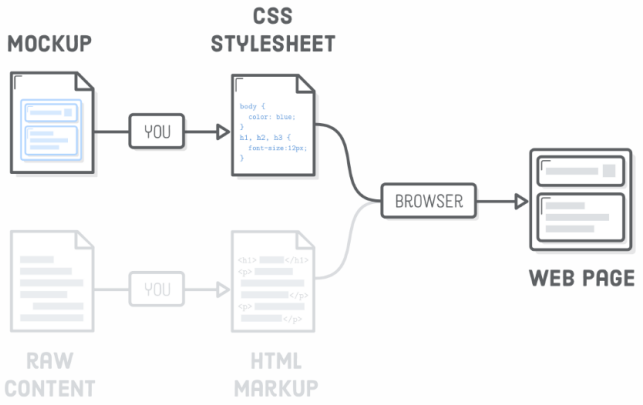
HELLO, CSS

Nº 4. OF HTML & CSS IS HARD

A friendly tutorial for crafting slightly prettier websites

The first few chapters of this tutorial focused exclusively on HTML. Now, it's time to make things pretty (sort of) with Cascading Style Sheets (CSS). You can think of CSS as defining the "design" of a web page. It determines things like font size, margins, and colors using a language entirely separate from HTML.

Why is it a separate language? Well, it serves a completely different purpose. HTML represents the content of your web page, while CSS defines how that content is presented to the user. This is a fundamental distinction central to modern web development.



CSS provides the vocabulary to tell a web browser things like, "I want my headings to be really big and my sidebar to appear on the left of the main article." HTML doesn't have the terminology to make those kinds of layout decisions—all it can say is, "that's a heading and that's a sidebar."

In this chapter, we'll explore the basic syntax of CSS, as well as how to connect it to our HTML documents. The goal isn't so much to become a CSS expert or memorize all the available styles, but rather to understand how CSS and HTML interact. CSS typically lives in its own file, so as in the previous chapter, good file organization will be paramount.

SETUP

To keep things simple, we'll store the example for each chapter of this tutorial in a separate folder. Using Atom, create a new project called `hello-css`. We're going to be styling an existing page called `hello-css.html`, so go ahead and create that, then add the following markup:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Hello, CSS</title>
  </head>
  <body>
    <h1>Hello, CSS</h1>

    <p>CSS lets us style HTML elements. There's also
       <a href='dummy.html'>another page</a> associated with this example.</p>

    <h2>List Styles</h2>

    <p>You can style unordered lists with the following bullets:</p>

    <ul>
      <li>disc</li>
      <li>circle</li>
      <li>square</li>
    </ul>

    <p>And you can number ordered lists with the following:</p>

    <ol>
      <li>decimal</li>
      <li>lower-roman</li>
      <li>upper-roman</li>
      <li>lower-alpha</li>
      <li>upper-alpha</li>
      <li>(and many more!)</li>
    </ol>
  </body>
```


```
    <p>You can style unordered lists with the following bullets:</p>

    <ul>
      <li>disc</li>
      <li>circle</li>
      <li>square</li>
    </ul>

    <p>And you can number ordered lists with the following:</p>

    <ol>
      <li>decimal</li>
      <li>lower-roman</li>
      <li>upper-roman</li>
      <li>lower-alpha</li>
      <li>upper-alpha</li>
      <li>(and many more!)</li>
    </ol>
  </body>
</html>
```


In addition, we'll need a small dummy page to learn how CSS styles can be applied to multiple web pages. Create `dummy.html` and add the following:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Dummy</title>
  </head>
  <body>
    <h1>Dummy</h1>
      <li>lower-alpha</li>
      <li>upper-alpha</li>
      <li>(and many more!)</li>
    </ol>
  </body>
</html>
```

In addition, we'll need a small dummy page to learn how CSS styles can be applied to multiple web pages. Create `dummy.html` and add the following:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Dummy</title>
  </head>
  <body>
    <h1>Dummy</h1>

    <p>This is a dummy page that helps us demonstrate reusable CSS
       stylesheets. <a href='hello-css.html'>Go back</a>.</p>
    </ol>
  </body>
</html>
```

In addition, we'll need a small dummy page to learn how CSS styles can be applied to multiple web pages. Create `dummy.html` and add the following:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Dummy</title>
  </head>
  <body>
    <h1>Dummy</h1>

    <p>This is a dummy page that helps us demonstrate reusable CSS
       stylesheets. <a href='hello-css.html'>Go back</a>.</p>

    <p>Want to try crossing out an <a href='nowhere.html'>obsolete link</a>? Th
       is your chance!</p>
  </body>
</html>
```

applied to multiple web pages. Create `dummy.html` and add the following:

```
<!DOCTYPE html>
<html lang='en'>
  <head>
    <meta charset='UTF-8'/>
    <title>Dummy</title>
  </head>
  <body>
    <h1>Dummy</h1>

    <p>This is a dummy page that helps us demonstrate reusable CSS
       stylesheets. <a href='hello-css.html'>Go back</a>.</p>

    <p>Want to try crossing out an <a href='nowhere.html'>obsolete link</a>? Th
       is your chance!</p>
  </body>
</html>
```

CSS STYLESHEETS

CSS stylesheets reside in plaintext files with a `.css` extension. Create a new file called `styles.css` in our `hello-css` folder. This will house all our example snippets for this chapter. Let's add one CSS rule so that we can tell if our stylesheet is hooked up to our HTML pages properly.

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

A CSS "rule" always start with a "selector" that defines which HTML

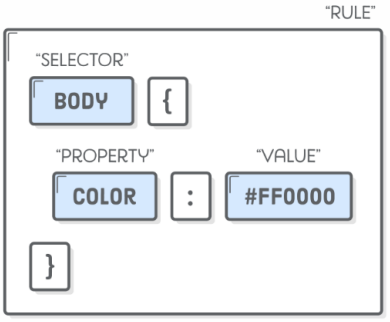
elements it applies to. In this case, we're trying to style the `<body>` element. After the selector, we have the "declarations block" inside of some curly

CSS STYLESHEETS

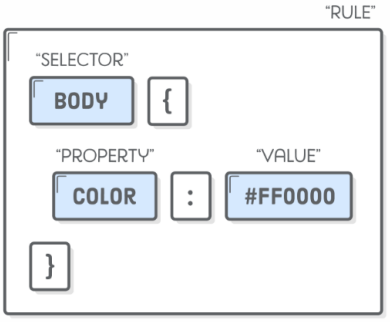
CSS stylesheets reside in plaintext files with a `.css` extension. Create a new file called `styles.css` in our `hello-css` folder. This will house all our example snippets for this chapter. Let's add one CSS rule so that we can tell if our stylesheet is hooked up to our HTML pages properly.

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

A CSS "rule" always start with a "selector" that defines which HTML elements it applies to. In this case, we're trying to style the `<body>` element. After the selector, we have the "declarations block" inside of some curly braces. Any "properties" we set in here will affect the `<body>` element.



`color` is a built-in property defined by the CSS specification that determines elements it applies to. In this case, we're trying to style the `<body>` element. After the selector, we have the "declarations block" inside of some curly braces. Any "properties" we set in here will affect the `<body>` element.



`color` is a built-in property defined by the CSS specification that determines the text color of whatever HTML elements have been selected. It accepts a hexadecimal value representing a color. `#FF0000` means bright red.

CSS properties are kind of like HTML attributes in that they both deal with key-value pairs. Except, here we're defining *presentational* information instead of contributing to the *semantic* meaning of the underlying content.

LINKING A CSS STYLESHEET

If you try loading either of the HTML pages in a browser, you won't see our stylesheet in action. That's because we didn't link them together yet. This is

what the HTML <link/> element is for. In hello-css.html, change <head> to the following:

```
<head>
  <meta charset='UTF-8'/>
  <title>Hello, CSS</title>
  <link rel='stylesheet' href='styles.css'/>
</head>
```

LINKING A CSS STYLESHEET

If you try loading either of the HTML pages in a browser, you won't see our stylesheet in action. That's because we didn't link them together yet. This is what the HTML <link/> element is for. In hello-css.html, change <head> to the following:


```
<head>
  <meta charset='UTF-8'/>
  <title>Hello, CSS</title>
  <link rel='stylesheet' href='styles.css'/>
</head>
```


This <link/> element is how browsers know they need to load styles.css when they try to render our hello-css.html page. We should now see

LINKING A CSS STYLESHEET

If you try loading either of the HTML pages in a browser, you won't see our stylesheet in action. That's because we didn't link them together yet. This is what the HTML <link/> element is for. In hello-css.html, change <head> to the following:

```
<head>
  <meta charset='UTF-8'/>
  <title>Hello, CSS</title>
  <link rel='stylesheet' href='styles.css'/>
</head>
```

This <link/> element is how browsers know they need to load styles.css when they try to render our hello-css.html page. We should now see

LINKING A CSS STYLESHEET

If you try loading either of the HTML pages in a browser, you won't see our stylesheet in action. That's because we didn't link them together yet. This is what the HTML <link/> element is for. In hello-css.html, change <head> to the following:

```
<head>
  <meta charset='UTF-8'/>
  <title>Hello, CSS</title>
  <link rel='stylesheet' href='styles.css'/>
</head>
```

This <link/> element is how browsers know they need to load styles.css when they try to render our hello-css.html page. We should now see

LINKING A CSS STYLESHEET

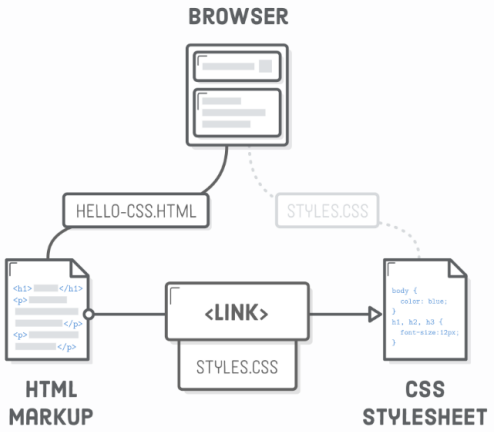
If you try loading either of the HTML pages in a browser, you won't see our stylesheet in action. That's because we didn't link them together yet. This is what the HTML `<link/>` element is for. In `hello-css.html`, change `<head>` to the following:

```
<head>
  <meta charset='UTF-8'/>
  <title>Hello, CSS</title>
  <link rel='stylesheet' href='styles.css'/>
</head>
```

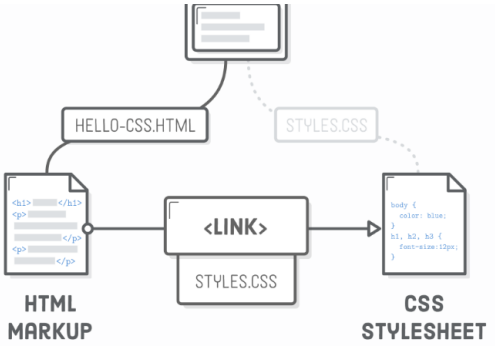
This `<link/>` element is how browsers know they need to load `styles.css` when they try to render our `hello-css.html` page. We should now see blindingly red text everywhere:

The `<link/>` element is just like the `<a>` element, but it's only meant to be used inside of `<head>`. Since it's in the head of the document, `<link/>` connects to *metadata* that's defined outside of the current document. Also notice that it's an empty element, so it doesn't need a closing tag.

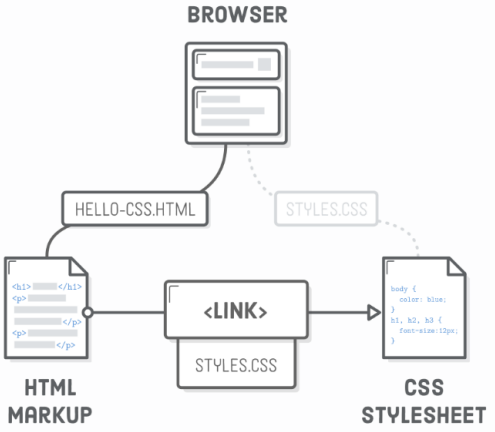
The `rel` attribute defines the relationship between the resource and the HTML document. By far the most common value is `stylesheet`, but there are a few other options. The `href` attribute works the same as in the previous chapter, only it should point to a `.css` file instead of another web page. The value for `href` can be an absolute, relative, or root-relative link.



Note that there's no *direct* connection between the browser and our



Note that there's no *direct* connection between the browser and our stylesheet. It's only through the HTML markup that the browser can find it. CSS, images, and even JavaScript all rely on an HTML page to glue



Note that there's no *direct* connection between the browser and our stylesheet. It's only through the HTML markup that the browser can find it. CSS, images, and even JavaScript all rely on an HTML page to glue everything together, making HTML the heart of most websites.

CSS COMMENTS

Now that our stylesheet is hooked up, let's play with it a little bit. That red is horrible. Let's tone it down to a nice gray:

```
body {
  color: #414141;    /* Dark gray */
}
```

Notice that comments in CSS are a little different than their HTML counterparts. Instead of the <!-- --> syntax, CSS ignores everything between /* and */ characters.

CSS COMMENTS

Now that our stylesheet is hooked up, let's play with it a little bit. That red is horrible. Let's tone it down to a nice gray:

```
body {
  color: #414141;    /* Dark gray */
}
```

Notice that comments in CSS are a little different than their HTML counterparts. Instead of the <!-- --> syntax, CSS ignores everything between /* and */ characters.

CSS COMMENTS

Now that our stylesheet is hooked up, let's play with it a little bit. That red is horrible. Let's tone it down to a nice gray:

```
body {
  color: #414141;    /* Dark gray */
}
```

Notice that comments in CSS are a little different than their HTML counterparts. Instead of the <!-- --> syntax, CSS ignores everything between /* and */ characters.

CSS COMMENTS

Now that our stylesheet is hooked up, let's play with it a little bit. That red is horrible. Let's tone it down to a nice gray:

```
body {
  color: #414141;    /* Dark gray */
}
```

Notice that comments in CSS are a little different than their HTML counterparts. Instead of the <!-- --> syntax, CSS ignores everything between /* and */ characters.

CSS COMMENTS

Now that our stylesheet is hooked up, let's play with it a little bit. That red is horrible. Let's tone it down to a nice gray:

```
body {
  color: #414141;    /* Dark gray */
}
```

Notice that comments in CSS are a little different than their HTML counterparts. Instead of the <!-- --> syntax, CSS ignores everything between /* and */ characters.

CSS COMMENTS

Now that our stylesheet is hooked up, let's play with it a little bit. That red is horrible. Let's tone it down to a nice gray:

```
body {
  color: #414141;    /* Dark gray */
}
```

Notice that comments in CSS are a little different than their HTML counterparts. Instead of the `<!-- -->` syntax, CSS ignores everything between `/*` and `*/` characters.

SETTING MULTIPLE PROPERTIES

You can stick as many properties as you want in the declarations block of a CSS rule. Try setting the background color of the entire web page by changing our rule to the following:

```
body {
  color: #414141;               /* Dark gray */
  background-color: #EEEEEE;    /* Light gray */
}
```

The `background-color` property is very similar to the `color` property, but it defines the background color of whatever element you selected. Take a second to admire those semicolons at the end of each declaration. Removing

SETTING MULTIPLE PROPERTIES

You can stick as many properties as you want in the declarations block of a CSS rule. Try setting the background color of the entire web page by changing our rule to the following:

```
body {
  color: #414141;               /* Dark gray */
  background-color: #EEEEEE;    /* Light gray */
}
```

The `background-color` property is very similar to the `color` property, but it defines the background color of whatever element you selected. Take a second to admire those semicolons at the end of each declaration. Removing them will break the CSS rule, so *always mind your semicolons!*

Why did we pick shades of grays instead of black and white? Using a `#000000` background with a `#FFFFFF` text color is too high of a contrast. It makes it look like the page is vibrating, which can be very distracting for readers.

SELECTING DIFFERENT ELEMENTS

Of course, you'll want to apply styles to elements other than `<body>`. For that, simply add more CSS rules with different selectors. We can change the font size of our `<h1>` headings like so:

```
body {
  color: #414141;               /* Dark gray */
  background-color: #EEEEEE;    /* Light gray */
}

h1 {
  font-size: 36px;
}
```

And, if you want to alter `h2` headings, add another rule:

```
h2 {
  font-size: 28px;
}
```


```
body {
  color: #414141;               /* Dark gray */
  background-color: #EEEEEE;    /* Light gray */
}

h1 {
  font-size: 36px;
}
```

And, if you want to alter `h2` headings, add another rule:

```
h2 {
  font-size: 28px;
}
```

```
body {
  color: #414141;               /* Dark gray */
  background-color: #EEEEEE;    /* Light gray */
}

h1 {
  font-size: 36px;
}
```

And, if you want to alter `h2` headings, add another rule:

```
h2 {
  font-size: 28px;
}
```


UNITS OF MEASUREMENT

Many CSS properties require a unit of measurement. There's a lot of units available, but the most common ones you'll encounter are `px` (pixel) and `em` (pronounced like the letter *m*). The former is what you would intuitively call a pixel, regardless of whether the user has a retina display or not, and the latter is the current font size of the element in question.

UNITS OF MEASUREMENT

Many CSS properties require a unit of measurement. There's a lot of units available, but the most common ones you'll encounter are `px` (pixel) and `em` (pronounced like the letter *m*). The former is what you would intuitively call a pixel, regardless of whether the user has a retina display or not, and the latter is the current font size of the element in question.

The `em` unit is very useful for defining sizes relative to some base font. In the above diagram, you can see `em` units scaling to match a base font size of `12px`, `16px`, and `20px`. For a concrete example, consider the following alternative to the previous code snippet:

```
body {
  color: #414141;               /* Dark gray */
  background-color: #EEEEEE;    /* Light gray */
  font-size: 18px;
}

h1 {
  font-size: 2em;
}

h2 {
  font-size: 1.6em;
}
```

The `em` unit is very useful for defining sizes relative to some base font. In the above diagram, you can see `em` units scaling to match a base font size of `12px`, `16px`, and `20px`. For a concrete example, consider the following alternative to the previous code snippet:

```
body {
  color: #414141;               /* Dark gray */
  background-color: #EEEEEE;    /* Light gray */
  font-size: 18px;
}

h1 {
  font-size: 2em;
}

h2 {
  font-size: 1.6em;
}
```

This sets our base font size for the document to `18px`, then says that our `<h1>` elements should be twice that size and our `<h2>`'s should be 1.6 times bigger. If we (or the user) ever wanted to make the base font bigger or smaller, `em` units would allow our entire page to scale accordingly.

SELECTING MULTIPLE ELEMENTS

What if we want to add some styles to *all* our headings? We don't want to have redundant rules, since that would eventually become a nightmare to maintain:

This sets our base font size for the document to `18px`, then says that our `<h1>` elements should be twice that size and our `<h2>`'s should be 1.6 times bigger. If we (or the user) ever wanted to make the base font bigger or smaller, `em` units would allow our entire page to scale accordingly.

SELECTING MULTIPLE ELEMENTS

What if we want to add some styles to *all* our headings? We don't want to have redundant rules, since that would eventually become a nightmare to maintain:

```
/* (You'll regret creating redundant styles like this) */
h1 {
  font-family: "Helvetica", "Arial", sans-serif;
}

h2 {
  font-family: "Helvetica", "Arial", sans-serif;
}

h3 {
  font-family: "Helvetica", "Arial", sans-serif;
}
/* (etc) */
```

What if we want to add some styles to *all* our headings? We don't want to have redundant rules, since that would eventually become a nightmare to maintain:

```
/* (You'll regret creating redundant styles like this) */
h1 {
  font-family: "Helvetica", "Arial", sans-serif;
}

h2 {
  font-family: "Helvetica", "Arial", sans-serif;
}

h3 {
  font-family: "Helvetica", "Arial", sans-serif;
}
/* (etc) */
```

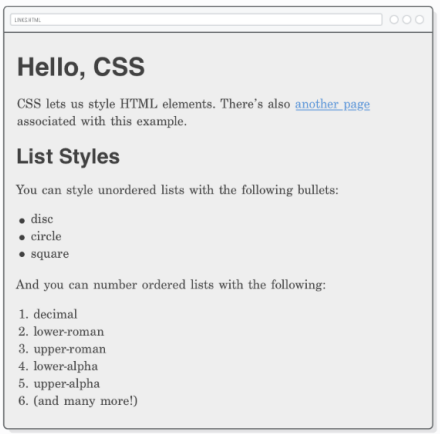
Instead, we can select multiple HTML elements in the same CSS rule by separating them with commas. Add this to our `styles.css` file:
maintain:

```
/* (You'll regret creating redundant styles like this) */
h1 {
  font-family: "Helvetica", "Arial", sans-serif;
}

h2 {
  font-family: "Helvetica", "Arial", sans-serif;
}

h3 {
  font-family: "Helvetica", "Arial", sans-serif;
}
/* (etc) */
```

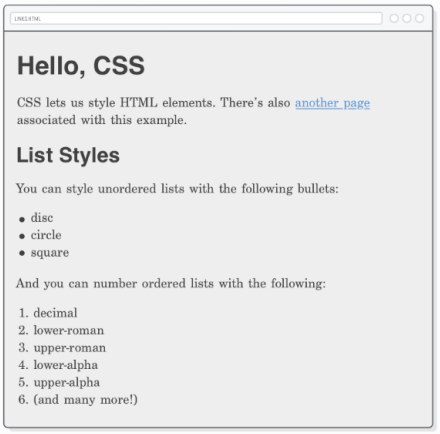
Instead, we can select multiple HTML elements in the same CSS rule by separating them with commas. Add this to our `styles.css` file:

```
h1, h2, h3, h4, h5, h6 {
  font-family: "Helvetica", "Arial", sans-serif;
}
```

This defines the font to use for all of our headings with a single rule. That's great, 'cause if we ever want to change it, we only have to do so in one place. Copying and pasting code is usually a bad idea for web developers, and multiple selectors can help reduce that kind of behavior quite a bit.



and multiple selectors can help reduce that kind of behavior quite a bit.



DEFINING FONTS

`font-family` is another built-in CSS property that defines the typeface for whatever element you selected. It accepts multiple values because not all users will have the same fonts installed. With the above snippet, the browser tries to load the left-most one first (`Helvetica`), falls back to `Arial` if the user doesn't have it, and finally chooses the system's default sans serif font.

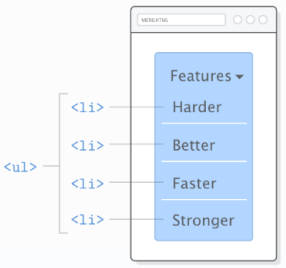
Relying on the user's built-in fonts has historically been incredibly limiting for web designers. Nowadays, system fonts have been largely superseded by web fonts. You can read more about this in the Web Typography chapter of this tutorial.

LIST STYLES

The `list-style-type` property lets you alter the bullet icon used for `<li>` elements. You'll typically want to define it on the parent `<ul>` or `<ol>` element:

```
ul {
  list-style-type: circle;
}

ol {
  list-style-type: lower-roman;
}
```

navigation with a `<ul>` list. The `none` value allows the menu's list items to be styled more like buttons. In the Advanced Positioning chapter, we'll actually use this technique to create the navigation menu shown below.



This is good example of the separation of content from presentation. A navigation menu *is* an unordered list, but it also makes sense to display them as buttons instead of a typical bulleted list. Intelligently designed HTML allows search engines to infer the structure of our content, while CSS lets us display it to humans in beautiful ways.

You can even create custom bullets for `<li>` elements with the `list-style-image` property (see MDN for details).