - Rounded Corners

You may remember from our lecture on Background Images that providing "rounded corners" to an element required a fair amount of wizardry in CSS 2.1. While the techniques discussed there are still valuable for for unique designs, CSS 3 provides us with a new property for easily adding rounded corners to elements:

CSS

```
div
{
   border:5px solid #597CBB;
   border-radius:15px;
}
```

Browser

Like the 'border' property, 'border-radius' can take a single value (used for all four corners), two values (used for the top-left/bottom-right and top-right/bottom-left corners respectively), or four values (used for the top-left, top-right, bottom-right, and bottom-left corners respectively).

CSS

CSS 3 - Rounded Corners

We can also specify border radius for a specific corner. If we provide a single value, it is essentially the same as our border-radius property, only being applied to a single corner. However, if we provide two values, we'll find that we can specify the horizontal and vertial radii separately. For example, if we wanted our top-left corner to have a horizontal radius of 30px and a vertical radius of 15px, then we could use the following:

CSS

```
        border-top-left-radius:30px 15px;
```

We can do this for all corners by providing two values for our 'border-radius' property, separated by a slash:

CSS

```
        border-radius:30px / 90px;
```

Browser

CSS

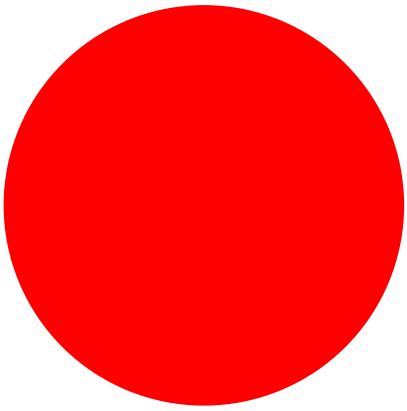
CSS 3 - Rounded Corners & Shapes

Using border-radius, we can now transform our elements into a number of shapes that previously wouldn't have been possible without relying on background images.

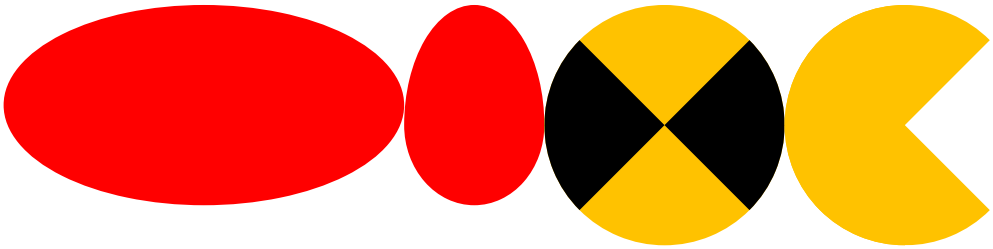
CSS

```
div
{
   width:200px;
   height:200px;
   border-radius:100px;
}
```

Browser



In addition to circles, we can use CSS 3 to create ovals, egg shapes, biohazard symbols, and even famouse video game characters.



In addition to circles, we can use CSS 3 to create ovals, egg shapes, biohazard symbols, and even famouse video game characters. An designer went sofar as to create an entire icon library using just CSS 3!

CSS

border-radius Property

CSS

```
div { border-radius: 30px; }
```

```
div { border-radius: 10px 5px; }
```

```
div { border-radius: 30px  / 15px; }
```

```
/*
 * Specifies the shape of the borders
 * for the given element.
 *
 *
 * Default Value: 0
 * Inherited: No
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 9 only
```

CSS

CSS 3 - Box Shadows

Another task that required using background images was creating the effect of a shadow underneath an element. CSS 3 offers us the "box-shadow" property which takes 5 values:

CSS

```
div
{
   box-shadow: horizontalOffset verticalOffset blurSize spreadSize color;
}
```

Using this property, we have a wide array of design opportunities available to us just using CSS. For example, if we wanted to create a simple 10px shadow that was offset by 10px horizontally and vertically, we could use the following:

CSS

```
div
{
   border: 2px solid #000;
   box-shadow: 10px 10px 0px 0px #888;
}
```

Browser

By setting the blur value to 0, we've taken away all of the shadow's "softness". We could also adjust the size of the shadow by changing the spreadSize. If our element had a border radius, the shadow would also be rendered with the same radius to maintain the realistic look and feel.

CSS

CSS 3 - Box Shadows

We could also flip our shadow so that it displays inside of our element by adding the keyword "inset" to the box-shadow property:

CSS

```
div
{
   border: 2px solid #000;
   box-shadow: 0px 0px 10px 10px #888 inset;
   border-radius: 50px;

}
```

However, both this technique and using a large border radius can result in overflow/readability issues as seen below:

Browser

To address these issues, we could simply add some additional padding to our element, pushing the text further inside the borders and box shadow.

It is also worth noting that, when printing, some browsers will print box shadows, even if background images are not printed. Additionally, some may ignore the blurSize value and even shadow colors when printing, creating large black boxes on your page instead. For the best user experience, it is recommended that you turn off box shadows when printing.

CSS

CSS 3 - Box Shadows

One of the other cool features introduced by CSS 3 is the idea of comma-separated values, allowing us to set multiple values for a single property. While this isn't an option for every CSS property (can you imagine a case where you'd want to set two text color values for a single element?), it is an option for box-shadow:

CSS

```
div
{
   box-shadow:  10px  10px 10px 0px #0f0,
               -10px -10px 10px 0px #f00;
}
```

Browser

While there is not a technical limit to how many box shadows you can apply to an element, there is a practical limit. Use your judgement and always be thinking about usability and readability.

CSS

box-shadow Property

CSS

```
div { box-shadow: 10px 10px 10px 0px #888; }
```

```
div { box-shadow:  10px  10px 10px 0px #888 inset,
                  -10px -10px 10px 0px #888;
}
```

```
/*
 * Creates a shadow effect on
 * a given element.
 *
 *
 * Default Value: none
 * Inherited: No
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 9 only
```

CSS

CSS 3 - Text Shadows

Like box shadows, CSS 3 also provides us with the option of setting a shadow underneath the text of an element (not the box itself).

CSS

```
div
{
   text-shadow: 2px 2px 2px #000;
}
```

This property is a double-edged sword! In some cases, adding a text shadow can create additional contrast (particularly when displaying text on top of a photograph or background image). In other cases, the legibility of the text can be hindered by adding a text-shadow. Use your best judgement.

It is also worth noting that text-shadows aren't printed by some browsers.

CSS

text-shadow Property

CSS

```
div { text-shadow: 2px 2px 2px #000; }
```

```
/*
 * Creates a shadow effect on
 * the text of a given element.
 *
 *
 * Default Value: none
 * Inherited: No
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 10 only
```

CSS

CSS 3 - Multiple Backgrounds

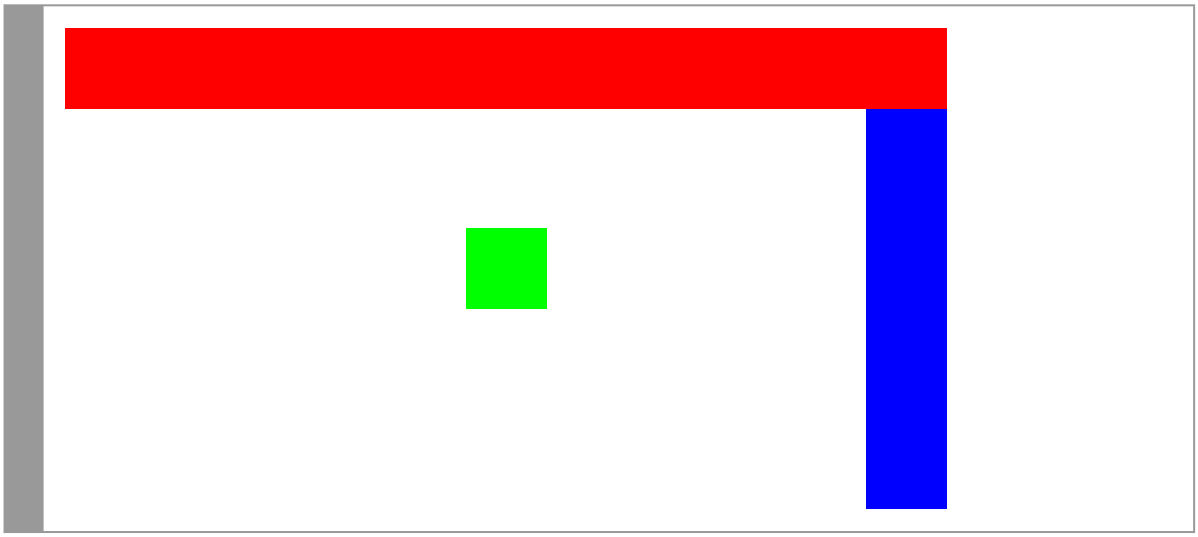
Like box-shadow and text-shadow, multiple backgrounds can now be applied using the background-image property. As seen below, we can apply multiple background images to a single element, affording us the ability to create complex backgrounds using multiple images and/or tiles.

In the example below, we use three images: a red background tile, a blue background tile, and a green background tile. Note that the green tile is placed first, then the blue tile placed on top of it, followed by the red tile on the very top. We control the position and repeat properties of these background tiles using the same comma-separated syntax that we used for the background-image property.

CSS

```
div
{
   background-image: url('redTile.png'),   /* Red Background   */
                     url('blueTile.png'),  /* Blue Background  */
                     url('greenTile.png'); /* Green Background */
   background-position: left top, right bottom, center center;
   background-repeat: repeat-x, repeat-y, no-repeat;
}
```

Browser



As we noted with the calc() function, you will want to provide graceful degradation by supplying a CSS 2.1 value for the background properties before adding the CSS 3 values.

CSS

```
div
{
   background-image: url('mainBackground.png'); // For older browsers.

   background-image: url('overlay.png'),
                     url('mainBackground.png');
}
```

CSS

(Multiple) background-image Property

CSS

```
div
{
   background-image: url('background1.png'),
                     url('background2.png');
}
```

```
/*
 * Specify one or more background
 * images for an element.
 *
 *
 * Default Value: none
 * Inherited: No
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 9 only
```

CSS

CSS 3 - Background Size

Another common issue when developing with CSS 2.1 was running into problems with the size of background images. Often, images were too small for a given container. In other cases, they were too large, resulting in the image being cropped by the edge of the element. For example, consider the following with a small background image of kittens.

CSS

```
div
{
   background-image: url('cats.jpg');
   background-repeat: no-repeat;

}
```

Browser

CSS

CSS 3 - Background Size

Because the image is smaller than the itself, we have empty space on the left and bottom of our element. CSS 3 provides us some control over this problem with the "background-size" property:

CSS

```
div
{
   background-image: url('cats.jpg');
   background-repeat: no-repeat;
   background-size:100% 100%;
}
```

Using the value "100% 100%", we can tell the browser that the background image should be stretched to fill 100% of the horizontal space and 100% of the vertical space respectively.

Browser

We can also use the keyword "contain" to specify that the background image should be resized so that its full height and width can both fit inside of the element. Alternatively, we could use the keyword "cover" to resize the image so that the entire element is covered by the background image. When using "cover", be aware that the image may be resized so that parts may not be visible to users.

CSS

background-size Property

CSS

```
div
{
   background-image: url('background1.png');
   background-size: 100% 100%;
}
```

```
div { background-size: cover;   }
```

```
/*
 * Specify the size of a background image as an explicit
 * length, percentage, or using one of the keywords 'cover' or 'contain'.
 *
 * Default Value: auto
 * Inherited: No
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 9 only
```

CSS

CSS 3 - Background Origin

By default, background images are placed based on what is known as the padding box. However, a designer can now control where background images are initially placed using the "background-origin" property. For example, if we wanted to place the background image underneath the border, we could do the following:

CSS

```
div
{
   padding: 50px;
   background-image: url('cats.jpg');
   background-repeat: no-repeat;
   background-origin:border-box;
}
```

Browser

Note that the background image is covered by the dashed border (something that wouldn't happen by default). Alternatively, designers can use the keyword 'content-box' to place the background image inside of the padding. For what it's worth, I don't know any designers who have used this in a website design for a paying client...

CSS

background-origin Property

CSS

```
div
{
   background-image: url('background1.png');
   background-origin: content-box;
}
```

```
/*
 * Specify what part of the box model background images
 * should be relative to. One of the following:
 * padding-box, border-box, or content-box.]
 *
 *
 * Default Value: padding-box
 * Inherited: No
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 9 only
```

CSS

CSS 3 - Opacity

Before CSS 3, creating semi-transparent elements was no easy task. It usually involved a combination of semi-transparent PNG images, CSS 2.1 background properties, and, often, proprietary filters to make the effect work in all browsers.

CSS 3 removed all of that overhead by introducing the 'opacity' property. By default, all elements have an opacity of 1. However, we can adjust this value down to any number between 0.0 and 1.0, with lower values resulting in greater transparency.

For example, if we had a background image that we wanted to show through our heading, we could set the background color of the heading to white and then reduce the opacity to 0.8 (i.e., 80%).

CSS

```
h1
{
   background-color: #fff;
   opacity: 0.8;
}
```

Browser

Note that while this does achieve our desired goal of allowing our background image to be seen through the heading, the approach leaves something to be desired with regards to readability. By setting the opacity to 80%, we've made every part of our element, including the text, semi-transparent. This is an important point to keep in mind when using the 'opacity' property.

CSS

opacity Property

CSS

```
div
{
   background-image: url('background.png');
   opacity: 0.5;
}
```

```
/*
 * Specify the opacity of an element. Values can range
 * from 0.0 (fully transparent) to 1.0 (fully visible)
 *
 * Default Value: 1.0
 * Inherited: No
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 9 only
```

CSS

CSS 3 - RGBA

As you may recall from our earlier lecture covering CSS color values, you can specify CSS colors using color keywords, hexidecimal values, or **by specifying RGB values using the following form**:

CSS

```
        background-color:  rgb( redVal, blueVal, greenVal);
```

CSS 3 introduced some additional options for specifying color values. The most highly anticipated of these options was the RGBa syntax:

CSS

```
        background-color: rgba( redVal, blueVal, greenVal, alphaVal);
```

Using RGBA, we can now specify a color using RGB values along with an *alpha* value (i.e., opacity). For example, let's consider the case we looked at for 'opacity'. When we adjusted the opacity for the heading, both the background color and the text itself all became semi-transparent, achieving our goal but hurting the overall readability of the text. Instead of using 'opacity', we can specify a semi-transparent background color using RGBA to achieve the same effect:

CSS

```
h1
{
   background-color: rgba(255, 255, 255, .5);
}
```

The text of our heading will have an opacity of 1.0 but the background image will be visible through the semi-transparent background color of the heading.

As with our other CSS 3 properties, it is important to think about older browsers when using RGBA. Older browsers will ignore any properties that use RGBA so it is important to provide a CSS 2.1 compliant property value first.

CSS

```
h1
{
   background-color: rgb(255, 255, 255);      /*** For older browsers ***/
   background-color: rgba(255, 255, 255, .5);
}
```

The CSS 3 Color Module also introduced the hue-saturation-lightness (HSL) format, as well as the related HSLA format. Because this is a bit of a digression from our current understanding of color values, we opt to not cover this in our course. However, it is an interesting topic worth exploring in your own time.

The CSS 3 Color Module also added (or extended the use of) some additional color keywords. For example, the keyword "transparent" can now be used as a value for any property that accepts a color value. The W3C also added additional color keywords to more closely match those used by the SVG specification. Lastly, the W3C added the keyword 'currentColor', which can be used to match the current 'color' value of an element.

CSS

rgba Color Values

CSS

```
div
{
   background-color: rgba(255, 255, 255, .5);
}
```

```
/*
 * Allows us to specify alpha-transparency for a given
 * color.
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 9 only
```

CSS

CSS 3 - Linear Gradients

You may have noticed that many of the design limitations of CSS 2.1 were solved by introducing additional background images. While necessary to achieve design goals, this added a fair amount of overhead for the browser, requiring multiple HTTP requests just to get all of the elements required for the page to look the way the designer intended.

As we've seen already, CSS 3 introduced a number of CSS properties to fix this limitation. At the time of its inception, gradients were a big target for CSS 3. A number of designers used linear or radial gradients in their designs, each of which required the use of a finely-tuned background image (or images) and all of which were somewhat static in nature (i.e., adjusting them was not a trivial task). CSS 3 solved this issue by introducing the linear gradient function:

CSS

```
body
{
   background-image: linear-gradient(#3FAECA, #fff);
}
```

Browser

CSS

CSS 3 - Linear Gradients

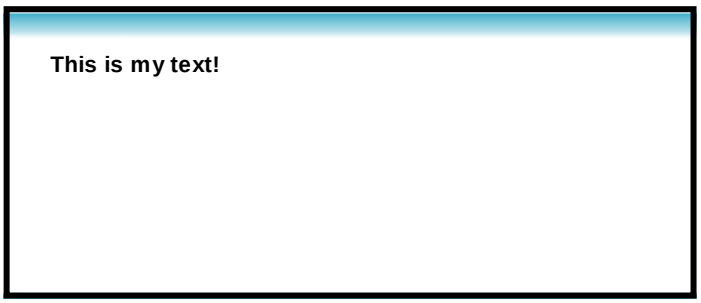
In our previous example, we simply supplied two colors for our gradient, resulting in an evenly distributed gradient from the top of our element to the bottom. We can actually control the distribution of the gradient colors by providing explicit *color stop* values.

Color stop values tell the browser where one color should stop and the next color should begin. It typically is specified as a percentage value, though you can also use explicit pixel or *em* values. The example below uses our same gradient but specifies that the last color should stop at 10% of the element's height, thus ending the gradient.

CSS

```
body
{
   background-image: linear-gradient(#3FAECA, #fff 10%);
}
```

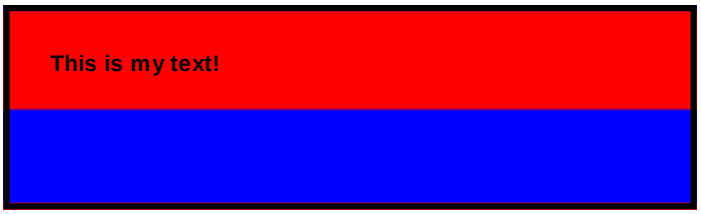
Browser



It is also worth noting that if two color stops have the same value, the browser will see this as an 'infintesimal transition', resulting in a sudden color change from one to another (i.e., like solid stripes).

```
        background-image: linear-gradient(#f00 50%, #00f 50%);
```

Browser



While we've only used two colors for our gradient here, we can use as many as our design calls for. That said, too many colors can quickly turn into an eyesore.

CSS

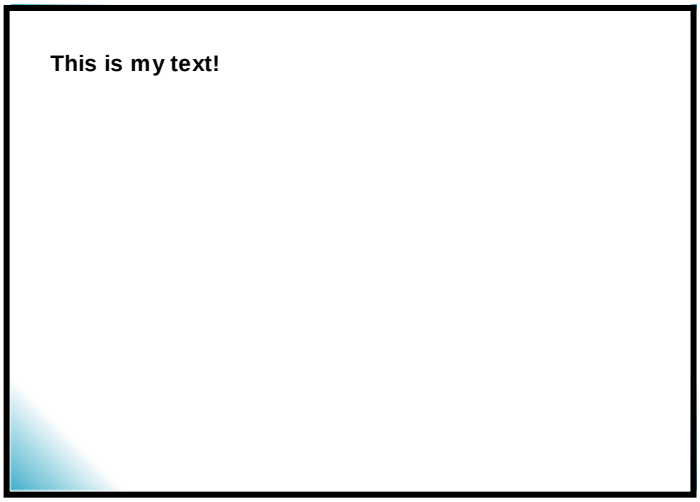
CSS 3 - Linear Gradients

We can also specify the gradient's line direction by providing an angle as the first argument to the function.

CSS

```
body
{
   background-image: linear-gradient(45deg, #3FAECA, #fff 10%);
}
```

Browser



When using angles, specify the angle as an integer followed by the letters 'deg' (without any spaces). For our purposes, '0deg' is upwards, '90deg' is to the right, '180deg' is downwards, and '270deg' is to the left. For these values, we can also use the following keyword pairs: 'to top', 'to right', 'to bottom', or 'to left'.

CSS

linear-gradient Values

CSS

```
div { background-image: linear-gradient(rgba(0, 0, 255, .25), #fff); }
```

```
div { background-image: linear-gradient(left, #3FAECA 10%, #fff 80%); }
```

```
div { background-image: linear-gradient(60deg, #f00, #0f0, #00f); }
```

```
/*
 * Allows us to specify a linear gradient in place
 * of a background image.
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 10 only
```

It is worth noting that we can combine some of our CSS 3 properties to create complex visual effects. For example, we can use a combination of multiple background images, background size, linear gradients, and RGBA values to add texture to a webpage by including a large background image (such as a close-up of cement, wooden panels, etc.) that show slightly through the background color of the page body.

CSS

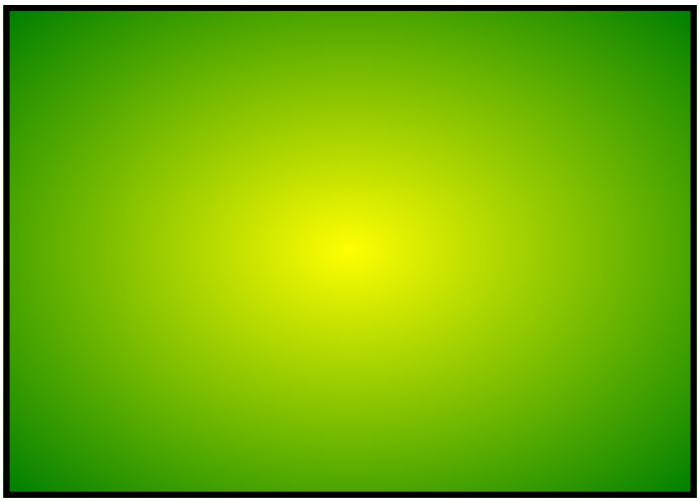
CSS 3 - Radial Gradients

Like linear gradients, we can also use CSS 3 to produce radial gradients which emerge from a single point and spread outwards in a elliptical (or circular shape).

CSS

```
div
{
   background-image: radial-gradient(#ff0, #008000);
}
```

Browser



There are a number of additional options for specifying exactly how a radial gradient should be displayed.

CSS

CSS 3 - Radial Gradients

We can specify color stops for radial gradients using the same syntax that we used for linear gradients.

CSS

```
div
{
   background-image: radial-gradient(#ff0, #008000 50%);
}
```

We can also specify whether the radial gradient is an ellipse (default) or a circle.

CSS

```
div
{
   background-image: radial-gradient(circle, #ff0, #008000 50%);
}
```

We can dictate where the origin of the radial gradient is using the same form we use for positioning background images (i.e., keywords like 'left' and 'center' or by using percentages or explicit values.

CSS

```
div
{
   background-image: radial-gradient(at left bottom, #ff0, #008000 50%);
}
```

Lastly, we can specify how the size of the gradient should be calculated: 'farthest-corner' to base it on the farthest corner from the origin of the gradient, 'closest-corner', 'farthest-side', or 'closest-side'. Below is an example of using the closest side to determine the size of the gradient.

CSS

```
div
{
   background-image: radial-gradient(closest-side at 20px 50px, #ff0, #008000 50%);
}
```

CSS

CSS 3 - Repeating Linear Gradients

Now imagine a scenario in which you've used CSS 3 to create a striped background for your page. You might use something like the following:

CSS

```
div
{
   background-image: linear-gradient(#f00 25%,
                                     #fff 25%,
                                     #fff 50%,
                                     #f00 50%,
                                     #f00 75%,
                                     #fff 75%);
}
```

Using a shared color stop for values, we've created the effect of having four stripes, each of which takes up roughly a quarter of the page.

Browser

If our client was to ask us for smaller stripes (say about 5% tall), we would be forced to add an additional 34 color stops, making our CSS less readable and less manageable with each stripe. Fortunately, CSS 3 offers us a better solution...

CSS

CSS 3 - Repeating Linear Gradients

Using the 'repeating-linear-gradient' function, we can specify our linear gradient just as we normally would. The only difference is that the browser will repeat the color stops infinitely in both directions.

CSS

```
div
{
   background:repeating-linear-gradient(#f00 0%, #f00 5%, #fff 5%, #fff 10%);
}
```

In the example above, we've created a "stripe" that takes up roughly 5% of the page height. Because we've used a repeating gradient, the browser will repeat the gradient when it reaches the last stop (10%), and will continue to do so until it has filled the entire background of the element.

Browser

The repeating gradient renders the same as if we were to use something like the following non-repeating gradient:

CSS

```
div
{
   background: linear-gradient(#f00 0%, #f00 5%, #fff 5%, #fff 10%, #f00 10%, #f00 15%, #fff 15%,
#fff 20%, #f00 20% ..., #fff 95%, #fff 100%);
}
```

CSS

repeating-linear-gradient Values

CSS

```
div { background-image: repeating-linear-gradient(#eee 0%, #fff 5%); }
```

```
/*
 * Allows us to specify a repeating linear gradient in place
 * of a background image.
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: >= IE 10 only
```

It is worth noting that there is also a *repeating-radial-gradient* property that works in a similar fashion.

CSS

CSS 3 - Fonts

As previously discussed, there are some significant limitations to typography when using CSS 2.1. In particular, we are limited to the fonts that are available on users' computers. In the case that our specified font wasn't available on the user's computer, we specified fall-back fonts that could be used instead. While this was a functional approach, it seriously hindered the ability to use all of the typographical tools that the modern age has to offer.

CSS 3 allows us to specify fonts that can be used within our web pages **that can be downloaded automatically from the server** if the user doesn't have a copy. The browser then uses the downloaded font to render the text on the web page. **Warning: Many fonts are not licensed for this kind of use on the web. Make sure you read the fine print!**

To specify a custom font in CSS 3, start by creating an 'at-rule' named "font-face". Within that rule, you'll specify two things: a font family name and a source. An example can be seen below:

CSS

```
@font-face
{
   font-family: 'MySpecialFont';                         /* Font family name */
   src: url('fontfile.eot');                             /* IE 9 Compat Mode */
   src: local('MySpecialFont'),                          /* Local */
        url('fontfile.eot') format('embedded-opentype'), /* IE6 - IE8 */
        url('fontfile.woff') format('woff'),             /* Modern Browsers */
        url('fontfile.ttf') format('truetype'),          /* Safari, Android, iOS */
        url('fontfile.svg#svgFontName') format('svg');   /* Legacy iOS */
}
```

You'll notice that we included a number of source files for our font in the example above. This is because different browsers support different types of font files. To ensure the most consistent cross-browser user experience, we provide a font file for each browser to use.

Once we've properly included our at-rule for the font face, we can now use it in our other CSS rules just as we would any other font.

CSS

```
div
{
   font-family: MySpecialFont, serif;
}
```

There are a number of online services that will provide you with fonts that can be used online. Some of them will even generate the necessary CSS files for you or allow you to link directly to a CSS file located on their server. Google Web Fonts is a good free option. Alternatively, you can also upload a font file that you are **legally** allowed to use on the web to FontSquirrel. This website will then provide you with all of the necessary font files for you to host your own font on your server.

CSS

@font-face Property

CSS

```
@font-face
{
   font-family: 'FontName';    /* Font name */
   src: url('fontfile.eot');   /* Font file location */
}
```

```
h1 { font-family: 'FontName', arial, serif; }
```

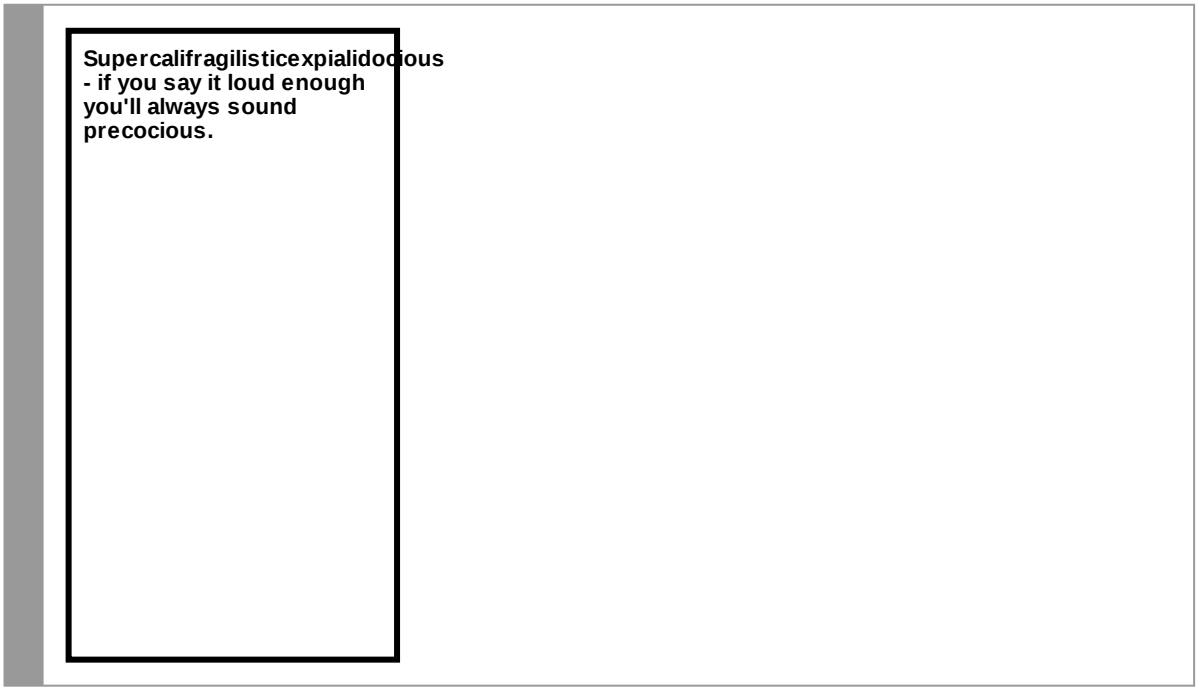
```
/*
 * Specifies a font family to download and
 * use within your website.
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: Yes
```

CSS

CSS 3 - Typography

When we have an element with a fixed width (or a very small screen), there may be instances when a word is simply too long to fit inside of the element. Take the following song lyrics as an example:

Browser



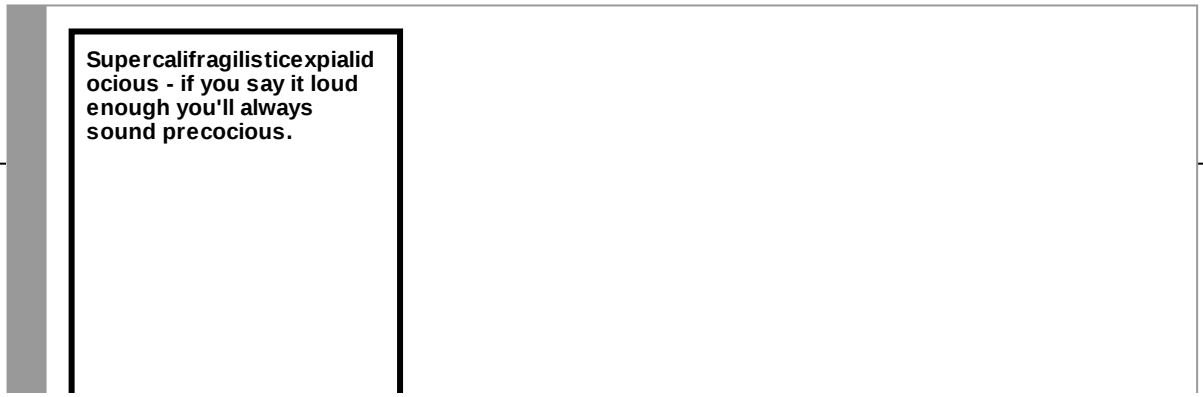
The word 'Supercalifragilisticexpialidocious' simply isn't going to fit inside of our small container. We either have to accept that the text will overflow the borders or set overflow to 'hidden' (which will cut off part of the word). Neither of these are good options. Using CSS 3, we can use the 'word-wrap' property to tell the browser that it should break the word into smaller pieces rather than allow it to overflow the borders:

CSS

```
div
{
   width: 150px;
   border: 3px solid #000;
   word-wrap: break-word;
}
```

Now the browser knows that it can break the word into multiple lines if necessary, solving our overflow issue (though reducing the readability of the word somewhat...)

Browser



CSS

word-wrap Property

CSS

```
div
{
   width: 150px;
   border: 3px solid #000;
   word-wrap: break-word;
}
```

want to hide any part of a URL, you also wouldn't want it messing up your layout. In that case, using the word-wrap property might be a good idea.

```
/*
 * Specifies whether the browser
 * can break words across multiple lines
 * if they don't fit within the element.
 *
 */

Browser Support:
Chrome: Yes
Firefox: Yes
Opera: Yes
Safari: Yes
MS Edge: Yes
IE: Yes
```

CSS

CSS 3 - Hyphens

While the word-wrap property will help prevent overflow issues, it doesn't address the bigger shortcoming of web typography in that there is no hyphenation.

However, CSS 3 is hoping to solve that problem soon as well! Consider the following paragraph:

Browser

This is a paragraph that includes an unusually elevated frequency of a few prolonged words, thus increasing the probability of a hyphen being needed.

Because we've used justified text alignment, the need for hyphens becomes much more obvious, as our text is littered with lots of excess white space. With CSS 3, we can use the new 'hyphens' property to specify that we want browsers who support the property to automatically add hyphens where needed, ideally reducing the excess white space and giving us nicely justified text.

CSS

```
div
{
   width: 200px;
   text-align: justify;
   hyphens: auto;
}
```

Browser

This is a paragraph that includes an unusually elevated frequency of a few prolonged words, thus increasing the probability of a hyphen being needed.

CSS

CSS 3 - Web Prefixes

When experimenting with new CSS properties, most browser makers will start by implementing their own version of the property, based on their understanding of the specification. To do this safely, they use a browser prefix on the property. Browser prefixes are as follows:

- -webkit- ***(Used by Chrome, Safari, and recent versions of Opera)***
- -moz- ***(Used for Firefox)***
- -ms- ***(Used for IE and Edge)***
- -o- ***(Used for older versions of Opera)***

Well, depending on what browser you're using, the above will either have some effect or will be completely ignored. Unfortunately, this is one of those newer experimental properties that browser makers are still playing with. In fact, at this point, only one browser (Firefox) supports this property fully. However, there's more to the story than just that...
In order to get these experimental properties to work in browsers, both now and in the future, you'll want to use browser prefixes as well as the un-prefixed property. For example, for 'hyphens', we'd do the following:

CSS

```
div
{
   -webkit-hyphens: auto;
      -moz-hyphens: auto;
       -ms-hyphens: auto;
        -o-hyphens: auto;
            hyphens: auto;
}
```

Note that the un-prefixed version is the last one specified. This will ensure that, when supported, browsers will use that to set the property for the element. For what we've covered in CSS 3 thus far, browser prefixes won't play much of a role. However, as we look at more advanced topics, they'll become more important.

CSS

hyphens Property

CSS

```
div { hyphens: auto; }
```

```
/*
 * Allows us to tell the browser
 * to hyphenate words when necessary.
 *
 * -- Browser prefix recommended --
 */

Browser Support:
Chrome: No
Firefox: Yes
Opera: No
Safari: >= Safari - with -webkit- prefix only
MS Edge: >= MS Edge - with -ms- prefix only
IE: >= IE - IE 10+, with -ms- prefix only
```