**Best Practices**

**1. Make it Readable**

The readability of your CSS is incredibly important, though most people overlook *why* it's important. Great readability of your CSS makes it much easier to maintain in the future, as you'll be able to find elements quicker. Also, you'll never know who might need to look at your code later on.

**Group 1: All on one line**

|  |  |
| --- | --- |
| 1 | .someDiv { background: red; padding: 2em; border: 1px solid black; } |

**Group 2: Each style gets its own line**

|  |  |
| --- | --- |
| 1  2  3  4  5 | .someDiv {    background: red;    padding: 2em;    border: 1px solid black;  } |

**Keep it Consistent**

Along the lines of keeping your code readable is making sure that the CSS is consistent. You should start to develop your own "sub-language" of CSS that allows you to quickly name things. There are certain classes that I create in nearly every theme, and I use the same name each time. For example, I use ".caption-right" to float images which contain a caption to the right.

Think about things like whether or not you'll use underscores or dashes in your ID's and class names, and in what cases you'll use them. When you start creating your own standards for CSS, you'll become much more proficient.

**Start with a Framework**

Some design purists scoff at the thought of using a [CSS framework](http://www.smashingmagazine.com/2007/09/21/css-frameworks-css-reset-design-from-scratch/) with each design, but I believe that if someone else has taken the time to maintain a tool that speeds up production, why reinvent the wheel? I know frameworks shouldn't be used in every instance, but most of the time they can help.

Many designers have their own framework that they have created over time, and that's a great idea too. It helps keep consistency within the projects.

**Use a Reset**

Most CSS frameworks have a reset built-in, but if you're not going to use one then *at least* consider using a reset. Resets essentially eliminate browser inconsistencies such as heights, font sizes, margins, headings, etc. The reset allows your layout look consistent in all browsers.

**Organize the Stylesheet with a Top-down Structure**

It always makes sense to lay your stylesheet out in a way that allows you to quickly find parts of your code. I recommend a top-down format that tackles styles as they appear in the source code. So, an example stylesheet might be ordered like this:

1. Generic classes (body, a, p, h1, etc.)
2. #header
3. #nav-menu
4. #main-content

|  |  |
| --- | --- |
| Example:  1  2  3  4  5  6  7 | /\*\*\*\*\*\* main content \*\*\*\*\*\*\*\*\*/    styles goes here...    /\*\*\*\*\*\* footer \*\*\*\*\*\*\*\*\*/    styles go here... |

**Combine Elements**

Elements in a stylesheet sometimes share properties. Instead of re-writing previous code, why not just combine them? For example, your h1, h2, and h3 elements might all share the same font and color:

|  |  |
| --- | --- |
| 1 | h1, h2, h3 {font-family: tahoma, color: #333} |

We could add unique characteristics to each of these header styles if we wanted (ie. h1 {size: 2.1em}) later in the stylesheet.

**Create Your HTML First**

Many designers create their CSS at the same time they create the HTML. It seems logical to create both at the same time, but actually you'll save even more time if you create the *entire* HTML mockup first. The reasoning behind this method is that you know all the elements of your site layout, but you don't know what CSS you'll need with your design. Creating the HTML layout first allows you to visualize the entire page as a whole, and allows you to think of your CSS in a more holistic, top-down manner.

**Use Multiple Classes**

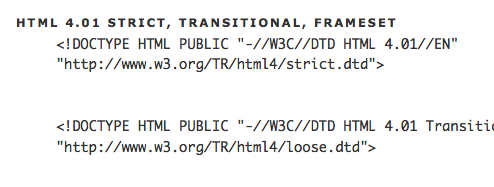
Sometimes it's beneficial to add multiple classes to an element. Let's say that you have a <div> "box" that you want to float right, and you've already got a class .right in your CSS that floats everything to the right. You can simply add an extra class in the declaration, like so:

|  |  |
| --- | --- |
| 1 | <div class="box right"></div> |

You can add as many classes as you'd like (**space** separated) to any declaration.

**Use the Right Doctype**

The doctype declaration matters a whole lot on whether or not your markup and CSS will validate. In fact, the entire look and feel of your site can change greatly depending on the DOCTYPE that you declare.



**Use Shorthand**

You can shrink your code considerably by using shorthand when crafting your CSS. For elements like padding, margin, font and some others, you can combine styles in one line. For example, a div might have these styles:

|  |  |
| --- | --- |
| 1  2  3  4  5 | #crayon {      margin-left:    5px;      margin-right:   7px;      margin-top: 8px;  } |

You could combine those styles in one line, like so:

|  |  |
| --- | --- |
| 1  2  3 | #crayon {      margin: 8px 7px 0px 5px; // top, right, bottom, and left values, respectively.  } |

If you need more help, here's a [comprehensive guide on CSS shorthand properties](http://www.dustindiaz.com/css-shorthand/).

**Comment your CSS**

Just like any other language, it's a great idea to comment your code in sections. To add a comment, simply add /\* behind the comment, and \*/ to close it, like so:

|  |  |
| --- | --- |
| 1 | /\* Here's how you comment CSS \*/ |

**Understand the Difference Between Block vs. Inline Elements**

Block elements are elements that naturally clear each line after they're declared, spanning the whole width of the available space. Inline elements take only as much space as they need, and don't force a new line after they're used.

Here are the lists of elements that are either inline or block:

|  |  |
| --- | --- |
| 1 | span, a, strong, em, img, br, input, abbr, acronym |

And the block elements:

|  |  |
| --- | --- |
| 1 | div, h1...h6, p, ul, li, table, blockquote, pre, form |

**Alphabetize your Properties**

While this is more of a frivolous tip, it can come in handy for quick scanning.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #cotton-candy {      color: #fff;      float: left;      font-weight:      height: 200px;      margin: 0;      padding: 0;      width: 150px;  } |

**Make Use of Generic Classes**

You'll find that there are certain styles that you're applying over and over. Instead of adding that particular style to each ID, you can create generic classes and add them to the IDs or other CSS classes (using tip #8).

For example, I find myself using float:right and float:left over an over in my designs. So I simply add the classes .left and .right to my stylesheet, and reference it in the elements.

|  |  |
| --- | --- |
| 1  2  3  4 | .left {float:left}  .right {float:right}    <div id="coolbox" class="left">...</div> |

This way you don't have to constantly add "float:left" to all the elements that need to be floated.

**Use "Margin: 0 auto" to Center Layouts**

Many beginners to CSS can't figure out why you can't simply use *float: center* to achieve that centered effect on block-level elements. If only it were that easy! Unfortunately, you'll need to use

|  |  |
| --- | --- |
| 1 | margin: 0 auto; // top, bottom - and left, right values, respectively. |

to center divs, paragraphs or other elements in your layout.

**Don't Just Wrap a DIV Around It**

When starting out, there's a temptation to wrap a div with an ID or class around an element and create a style for it.

|  |  |
| --- | --- |
| 1 | <div class="header-text"><h1>Header Text</h1></div> |

Sometimes it might seem easier to just create unique element styles like the above example, but you'll start to clutter your stylesheet. This would have worked just fine:

|  |  |
| --- | --- |
| 1 | <h1>Header Text</h1> |

Then you can easily add a style to the h1 instead of a parent div.

**Use Firebug**

If you haven't downloaded [Firebug](http://www.getfirebug.com/) yet, stop and go do it. Seriously. This little tool is a *must have* for any web developer. Among the many features that come bundled with the Firefox extension (debug JavaScript, inspect HTML, find errors), you can also visually inspect, modify, and edit CSS in real-time. You can learn more about what Firebug does on the official [Firebug website](http://www.getfirebug.com/).



**Use Absolute Positioning Sparingly**

[Absolute positioning](http://www.w3schools.com/Css/pr_class_position.asp) is a handy aspect of CSS that allows you to define where *exactly* an element should be positioned on a page to the exact pixel. However, because of absolute positioning's disregard for other elements on the page, the layouts can get quite hairy if there are multiple absolutely positioned elements running around the layout.

**Use Text-transform**

[Text-transform](http://www.w3schools.com/Css/pr_text_text-transform.asp) is a highly-useful CSS property that allows you to "standardize" how text is formatted on your site. For example, say you're wanting to create some headers that only have lowercase letters. Just add the text-transform property to the header style like so:

|  |  |
| --- | --- |
| 1 | text-transform: lowercase; |

Now all of the letters in the header will be lowercase by default. Text-transform allows you to modify your text (first letter capitalized, all letters capitalized, or all lowercase) with a simple property.

**Don't use Negative Margins to Hide Your h1**

Oftentimes people will use an image for their header text, and then either use display:none or a negative margin to float the h1 off the page. [Matt Cutts](http://www.mattcutts.com/), the head of Google's Webspam team, has officially said that this is a bad idea, as Google might think it's spam.

As Mr. Cutts explicitly says, avoid hiding your logo's text with CSS. Just use the alt tag. While many claim that you can still use CSS to hide a h1 tag as long as the h1 is the same as the logo text, I prefer to err on the safe side.

**Validate Your CSS and XHTML**

Validating your CSS and XHTML does more than give a sense of pride: it helps you quickly spot errors in your code. If you're working on a design and for some reason things just aren't looking right, try running the [markup](http://validator.w3.org/) and [CSS validator](http://jigsaw.w3.org/css-validator/) and see what errors pop up. Usually you'll find that you forgot to close a div somewhere, or a missed semi-colon in a CSS property.

**Ems vs. Pixels**

There's always been a strong debate as to whether it's better to use pixels (px) or ems (em) when defining font sizes. Pixels are a more static way to define font sizes, and ems are more scalable with different browser sizes and mobile devices. With the advent of many different types of web browsing (laptop, mobile, etc.), ems are increasingly becoming the default for font size measurements as they allow the greatest form of flexibility. You can read more about why you should use ems for font sizes in [this thoughtful forum thread](http://www.astahost.com/Sizes-Webdesign-Em-Vs-Px-t8926.html). About.com also has a great article on the [differences between the measurement sizes](http://webdesign.about.com/cs/typemeasurements/a/aa042803a.htm).

**Don't Underestimate the List**

Lists are a great way to present data in a structured format that's easy to modify the style. Thanks to the display property, you don't have to just use the list as a text attribute. Lists are also great for creating navigation menus and things of the sort.

Many beginners use divs to make each element in the list because they don't understand how to properly utilize them. It's well worth the effort to use brush up on learning list elements to structure data in the future.

**Avoid Extra Selectors**

It's easy to unknowingly add extra selectors to our CSS that clutters the stylesheet. One common example of adding extra selectors is with lists.

|  |  |
| --- | --- |
| 1 | body #container .someclass ul li {....} |

In this instance, just the *.someclass li* would have worked just fine.

|  |  |
| --- | --- |
| 1 | .someclass li {...} |

Adding extra selectors won't bring Armageddon or anything of the sort, but they do keep your CSS from being as simple and clean as possible.

**Add Margins and Padding to All**

Different browsers render elements differently. IE renders certain elements differently than Firefox. IE 6 renders elements differently than IE 7 and IE 8. While the browsers are starting to adhere more closely to [W3C standards](http://www.w3.org/), they're still not perfect (\*cough IE cough\*).

One of the main differences between versions of browsers is how padding and margins are rendered. If you're not already using a reset, you might want to define the margin and padding for all elements on the page, to be on the safe side. You can do this quickly with a global reset, like so:

|  |  |
| --- | --- |
| 1 | \* {margin:0;padding:0;} |

Now *all* elements have a padding and margin of 0, unless defined by another style in the stylesheet.

**When Ready, Try Object Oriented CSS**

Object Oriented programming is the separation of elements in the code so that they're easier to maintain reuse. [Object Oriented CSS](http://wiki.github.com/stubbornella/oocss) follows the same principle of separating different aspects of the stylesheet(s) into more logical sections, making your CSS more modular and reusable.

**Use Multiple Stylesheets**

Depending on the complexity of the design and the size of the site, it's sometimes easier to make smaller, multiple stylesheets instead of one giant stylesheet. Aside from it being easier for the designer to manage, multiple stylesheets allow you to leave out CSS on certain pages that don't need them.

For example, I might having a polling program that would have a unique set of styles. Instead of including the poll styles to the main stylesheet, I could just create a *poll.css* and the stylesheet only to the pages that show the poll.

**Check for Closed Elements First When Debugging**

If you're noticing that your design looks a tad wonky, there's a good chance it's because you've left off a closing *</div>*. You can use the [XHTML validator](http://validator.w3.org/) to help sniff out all sorts of errors too.