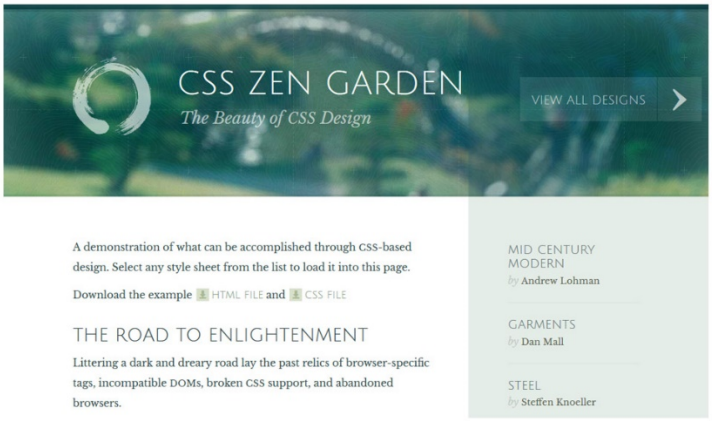
INTRODUCTION

CSS stands for Cascading Style Sheets, and is used to define a web page's colors, typography, and layout. The website csszengarden.com shows how HTML markup can be rendered with completely different looks using CSS rules. The following HTML snippet is from the CSS Zen Garden web site.Two different designs from the site are shown below, along with some of the CSS code used in each design.

SAMPLE HTML:

```
<section class="intro" id="zen-intro">
	<header role="banner">
	<h1>CSS Zen Garden</h1>
	<h2>The Beauty of <abbr title="Cascading Style Sheets">CSS</abbr> Design</h2>
</header>
```

SAMPLE CSS #1:

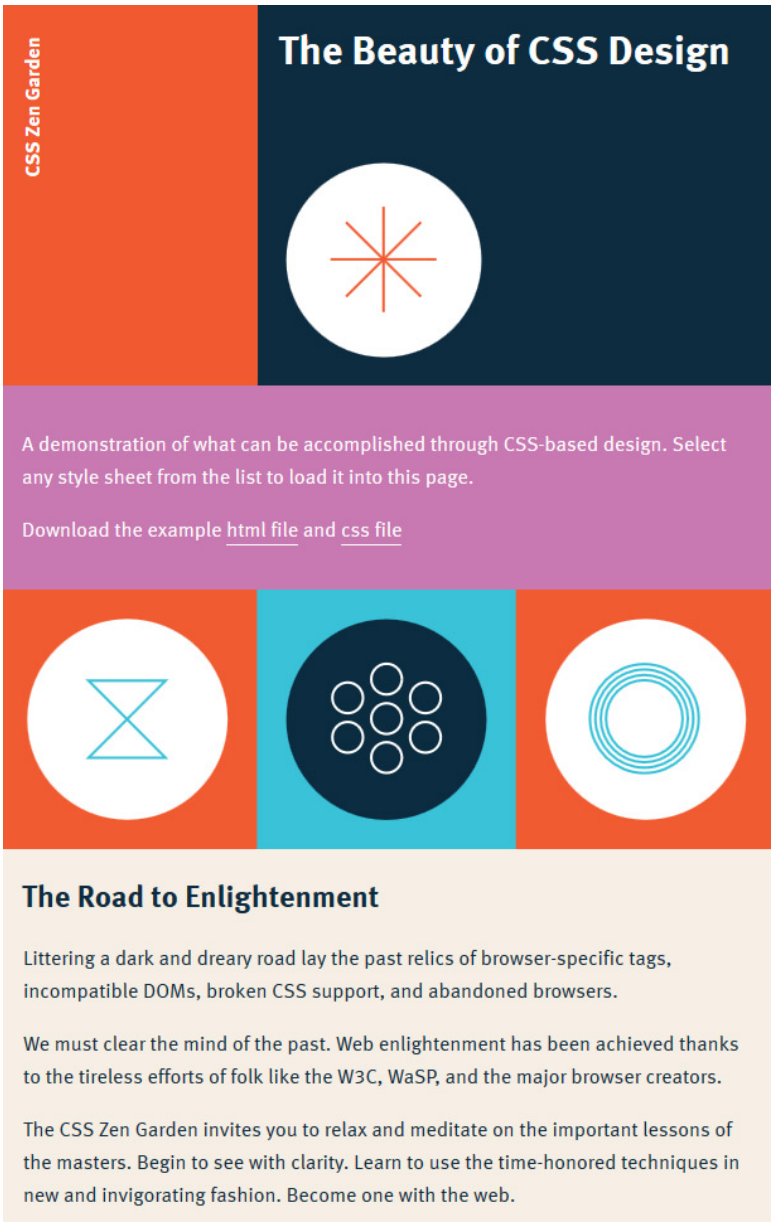



```
header {
  height: 230px;
  padding: 20px 0 70px 0;

  background: #2d6360 50% 0
url(huntington.jpg) no-repeat; /*
old IE fallback */

  background-attachment: fixed,
fixed, fixed, scroll;
  background-image:
url(contours.png), url(noise.png),
url(gridlines.png),
url(huntington.jpg);
  background-position: 0 0, 0 0, -
5px -25px, 0 50%;
  background-repeat: repeat, repeat,
repeat, no-repeat;
  background-size: auto, auto, auto,
cover;
  text-align: center;
}
```

SAMPLE CSS #2:




```
.intro {
  overflow: hidden;
}

.intro header {
  position: relative;
  background: #c879b2;
  overflow: hidden;
  width: 40%;
  height: 50em;
  float: left;
  -webkit-transition: all 0.4s ease;
  -moz-transition: all 0.4s ease;
  -o-transition: all 0.4s ease;
  -ms-transition: all 0.4s ease;
  transition: all 0.4s ease;
}

  @media (max-width: 70em) {
    .intro header {
      background: #f15a30;
      width: 100%;
      height: 27em;
    }
  }
  @media (max-width: 53em) {
    .intro header {
      height: 20em;
    }
  }

.intro header h1 {
  position: absolute;
  color: #ffffff;
  font-size: 100%;
  line-height: 2em;
  top: 4.2em;
  left: -3em;
  width: auto;
  margin: 0;
  padding: 1em;
  -webkit-transform: rotate(-90deg);
  -moz-transform: rotate(-90deg);
  -ms-transform: rotate(-90deg);
  -o-transform: rotate(-90deg);
  filter:
progid:DXImageTransform.Microsoft.BasicI
mage(rotation=3);
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
```

CSS Syntax

A CSS rule consists of a selector followed by a declaration block.

A **selector** declares which HTML elements should be styled.

A **declaration block** defines how selected elements should be styled. Declaration blocks are enclosed in curly braces and contain a list of **properties** and **values**. Place colons between the properties and values. Add semicolons at the end of each line, though they are optional for the last declaration in each list.

A **comment** begins with the /* characters and ends with */

```
/* comment */

selector {
	property: value;
	property2: value2;
}
```

Way to Insert CSS

1. Inline styles are added directly to HTML markup using the style attribute. They do not contain selectors; only declaration blocks.

```
<p style="color:red; font-weight:bold">This is a bold red paragraph.</p>
```

The use of Inline Styles is not recommended. Scattering style information in HTML files violates the principle of separation of concerns.

2. Internal styles are defined using a style element in the header.

```
<head>
	...
	<style>
		p {
			color: red;
			font-weight:bold;
		}
	</style>
</head>
```

Internal Styles are not recommended for multiple page sites. They would have to be put on every page.

3. External style sheets are separate files saved with the .css extension. This is the best place to put style rules. Styles defined here are consistent across all pages where they are included. Changing styles for the entire site is easy because they only must be edited in one place.

Use the **<link>** element to connect a stylesheet to a web page.

style.css

```
p {
	color: red;
	font-weight:bold;
}
```

index.html

```
<head>
	...
	<link rel="stylesheet" href="file.css">
</head>
```

CSS SELECTORS

Selectors define patterns that identify a single element or a group of elements in a web page document.

Selector Writing the name of an element is the simplest pattern. Choose an element, and the declaration will apply to all instances of that element in the web page document. In this example, the declaration will apply to all paragraphs.

```
p { ... }
```

Multiple Selectors Writing the name of multiple elements separated by commas means the declaration will apply to all the elements.

In this example, the declaration will apply to both headers and footers.

```
header, footer { ... }
```

Descendant Combinators select all elements that are nested inside the first specified element. Separate the selectors with spaces.

In this example, the declaration will apply to only paragraphs located inside an article element.

```
article p { ... }
```

Child Combinators select only elements that are a direct child of the first selector. Separate them with a right-angle bracket '>'.

The following combinator will only select the first paragraph in each article. The second paragraph in this example is a grandchild of the article element, so it would not be selected.

```
article > p { ... }

...

<article>
	<p>This 'child' paragraph is selected.</p>
	<div> <p>This 'grandchild' paragraph is NOT selected.</div> </p>
</article>
```

ID selectors are used to specify a style for a single, unique element marked with an **id** attribute in the HTML document. It is identified using the # symbol. **An id can appear only once in an html page.**

```
#warning
{
	color:red;
}

...

<p id="warning">
```

Note: The ID selector can be limited to a specific element, by appending it after an element name.
p#warning would select only paragraphs with the warning id.

Class Selectors are used to specify a style for a group of elements marked with a **class** attribute in the HTML document. It is defined with a period symbol. **A class may be reused throughout an html page.**

```
.highlight
{
	color:red;
}

...

<p class="warning">
```

Note: The class selector can be limited to a specific element, by appending it after an element name.
p.warning would select only paragraphs with the warning class.

Pseduo-classes are used to select a specific state. It is defined with the colon symbol. In the following example, only the first element inside each of the document's articles will be colored red.

```
article:first-child
{
	color:red;
}

...

<article>
	<p>This 'first child' paragraph is selected.</p>
	<p>This 'second child' paragraph is NOT selected.</p>
	<p>This 'third child' paragraph is also NOT selected.</p>
</article>
```

The four links states are styled using pseudo-classes:

- a:link – an unvisited link
- a:visited – a link to content the user has previously visited
- a:hover – the state when the user's mouse is over the link
- a:active – the state when the user's mouse clicks down on a link

```
a:link { color:#FF0000; }
a:visited { color:#00FF00; }
a:hover { color:#FF00FF; }
a:active { color:#0000FF; }
```

Combinators can be used to apply different link styles within a page, like links within a nav element:

```
nav a:link { color:#FF0000; }
nav a:visited { color:#00FF00; }
nav a:hover { color:#FF00FF; }
nav a:active { color:#0000FF; }
```

Or links within an element modified with a 'sidebar' class:

```
a.sidebar:link { color:#FF0000; }
a.sidebar:visited { color:#00FF00; }
a.sidebar:hover { color:#FF00FF; }
a.sidebar:active { color:#0000FF; }
```

Attribute selectors are used to select elements that have an attribute with a specified value.

```
input[type="button"] {
	color:red;
}
```

THE CASCADE

Sometimes, an element can receive conflicting style declarations. This happens when multiple CSS rules select the same elements. There are a set of rules that determine which declarations will be applied to each element in the document.

① — CSS rules are defined in several places, called the **cascade**. The browser checks these sources in order. The rules 'cascade' from one source to the next, until they are overridden by another rule which has greater priority.

1. The browser will first apply a default user agent style sheet for all elements. User agent styles have the lowest priority in the cascade.

2. Users can edit the default user agent style sheet, but in practice they rarely change default styles. User edits override the default browser styles.

3. Rules written by web site authors, who create the HTML and CSS pages for the site, are the next source in the cascade. They override default browser and user generated styles.

4. Author-created CSS animations, using @keyframe rules, are the next source. They override all other rules written by the site author.

5. Finally, declarations with the !important statement are given higher priority than normal declarations. Web site authors shouldn't use this, as it is a sign of poorly organized CSS rules. The !important statement can appear in the user agent style sheet for accessibility; for example a vision impaired can force a site to use a larger font size.

② — When two rules are "tied" because they appear in sources with the same priority, selectors with a higher **specificity** are given preference. Selectors that target elements have the lowest specificity, and selectors that target ids have the highest specificity. ID selectors always have priority, even if other rules appear later.

1. *Element selectors* (h1) have a specificity of 0 (lowest priority).

2. The following – *class selectors* (.example), *attribute selectors* ([type="button"]), and *pseudo selectors* (:hover) – all have a specificity of 1.

3. *ID selectors* (#example) have a specificity of 2 (highest priority).

```
<p class="class-selector" id="id-selector"> …          // id selector has precedence
```

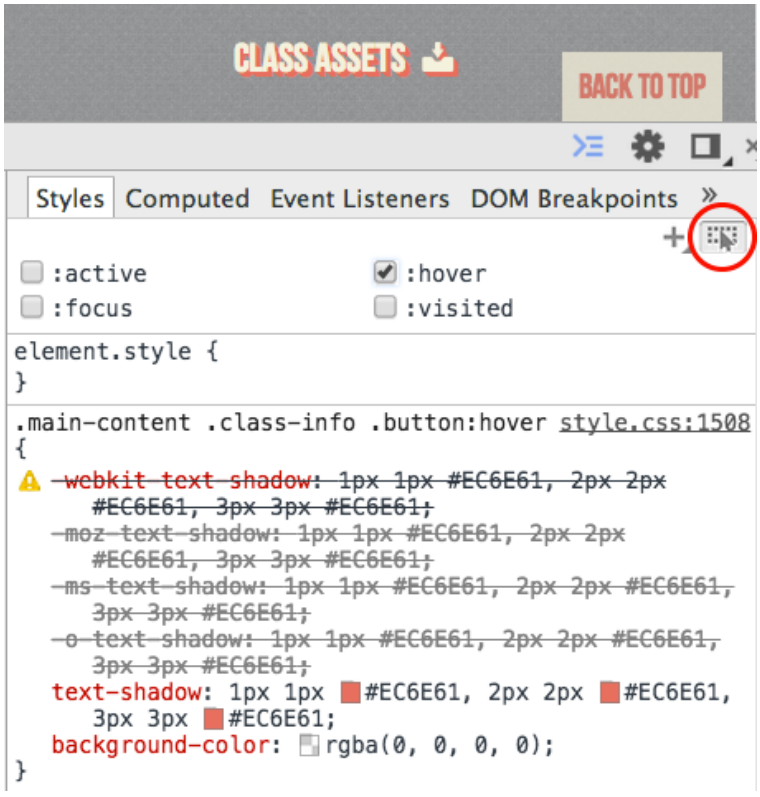
③ — When two rules are still “tied”, count the number of times each specificity level is targeted. If a selector targets more than one component with the same specificity level, it has precedence over a selector that targets fewer components. First try breaking the tie by counting the number of IDs targeted. Work your way down the specificity levels until the tie is broken. In the following example, the *first* rule with the sidebar class selector has precedence because it targets a greater number of classes, attributes, and pseudo selectors:

```
a:hover {...          // id count = 0
                      // class + attribute + pseudo selector count = 1
                      // element count = 1

a.sidebar:hover {...  // id count = 0
                      // class + attribute + pseudo selector count = 2
                      // element count = 1
```

④ — If two CSS rules are still tied, the one which appears later is given priority. To make CSS code easier to maintain, put rules with a higher specificity below ones with a lower specificity.

Note: Elements inherit styles from their parents. However, rules that directly target elements will override inherited styles.



richfinelli.com/troubleshooting-html-and-css/

The Chrome Inspector tool shows which CSS rules were applied and where they came from. Greyed out rules were overridden by another rule in the cascade.

CSS Color Values

Two ways to specify CSS colors are named color keywords and hex codes. A list of CSS **color keywords**, and other ways to define colors, can be found online.

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

```
p { color: red; }
```

In **hex codes**, the first two digits specifies the amount of red, the second two digits specifies the amount of green, and the last two digits specifies the amount of blue. Use the pound (hashtag) symbol before hexadecimal codes. The codes can be found in Photoshop's color picker and on sites like Adobe Color and Paletton.

```
p { color: #FF0000; }
```

Text colors are defined using the color property:

```
body { color: #333333; }
```

Background colors are defined using the background-color property:

```
body { background-color: #FDFDFD; }
```

Border colors can also be chosen:

```
.sidebar { border-color: #333333; }
```

CSS Size Values

There are several options for setting sizes in CSS.

Images, div heights and widths, font sizes, margins, padding, and other layout elements can be defined using px (pixel values), pt (point values), % (percentages), em (ems), and a few other specialized unit. Do not put a space between the value and the unit:

```
width: 50em;
```

A debate about which units should be used has lasted for many years and created much confusion. The following notes show the simplest way to use the units as intended to create accessible, easily maintainable designs.

Point units (pt) were created for printed materials. It is a fixed-size measurement where one point is equal to 1/72 of an inch on printed paper. Other absolute units include cm, mm, and in (centimeters, millimeters, and inches). **There really isn't any reason to use point sizes in screen based interfaces.** Using em units also look fine in print without any additional work.

Pixel units (px) are another absolute unit size. They were meant to be used for the screen. A logical pixel is often defined as 1/96 of an inch. The actual number of hardware pixels may vary. For example, retina displays from Apple have twice the pixel density per inch. Most current browsers and operating systems will automatically translate the logical pixels defined in CSS rules to the number of hardware pixels necessary to render the design at a reasonable size.

Pixels units have retained popularity among many designers because it is easier to think in terms of fixed sizes than fluid, responsive sizes.

However, pixel based designs can be harder to maintain. Because they are absolute units, changing the overall size of a design requires calculating new pixel values for every single design element in the page.

Another problem with pixel based designs is reduced accessibility. If a font size is increased by the user agent for readability by a visually impaired person or for clarity on a projected screen, text could spill out of fixed size containers. Other use cases and future devices that allow fonts to be scaled at different sizes than the rest of the design could also break the layout.

Pixel units can be used for border widths, which usually shouldn't scale with the rest of the design.

```
border-width: 1px;
```

Pixel units can also be created for designs with absolute positioning and sizing where the content doesn't include ANY text.

Percent units (%) are relative to the size of the container. That could be the width of the screen, or the width of a smaller div somewhere within the screen layout.

Percent units can be used to easily generate multiple fluid variable-width columns in a layout. For example, in a two-column layout, the width of each column would be 50%. Browsers need to automatically expand columns vertically downward, so all content will fit, so it is important to remember this: *never specify heights in CSS!*

```
.sidebar {
	width: 25%;
}
```

Percent units are the best solution for creating responsive images. In desktop layouts, it is ok for images to take up up a larger percentage of screen real estate than on smaller mobile devices. A lot of text could fit underneath a hero image that fills the top 50% of a browser window on a desktop. Very little could fit underneath that on a mobile phone.

Also, containers in fluid layouts could have constantly changing sizes. It would be nice to have images automatically rescale to fit their containers.

To write the CSS for a responsive image, first define the width of a container using percentage or em units. Then specify that the width of the image should be 100%, and it will fill the container.

```
.image {
	width: 25em;
}

img {
	width: 100%;
}

...

<div class="image">
	<img src=...>
</div>
```

Make sure your images have a large enough resolution, so they don't get scaled up and pixelated. On the other hand, don't make the images so large that a lot of bandwidth is wasted on resolutions that are much higher than what a large screen can display.

Em units (em) are sizes relative to the height of the current font. The default size of one em is usually 16px, which is a good size for body text on most devices. Headings and other larger and smaller elements should be scaled using a typographic ratio that is pleasing to the eye. Defining your own font sizes usually isn't necessary, because default browser styles already include a good typographic scale.

If you do define your own font sizes, note that nested em values are compounded and multiplied. If a sidebar's font size is set to .75em, and a header's font size is set to 2em, then a header inside the sidebar would be sized at 1.5em (.75 x 2 = 1.5). Some developers find ems difficult to work with, because there is no easy way to override the compounding effect or ignore a parent's font-size setting. However, because ems are relative units, trying to use them to define specific sizes doesn't make sense. The em unit should be used in a relative way, as demonstrated in the following examples:

You can use ems, pixels, or percentages to define the base text size for the page. Making the default text size smaller than 100%, 1 em, or 16 px is not recommended to maintain good readability.

```
body {
	font-size: 1.2em;
}
```

If you do wish to specify font sizes for specific elements, **add em font sizes to the element that directly wraps the text**. In the following example, note that the font size of the main menu links targets the <a> element wrapping the text, not the complete <nav> container.

```
nav a {
	font-size: 1.5em;
}
```

When adding contrasting sizes like large headers or small fine print, choose a good typographic scale. The scales calculated at http://type-scale.com/ will create balanced sizing with good harmony.

Add ems to containers like divs and other sectioning elements only when the entire section and all its contents should be scaled. For example, if you want all the contents in a sidebar to be smaller than the surrounding body text, the relative em unit is a great way to scale everything down with one line of code. That works because all measurements in the aside's child elements will be calculated relative to the font size of the aside.

```
aside {
	font-size: 1.2em;
}
```

Because whitespace should scale with the font size, **use ems to specify margins, and padding**.

```
article {
	margin: 4em;
	padding: 2em;
}
```

Containers like divs, sections, and columns that have a fixed width should also use em units. Columns, divs, and other sections could be defined as a percentage of the total screen width, but they could also have a fixed width. When a fixed width is desired anywhere text could be included, use em units (not pixels) so the size of the container will scale to fit the font size of the text.

```
aside {
	width: 50em;
}
```

Use em units to define media query breakpoints. Media queries are used to present different layouts on devices with different widths. Pixel width breakpoints won't work as intended if the user agent overrides the base font size.

```
@media only screen and (max-width: 37.5em) { ... }
```

Line height is a special case. Use a unitless value instead of ems. That way line spacing will be based on its own font size instead of the container's.

Extra line spacing is needed for web typography because screen text is harder to read. A line height of 1.4 is a typical value for an average font family and column width. This will automatically scale with the font size. So, if a larger font is used somewhere, the line-height will increase by an appropriate amount.

```
body {
	line-height: 1.4;
}
```

CSS TYPOGRAPHY

font-family usually requires uploading the font files to the user, or using web safe font stacks, because not all users have all fonts installed.

Family names with spaces should be enclosed in double quotes.

In a **font stack**, If the first one listed is available on the user's system, it is used. Otherwise, the browser attempts to fall back to the other fonts on the list. A good list includes fonts that are commonly installed on Windows, Mac, and Linux systems. End each font stack with one of the generic default font families: *serif, sans-serif, monospace, fantasy, cursive*. Providing a generic backup alternative for browsers that don't support a feature is called **graceful degradation**.

```
p { font-family: "Times New Roman", Times, serif; }
```

The following chart includes some popular font stacks. (more at http://www.cssfontstack.com/

- Blue color = Mac installed font, Purple color = Windows font, Black color = cross-platform
- The greed underlined fonts were cross-platform fonts designed for web page viewing.
- The names in grey at the end of each list are the generic family of each font, used by default if the user doesn't have any of the other fonts installed.

Web Safe Font for Body Text
Verdana, sans-serif
Georgia, serif
Arial, sans-serif
"Times New Roman", Times, serif
Trebuchet MS, sans-serif
Tahoma, Geneva, sans-serif
Palatino, "Palatino Linotype", serif
"Courier New", monospace

Web Safe Fonts for Headlines and Display
"Helvetica Neue", Helvetica, Calibri, Arial, sans-serif
Arial Black, Gadget, "Arial Bold", sans-serif
Impact, Charcoal, sans-serif

@font-face is a CSS rule that allows fonts to be embedded in web sites. If you buy fonts licensed for the web, you can put the files in a font/ folder, and use @font-face to specify the family name and file path.

Google Fonts and **Adobe Typekit** are online libraries containing fonts you can use in your site. You can download the fonts to work on your design mockups, but you don't need to upload them to your web hosting space because the files are served from online content delivery networks.

To use a Google font, follow these directions:

1. Click the red plus button that says, 'Select this Font.' Repeat for all your fonts

2. Click the 'Families Selected' link in the black bar at the bottom right corner of the page. If you wish to download the font to install it on your computer, press the red download link icon in the top right corner of the popup window.

3. Click the 'Customize' button in the popup window. You should select italic and bold weights for your body copy fonts. You usually don't need to do this for your headline and display fonts.

4. Add the font embed to your code. The 'Standard' code should be placed in the <head> section on all your pages. Alternatively, you can use the '@import' code, and copy just the @import line to the top of your CSS file. Google also shows the font stack that should be used in CSS files.

An advanced JavaScript font loading method exists to avoid flashes of invisible or unstyled text while the web fonts load: https://github.com/typekit/webfontloader, https://css-tricks.com/loading-web-fonts-with-the-web-font-loader/

Other Typography Properties

font-size (css size)

font-weight bold, normal, inherit

font-style italic, normal, oblique, inherit

color (css color)

text-align left, center, right, justified (applies to the element's contents)

line-height (css size or number multiplied by current font size)

text-indent (css size)

text-transform uppercase, lowercase, capitalize

text-decoration underline, none (remove underlines from hyperlinks)

STYLESHEET BOILERPLATE

Below are declarations commonly used in CSS stylesheets to define base color and typography stykes. Put them in a file like global.css, and link to the file from your html page.

```
html {
	background-color: #FDFDFD;
	font-size: 100%;
}

body {
	font-family: "Palatino Linotype", "Book Antiqua", Palatino, Georgia, serif;
	line-height: 1.4;
	color: #333333
}

h1, h2 {
	font-family: “Helvetica Neue”, Helvetica, Arial, sans-serif;
}

a:link {
	color:#FF0000;
}

a:visited {
	color:#00FF00;
}

a:hover {
	color:#FF00FF;
}

a:active {
	color:#0000FF;
}
```

RESET CSS

More advanced developers use a reset.css file found on the web. The file contains CSS rules to remove all margins, padding, and default browser styles. It is always the first CSS file included in a web page. A global CSS file or section is then used to override the reset styles.

Reset CSS files are popular because different browsers don’t use the same default user agent style sheets. Writing new default values for all key typographic and layout property values is a lot of extra work for the CSS author, but the extra effort ensures consistency across all browsers.

User agent style sheets are similar enough in different browsers that beginning web developers can skip this step.