Introduction to CSS

CSS (**C**ascading **S**tyle **S**heets) is used to style your webpages. Whereas HTML is purely for structuring your page **content**, CSS is used to **style** your page. In this way, you can separate your content from your presentation (styling).

When talking about styling, there are different sources of styling: the browser defaults, the user (usually there's a ***user.css*** or something similar for your browser), and the author of the document (your webpage's styles). The browser *cascades* through the different sources to render the final styling, hence **Cascading Style Sheets**.

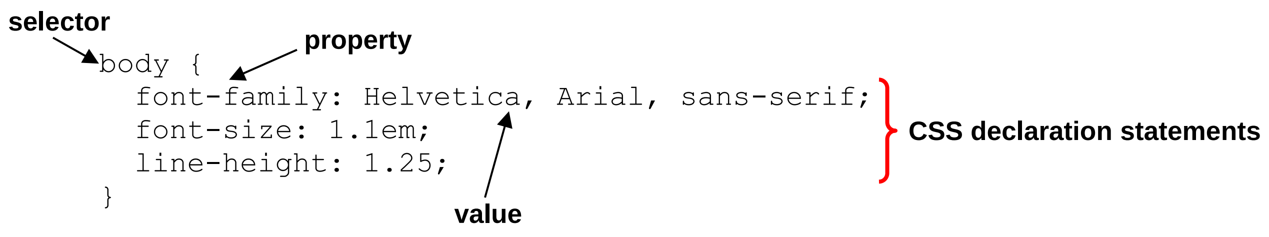
Beyond this cascade, the browser gives a weight to your CSS rules to determine which styles are applied. This weighting refers to the specificity of your CSS rules but this will be discussed later in this document.

Syntax

Typical CSS for an element looks like the code below:

```
body {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 1.1em;
  line-height: 1.25;
}
```

To understand the text talking about CSS, you should understand the terminology. The syntax for a piece of CSS is made up of the following parts:



```
body {
  font-family: Helvetica, Arial, sans-serif;
  font-size: 1.1em;
  line-height: 1.25;
}
```

The above code is an example of a CSS declaration block. The entire block, including the selector and declaration statements, is called a rule.

The selector "selects" the element, class, or ID that you want to style. For that selector, you then add your CSS declarations within curly brackets.

A declaration statement contains a *property* followed by a colon then a *value* for that property. The statement is ended with a semicolon. Don't forget the semicolon or else the code compiler won't know that the statement was completed. (Technically, you don't need the semicolon on the last declaration statement in a block, but it is best practice to use one anyway so that you don't forget to add one if you add additional declaration statements afterwards.)

These parts are what make up your styling.

CSS Comments

The following is the syntax for CSS comments:

```
/* This is a comment. */
/* Comments can also span
   multiple lines. */
```

CSS Selectors

Your CSS selector can specify an element, class, or ID.

The selector for an element is just the element name. For example, the selector for the `<body>` element is:

```
body {
  ...
}
```

Note that there are no angle brackets, just the element name followed by the declaration block.

The selector for a class name is preceded by a dot (.) with no space in between the dot and the class name. For example, to specify the selector for a class named "important-text", use:

```
.important-text {
  ...
}
```

You can target a particular element type with a certain class with:

```
p.important-text {
  ...
}
```

The selector for an ID (e.g. an element with the ID "left-sidebar") is the pound sign (#) followed by the ID name. There is no space between the "#" and the ID name. Using the example of an element

```
<aside id="left-sidebar">...</aside>
```

the selector would be as follows:

```
#left-sidebar {
  ...
}
```

Now you can see the usefulness of classes and IDs. Using classes and IDs give a lot more flexibility when choosing elements to style. (You do not need to specify the element when using an ID because IDs are already unique.)

You can also apply the same rule to multiple selectors by separating the selectors with a comma:

```
button, a.button {
   /* apply this rule (styles in curly brackets) to button
      elements and links with the class "button" */
}
```

There is a special selector called the universal selector. This selector is the star symbol (*). This means that the CSS declaration block applies to **everything**. Sometimes you may want to use this to reset all margins and paddings. (Different browsers may have different default margins and paddings for each element. For example, margins and paddings for the `<body>` element or `<input>` element.)

Pseudo-classes

CSS also includes "pseudo-classes" which allow you to select elements in a certain **state**. Pseudo-classes are preceded by a single colon (:). Usually, the selector is the element name followed by a colon followed by the pseudo-class for the state of the element you want to style (e.g. `a:hover` to style a link when the mouse is hovering over it). The most commonly used are the following:

Pseudo-class	Use
`:link`	Select links inside an element
`:visited`	Style visited links (by default visited links are usually purple).
`:active`	Style an element that has been "activated" by the user usually by mouse or keyboard key. In links, this state occurs after the mouse click but before the mouse button is released.
`:hover`	Style an element where the mouse is hovering over it.
`:focus`	Style an element that is in focus. For accessibility, you may wish to have a thick border around an element that is in focus so that users know where they are in the page (especially helpful for keyboard users).

`:first-child`	Style the **first** child of an element. For example, `p:first-child` selects the **first** `<p>` element within a parent container (in this case `#left-sidebar`).
	
	`<aside id="left-sidebar">`
	`  <p>I will be selected because I'm`
	`  the first child of #left-sidebar.`
	`  </p>`
	`  <p>I am the second and last`
	`  child.</p>`
	`</aside>`

`:last-child`	Style the **last** occurring child matching the selector within its parent container.
`:nth-child()`	Style a specific child (e.g. 4th child) or children of a numerical pattern (e.g. 2n for even children).
`:checked`	Style a radio button, checkbox, or dropdown option which has been checked or selected.

Styling for links

In the above table, there are four particular pseudo-classes to use to style links:

```
a, a:link {
   ...
}
a:visited {
  ...
}
a:hover {
  ...
}
a:active {
  ...
}
```

When you style links with the above pseudo-classes, be sure to use the above order to avoid problems with styles not being applied. (Note that you can use a comma to separate items to style so you can style more than one item with the same styling at once.)

Selectors can also include the following special symbols:

> (E > A: select A which is a child of element E)

This specifies a direct descendant, or child, of an element. For example:

```
#left-sidebar > p {
  ...
}
```

The above selector means that the styling in the declaration block will only apply to paragraphs which are children (i.e. direct descendants) of the element with the ID "left-sidebar".

To use > or not to use >

You don't need to use ">" but if you do not then the selector will apply to any element which matches and is a descendant of the element no matter how deeply nested.

For example:

```
#left-sidebar > p {
  ...
}
```

will apply to:

```
<aside id="left-sidebar">
  <p>I will be selected.</p>
  <p>So will I.</p>
</aside>
```

because both paragraphs (above) are children of #left-sidebar, but the above CSS rule will not apply for the paragraph in the example below:

```
<aside id="left-sidebar">
  <section>
    <p>I am not a child so I will not be styled.</p>
  </section>
</aside>
```

On the other hand,

```
#left-sidebar p {
  ...
}
```

will work on both paragraphs below (i.e. it will work on both the child p element as well as the grand-child p element because they are descendants of #left-sidebar):

```
<aside id="left-sidebar">
  <p>I will be styled.</p>
  <section>
    <p>I will be styled as well because I match the criteria.</p>
  </section>
</aside>
```

Pseudo-elements

Pseudo-elements allow you to access abstract "elements" which are not otherwise specified in the DOM. Two such pseudo-elements are given below (::before and ::after). Pseudo-elements are preceded with a double colon (::).

`[element/class/ID]::before`

You can use `::before` to add some content before an element using the `content` property. This allows you to add some stuff in a way that will not affect your semantic markup.

`[element/class/ID]::after`

Similarly, you can use `::after` to add some content after an element using the `content` property.

Note

The double colon was introduced at a later point to differentiate pseudo-elements from pseudo-classes. Prior to the double colon, pseudo-elements were preceded by a single colon like pseudo-classes. Some older browsers still need the single colon. For this reason, browsers still accept pseudo-elements with single colons for backwards compatibility.

As of right now, only **ONE** pseudo-element is allowed per selector.

Adding CSS to your HTML

There are three main ways to add CSS to your HTML page:

1. in the `<head>` section within `<style>` tags (not recommended)
2. inline as an attribute of an element (not recommended)
3. link to a separate CSS file in the `<head>` section (**RECOMMENDED**)

In the `<head>` section, you can place some CSS like so:

```
<head>
  <meta charset="utf-8">
  <title>Test CSS</title>
  <style type="text/css">
    body {
      background-color:#000; /* make the background color of the page black */
      color:#fff; /* make the text color white */
    }
  </style>
</head>
```

To add styling inline in an element, you add it as a style attribute to the element in question. For example:

```
<p style="color: #ff0000;">A red paragraph.</p>
```

The above is equivalent to the following in a CSS file:

```
p {
  color: #ff0000;
}
```

Although you can place CSS within your HTML <head> element (not <header>) and inline, the recommended way to **add CSS is via a separate file** which you link to. For example, if you have a CSS file named ***style.css*** in the same directory as your HTML page, you would link to ***style.css*** by placing the following line in your <head> section (**RECOMMENDED**):

```
<link rel="stylesheet" type="text/css" href="style.css">
```

The **`href`** specifies the path to the CSS file.

Note

You can add multiple CSS files per page. This allows you to separate styles for better organization. One thing to note is that from IE6 to IE9, only up to 31 stylesheets can be loaded. For the most part, this is not an issue but you may come across it if you are using a framework or a theme for a content management system and your client is using an older browser out of necessity (e.g. an enterprise solution). For example, in an older version of Zen Grids, this error occurred because of the many stylesheets used.

From now on, you should always place your styling in a separate CSS file. This allows for more reuseable styling that you can use across multiple pages (e.g. to style an entire website instead of adding styles again for every page).

To play around with CSS, create a ***layout.html*** file to define a layout that we'll play around with. Use the following HTML to get started (this will eventually be a 3-column layout, but this document will only go over some CSS properties without building a layout):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Layout example</title>
  </head>
  <body>
    <header id="header">
      <h2 id="site-name"><a href="#">Sample site</a></h2>
      <nav id="main-menu" aria-label="Main navigation">
        <ul>
          <li><a href="#">Link one</a></li>
          <li><a href="#">Link two</a></li>
          <li><a href="#">Link three</a></li>
        </ul>
      </nav>
    </header>
    <main id="main">
      <h1>Layout basics</h1>
      <p>Some main content.</p>
    </main>
    <aside id="sidebar-one">
      <p>Some left sidebar content.</p>
    </aside>
    <aside id="sidebar-two">
      <p>Some right sidebar content.</p>
    </aside>
    <footer id="footer">
      <p>Some footer content.</p>
    </footer>
  </body>
</html>
```

Note that although only one `<main>` can exist in a page, it still has an ID (main). This is because not all browsers may support the `<main>` element. In those cases, it will be treated as a div. Having an ID also makes it easier for applying CSS. Header and footer also have IDs because more than one may present on the page and having IDs make it easier to style.

Also note that the main section was placed above both sidebars. Although it is a little easier and more intuitive to style with the left sidebar placed *before* the main section, placing main above is better for screen readers because users browse to the main content faster.

Also create a ***style.css*** file in the same directory and link it to the HTML page.

1. Create a new file, ***style.css***, in the same directory as ***layout.html*** (the HTML page above).
2. Open ***layout.html*** in a plain-text editor if it is not already open.
3. Add the following line in the `<head>` section of the page:

   ```
   <link rel="stylesheet" type="text/css" href="style.css">
   ```

   (Don't confuse `<head>` with `<header>`. The `<head>` is where you put meta-data.)

4. Save ***layout.html***. We can then just make our styling changes to only ***style.css***.

Essential CSS

Margin

Margins are spacing outside of an element's border edge (see *CSS Box Model* PDF). Even if there is no border, the margin will lie outside of where the border *would* lie if there was one.

```
margin: [margin-top] [margin-right] [margin-bottom] [margin-left];
```

Margins can be a length in px, em, rem or percentages. In most cases, I would use em because this is a relative unit which makes it more useful for a responsive design. I would only use px if the margin is very small (e.g. under 10px). Large margins in pixels can lead to major issues with responsiveness.

px, em and rem

px

This is the simplest unit to grasp because we're used to thinking of pixels when it comes to computer screens, but for modern CSS, you should **avoid using pixels in most cases** because pixels are the **least** flexible. Pixels are an absolute value which will make it difficult to style a responsive design. Designing with pixels make the design only usable for one screen size.

em

em is a relative unit which is **relative to the default font size of the selected element**. The number for em is a multiplier. For example, if the default font size of `<p>` comes to 16px in desktop (typically), a size of 1.2em would be (16px * 1.2 = 19.2px).

In the case of a desktop browser, the default font size is usually 16px. em is the best choice to use when setting font sizes in your custom CSS because the browser's default font size is usually whatever is deemed an appropriate size for the device (by the browser). This means that for mobile, the default font size set by the browser may be different but we want to base our styling on this because we know that the given size is already appropriate for the device so we only need minor tweaks. Using pixels may result in the font being extremely small for mobile.

rem (root em)

rem is similar to em in that it's based on a font size and is prepended with a multiplier number, but where em is relative to the default font size for the current element, rem is always based on the font size of the **root** element (`<html>` element).

If you want to set the top and bottom margins as the same value and make the left and right margins the same, you can use the shortcut:

```
margin: [margin-top-and-bottom] [margin-left-and-right];
```

If all margins are the same, you can just use one value.

More specifically, you can specify each margin individually using the properties `margin-top`, `margin-left`, `margin-right`, or `margin-bottom`.

Padding

Padding is the spacing **within the border edge** of an element. When you want consistent spacing on the inside perimeter of an element (e.g. to have a gap around the edges of a container for the contents), use padding on the element so that the inside contents align consistently all around.

For example, if you want to evenly add a gap around the inside edges of your header, use padding on the header so that you don't have to add gaps for every element inside header. In the case of a header with a nothing else above it, when adding a gap around a logo, padding is recommended **on the header** because a margin around the logo may result in an odd gap above the header (noticeable if there's a background colour). This is because margins are outside of the border edge and may not always register in the browser so that the header height expands (this usually happens at the top of the page). Padding, on the other hand, is within the border edge and does register so that the outer container expands.

Padding, like margin, goes in the order top, right, bottom, left (i.e. start at the top edge and go clockwise) and has corresponding shortcuts and individual properties:

```
padding: [padding-top] [padding-right] [padding-bottom] [padding-left];
```

```
padding: [padding-top-and-bottom] [padding-left-and-right];
```

```
padding-top: [length];
padding-bottom: [length];
padding-left: [length];
padding-right: [length];
```

Fonts

The following are property-value pairs applicable for fonts:

```
font-family: Helvetica, Arial, sans-serif;
```

Font Stacks

When specifying fonts, you can specify a font you want to be used followed by suggested fonts if the font is unavailable on the user's computer. For example:

```
font-family: Helvetica, Arial, sans-serif;
```

means that the font used should be Arial, but if Arial is unavailable, use Helvetica. If Helvetica is also unavailable, use the system's default sans-serif font. This list of fonts to use or try is called a **font stack**. You should always put the generic font (in the above example, "sans-serif") at the end of the font stack. Each successive font is a fallback for the previous font so the most generic should be last.

There are some common font stacks to specify fonts available on most computers:

- "Times New Roman", Times, serif
- Georgia, Times, "Times New Roman", serif
- Geneva, "Lucida Sans", "Lucida Grande", "Lucida Sans Unicode", Verdana, sans-serif
- "Trebuchet MS", Trebuchet, Tahoma, Arial, sans-serif
- Impact, Haettenschweiler, "Arial Narrow Bold", sans-serif
- Futura, "Century Gothic", AppleGothic, sans-serif

Whichever font stack you use, make sure you know what the fonts look like so you don't end up with weird mismatched typography.

(These font stacks are not so much an issue when you are using an embedded web font. For example, when you use a font from Google Fonts. We'll get to Google Fonts in week 5 but if you wanted to try early, go to Google Fonts, select a font and choose a style by scrolling to the styles and clicking on a + button.)

For more information about font stacks see:

http://www.smashingmagazine.com/2009/09/complete-guide-to-css-font-stacks/

`font-size:` *[numeric size in em/rem/px]*`;`

The recommendation is to set font sizes in ems (you can set the base font size in pixels if you want). Ems size the font as a percentage of the parent element's font size. For example, "`font-size: 0.9em;`" means that the font size for the given element is 90% of the parent element's font size.

Rem (root em), on the other hand, is based on the *root* font size of the <html> element. Rem units then calculate the font size as a percentage of this root font size. This avoids the cascading behaviour of em. (You may find the cascading behaviour of em handy as setting the em size for an element affects all font sizes of child elements. This keeps the relative sizing consistent.)

Pixels are no longer recommended to use to set the font size outside of setting the base or root font size. The reason is that relative sizing is preferable for responsive design. Devices and browsers usually have their own base font size and using pixels can sometimes lead to improper sizes on mobile devices.

`font-style:` *[italic/normal]*`;`

Italic, as it implies, italicizes the font. Normal is just the regular, non-italicized font.

`font-weight:` *[normal/bold/a number from 100 to 900/lighter/bolder]*`;`

The font weight can be normal, bold, a number (usually, 100, 200, etc.), lighter (relative to the parent element's font weight) or bolder (relative to the parent element's font weight). The weight number goes from lighter to bolder as the number gets larger.

`line-height:` *[a number]*`;`

The line height is the height of a line of text. The spacing between lines of text is calculated by the number multiplied with the font size.

Although units are allowed (e.g. 1em), it is preferred to use unitless numbers because having a specified length may cause overlapping lines of text if the font size is larger than the line-height.

`letter-spacing:` *[normal/length]*`;`

Letter spacing adjusts the kerning or space between letters. You can use negative lengths to make the letters closer together. Positive numbers move letters farther apart.

Adjust the font for ***layout.html***.

1. In ***style.css***, set the base font size to 18px.

   ```
   html {
     font-size: 18px;
   }
   ```

2. Change the body font family to Helvetica/Arial

   ```
   body {
     font-family: Helvetica, Arial, sans-serif;
   }
   ```

3. Change the font-family to Trebuchet for the headings H1 to H3.

   ```
   h1, h2, h3 {
     font-family: "Trebuchet MS", Trebuchet, Tahoma, Arial, sans-
   serif;
   }
   ```

4. Change the styling for the body links and menu links:
   a. Change the link color.

      ```
      a, a:link,
      #main-menu a, #main-menu a:link, #main-menu a:visited {
        color: #598598;
      }
      ```

b. Remove the underline from the main menu links. This is found in the `text-decoration` property.

```
#main-menu a, #main-menu a:link {
  text-decoration: none;
}
```

The `text-decoration` would be set to **underline** to add the underline.

c. Set the visited link colour as a slightly darker grey colour. Place this block *after* the `a, a:link` rule.

```
a:visited {
  color: #555555;
}
```

d. Make the hover link color black.

```
a:hover {
  color: #000;
}
```

5. Add a 2px solid black bottom border to the header.

```
#header {
  border: 1px dashed blue;
  margin-bottom: 1em;
  border-bottom: 2px solid #000;
}
```

Here's a summary of the text-related properties mentioned above that you can use (plus a few others):

`font-family:` *[font stack from specific to generic]*`;`

`font-size:` *[numeric size in em/rem/px, em recommended]*`;`

`font-style:` *[italic/normal]*`;`

`font-weight:` *[normal/bold/a number from 100 to 900/lighter/bolder]*`;`

`line-height:` *[a number]*`;` (spacing between lines of text)

`letter-spacing:` *[normal/length]*`;` (spacing between letters)

`text-decoration:` *[underline/overline/line-through/underline overline/none]*`;`

`color:` *[color name/rgb(R_{val}, G_{val}, B_{val})/hex code]*`;` (text colour)

You can also use the short-hand font property to set the font size, weight, style, line-height, and font family all in one go.

```
font: [style] [size]/[line-height] [family];
```

Borders

You can give borders to any element. (Recall the CSS box model as to where the border will be located.)

`border-width:` *[length]*`;`

`border-style:` *[solid/dashed/dotted/none]*`;`

`border-color:` *[named_color/hex color/rgba]*`;`

You can also specify styling for each border using the properties below. This is a shortcut for each element side's border.

```
border-top: [width] [style] [color];
border-right: ...
border-bottom: ...
border-left: ...
```

There is a shortcut apply the border style, width, and colour to all four border sides.

```
border: [width] [style] [color];
```

You can add a radius to your border to create rounded corners.

`border-radius:` *[length]*`;`

You can have a more complex value. The following list shows which values apply for which border corner when you have multiple radius values.

`border-radius:` *[value for top-left and bottom-right] [value for top-right and bottom-left]*`;`
`border-radius:` *[top-left value] [top-right value] [bottom-right value] [bottom-left value]*`;`
`border-radius:` *[top-left value] [value for top-right and bottom-left] [bottom-right value]*`;`

More information:

https://css-tricks.com/almanac/properties/b/border-radius/

Backgrounds

Backgrounds can be colours but can also specify images.

`background-color:` *[named color/hex color/rgba/transparent/hsl/]*`;`
`background-image: url(`*[path to image OR none]*`);`
`background-position:` *[top/bottom/left/right/center/x-value y-value]*`;`
`background-repeat:` *[repeat-x/repeat-y/repeat/no-repeat]*`;`

There is a shortcut for the above in the following format:

```
background: [image] [position] [repeat] [color];
```

For more information about the different properties for backgrounds, view the following page:

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

CSS reset

You may have noticed some differences in default styles if you've checked your HTML page in different browsers. This is because every browser has its own internal CSS which determines the default HTML styling. This means that when we are adding our *own* CSS, what we're actually doing is we're overriding the browser's CSS.

Since every browser has its own internal CSS, this means that there can be some small inconsistencies between browsers—usually regarding margins and paddings (especially in forms).

To reset some of those default styles for a more consistent look where you define all your margins and paddings (to help eliminate inconsistencies across browsers), you can use something called a CSS reset. A CSS reset is a set of CSS rules which reset some of the default browser CSS so that you are starting from a consistent clean slate.

A commonly-used CSS reset is provided by Eric Meyer (see link below). Note that **you don't have to use the reset as is because you *should* tweak it according to what you need** (e.g. if you don't want to remove default padding for headings and paragraphs, remove them from the CSS rule which resets margins and paddings). This is best because if you don't need to remove margins and paddings, it may result in more styling than necessary because you set them to zero with the reset.

Eric Meyer's CSS reset: https://meyerweb.com/eric/tools/css/reset/

Specificity

Now that you've seen some CSS, it is time to introduce an important concept of CSS: **specificity**. CSS styles *cascade* through a page, hence cascading style sheets. (Recall: Browsers have their own default styles which are overridden by user styles, which are *also* overridden by author styles. The CSS file we've created for our webpage is the author style.) CSS follows an order of specificity and goes top-down.

Specificity refers to the "specific-ness" of your CSS selector. The more specific selector overwrites styles in the CSS (i.e. is prioritized) because it has higher specificity. The following list indicates the differing types of selectors in order of specificity:

1. Universal selector (*): The lowest specificity. Any other styles more specific than this will overwrite anything in the * { ... } block.
2. Type or element selector (e.g. h1) and pseudo-elements
3. Class selectors and pseudo-classes
4. ID selectors

Tip!

If you are working on your styling and the CSS for one of your styles is never applied in the browser, try checking your specificity. Remember that higher specificity somewhere in your CSS will cause the styling to be overwritten. Another cause could be that you have two CSS rules with the same specificity which are relevant to the element you want to style (and are styling the same property) but because CSS is read top-down, only the last rule is applied.

Another tip!

Inline styles (i.e. styling via the **style** attribute for an element) have the highest specificity, so they will override your styles in your external stylesheet. Inline styles are not only a bad idea because you're mixing content with styling but it is also a bad idea because some styling you put in your CSS *file* may not work because the of the styling you placed within the HTML document itself.

Inheritance

CSS styles for elements are inherited by their child elements. For example, if you specify paragraphs to use the Arial font, all child elements of paragraphs will, by default, inherit that styling and also use the Arial font. The inherited styling will cascade through all child elements unless overridden, hence cascading style sheet.

For example:

```
<p>This is a <span>paragraph</span>.</p>
```

.
The styling for `<p>` will be *inherited* by `<span>`, so unless `<span>`'s styling has been overridden, it will use the same styling as `<p>` because it is a child element of `<p>`.

CSS Organization

You should use your CSS comments to good effect for organization. When your CSS gets long, you need an easy way to find a style to edit.

One way I do this, is to create a table of contents in the comments, such as:

```
/******************************************************
 * TABLE OF CONTENTS
 * ===================
 * 1. LAYOUT
 * 2. FONT/TYPOGRAPHY
 * 3. LINKS
 ******************************************************/
```

Then, you can place a comment over the group of CSS that corresponds to each table of contents item.

For example:

```
/* 1. LAYOUT */
#header {
  ...
}
...
```

This allows you to easily find a section by doing a document search (**<Ctrl> + F** in Windows).

You do not need to follow the exact organization method above but you should have at least some form of organization and not just a jumble of code. When it comes time to debug your code, you'll be thankful for the organization.