Chapter 3: Basics of CSS

Now that we’ve covered quite a bit of HTML, let’s move on to CSS. CSS stands for Cascading Stylesheet and as the name suggests, CSS is all about styling and making your website look gorgeous.

The latest version of CSS is CSS3. Unlike previous versions of CSS (namely CSS1 and CSS2.1), CSS3 splits the language into different modules so that each module can be developed separately at a different pace. Each module adds new features or extends the capabilities of features previously defined in CSS 2.1. Essentially, CSS3 is simply an extension of CSS2.1.

This book covers the core properties of CSS2.1 as well as a few new properties that are introduced in CSS3. Once you master the core properties, you will have no problems moving on to more advanced properties that are newly added in CSS3. These advanced properties allow for more fanciful styling of your website, such as adding transitions and animations.

In this chapter, we’ll be covering the basics of CSS, including its syntax and order of precedence. However, before going into the syntax of CSS, let’s first learn how to add CSS rules to our web site.

Applying CSS Code

There are three ways to apply CSS code to our site.

The first is by linking to an external file. This is the recommended method. To do linking, you need to write your CSS rules in a separate text file and save it with a `.css` extension. The syntax for adding the rules to your HTML code is

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

You add the `<link>` tag to the `head` element, between the `<head>...</head>` tags. The first two attributes `rel` and `type` tell the browser that this is a CSS stylesheet. You do not need to modify them. The last attribute `href` is where you specify the path to the CSS file that you want to link to. A CSS file is simply a file that contains CSS rules, without any HTML tags. An example is shown below. Don't worry if the code does not make sense to you, we'll cover them very soon.

```
body {
       margin: 0;
       background-color: green;
}
```

Save this code as "style.css" in the same folder as the .html file. You can then use the `<link>` tag above to link this CSS file to your HTML file.

The second method to add CSS rules to our site is to embed the code directly into our HTML source code, within the `head` element. This is done with the `<style>` tag. An example is shown below. The embedded CSS code starts after the `<style>` start tag and ends before the `</style>` end tag.

```
<head>
    <style>
    div {
             color: blue;
             width: 100px;
             height: 200px;
    }
    </style>
</head>
```

The last method is to use inline CSS. Inline CSS is specified in the start tag of the element you want to apply it to, using the `style` attribute. Each rule ends with a semi-colon (;). An example is:

```
<div style="text-decoration:underline; color:blue;">Some text</div>
```

Out of the three methods, linking is the preferred method. Linking separates the HTML content from the styling rules and makes it easier to maintain our codes. It is also extremely useful when we need to apply the same CSS rules to multiple web pages.

Embedded CSS, on the other hand, is commonly used when the rules only apply to one web page.

Inline CSS is handy when you need to apply a rule to only <u>one element</u>, or when you want to override other CSS rules that apply to the same element. This is because inline CSS has a higher precedence than CSS code added using the other two methods. We'll discuss order of precedence later in this chapter. However, inline CSS mixes styling with content and should be avoided whenever possible.

Syntax of a CSS rule

Now that we know how to apply CSS rules to our HTML files, let's move on to learn some actual CSS code. The first thing to learn about CSS is its syntax, which is relatively straightforward. The syntax is:

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

For instance, if you want to style the contents inside a `<div>` tag, you write the rule as

```
div {
  background-color: green;
  font-size: 12px;
}
```

The first word `div` is the selector. It tells the browser that the set of rules inside the curly brackets { } applies to all elements with the `<div>` tag.

Inside the curly brackets, you write all your declarations. You start by declaring the property that you want to set (`background-color` in the first declaration), followed by a colon (:). Next, you give the value that you want (`green` in this case). Finally, you end each declaration with a semi-colon (;).

Indentation and line breaks do not matter in CSS. You can also write your declarations like this:

```
div { background-color: green; font-size: 12px; }
```

Pretty straightforward right? Great! Let's move on...

Selecting an Element

In the example above, the rules declared in the curly brackets will apply to ALL elements with a `<div>` tag. However, most of the time, we want greater variation. Suppose you want one `<div>` element to have a font size of 12px and another to have a font size of 14px. How would you do it?

Selecting Classes and IDs

There are basically two ways to do it. The first method is to use the `id` attribute. In your HTML document, instead of just using the `<div>` tag, you can add an `id` attribute to it. For instance, you can write

```
<div id=”para1”>
    Some text.
</div>

<div id=”para2”>
    More text.
</div>
```

In our CSS code, we can then select the respective `id` by adding a # sign in front of the `id` name. An example is shown below:

```
div {
  background-color: green;
}

#para1
{
  font-size: 12px;
}

#para2
{
  font-size: 14px;
}
```

The first rule applies to all elements with the `<div>` tag. The second rule only applies to the element with `id=”para1”`. The third rule only applies to the element with `id=”para2”`.

In addition to using the selector `#para1`, you can also be more specific and write `div#para1`, with no space before and after the # sign. Both methods will select the same element, but the second method has a higher precedence (more on this later).

Note that an `id` should be unique within a page. Two `<div id=”para1”>` tags is not allowed. One `<div id=”para1”>` and one `<p id=”para1”>` tag is also not allowed as both have the same `id`. Although your HTML and CSS code will work even if you have two elements with the same `id`, problems will arise when you start using Javascript or other scripting languages on your website.

If you need to apply the same CSS rules to two different elements, you can use a `class`. A `class` is similar to an `id`, with the exception that a `class` need not be unique. In addition, an `id` has a higher precedence than a `class`.

For now, let’s consider the following code:

```
<div class=”myclass1”>
    Some text.
</div>

<p class=”myclass1”>
    More text.
</p>

<div>
    Yet more text.
</div>
```

If you want to select all `<div>` elements (i.e. the first and third element), you write

```
div { … }
```

If you want to select all elements with `class=”myclass1”` (i.e. the first and second element), you add a dot (.) in front of the `class` name, like this:

```
.myclass1 { … }
```

If you only want to select `<p>` tags with `class=”myclass1”` (i.e. the second element), you write

```
p.myclass { … }
```

There should be no space before and after the dot.

An element can have more than one classes. Multiple classes are separated with a space in the HTML attribute. For instance, the `div` below has two classes: `myclass1` and `myclass2`.

```
<div class=”myclass1 myclass2”>
…
</div>
```

If we have the following CSS code,

```
.myclass1 { … }

.myclass2 { … }
```

the rules for both `myclass1` and `myclass2` will apply to the above `<div>`.

More Selectors

In addition to selecting an element by `id` and `class`, CSS offers a large variety of ways to specify the elements that we want to select.

Selecting Multiple Elements

For instance, we can select multiple elements at one go. If we want to select the `<div>`, `<p>` and `<ul>` elements, we write:

```
div, p, ul { … }
```

Selecting Child Elements

If we want to select all the `<p>` elements ***inside*** `<div>` elements, we write

```
div p { … }
```

Note that there is no comma between `div` and `p`. In this case, the CSS rules will only apply to `<p>` elements that are inside `<div>` elements. For instance, if we have the HTML structure below, the rules will apply to `‘I am a paragraph inside div’` and not to `‘I am a stand-alone paragraph’`.

```
<div>
       <p>I am a paragraph inside div</p>
</div>

<p>I am a stand-alone paragraph</p>
```

The first paragraph `‘I am a paragraph inside div’` is called a child element of the `<div>` tag as its start and end tags (`<p>` and `</p>`) lie entirely within the `<div>...</div>` tags.

Selecting by Attribute

You can also select an element based on its attribute. If you want to select all hyperlinks that link to http://www.learncodingfast.com, you write

```
a[href=”http://www.learncodingfast.com”]  { … }
```

There should be no space before the square bracket. If you have the following HTML code, only the first link will be selected.

```
<a href=”http://www.learncodingfast.com”>Learn Coding Fast</a>
<a href=”http://www.google.com”>Google</a>
```

Selecting Pseudo-classes

Another commonly used selector is the pseudo-class selector. A pseudo-class refers to a *special state of an element*. The most common pseudo-classes are those for the `<a>...</a>` element. A hyperlink can be in one of four states:

`link` (an unvisited link)
`visited` (a visited link)
`hover` (when the user mouses over it), or
`active` (when the link is clicked).

We can select a hyperlink based on the state it is in. For instance, to select the hover state, we write

```
a:hover { … }
```

The keyword `hover` is added to the back of the `a` selector using a colon (:), with no spaces before and after the colon. We’ll come back to the concept of selecting and styling different states of a hyperlink in Chapter 9.

In addition to selecting different states of a hyperlink, we can also use pseudo-classes is to select child elements. Suppose we have a `<div>` element with three `<p>` child elements:

```
<div>
       <p>I am the first child</p>
       <p>I am the second child</p>
       <p>I am the third child</p>
</div>
```

We can use the `first-child` pseudo-class to select the first `<p>` element. We can also use the `last-child` selector to select the last child or the `nth-child(n)` selector to select the nth child.

For instance, if we write

```
p:nth-child(2) { … }
```

we’ll be selecting the paragraph `‘I am the second child’` because of the number `‘2’` in the parenthesis ( ).

Selecting Pseudo-elements

In addition to pseudo-classes, CSS also has the concept of pseudo-elements. A pseudo-element refers to a specified part of an element, such as the first letter or the first line of an element.

For instance, if we have the following `<p>` element:

```
<p>This is some text.</p>
```

We can select the first letter (T) by writing

```
p::first-letter { … }
```

Note that a double colon is used in this case. Another pseudo-element is the `first-line` element. This will select the first line of the text.

Finally, we can use the `before` and `after` pseudo-elements to insert content before, or after, the content of an element. For instance, if we want to add an exclamation mark after all `H1` elements, we can write

```
h1::after {
    content: “!”;
}
```

This will automatically append an exclamation mark after all `H1` elements. If we have the following HTML code

```
<h1>This is a heading</h1>
```

we’ll get

```
This is a heading!
```

Case Insensitivity

For the most part, CSS selectors and rules are case-insensitive. Hence, you can either write

```
div {
       Background-color: GREEN;
}
```

or

```
DIV {

       background-coloR: green;
}
```

Both will work equally well. The only exception to this case-insensitivity is when selecting classes and ids.

If we have

```
<div id= “myID”>Some text</div>
```

`div#myID` will select the above element while `div#MYID` will not.

Order of Precedence

Now that we’ve learnt how to select elements, let us move on to a very important concept in CSS: order of precedence.

As mentioned earlier, we can apply CSS code to our website in three different ways. It is common for a programmer to use more than one way to apply CSS code to a site. For instance, a website may have CSS rules defined in an external file AND some additional CSS rules embedded within its `<style>...</style>` tags. This may result in more than one rule being applied to the same element. One of the most frustrating experience about working with CSS, especially when you are first starting out, is when you try to apply a css style to an element and the page simply seems to ignore your rule. Most of the time, this is due to the order of precedence. Specifically, this happens when more than one rule applies to the same element, and another rule has a higher precedence than the one you are specifying.

Three principles control which CSS rule has a higher precedence.

Principle 1: The more specific the selector, the higher the precedence

We won’t go into details about how to calculate the specificity of a selector. The main point to remember is that an `id` is considered to be more specific than a `class`, and a `class` more specific than an element. Let’s consider the code below:

```
div { font-size: 10px; }
#myId { font-size: 12px; }
.myClass { font-size: 14px; }

<div id=”myId” class=”myClass”>Some text</div>
```

Since the `<div>` element has `class=”myClass”` and `id=”myId”`, all three rules `div`, `#myId` and `.myClass` will apply to the `<div>` element. However, as `id` has the highest precedence, `“Some text”` will be displayed with a font size of 12px.

In addition, another point to note about specificity is that the more detailed your selector, the higher the precedence. For instance, `div#myId` has a higher precedence than `#myId`. This is because `div#myId` is considered to be more detailed as it tells us that `myId` is an `id` of the `div` element. In the sample code below, the color yellow will be applied.

```
div { color: red; }
div#myId { color: yellow; }
#myId { color: blue; }
.myClass { color: green; }

<div id=”myId” class=”myClass”>Some text</div>
```

Principle 2: If no style is specified, elements inherit styles from their parent container

A child element is an element which lies entirely within the start and end tags of another element. For instance, in the code below, `<p>` is a child element of the `<body>` element. Since the font size of `<p>` is not defined, it’ll inherit this property from the `<body>` element for which the property is defined.

```
body {
       font-size: 1.5em;
}

<body>
       <p>Some text</p>
</body>
```

If the `font-size` property is also not defined for the `<body>` element, the browser’s default font size will be used.

Principle 3: All else being equal, the last declared rule wins

Suppose you have the following CSS declaration in your HTML `<head>` element.

```
<head>
<style>
       p { font-size: 20px; }
</style>
</head>
```

Further down the HTML document, you have the following HTML code, with an inline CSS rule:

```
<p style=”font-size: 30px;”>Some text</p>
```

Which rule do you think will be applied to the words “Some text”?

The correct answer is the inline rule. This is because all things being equal, the rule that is declared last has the highest precedence. Since inline CSS is declared within the HTML code, it is declared later than the embedded CSS which is declared in the `head` section. Hence, a font size of 30px will be applied.

Display Inconsistency

Another issue to deal with when working with CSS is the problem of display inconsistency across browsers. You may find that your website looks slightly (or drastically) different in different browsers. Most display issues tend to occur in older versions of Internet Explorer, although issues can occur in other browsers too (especially mobile browsers).

Display inconsistencies occur because different browsers use different layout engines to interpret the site’s CSS code. For instance, Safari and Chrome use the WebKit engine while Firefox uses the Gecko engine. One engine may calculate and display a page differently from another engine. For instance Trident, the engine used by Internet Explorer, automatically widens a page’s pixel width for certain page designs. This can lead to the sidebar being pushed to the bottom due to insufficient width.

Another problem causing display inconsistency is the lack of universal support for some CSS properties. Some properties are not supported by all browsers. You can go to the site http://www.caniuse.com to check if a certain CSS property is supported by the browser that you are developing for.

Sometimes, a certain CSS property is supported by a particular browser only when we add a prefix to our CSS rules. This is especially true for newer properties in CSS3. An example is the `column-count` property in CSS3. This property divides an element into multiple columns. For instance, we can divide a `div` element into three columns by writing `column-count: 3`.

This property is not supported by older versions of Firefox, Chrome, Safari and

Opera. To enable the property to work on these browsers, you have to write it as three declarations,

```
-webkit-column-count: 3;
-moz-column-count: 3;
column-count: 3;
```

instead of just

```
column-count: 3;
```

The `-webkit-` prefix adds support for older versions of Chrome, Safari and Opera while the `-moz-` prefix adds support for Firefox. In addition, we also have the `-ms-` prefix that adds support for Internet Explorer.

When creating your website, it is useful to test it on various browsers to ensure that nothing is broken. The way to fix a ‘broken’ display depends on the issue causing it. If you are really stuck, I suggest searching or posting the question on http://stackoverflow.com, which is a very useful online community for programmers.

Comments

The last thing to cover in this chapter is comments. In CSS, we add comments to our code using the `/*...*/` symbols. An example is as follows:

```
/*
The rules below are comments.

p {
       background-color: black;
       font-size: 20px;
       color: white;
}
*/
```

Everything between the `/*` and `*/` symbols is ignored by the browser.

Exercise 3