**CSS Beginner Tutorial**

Like the [HTML Beginner Tutorial](http://www.htmldog.com/guides/html/beginner/), the *CSS Beginner Tutorial* assumes that you know as much about CSS as you do about the cumulative effects of sea squirt discharge on the brain chemistry of Germanic ammonites. The purpose of this guide is to teach the bare essentials - just enough to get started. The [CSS Intermediate Tutorial](http://www.htmldog.com/guides/css/intermediate/) and [CSS Advanced Tutorial](http://www.htmldog.com/guides/css/advanced/) go into more depth about CSS.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

*CSS*, or *Cascading Styles Sheets*, is a way to style and present HTML. Whereas the HTML is the *meaning* or *content*, the style sheet is the *presentation* of that document.

Styles don’t smell or taste anything like HTML, they have a format of ‘*property: value*’ and most properties can be applied to most HTML tags.

**Contents**

* [Applying CSS](http://www.htmldog.com/guides/css/beginner/applyingcss/) - The different ways you can apply CSS to HTML.
* [Selectors, Properties, and Values](http://www.htmldog.com/guides/css/beginner/selectors/) - The bits that make up CSS.
* [Colors](http://www.htmldog.com/guides/css/beginner/colors/) - How to use color.
* [Text](http://www.htmldog.com/guides/css/beginner/text/) - How to manipulate the size and shape of text.
* [Margins and Padding](http://www.htmldog.com/guides/css/beginner/margins/) - How to space things out.
* [Borders](http://www.htmldog.com/guides/css/beginner/borders/) - Erm. Borders. Things that go around things.
* [Putting It All Together](http://www.htmldog.com/guides/css/beginner/conclusion/) - Throwing all of the above ingredients into one spicy hotpot.

# Applying CSS

There are three ways to apply CSS to HTML: In-line, internal, and external.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## In-line

In-line styles are plonked straight into the HTML tags using the style attribute.

They look something like this:

<p style="color: red">text</p>

This will make that specific paragraph red.

But, if you remember, the best-practice approach is that the HTML should be a stand-alone, presentation free document, and so in-line styles should be avoided wherever possible.

## Internal

Embedded, or internal, styles are used for the whole page. Inside the [head](http://www.htmldog.com/reference/htmltags/head/) element, the [style](http://www.htmldog.com/reference/htmltags/style/) tags surround all of the styles for the page.

<!DOCTYPE html>

<html>

<head>

<title>CSS Example</title>

*<style>*

p {

color: red;

}

a {

color: blue;

}

*</style>*

...

This will make all of the paragraphs in the page red and all of the links blue.

Although preferable to soiling our HTML with inline presentation, it is similarly usually preferable to keep the HTML and the CSS files separate, and so we are left with our savior…

## External

External styles are used for the whole, multiple-page website. There is a separate CSS file, which will simply look something like:

p {

color: red;

}

a {

color: blue;

}

If this file is saved as “style.css” in the same directory as your HTML page then it can be linked to in the HTML like this:

<!DOCTYPE html>

<html>

<head>

<title>CSS Example</title>

<link rel="stylesheet" href="style.css">

...

## Apply!

To get the most from this guide, it would be a good idea to try out the code as we go along, so start a fresh new file with your text-editor and save the blank document as “style.css” in the same directory as your HTML file.

Now change your HTML file so that it starts something like this:

<!DOCTYPE html>

<html>

<head>

<title>My first web page</title>

<link rel="stylesheet" href="style.css">

</head>

...

Save the HTML file. This now links to the CSS file, which is empty at the moment, so won’t change a thing. As you work your way through the CSS Beginner Tutorial, you will be able to add to and change the CSS file and see the results by simply refreshing the browser window that has the HTML file in it, as we did before.

**Selectors, Properties, and Values**

Whereas HTML has *tags*, CSS has *selectors*. Selectors are the names given to styles in internal and external style sheets. In this CSS Beginner Tutorial we will be concentrating on *HTML selectors*, which are simply the names of HTML tags and are used to change the style of a specific type of element.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

For each selector there are “*properties*” inside *curly brackets*, which simply take the form of words such as [color](http://www.htmldog.com/reference/cssproperties/color/), [font-weight](http://www.htmldog.com/reference/cssproperties/font-weight/) or [background-color](http://www.htmldog.com/reference/cssproperties/background-color/).

A *value* is given to the property following a *colon* (NOT an “equals” sign) and *semi-colons* separate the properties.

body {

font-size: 14px;

color: navy;

}

This will apply the given values to the [font-size](http://www.htmldog.com/reference/cssproperties/font-size/) and [color](http://www.htmldog.com/reference/cssproperties/color/) properties to the body selector.

So basically, when this is applied to an HTML document, text between the body tags (which is the content of the whole window) will be 14 pixels in size and navy in color.

**Lengths and Percentages**

There are many property-specific units for values used in CSS, but there are some general units that are used by a number of properties and it is worth familiarizing yourself with these before continuing.

* px (such as font-size: 12px) is the unit for pixels.
* em (such as font-size: 2em) is the unit for the calculated size of a font. So “2em”, for example, is two times the current font size.
* pt (such as font-size: 12pt) is the unit for points, for measurements typically in printed media.
* % (such as width: 80%) is the unit for… wait for it… percentages.

Other units include pc (picas), cm (centimeters), mm (millimeters) and in (inches).

When a value is *zero*, you do not need to state a unit. For example, if you wanted to specify no border, it would be border: 0.

“px” in this case, doesn’t actually necessarily mean pixels - the little squares that make up a computer’s display - all of the time. Modern browsers allow users to zoom in and out of a page so that, even if you specify font-size: 12px, or height: 200px, for example, although these will be the genuine specified size on a non-zoomed browser, they will still increase and decrease in size with the user’s preference. It’s a good thing. Trust me.

**Colors**

CSS brings *16,777,216* colors to your disposal. They can take the form of a *name*, an *RGB* (red/green/blue) value or a *hex* code.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

The following values, to specify full-on as red-as-red-can-be, all produce the same result:

* red
* rgb(255,0,0)
* rgb(100%,0%,0%)
* #ff0000
* #f00

Predefined color names include aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow. transparent is also a valid value.

With the possible exception of black and white, color names have limited use in a modern, well-designed web sites because they are so specific and limiting.

The three values in the RGB value are from 0 to 255, 0 being the lowest level (no red, for example), 255 being the highest level (full red, for example). These values can also be a percentage.

*Hexadecimal* (previously and more accurately known as “*sexadecimal*”) is a *base-16* number system. We are generally used to the *decimal* number system (*base-10*, from 0 to 9), but hexadecimal has 16 digits, from 0 to f.

The hex number is prefixed with a hash character (*#*) and can be three or six digits in length. Basically, the three-digit version is a compressed version of the six-digit (#ff0000 becomes #f00, #cc9966 becomes #c96, etc.). The three-digit version is easier to decipher (the first digit, like the first value in RGB, is red, the second green and the third blue) but the six-digit version gives you more control over the exact color.

CSS3, the latest version of CSS, also allows you to define *HSL* colors - hue, saturation and lightness. [More on this, along with semi-transparent colors, can be found in the CSS Advanced Tutorial.](http://www.htmldog.com/guides/css/advanced/colors/)

**color and background-color**

Colors can be applied by using [color](http://www.htmldog.com/reference/cssproperties/color/) and [background-color](http://www.htmldog.com/reference/cssproperties/background-color/) (note that this must be the American English “color” and not “colour”).

A blue background and yellow text could look like this:

h1 {

*color: yellow;*

*background-color: blue;*

}

These colors might be a little too harsh, so you could change the code of your CSS file for slightly different shades:

body {

font-size: 14px;

color: navy;

}

h1 {

*color: #ffc;*

*background-color: #009;*

}

Save the CSS file and refresh your browser. You will see the colors of the first heading (the [h1](http://www.htmldog.com/reference/htmltags/h1h2h3h4h5h6/) element) have changed to yellow and blue.

You can apply the [color](http://www.htmldog.com/reference/cssproperties/color/) and [background-color](http://www.htmldog.com/reference/cssproperties/background-color/) properties to most HTML elements, including [body](http://www.htmldog.com/reference/htmltags/body/), which will change the colors of the page and everything in it.

**Text**

You can alter the size and shape of the text on a web page with a range of properties.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

**font-family**

This is the font itself, such as Times New Roman, Arial, or Verdana.

The user’s browser has to be able to find the font you specify, which, in most cases, means it needs to be on *their* computer so there is little point in using obscure fonts that are only sitting on *your* computer. There are a select few “*safe*” fonts (the most commonly used are Arial, Verdana and Times New Roman), but you can specify more than one font, separated by *commas*. The purpose of this is that if the user does not have the first font you specify, the browser will go through the list until it finds one it does have. This is useful because different computers sometimes have different fonts installed. So font-family: arial, helvetica, serif, will look for the Arial font first and, if the browser can’t find it, it will search for Helvetica, and then a common serif font.

Note: if the name of a font is more than one word, it should be put in quotation marks, such as font-family: "Times New Roman".

You can use a wider selection than the “safe” fonts using several methods outlined in the CSS Advanced Tutorial but if you’re just getting to grips with CSS, we suggest sticking with this basic standard approach for the moment.

**font-size**

The size of the font. Be careful with this - text such as headings should not just be an HTML paragraph ([p](http://www.htmldog.com/reference/htmltags/p/)) in a large font - you should still use headings ([h1](http://www.htmldog.com/reference/htmltags/h1h2h3h4h5h6/), [h2](http://www.htmldog.com/reference/htmltags/h1h2h3h4h5h6/) etc.) even though, in practice, you could make the font-size of a paragraph larger than that of a heading (not recommended for sensible people).

**font-weight**

This states whether the text is bold or not. Most commonly this is used as font-weight: bold or font-weight: normal but other values are bolder, lighter, 100, 200, 300, 400 (same as normal), 500, 600, 700 (same as bold), 800 or 900.

Play around with these font-weight values if you want see their effect but, keep in mind, that some older browsers become a little confused with anything other than bold and normal so we suggest sticking to those unless you’re a typography ninja.

**font-style**

This states whether the text is italic or not. It can be font-style: italic or font-style: normal.

**text-decoration**

This states whether the text has got a line running under, over, or through it.

* text-decoration: underline, does what you would expect.
* text-decoration: overline places a line above the text.
* text-decoration: line-through puts a line through the text (“strike-through”).

This property is usually used to decorate links and you can specify no underline with text-decoration: none.

Underlines should only really be used for links. They are a commonly understood web convention that has lead users to generally expect underlined text to be a link.

**text-transform**

This will change the case of the text.

* text-transform: capitalize turns the first letter of every word into uppercase.
* text-transform: uppercase turns everything into uppercase.
* text-transform: lowercase turns everything into lowercase.
* text-transform: none I’ll leave for you to work out.

So, a few of these things used together might look like this:

body {

*font-family: arial, helvetica, sans-serif;*

*font-size: 14px;*

}

h1 {

*font-size: 2em;*

}

h2 {

*font-size: 1.5em;*

}

a {

*text-decoration: none;*

}

strong {

*font-style: italic;*

*text-transform: uppercase;*

}

**Text spacing**

Before we move on from this introduction to styling text, a quick look at how to space out the text on a page:

The [letter-spacing](http://www.htmldog.com/reference/cssproperties/letter-spacing/) and [word-spacing](http://www.htmldog.com/reference/cssproperties/word-spacing/) properties are for spacing between letters or words. The value can be a length or normal.

The [line-height](http://www.htmldog.com/reference/cssproperties/line-height/) property sets the height of the lines in an element, such as a paragraph, without adjusting the size of the font. It can be a number (which specifies a multiple of the font size, so “2” will be two times the font size, for example), a length, a percentage, or normal.

The [text-align](http://www.htmldog.com/reference/cssproperties/text-align/) property will align the text inside an element to left, right, center, or justify.

The [text-indent](http://www.htmldog.com/reference/cssproperties/text-indent/) property will indent the first line of a paragraph, for example, to a given length or percentage. This is a style traditionally used in print, but rarely in digital media such as the web.

p {

*letter-spacing: 0.5em;*

*word-spacing: 2em;*

*line-height: 1.5;*

*text-align: center;*

}

# Margins and Padding

[margin](http://www.htmldog.com/reference/cssproperties/margin/) and [padding](http://www.htmldog.com/reference/cssproperties/padding/) are the two most commonly used properties for spacing-out elements. A margin is the space outside something, whereas padding is the space inside something.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

Change the CSS code for h2 to the following:

h2 {

font-size: 1.5em;

*background-color:* #ccc*;*

*margin:* 20px*;*

*padding:* 40px*;*

}

This leaves a 20-pixel width space around the secondary header and the header itself is [fat](http://www.htmldog.com/guides/css/beginner/margins/) from the 40-pixel width padding.

The four sides of an element can also be set individually. [margin-top](http://www.htmldog.com/reference/cssproperties/margin/), [margin-right](http://www.htmldog.com/reference/cssproperties/margin/), [margin-bottom](http://www.htmldog.com/reference/cssproperties/margin/), [margin-left](http://www.htmldog.com/reference/cssproperties/margin/), [padding-top](http://www.htmldog.com/reference/cssproperties/padding/), [padding-right](http://www.htmldog.com/reference/cssproperties/padding/), [padding-bottom](http://www.htmldog.com/reference/cssproperties/padding/) and [padding-left](http://www.htmldog.com/reference/cssproperties/padding/) are the self-explanatory properties you can use.

## The Box Model

Margins, padding and borders (see [next page](http://www.htmldog.com/guides/css/beginner/borders/)) are all part of what’s known as the Box Model. The Box Model works like this: in the middle you have the content area (let’s say an image), surrounding that you have the padding, surrounding that you have the border and surrounding that you have the margin. It can be visually represented like this:

Margin box

Border box

Padding box

Element box

You don’t have to use all of these, but it can be helpful to remember that the box model can be applied to every element on the page, and that’s a powerful thing!

# Borders

Borders can be applied to most HTML elements within the body.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

To make a border around an element, all you need is [border-style](http://www.htmldog.com/reference/cssproperties/border-style/). The values can be solid, dotted, dashed, double, groove, ridge, inset and outset.

[border-width](http://www.htmldog.com/reference/cssproperties/border-width/) sets the width of the border, most commonly using pixels as a value. There are also properties for [border-top-width](http://www.htmldog.com/reference/cssproperties/border-width/), [border-right-width](http://www.htmldog.com/reference/cssproperties/border-width/), [border-bottom-width](http://www.htmldog.com/reference/cssproperties/border-width/) and [border-left-width](http://www.htmldog.com/reference/cssproperties/border-width/).

Finally, [border-color](http://www.htmldog.com/reference/cssproperties/border-color/) sets the color.

Add the following code to the CSS file:

h2 {

*border-style:* dashed*;*

*border-width:* 3px*;*

*border-left-width:* 10px*;*

*border-right-width:* 10px*;*

*border-color:* red*;*

}

This will make a red dashed border around all HTML secondary headers (the [h2](http://www.htmldog.com/reference/htmltags/h1h2h3h4h5h6/) element) that is 3 pixels wide on the top and bottom and 10 pixels wide on the left and right (these having over-ridden the 3 pixel wide width of the entire border).

# Putting It All Together

You should already have an HTML file like the one made at [the end of the HTML Beginner Tutorial](http://www.htmldog.com/guides/html/beginner/conclusion/), with a line that we added at [the start of this CSS Beginner Tutorial](http://www.htmldog.com/guides/css/beginner/applyingcss/), linking the HTML file to the CSS file.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

The code below covers all of the CSS methods in this section. If you save this as your CSS file and look at the HTML file then you should now understand what each CSS property does and how to apply them. The best way to fully understand all of this is to mess around with the HTML and the CSS files and see what happens when you change things.

body {

font-family: arial, helvetica, sans-serif;

font-size: 14px;

color: black;

background-color: #ffc;

margin: 20px;

padding: 0;

}

/\* This is a comment, by the way \*/

p {

line-height: 21px;

}

h1 {

color: #ffc;

background-color: #900;

font-size: 2em;

margin: 0;

margin-bottom: 7px;

padding: 4px;

font-style: italic;

text-align: center;

letter-spacing: 0.5em;

border-bottom-style: solid;

border-bottom-width: 0.5em;

border-bottom-color: #c00;

}

h2 {

color: white;

background-color: #090;

font-size: 1.5em;

margin: 0;

padding: 2px;

padding-left: 14px;

}

h3 {

color: #999;

font-size: 1.25em;

}

img {

border-style: dashed;

border-width: 2px;

border-color: #ccc;

}

a {

text-decoration: none;

}

strong {

font-style: italic;

text-transform: uppercase;

}

li {

color: #900;

font-style: italic;

}

table {

background-color: #ccc;

}

# CSS Intermediate Tutorial

Like the HTML Intermediate Tutorial, this CSS Intermediate Guide should not be that difficult, but rather build on the basics of the [CSS Beginner Tutorial](http://www.htmldog.com/guides/css/beginner/).

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Contents

* [Class and ID Selectors](http://www.htmldog.com/guides/css/intermediate/classid/): Make your own selectors without the need for sticky-backed plastic!
* [Grouping and Nesting](http://www.htmldog.com/guides/css/intermediate/grouping/): Properties assigned to multiple selectors or selectors within selectors.
* [Pseudo Classes](http://www.htmldog.com/guides/css/intermediate/pseudoclasses/): Defining various states of a link selector.
* [Shorthand Properties](http://www.htmldog.com/guides/css/intermediate/shorthand/): Various properties, such as borders and margins that amalgamate other properties into one.
* [Background Images](http://www.htmldog.com/guides/css/intermediate/backgroundimages/): Guess.
* [Specificity](http://www.htmldog.com/guides/css/intermediate/specificity/): How a browser will deal with conflicting CSS rules.
* [Display](http://www.htmldog.com/guides/css/intermediate/display/): Specifying the characteristics of a box.
* [Pseudo Elements](http://www.htmldog.com/guides/css/intermediate/pseudoelements/): Styling first letters, first lines and placing content before and after elements.
* [Page Layout](http://www.htmldog.com/guides/css/intermediate/layout/): Floating and positioning boxes.

# Class and ID Selectors

For the [CSS Beginner Tutorial](http://www.htmldog.com/guides/css/beginner/) we looked solely at [HTML selectors](http://www.htmldog.com/guides/css/beginner/selectors/) - those that represent an HTML tag.

You can also define your own selectors in the form of class and ID selectors.

The benefit of this is that you can have the same HTML element, but present it differently depending on its class or ID.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

In the CSS, a class selector is a name preceded by a full stop (“.”) and an ID selector is a name preceded by a hash character (“#”).

So the CSS might look something like:

#top {

background-color: #ccc;

padding: 20px

}

.intro {

color: red;

font-weight: bold;

}

The HTML refers to the CSS by using the attributes id and class. It could look something like this:

<div *id*=*"top"*>

<h1>Chocolate curry</h1>

<p *class*=*"intro"*>This is my recipe for making curry purely with chocolate</p>

<p *class*=*"intro"*>Mmm mm mmmmm</p>

</div>

The difference between an ID and a class is that an ID can be used to identify one element, whereas a class can be used to identify more than one.

You can also apply a selector to a specific HTML element by simply stating the HTML selector first, so p.jam { /\* whatever \*/ } will only be applied to paragraph elements that have the class “jam”.

# Grouping and Nesting

Two ways that you can simplify your code - both HTML and CSS - and make it easier to manage.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Grouping

You can give the same properties to a number of selectors without having to repeat them.

For example, if you have something like:

h2 {

color: red;

}

.thisOtherClass {

color: red;

}

.yetAnotherClass {

color: red;

}

You can simply separate selectors with commas in one line and apply the same properties to them all so, making the above:

h2, .thisOtherClass, .yetAnotherClass {

color: red;

}

## Nesting

If the CSS is structured well, there shouldn’t be a need to use many class or ID selectors. This is because you can specify properties to selectors within other selectors.

For example:

#top {

background-color: #ccc;

padding: 1em

}

#top h1 {

color: #ff0;

}

#top p {

color: red;

font-weight: bold;

}

This removes the need for classes or IDs on the [p](http://www.htmldog.com/reference/htmltags/p/) and [h1](http://www.htmldog.com/reference/htmltags/h1h2h3h4h5h6/) tags if it is applied to HTML that looks something like this:

<div id="top">

<h1>Chocolate curry</h1>

<p>This is my recipe for making curry purely with chocolate</p>

<p>Mmm mm mmmmm</p>

</div>

This is because, by separating selectors with spaces, we are saying “[h1](http://www.htmldog.com/reference/htmltags/h1h2h3h4h5h6/) inside ID top is colour #ff0” and “[p](http://www.htmldog.com/reference/htmltags/p/) inside ID top is red and bold”.

This can get quite complicated (because it can go for more than two levels, such as this inside this inside this inside this etc.) and may take a bit of practice.

**Pseudo Classes**

*Pseudo classes* are bolted on to selectors to specify a state or relation to the selector. They take the form of *selector:pseudo\_class { property: value; }*, simply with a colon in between the selector and the pseudo class.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

**Links**

link, targeting *unvisited links*, and visited, targeting, you guessed it, *visited links*, are the most basic pseudo classes.

The following will apply colors to all links in a page depending on whether the user has visited that page before or not:

a*:link* {

color: blue;

}

a*:visited* {

color: purple;

}

**Dynamic Pseudo Classes**

Also commonly used for links, the dynamic pseudo classes can be used to apply styles when something happens to something.

* active is for when something activated by the user, such as when a link is clicked on.
* hover is for a when something is passed over by an input from the user, such as when a cursor moves over a link.
* focus is for when something gains focus, that is when it is selected by, or is ready for, keyboard input.

focus is most often used on *form elements* but can be used for *links*. Although most users will navigate around and between pages using a pointing device such as a mouse those who choose note to, or are unable to do so, such as those with motor disabilities, may navigate using a keyboard or similar device. Links can be jumped between using a tab key and they will gain focus one at a time.

a*:active* {

color: red;

}

a*:hover* {

text-decoration: none;

color: blue;

background-color: yellow;

}

input*:focus*, textarea*:focus* {

background: #eee;

}

focus isn’t supported by some of the older browsers so be careful not to use it for anything vital.

**First Children**

Finally (for this article, at least), there is the first-child pseudo class. This will target something only if it is the very first descendant of an element. So, in the following HTML…

<body>

<p>I’m the body’s first paragraph child. I rule. Obey me.</p>

<p>I resent you.</p>

...

…if you only want to style the *first* paragraph, you could use the following CSS:

p*:first-child* {

font-weight: bold;

font-size: 40px;

}

CSS3 has also delivered a whole new set of pseudo classes: last-child, target, first-of-type, and more.

# Shorthand Properties

Some CSS properties allow a string of values, replacing the need for a number of properties. These are represented by values separated by spaces.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Margins and Padding

[margin](http://www.htmldog.com/reference/cssproperties/margin/) and [padding](http://www.htmldog.com/reference/cssproperties/padding/) allow you to amalgamate margin-top-width, margin-right-width, margin-bottom-width etc. in the form of property: top right bottom left;

So:

p {

margin-top: 1px;

margin-right: 5px;

margin-bottom: 10px;

margin-left: 20px;

}

Can be summed up as:

p {

*margin:* 1px 5px 10px 20px*;*

}

By stating just two values (such as padding: 1em 10em;), the first value will be the top and bottom and the second value will be the right and left.

## Borders

[border-width](http://www.htmldog.com/reference/cssproperties/border-width/) can be used in exactly the same was as [margin](http://www.htmldog.com/reference/cssproperties/margin/) and [padding](http://www.htmldog.com/reference/cssproperties/padding/). You can also combine [border-width](http://www.htmldog.com/reference/cssproperties/border-width/), [border-color](http://www.htmldog.com/reference/cssproperties/border-color/), and [border-style](http://www.htmldog.com/reference/cssproperties/border-style/) with the [border](http://www.htmldog.com/reference/cssproperties/border/) property. So:

p {

border-width: 1px;

border-color: red;

border-style: solid;

}

Can be simplified to be:

p {

*border:* 1px red solid*;*

}

The width/color/style combination can also be applied to [border-top](http://www.htmldog.com/reference/cssproperties/border/), [border-right](http://www.htmldog.com/reference/cssproperties/border/) etc.

## Fonts

Font-related properties can also be gathered together with the [font](http://www.htmldog.com/reference/cssproperties/font/) property:

p {

*font:* italic bold 12px/2 courier*;*

}

This combines [font-style](http://www.htmldog.com/reference/cssproperties/font-style/), [font-weight](http://www.htmldog.com/reference/cssproperties/font-weight/), [font-size](http://www.htmldog.com/reference/cssproperties/font-size/), [line-height](http://www.htmldog.com/reference/cssproperties/line-height/), and [font-family](http://www.htmldog.com/reference/cssproperties/font-family/).

So, to put it all together, try this code:

p {

font: 14px/1.5 "Times New Roman", times, serif;

padding: 30px 10px;

border: 1px black solid;

border-width: 1px 5px 5px 1px;

border-color: red green blue yellow;

margin: 10px 50px;

}

Lovely.

# Background Images

Used in a very different way to the [img](http://www.htmldog.com/reference/htmltags/img/) HTML element, CSS background images are a powerful way to add detailed presentation to a page.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

To jump in at the deep end, the [shorthand property](http://www.htmldog.com/guides/css/intermediate/shorthand/) [background](http://www.htmldog.com/reference/cssproperties/background/) can deal with all of the basic background image manipulation aspects.

body {

*background:* white url(*http:*//www.htmldog.com/images/bg.gif) no-repeat top right*;*

}

This amalgamates these properties:

* [background-color](http://www.htmldog.com/reference/cssproperties/background-color/), which we have [come across before](http://www.htmldog.com/guides/css/beginner/colors/).
* [background-image](http://www.htmldog.com/reference/cssproperties/background-image/), which is the location of the image itself.
* [background-repeat](http://www.htmldog.com/reference/cssproperties/background-repeat/), which is how the image repeats itself. Its value can be:
  + repeat, the equivalent of a “tile” effect across the whole background,
  + repeat-y, repeating on the y-axis, above and below,
  + repeat-x (repeating on the x-axis, side-by-side), or
  + no-repeat (which shows just one instance of the image).
* [background-position](http://www.htmldog.com/reference/cssproperties/background-position/), which can be top, center, bottom, left, right, a length, or a percentage, or any sensible combination, such as top right.

Background-images can be used in most HTML elements - not just for the whole page (body) and can be used for simple but effective results. As an example, background images are used on this web site as the bullets in lists, as the magnifying glass in the search box, and as the icons in the top left corner of some notes, such as this one.

# Specificity

If you have two (or more) conflicting CSS rules that point to the same element, there are some basic rules that a browser follows to determine which one is most specific and therefore wins out.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

It may not seem like something that important, and in most cases you won’t come across any conflicts at all, but the larger and more complex your CSS files become, or the more CSS files you start to juggle with, the greater likelihood there is of conflicts arising.

## More Specific = Greater Precedence

If the selectors are the same then the last one will always take precedence. For example, if you had:

p { color: red }

p { color: blue }

The text in the box of [p](http://www.htmldog.com/reference/htmltags/p/) elements would be colored blue because that rule came last.

However, you won’t usually have identical selectors with conflicting declarations on purpose (because there’s not much point). Conflicts quite legitimately come up, though, when you have [nested selectors](http://www.htmldog.com/guides/css/intermediate/grouping/).

div p { color: red }

p { color: blue }

In this example it might seem that a [p](http://www.htmldog.com/reference/htmltags/p/) within a [div](http://www.htmldog.com/reference/htmltags/div/) would be colored blue, seeing as a rule to color [p](http://www.htmldog.com/reference/htmltags/p/) boxes blue comes last, but they would actually be colored red due to the specificity of the first selector. Basically, the more specific a selector, the more preference it will be given when it comes to conflicting styles.

## Calculating Specificity

The actual specificity of a group of nested selectors takes some calculating. Basically, you give every ID selector (“#whatever”) a value of 100, every class selector (“.whatever”) a value of 10 and every HTML selector (“whatever”) a value of 1. Then you add them all up and hey presto, you have the specificity value.

* [p](http://www.htmldog.com/reference/htmltags/p/) has a specificity of 1 (1 HTML selector)
* div p has a specificity of 2 (2 HTML selectors, 1+1)
* .tree has a specificity of 10 (1 class selector)
* div p.tree has a specificity of 12 (2 HTML selectors + a class selector, 1+1+10)
* #baobab has a specificity of 100 (1 id selector)
* body #content .alternative p has a specificity of 112 (HTML selector + id selector + class selector + HTML selector, 1+100+10+1)

So if all of these examples were used, div p.tree (with a specificity of 12) would win out over div p (with a specificity of 2) and body #content .alternative p would win out over all of them, regardless of the order.

# Display

A key trick to the manipulation of HTML elements is understanding that there’s nothing at all special about how most of them work. Most pages could be made up from just a few tags that can be styled any which way you choose. The browser’s default visual representation of most HTML elements consist of varying font styles, margins, padding and, essentially, display types.

The most fundamental types of display are inline, block and none and they can be manipulated with the [display](http://www.htmldog.com/reference/cssproperties/display/) property and the shockingly surprising values inline, block and none.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Inline

inline does just what it says - boxes that are displayed inline follow the flow of a line. Anchor (links) and emphasis are examples of elements that are displayed inline by default.

The following code, for example, will cause all list items in a list to appear next to each other in one continuous line rather than each one having its own line:

li { *display:* inline }

## Block

block makes a box standalone, fitting the entire width of its containing box, with an effective line break before and after it. Unlike inline boxes, block boxes allow greater manipulation of height, margins, and padding. Heading and paragraph elements are examples of elements that are displayed this way by default in browsers.

The next example will make all links in “nav” large clickable blocks:

#navigation a {

*display:* block*;*

padding: 20px 10px;

}

display: inline-block will keep a box inline but lend the greater formatting flexibility of block boxes, allowing margin to the right and left of the box, for example.

## None

none, well, doesn’t display a box at all, which may sound pretty useless but can be used to good effect with dynamic effects, such as switching extended information on and off at the click of a link, or in alternative stylesheets.

The following code, for example, could be used in a print stylesheet to basically “turn off” the display of elements such as navigation that would be useless in that situation:

#navigation, #related\_links { *display:* none }

display: none and visibility: hidden vary in that display: none takes the element’s box completely out of play, whereas visibility: hidden keeps the box and its flow in place without visually representing its contents. For example, if the second paragraph of 3 were set to display: none, the first paragraph would run straight into the third whereas if it were set to visibility: hidden, there would be a gap where the paragraph should be.

## Tables

OK. So that was the basics. Now for something a little more advanced and rarely used…

Perhaps the best way to understand the table-related [display](http://www.htmldog.com/reference/cssproperties/display/) property values is to think of HTML tables. table is the initial display and you can mimic the [tr](http://www.htmldog.com/reference/htmltags/tr/) and [td](http://www.htmldog.com/reference/htmltags/td/) elements with the table-row and table-cell property values respectively.

The [display](http://www.htmldog.com/reference/cssproperties/display/) property goes further by offering table-column, table-row-group, table-column-group, table-header-group, table-footer-group and table-caption as values, which are all quite self-descriptive. The immediately obvious benefit of these values is that you can construct a table by columns, rather than the row-biased method used in HTML.

Finally, the value inline-table basically sets the table without line breaks before and after it.

Be careful when using these values. Older browsers struggle with them and getting carried away with CSS tables can seriously damage your accessibility. HTML should be used to convey meaning, so if you have tabular data it should be arranged in HTML tables. Using CSS tables exclusively could result in a mash of data that is completely unreadable without the CSS. Bad. And not in a Michael Jackson way.

## Other display types

list-item displays a box in the way that you would usually expect an [li](http://www.htmldog.com/reference/htmltags/li/) HTML element to be displayed. To work properly then, elements displayed this way should be nested in a [ul](http://www.htmldog.com/reference/htmltags/ul/) or [ol](http://www.htmldog.com/reference/htmltags/ol/) element.

run-in makes a box either in-line or block depending on the display of its parent.

# Pseudo Elements

Pseudo elements suck on to selectors much like [pseudo classes](http://www.htmldog.com/guides/css/intermediate/pseudoclasses/), taking the form of selector:pseudoelement { property: value; }. There are four of the suckers.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## First Letters and First Lines

The first-letter pseudo element applies to the first letter inside a box and first-line to the top-most displayed line in a box.

As an example, you could create drop caps and a bold first-line for paragraphs with something like this:

p {

font-size: 12px;

}

*p:*first-letter {

font-size: 24px;

float: left;

}

*p:*first-line {

font-weight: bold;

}

The CSS 3 specs suggest pseudo elements should include two colons, so p::first-line as opposed to p:first-line. This is to differentiate them with pseudo classes. Aiming for backwards-compatibility (whereby older web pages will still work in new browsers), browsers will still behave if they come across the single colon approach and this remains the best approach in most circumstances due to some older browsers not recognizing the double colon.

## Before and After Content

The before and after pseudo elements are used in conjunction with the [content](http://www.htmldog.com/reference/cssproperties/content/) property to place content either side of a box without touching the HTML.

What?! Content in my CSS?! But I thought HTML was for content!

Well, it is. So use sparingly. Look at it like this: You are borrowing content to use solely as presentation, such as using “!” because it looks pretty. Not because you actually want to exclaim anything.

The value of the [content](http://www.htmldog.com/reference/cssproperties/content/) property can be open-quote, close-quote, any string enclosed in quotation marks, or any image, using url(imagename).

blockquote:before {

content: open-quote;

}

blockquote:after {

content: close-quote;

}

li:before {

content: "POW! ";

}

p:before {

content: url(images/jam.jpg);

}

The [content](http://www.htmldog.com/reference/cssproperties/content/) property effectively creates another box to play with so you can also add styles to the “presentational content”:

li:before {

content: "POW! ";

background: red;

color: #fc0;

}

# Page Layout

In the olden days, pre-hominid apes used HTML tables to layout web pages. Hilarious, right?! But CSS, that 2001: A Space Odyssey monolith, soon came along and changed all of that.

Layout with CSS is easy. You just take a chunk of your page and shove it wherever you choose. You can place these chunks absolutely or relative to another chunk.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Positioning

The [position](http://www.htmldog.com/reference/cssproperties/position/) property is used to define whether a box is absolute, relative, static or fixed:

* static is the default value and renders a box in the normal order of things, as they appear in the HTML.
* relative is much like static but the box can be offset from its original position with the properties top, right, bottom and left.
* absolute pulls a box out of the normal flow of the HTML and delivers it to a world all of its own. In this crazy little world, the absolute box can be placed anywhere on the page using top, right, bottom and left.
* fixed behaves like absolute, but it will absolutely position a box in reference to the browser window as opposed to the web page, so fixed boxes should stay exactly where they are on the screen even when the page is scrolled.

When we talk of absolutely positioned boxes being placed anywhere on the page, they’re actually still relatively positioned from the edges of the page. And to add another backtrack, the page doesn’t actually have to be the container - a box will be “absolutely” positioned in relation to any non-static positioned containing box. Just ignore that for now, though…

### Layout using absolute positioning

You can create a traditional two-column layout with absolute positioning if you have something like the following HTML:

<div id="navigation">

<ul>

<li><a href="this.html">This</a></li>

<li><a href="that.html">That</a></li>

<li><a href="theOther.html">The Other</a></li>

</ul>

</div>

<div id="content">

<h1>Ra ra banjo banjo</h1>

<p>Welcome to the Ra ra banjo banjo page. Ra ra banjo banjo. Ra ra banjo banjo. Ra ra banjo banjo.</p>

<p>(Ra ra banjo banjo)</p>

</div>

We’re being old-skool and using [div](http://www.htmldog.com/reference/htmltags/div/) elements so as not to introduce too many new things, but [Sectioning](http://www.htmldog.com/guides/html/intermediate/sectioning/) is sexier.

And if you apply the following CSS:

#navigation {

*position:* absolute*;*

*top:* 0*;*

*left:* 0*;*

width: 200px;

}

#content {

margin-left: 200px;

}

You will see that this will set the navigation bar to the left and set the width to 200 pixels. Because the navigation is absolutely positioned, it has nothing to do with the flow of the rest of the page so all that is needed is to set the left margin of the content area to be equal to the width of the navigation bar.

How stupidly easy! And you aren’t limited to this two-column approach. With clever positioning, you can arrange as many blocks as you like. If you wanted to add a third column, for example, you could add a “navigation2” chunk to the HTML and change the CSS to:

#navigation {

position: absolute;

top: 0;

left: 0;

width: 200px;

}

#navigation2 {

position: absolute;

top: 0;

right: 0;

width: 200px;

}

#content {

margin: 0 200px; /\* setting top and bottom margin to 0 and right and left margin to 200px \*/

}

The only downside to absolutely positioned boxes is that because they live in a world of their own, there is no way of accurately determining where they end. If you were to use the examples above and all of your pages had small navigation bars and large content areas, you would be okay, but, especially when using relative values for widths and sizes, you often have to abandon any hope of placing anything, such as a footer, below these boxes. If you wanted to do such a thing, one way would be to float your chunks, rather than absolutely positioning them.

## Floating

Floating a box will shift it to the right or left of a line, with surrounding content flowing around it.

Floating is normally used to shift around smaller chunks within a page, such as pushing a navigation link to the right of a container, but it can also be used with bigger chunks, such as navigation columns.

Using [float](http://www.htmldog.com/reference/cssproperties/float/), you can float: left or float: right.

Working with the same HTML, you could apply the following CSS:

#navigation {

*float:* left*;*

width: 200px;

}

#navigation2 {

*float:* right*;*

width: 200px;

}

#content {

margin: 0 200px;

}

Then, if you do not want the next box to wrap around the floating objects, you can apply the [clear](http://www.htmldog.com/reference/cssproperties/clear/) property:

* clear: left will clear left floated boxes
* clear: right will clear right floated boxes
* clear: both will clear both left and right floated boxes.

So if, for example, you wanted a footer in your page, you could add a chunk of HTML…

<div id="footer">

<p>Footer! Hoorah!</p>

</div>

…and then add the following CSS:

#footer {

*clear:* both*;*

}

And there you have it. A footer that will appear underneath all columns, regardless of the length of any of them.

This has been a general introduction to positioning and floating, with emphasis on the larger “chunks” of a page, but remember, these methods can be applied to any box within those boxes, too. With a combination of positioning, floating, margins, padding and borders, you should be able to represent any web design your mischievous little imagination can conjure. The best way to learn? Tinker! Play! Go!

# CSS Advanced Tutorial

We’re entering the deepest realms of CSS3 now - aspects that are often more specific than the [CSS Intermediate Tutorial](http://www.htmldog.com/guides/css/intermediate/) but, y’know, really biscuity cool.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Contents

* [Rounded Corners](http://www.htmldog.com/guides/css/advanced/roundedcorners/): Corners. That are rounded.
* [Shadows](http://www.htmldog.com/guides/css/advanced/shadows/): Adding “pop” to boxes and text.
* [Universal, Child, and Adjacent Selectors](http://www.htmldog.com/guides/css/advanced/selectors/): More precise aim with clever selectors.
* [Advanced Colors](http://www.htmldog.com/guides/css/advanced/colors/): Alpha transparency and HSL.
* [At-Rules](http://www.htmldog.com/guides/css/advanced/atrules/): Importing style sheets, styles for different media types, specifying the character set of a stylesheet and embedded fonts.
* [Attribute Selectors](http://www.htmldog.com/guides/css/advanced/attributeselectors/): Targeting boxes by their elements’ HTML attributes.
* [Transitions](http://www.htmldog.com/guides/css/advanced/transitions/): Creating smooth animations.
* [Backgrounds: Multiples, Size, and Origin](http://www.htmldog.com/guides/css/advanced/backgrounds/)
* [Transformations](http://www.htmldog.com/guides/css/advanced/transformations/): Molding the size and shape of a box and its contents.
* [Gradients](http://www.htmldog.com/guides/css/advanced/gradients/): Linear and radial gradients without image files.
* [Media Queries](http://www.htmldog.com/guides/css/advanced/mediaqueries/): Optimizing pages for different devices and screen sizes.

# Rounded Corners

Rounded corners used to be the stuff of constricting solid background images or, for flexible boxes, numerous background images, one per-corner, slapped on multiple nested [div](http://www.htmldog.com/reference/htmltags/div/) elements. Argh, ugly. Well, not any longer. Now, with simple CSS, you can lavish your designs with more curves than Marilyn Monroe.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Border radius?

Yeah. Border radius. Fear not, though - you don’t have to have borders. The border-radius property can be used to add a corner to each corner of a box.

#marilyn {

background: #fc0;

padding: 20px;

*border-radius:* 10px*;*

}

And you get something like this. Rounded corners, see? If you’ve got a sensible browser (see note below).

Or:

#ok\_a\_border\_then {

border: 5px solid #8b2;

padding: 20px;

*border-radius:* 5px*;*

}

Woop.

## Multiple values

border-top-left-radius, border-top-right-radius, border-bottom-right-radius, and border-bottom-left-radius can also be used to target specific corners.

Ever so slightly less horribly wordy, you can also define all corner radii (what a great word) individually with a space-separated list of values, working clockwise from top-left, just like other [shorthand properties](http://www.htmldog.com/guides/css/intermediate/shorthand/):

#monroe {

background: #fc0;

padding: 20px;

*border-radius:* 3px 6px 9px 12px*;*

}

Curvy.

By using two values instead of four you target top-left and bottom-right with the first length (or percentage) and top-right + bottom-left with the second. Three values? Top-left, then top-right + bottom-left, then bottom-right. Smashing.

Gah! There just had to be browser issues, didn’t there? Damn you, browsers.

Internet Explorer versions 8 and below don’t support border-radius. The only way you can deal with this is either to make do with a design in those browsers that doesn’t have rounded corners (most people can live with that), or revert to the old background images.

You might also stumble across similar proprietary properties, such as -webkit-border-radius and -moz-border-radius which are for older, barely-used versions of Safari and Firefox respectively. Our carefully worded professional advice? Screw ‘em.

## Ellipses

Are circles a bit too square for you? You can specify different horizontal and vertical radiiii by splitting values with a “/”.

#nanoo {

width: 100px;

height: 50px;

text-align: center;

padding-top: 100px;

background: #fc0;

*border-radius:* 50px/100px*;*

border-bottom-left-radius: 50px;

border-bottom-right-radius: 50px;

}

Nanoo.

# Shadows

Look! It’s like someone’s shining a torch over my web page!

You can give parts of your page “pop” by applying shadows both to boxes and to text.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Box Shadows

box-shadow is the standard CSS property to get you going and it can have a value comprised of several parts:

box-shadow: 5px 5px 3px 1px #999

* The first value is the horizontal offset - how far the shadow is nudged to the right (or left if it’s negative)
* The second value is the vertical offset - how far the shadow is nudged downwards (or upwards if it’s negative)
* The third value is the blur radius - the higher the value the less sharp the shadow. (“0” being absolutely sharp). This is optional - omitting it is equivalent of setting “0”.
* The fourth value is the spread distance - the higher the value, the larger the shadow (“0” being the inherited size of the box). This is also optional - omitting it is equivalent of setting “0”.
* The fifth value is a color. That’s optional, too.

### Inner shadows

You can also apply shadows to the inside of a box by adding “inset” to the list:

box-shadow: inset 0 0 7px 5px #ddd;

Splendid!

You might come across browser-specific versions of box-shadow, such as -moz-box-shadow and -webkit-box-shadow. Ignore ‘em. They’re old and stupid. The majority of modern browsers understand box-shadow, including Internet Explorer versions 9 and above.

## Text Shadows

box-shadow, as its name implies, only has eyes for boxes. Fickle beast. But you can also apply shadows to the outline of text with (surprise!) text-shadow:

text-shadow: -2px 2px 2px #999;

Similarly to box-shadow:

* The first value is the horizontal offset
* The second value is the vertical offset
* The third value is the blur radius (optional)
* The fourth value is the color (optional, although omitting this will make the shadow the same color as the text itself)

Note that there is no spread distance or inset option for text-shadow.

text-shadow has taken a little bit longer for browsers to figure out. Internet Explorer 9 and below won’t understand it so we suggest only using it in non-critical situations.

# Universal, Child, and Adjacent Selectors

In earlier tutorials, we have covered [HTML selectors](http://www.htmldog.com/guides/css/beginner/selectors/), [Class and ID selectors](http://www.htmldog.com/guides/css/intermediate/classid/), and how to [combine selectors](http://www.htmldog.com/guides/css/intermediate/grouping/) to target specific element boxes. With the use of three itty-bitty characters, you can further pinpoint an element, reducing the need to bloat your HTML with class and ID attributes.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Universal selectors

Using an asterisk (“ \* ”), you can target everything under the sun. You can use it by itself to set global styles for a page, or as a descendant of a selector to set styles of everything within something.

The following, for example, will set the margin and padding on everything in a page to zero and everything within an element with the ID “contact” to be displayed as a block:

\* {

margin: 0;

padding: 0;

}

#contact \* {

display: block;

}

Using a standalone universal selector is commonly used to “reset” many of a browser’s default styles. Setting a margin to zero, for example, will kill all spacing around the likes of paragraphs, headings and blockquotes.

## Child selectors

A greater-than symbol (“>”) can be used to specify something that is a child of something else, that is, something immediately nested within something.

So, with this HTML…

<ul id="genus\_examples">

<li>Cats

<ul>

<li>Panthera</li>

<li>Felis</li>

<li>Neofelis</li>

</ul>

</li>

<li>Apes

<ul>

<li>Pongo</li>

<li>Pan</li>

<li>Homo</li>

</ul>

</li>

</ul>

…and the following CSS…

#genus\_examples *>* li { border: 1px solid red }

…a red border would be drawn around “Cats” and “Apes” only, rather than around every single list item (which would be the case with #genus\_examples li { border: 1px solid red }). This is because the likes of “Panthera” and “Felis” are grandchildren of “genus\_examples”, not children.

## Adjacent selectors

A plus sign (“+”) is used to target an adjacent sibling of an element, essentially, something immediately following something.

With the following HTML:

<h1>Clouded leopards</h1>

<p>Clouded leopards are cats that belong to the genus Neofelis.</p>

<p>There are two extant species: Neofelis nebulosa and Neofelis diardi.</p>

…and CSS…

h1 + p { font-weight: bold )

Only the first paragraph, that following the heading, will be made bold.

A further, CSS 3, “general sibling” selector uses a tilde (“~”) and will match an element following another regardless of its immediacy. So, in the above example, h1 ~ p { font-weight: bold } will style all paragraphs after the top-level heading but if there were any [p](http://www.htmldog.com/reference/htmltags/p/)s preceding the [h1](http://www.htmldog.com/reference/htmltags/h1h2h3h4h5h6/), these would not be affected.

# Advanced Colors

We already know that [colors can be defined by name, RGB, or hex values](http://www.htmldog.com/guides/css/beginner/colors/), but CSS 3 also allows you to paint away with HSL - hue, saturation, and lightness - as well as stipulating transparency.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

There are no super special properties at play here - HSL and RGBa (the “a” standing for “alpha”, as in “alpha transparency”) can be applied to any property that has a color value, such as [color](http://www.htmldog.com/reference/cssproperties/color/), [background-color](http://www.htmldog.com/reference/cssproperties/background-color/), [border-color](http://www.htmldog.com/reference/cssproperties/border-color/) or box-shadow, to name a mere handful.

## Alpha transparency

RGBa opens up an exciting new dimension to web design, allowing you to set the transparency of a box or text. If you wanted a smidgen of a snazzy background image to peep through a heading, for example, you might use something like this:

h1 {

padding: 50px;

background-image: url(snazzy.jpg);

*color:* rgba(0,0,0,0.8)*;*

}

A standard value of rgb(0,0,0) would set the heading to pure black but that fourth value, in rgba, sets the level of transparency, “1” being completely opaque, “0” being completely transparent. So rgba(0,0,0,0.8) is saying red=“0”, green=“0”, blue=“0”, alpha=“0.8”, which, all together, makes it 80% black.

This doesn’t only apply to text, of course, you could apply a transparent background color to an entire box, a transparent box shadow… anywhere where you can use rgb, you can used rgba.

## Hue, saturation, and lightness

Color names aside, web colors have always been red-green-blue biased, be that through hex codes or explicit RBG (or RGBa). Although mildly less straightforward (especially if your brain is trained to break down colors into red, green and blue), HSL can actually be more intuitive because it gives you direct control over the aspects of a color’s shade rather than its logical ingredients.

It is used in a similar way to rgb:

#smut { color: hsl(36, 100%, 50%) }

Rather than each sub-value being a part of the color spectrum, however, they are:

* Hue (“36” in the above example): Any angle, from 0 to 360, taken from a typical color wheel, where “0” (and “360”) is red, “120” is green and “240” is blue.
* Saturation (“100%” in the example): How saturated you want the color to be, from 0% (none, so a level of grey depending on the lightness) to 100% (the whole whack, please).
* Lightness (“50%” in the example): From 0% (black) to 100% (white), 50% being “normal”.

So the example used here will produce an orange (36°) that is rich (100% saturation) and vibrant (50% lightness). It is the equivalent of #ff9900, #f90, and rgb(255, 153, 0).

### HSLa

Hey, man, this funky fresh transparency and HSL can be combined?! You’d better believe it. Here’s HSLa:

#rabbit { background: hsla(0, 75%, 75%, 0.5) }

You can figure out what that does, right?

hsl and hsla are supported by most modern browsers, including IE versions 9 and above.

# At-Rules: @import, @media, and @font-face

At-rules are clever, powerful little huggers that encapsulate a bunch of CSS rules and apply them to something specific. They can be used to import other CSS files, apply CSS to a particular media, or embed funkysexy uncommon fonts.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

Each at-rule starts with an apetail, or an “at sign”, if you want to be boring about it (“@”).

## Importing

Let’s start with the simple @import rule. This is used to bolt another stylesheet onto your existing one.

@import url(morestyles.css);

This can be used if a site requires long, complex stylesheets that might be easier to manage if they are broken down into smaller files.

@import rules must be placed at the top of a stylesheet, before any other rules.

## Targeting media types

@media can be used to apply styles to a specific media, such as print.

*@media print* {

body {

font-size: 10pt;

font-family: times, serif;

}

#navigation {

display: none;

}

}

Values that follow “@media” can include screen, print, projection, handheld, and all, or a comma-separated list of more than one, such as:

*@media screen*, projection {

/\* ... \*/

}

It doesn’t stop there, oh no. CSS 3 allows you to target not only specific media but also variables relating to that media, such as screen size (particularly helpful in targeting phones). Have a gander at the [Media Queries](http://www.htmldog.com/guides/css/advanced/mediaqueries/) page for more.

## Embedding fonts

@font-face has been around for a long time but was nigh-on useless for much of its life. CSS 3 has polished it up and it is now widely used as a technique for embedding fonts in a web page so that a typeface can be used even if it isn’t sitting on the user’s computer. So you no longer need to rely on “web safe” fonts such as Arial or Verdana. Exciting times.

Jumping in at the deep end…

@font-face {

font-family: "font of all knowledge";

src: url(fontofallknowledge.woff);

}

What this does is create a font named “font of all knowledge” using the [font-family](http://www.htmldog.com/reference/cssproperties/font-family/) descriptor and links the font file “fontofallknowledge.woff” to that name using the src descriptor. “font of all knowledge” can then be used in a standard font rule, such as:

p { font-family: "font of all knowledge", arial, sans-serif; }

The font will be downloaded (in this case from the same directory as the CSS file) and applied to paragraphs. If the browser is too decrepit to deal with sparkly new font-faces, it will simply revert to the next, standard, font in the list. Magic!

You can also look for a number of fonts to apply to the rule with a comma-separated list. Checking to see if a font is already present on a user’s computer, removing the need to download it, can also be accomplished by replacing “url” in the src value with “local”.

And because a font file might contain a whole host of weights or styles, you might also want to specify which one you’re interested in. So the @font-face rule could end up looking something like this:

@font-face {

font-family: "font of all knowledge";

src: local("font of all knowledge"), local(fontofallknowledge), url(fontofallknowledge.woff);

font-weight: 400;

font-style: normal;

}

Legally speaking, you can’t just throw any old font you feel like up on the Interweb because there are copyright and usage terms to consider, not to mention compatibility and optimization issues.

There are free web fonts out there that you can find, download, upload, and use, though. Hell, you could even create one yourself if you’re mad-scientist clever. There are also web font providers, such as Adobe’s [Typekit](https://typekit.com/), that have a large selection of fonts to choose from at a price.

[Google Web Fonts](http://www.google.com/fonts/) has a more limited selection but it’s free to use and makes things super, super (super, super, super) easy. All you need to do is link to one of their external CSS files, which is nothing more than a @font-face rule, and whammo - you’ve got a darling new font to play with.

**Attribute Selectors**

What? More selectors? Damn straight. Because someone somewhere really wants you to keep your HTML as tidy as possible. So before you add a class attribute to a tag see if you can simply target it by an attribute that might already be there.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

Attribute selectors allow you to style an element’s box based on the presence of an HTML attribute or of an attribute’s value.

**With the attribute…**

Appending an attribute name enclosed in square brackets, to style something that simply contains a certain *attribute*, you can do something like this:

*abbr[title]* { border-bottom: 1px dotted #ccc }

This basically says “shove a dotted line underneath all abbreviations with a title attribute”.

**With the attribute and it’s value…**

You can further specify that you want to style something with a specific *attribute value*.

*input[type=text]* { width: 200px; }

This example will apply a width of 200 pixels only to [input](http://www.htmldog.com/reference/htmltags/input/) elements that are specified as being text boxes (<input type="text">).

This approach can certainly be useful in forms where [input](http://www.htmldog.com/reference/htmltags/input/) elements can take on many guises using the type attribute.

You can also target more than one attribute at a time in a way similar to the following:

*input[type=text][disabled]* { border: 1px solid #ccc }

This will only apply a gray border to text inputs that are disabled.

CSS 3 further allows you to match an attribute without being exact:

* [attribute^=something] will match a the value of an attribute that *begins* with something.
* [attribute$=something] will match a the value of an attribute that *ends* with something.
* [attribute\*=something] will match a the value of an attribute that *contains* something, be it in the beginning, middle, or end.

So, as an example, you could style external links (eg. “http://www.htmldog.com”) differently to internal links (eg. “overhere.html”) in the following way:

a[href^=http] {

padding-right: 10px;

background: url(arrow.png) right no-repeat;

}

# Transitions

CSS transitions allow you to easily animate parts of your design without the need for the likes of JavaScript. How dangerous.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

At the most simplistic level, think of a traditional :hover state, where you might change the appearance of a link when a cursor fondles it:

a:link {

color: hsl(36,50%,50%);

}

a:hover {

color: hsl(36,100%,50%);

}

This creates a binary animation; a link switches from a subdued orange to a rich orange when it is hovered over.

The transition property, however, is much more powerful, allowing smooth animation (rather than a jump from one state to another). It is a [shorthand property](http://www.htmldog.com/guides/css/intermediate/shorthand/) that combines the following properties (which can be used individually if you so choose):

* transition-property: which property (or properties) will transition.
* transition-duration: how long the transition takes.
* transition-timing-function: if the transition takes place at a constant speed or if it accelerates and decelerates.
* transition-delay: how long to wait until the transition takes place.

Returning to the shorthand property, if a transition is applied like so…

a:link {

*transition:* all .5s linear 0*;*

color: hsl(36,50%,50%);

}

a:hover {

color: hsl(36,100%,50%);

}

…when the link is hovered over, the color will gradually change rather than suddenly switch. The transition property here is saying it wants all properties transitioned over half a second in a linear fashion with no delay.

For a transition to take place, only transition-duration is required, the rest defaulting to transition-property: all; transition-timing-function: ease; transition-delay: 0;. So you could, for example, simply declare transition: .5s.

## Targeting specific properties

While “all” can be used in transition-property (or transition), you can tell a browser only to transition certain properties, by simply plonking in the property name you want to change. So the previous example could actually include transition: color .5s ease 0, given only the color changes.

If you want to target more than one property (without using “all”), you can offer up a comma-separated list with transition-property:

a:link {

transition: .5s;

*transition-property:* color, font-size*;*

...

Or you can offer up a slew of rules for transitioning each property like so:

a:link {

*transition:* color .5s, font-size 2s*;*

...

## Easing

OK, so transition-timing-function (catchy!) is the least obvious fella. It effectively states how the browser should deal with the intermediate states of the transition.

It follows a cubic Bézier curve. Yeah. Obviously, we know all about them from infant school, but, to get down to it, at the most basic level you can use ease or linear.

ease produces a gradual acceleration at the start of the transition and a gradual deceleration at the end. linear maintains a constant speed throughout. Other values include ease-in and ease-out.

Guess what? None of this works! Woooooo! How practical. Don’t get angry, though, you’re not wasting your time by considering using transitions.

“None of this works” is a tad flippant - transition does work on some browsers (IE 10 and the latest versions of Chrome and Firefox) but if you want to make sure your transitions work, you should consider repeating the declarations with browser-specific ugliness, such as:

a:link {

-webkit-transition: .2s;

transition: .2s;

}

The -webkit-transition property will add a transition to browsers such as Safari and older versions of Chrome, not to mention a host of mobile browsers, that don’t understand the transition property.

The remaining bad news is that CSS transitions won’t work at all in Internet Explorer versions 9 and below. Booo.

# CSS Backgrounds: Multiples, Size, and Origin

As well as plastering a [single or tiling background image](http://www.htmldog.com/guides/css/intermediate/backgroundimages/) across parts of your page, you can also apply multiple backgrounds, adjust the size of background images, and define the origin of a background based on levels of the box model.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

## Multiple backgrounds

This is the boogie web designers have been crying out for since Bing Crosbie was topping the charts. CSS 3 allows you to apply multiple background images to a single box by simply putting image locations in a comma-separated list:

background-image: url(this.jpg), url(that.gif), url(theother.png);

More usefully, you can also define all of the other delightful background aspects. If you wanted to style a chunky button-like link with separate background, bullet, and arrow, for example, you could use something like:

background: url(bg.png), url(bullet.png) 0 50% no-repeat, url(arrow.png) right no-repeat;

Easy, right? It’s just the same as using [background-image](http://www.htmldog.com/reference/cssproperties/background-image/), [background-position](http://www.htmldog.com/reference/cssproperties/background-position/), [background-repeat](http://www.htmldog.com/reference/cssproperties/background-repeat/), [background-attachment](http://www.htmldog.com/reference/cssproperties/background-attachment/), and [background](http://www.htmldog.com/reference/cssproperties/background/), except you can specify more than one background with the aid of that helpful little comma.

## Background size

The background-size property allows you to stretch or compress a background image. The values can be:

* auto, which maintains the background image’s original size and width/height ratio.
* lengths, a width and a height, such as 100px 50px (100px wide, 50px high). Specifying a single length, such as 100px will result the equivalent of 100px auto.
* percentages, a width and a height, such as 50% 25% (50% of the width of the background area, 25% of the height of the background area). Specifying a single percentage, such as 50% will result the equivalent of 50% auto.
* A combination of lengths, percentages, and auto, such as 80px auto (80px wide, automatic height to maintain the image’s original ratio)
* contain, which maintains the background image’s original ratio and makes it as large as possible whilst fitting entirely within the box’s background area.
* cover, which maintains the background image’s original ratio and makes it large enough to fill the entire background area, which may result in cropping of either the height or width.

## Background origin

background-origin is the remarkably dull kid of the bunch. Not unintelligent, just dull. The kid that the other kids point and laugh at. The property defines where the background area of a box actually starts. If you think of the box model, when you set a background it should, by default, start from the upper-left corner of the padding box. So if you had…

#burrito {

width: 400px;

height: 200px;

border: 10px solid rgba(0,255,0,.5);

padding: 20px;

background: url(chilli.png) 0 0 no-repeat;

}

…the background image should appear in the top left corner, in the padding box, immediately after the inner edges of a green border. You can change this behavior, however, with background-origin. Its self-descriptive values can be padding-box (default, as described above), border-box, and content-box.

So adding background-origin: border-box to the previous example will cause the origin of the background image to be tucked up inside the border.

# Transformations

A woeful mega-budget Michael Bay movie about CSS?! Nay, the powerful manipulation of the shape of a box and its contents using the transform property.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

By default, CSS boxes, those visual representations of HTML elements, are so square. Rectangular, at least; level, with four straight sides and four boring right angles. But with the use of transform you can stretch and mold those boxes into all manner of shapes.

This page will only mention the transform and transform-origin properties but, in practice, you will probably want to duplicate these with -webkit-transform and -webkit-transform-origin to achieve the same results in the likes of Safari and Chrome as well as -ms-transform and -ms-transform-origin for Internet Explorer 9, which is the earliest version of IE that will support transforms.

Manipulating a box over two dimensions, transform can be used to rotate, skew, scale and translate a box and its contents.

## Rotating

The following would result in a square orange box, all of its contents - text, images, whatever - lying flat:

.note {

width: 300px;

height: 300px;

background: hsl(36,100%,50%);

}

But you can rotate it by adding a further declaration:

transform: rotate(-10deg);

This will tip the box and, importantly, everything in it, ten degrees anti-clockwise.

## Skewing

Skewing allows you to tip the horizontal and vertical so the following…

transform: skew(20deg,10deg);

…will tip over the x-axis by 20 degrees on the y-axis by 10 degrees.

You can also specify one angle, such as skew(20deg), which is the equivalent of skew(20deg,0).

## Scaling

Obviously, you can change [width](http://www.htmldog.com/reference/cssproperties/width/) and [height](http://www.htmldog.com/reference/cssproperties/height/) properties on a box, but that won’t affect the size of anything inside it. Scaling, however, will multiply not only the width and height, but the size of everything contained in the box as well:

transform: scale(2);

This will multiply the size by two. You can use any positive number, including a value less than “1”, such as “0.5”, if you want to use a shrink ray.

You can also scale the horizontal and vertical dimensions separately:

transform: scale(1,2);

This will leave the horizontal as is (because it’s a scale of 1) and multiply the vertical by two.

## Translating

You can shift a box horizontally and vertically using transform: translate:

transform: translate(100px,200px);

Similar to position: relative; left: 100px; top: 200px;, this will move a box 100 pixels to the right and 200 pixels down.

As well as the values mentioned, if you want to target an individual axis, you can also use skewX, skewY, scaleX, scaleY, translateX, and translateY.

## Combining transformations

Want to rotate and scale at the same time? You crazy kid. You can do this by simply separating values with spaces, such as:

transform: rotate(-10deg) scale(2);

The order of the values is important - the latter will be executed before the former. In the previous example, the box will be scaled and then rotated. It is, therefore, different to transform: scale(2) rotate(-10deg);, which will be rotated and then scaled.

Alternatively, you could use the matrix function. matrix does the whole lot - rotating, skewing, scaling, and translating. It takes six values:

transform: matrix(2,-0.35,0.35,2,0,0);

These values aren’t entirely straightforward and involve maths (or just one math if you’re of the American persuasion) that, if you really want to tackle (there are benefits not only in brevity but also in precision), it may be worth giving [the specs](http://www.w3.org/TR/css3-transforms/#mathematical-description) a gander.

## Origin

There’s one important aspect missing. If you are transforming a box, there is a further parameter involved: the origin of the transformation. If you are rotating, for example, it will, by default, turn on the point that is the center of the box; if you had a piece of card, stuck a pin right through the middle of it, and then stuck that to your forehead (don’t do this), the card would spin from the middle. You can change where in the card the pin is stuck, however:

transform-origin: 0 0;

This will tell the box to transform (rotate, in the previous example) from the top left, the first “0” being horizontal, the second being vertical, like all other x-y, and you can use the usual top, right, bottom, left, length and percentage values, including negatives.

And all of that’s just with two measly dimensions. transform is a leviathan with even greater power that can also be used for three-dimensional magic. On the most basic level, you can use rotateX and rotateY, which will rotate “towards” or “away” from you on the x- and y-axis, and there are the likes of translate3d, scale3d, and the intimidating matrix3d, all of which have even greater browser difficulties than their 2D counterparts.

**Gradients**

Images showing a smooth dissolve from one color to another are plastered all over the web. However, CSS 3 allows you to place them in your designs without having to create an actual image file.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

There is no special property for this; you simply use the [background](http://www.htmldog.com/reference/cssproperties/background/) or [background-image](http://www.htmldog.com/reference/cssproperties/background-image/) property and define your gradient in its value. You can create both *linear* and *radial* gradients this way.

**Linear gradients**

For an even spread of two colors, fading from one at the top to another at the bottom, a declaration can simply be something like:

background: linear-gradient(orange, red);

To manipulate the angle of the fading, you slot in “to” and the destination you want the transition to head to. You can head to one side:

background: linear-gradient(to right, orange, red);

Or one corner:

background: linear-gradient(to bottom right, orange, red);

Or any angle that tickles your fancy:

background: linear-gradient(20deg, orange, red);

Note that the “to” is excluded with angles because there isn’t an explicit destination.

And why stop at two colors? Specify as many as you dare:

background: linear-gradient(hsl(0,100%,50%),hsl(60,100%,50%),hsl(120,100%,50%),hsl(180,100%,50%),hsl(240,100%,50%),hsl(300,100%,50%));

To get gradients to work on as many browsers as possible, you will probably also want to use -webkit-linear-gradient to cover Safari and older versions of Chrome. *The values of these must also exclude the “to” part*, if used.

CSS gradients won’t play at all with IE 9 and below, but you can still make them, and any other incapable browser, use the traditional method of a good old fashioned image by specifying that first, as a fallback (proceeding declarations will just be ignored).

So, all-in, your gradients might look something like this:

body {

background: #666 url(fade.png) 0 0 repeat-y;

background: -webkit-linear-gradient(right, #000, #666);

background: linear-gradient(to right, #000, #666);

}

**Radial gradients**

Radial gradients, with one color starting from a central point and fading to another color, use a similar syntax:

background: radial-gradient(yellow, green);

You can also specify the shape of the fade. By default it is an ellipse, spreading to fill the background box, but you can force it to be circular, regardless of the shape of the box:

background: radial-gradient(circle, yellow, green);

Using “closest-side”, “closest-corner”, “farthest-side” and “farthest-corner” you can also specify if the gradient is contained by the sides or corners of the box nearest to or furthest away from the origin:

background: radial-gradient(circle closest-side, yellow, green);

And if you wanted to place the origin of the gradient somewhere specific, you can also use “at”:

background: radial-gradient(at top left, yellow, green);

More compatibility notes: additional -webkit-radial-gradient usage is wise. The order of the parameters needs to be changed here, and, like “to”, “at” is excluded. So:

background: radial-gradient(circle closest-side at 100px 200px , black, white);

Would be accompanied by:

background: -webkit-radial-gradient(100px 200px, circle closest-side, black, white);

**Color stops**

If you don’t want a uniform blend across your gradient, you can specify exactly where in the gradient each color kicks in, straight after each color, starting at “0” and ending at “100%” (although lengths can also be used).

So, just to make it clear before tinkering:

* linear-gradient(black 0, white 100%) is the equivalent of linear-gradient(black, white)
* radial-gradient(#06c 0, #fc0 50%, #039 100%) is the same as radial-gradient(#06c, #fc0, #039)
* linear-gradient(red 0%, green 33.3%, blue 66.7%, black 100%) will have the same result as linear-gradient(red, green, blue, black)

That’s because, when the positions are stated in these examples, they evenly space out the colors, which is the default when no color stops are explicitly defined.

So, to get on with that tinkering, you can pull and stretch away with those stops:

background: linear-gradient(135deg, hsl(36,100%,50%) 10%, hsl(72,100%,50%) 60%, white 90%);

**Repeating gradients**

A single gradient will fill a box with the previous methods but you can use “repeating-linear-gradient” and “repeating-linear-gradient” to build on the color stops and, well, repeat the gradient.

For basic bars of black-and-white bars, for example:

background: repeating-linear-gradient(white, black 10px, white 20px);

Or something a bit more solid:

background: repeating-radial-gradient(#8b2, #8b2 10px, #e90 10px, #e90 20px);

# Media Queries

@media at-rules, used to target styles at specific media, such as screen or print, [have already been covered](http://www.htmldog.com/guides/css/advanced/atrules/). But this can be pushed to an even greater level of sophistication, enabling you to specify different design choices depending on screen size. A page can then be optimized and laid out completely differently for mobile phones, tablets, and varying browser window sizes.

[](http://www.siteground.com/web-hosting.htm?afcode=4960306b51b3ed1ddecbf83b3076ee11)

To recap, if we want to apply some CSS solely to screen-based media, for example, one option would be to slot something like the following in at the bottom of a stylesheet:

@media screen {

body { font: 12px arial, sans-serif }

#nav { display: block }

}

## Browser-size specific CSS

To extend the previous example, not only can screen-based media be targeted, screen-based media of a certain size can be targeted as well:

*@media screen and* (*max-width:* 1000px) {

#content { width: 100% }

}

This is telling a browser to apply a block of CSS when its viewport is 1000 pixels wide or less. You could use this to do something as simple as making a content area or navigation narrower or you could completely re-arrange a page layout (like stacking page sections on top of one another instead of displaying them in columns).

You can apply more than one @media rule, so you could have a number of different designs that are browser size dependent:

@media screen and (max-width: 1000px) {

#content { width: 100% }

}

@media screen and (max-width: 800px) {

#nav { float: none }

}

@media screen and (max-width: 600px) {

#content aside {

float: none;

display: block;

}

}

Note that if, for example, a layout area was 600 pixels wide or less, all three of these would be applied (because it would be less than or equal to 1000px, 800px, and 600px).

As well as using a general max-width on the main content area (the article elements), this site also has a number of size-dependent CSS rules. If you’re able to do so, try resizing your browser to see the changes take effect.

You could also use “min-width” in place of “max-width” if you want to do things the other way around and apply CSS to sizes equal to or over a certain length.

## Orientation-specific CSS

If you have a hankering for applying CSS depending on the orientation of the browser, fill your boots with something like the following:

*@media screen and* (*orientation:* landscape) {

#nav { float: left }

}

*@media screen and* (*orientation:* portrait) {

#nav { float: none }

}

This could be especially useful with mobile devices…

## Device-specific CSS

We’re not talking different CSS for each and every brand and model of laptop, phone, and donkey - that would be sinful - but we can, with a relatively clear conscience (as long as we’re sensible), specify the likes of the device’s screen dimensions and its pixel ratio.

A “handheld” media type does exist and it could be used as @media handheld… but it isn’t widely supported and the distinction of what is “handheld” has become blurred due to the proliferation of all manner of devices with all manner of screen sizes.

### Width and height

device-width, device-height, min-device-width, max-device-width, min-device-height and max-device-height can be used to target the computed resolution of a device:

@media screen and (min-device-height: 768px) and (max-device-width: 1024px) {

/\* You can apply numerous conditions separated by "and" \*/

}

### Pixel ratio

It’s important to keep in mind that a CSS pixel need not be the same as a physical pixel. While a screen may physically be 720 pixels wide, a browser may actually apply CSS assuming that it is 480 pixels wide, for example. This is so that a standard designed web page will more likely fit on the screen. In this example, there is a pixel ratio of 1.5:1: There are 1½ physical pixels to every CSS pixel. A bog-standard desktop monitor will have a pixel ratio of 1:1: One CSS pixel to every physical pixel.

If you want to apply styles only to these devices, you can use something like:

@media (device-pixel-ratio: 2) {

body { background: url(twiceasbig.png) }

}

This would apply to the likes of the iPhone (4 and above), with a pixel ratio of 2:1. You can also use min-device-pixel-ratio and max-device-pixel-ratio.

Talking of iPhones, you know the deal: also use -webkit-device-pixel-ratio, etc, etc…

You might also want to fiddle with a device’s viewport to get it to play how you want. Leaping back over to HTML, the following [meta](http://www.htmldog.com/reference/htmltags/meta/) element will force a device to render a page at the width of the viewport (rather than attempting to show a wider-width design and zooming out, which it will do by default if the page can be wider than the viewport’s width) and also prevent the user from zooming in and out:

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">

The benefit of this is that you can control exactly what is applied to what physical screen size. And although it’s painful to remove user controls, if the design is delightfully swell and made just for that diddy little screen, there shouldn’t be a need to zoom.

The HTML Dog web site takes this approach: instead of a small device attempting to render a bigger, fatter web page by shrinking it down, the CSS turns it into a single-column design made specifically for such a device.

### Others

You can also apply styles depending on a device’s resolution:

@media screen and (resolution: 326dpi) { /\* \*/ }

@media screen and (min-resolution: 96dpi) { /\* \*/ }

Or on its aspect ratio:

@media screen and (device-aspect-ratio: 16/9) { /\* \*/ }

THAT’S ALL FOLKS ☺