CSS PROGRAMMING {FAST}

LEARN CSS PROGRAMMING **IN AN EASY WAY**

CSS Programming
Learn CSS Programming FAST!

Disclaimer

The information provided in this book is designed to provide helpful information on the subjects discussed. The author's books are only meant to provide the reader with the basics knowledge of a certain topic, without any warranties regarding whether the student will, or will not, be able to incorporate and apply all the information provided. Although the writer will make his best effort share his insights, learning is a difficult task and each person needs a different timeframe to fully incorporate a new topic. This book, nor any of the author's books constitute a promise that the reader will learn a certain topic within a certain timeframe.

Preface

CSS has been constructed into a great language as well as a language of great importance. Not only does CSS make life easier for programmers who are styling HTML documents. CSS creates the ability to make life easier for so many other professions. The formatting and styling can be used to create a sales funnel for a marketing company. The gradients and transitions can be used to lighten the workload of graphic designers. Perhaps most importantly the @media rules that allow responsive design to change the layout of a website based on screen size have made the web user friendly for millions and millions of regular people. Without this single feature the internet of things would be out the window, perhaps CSS is to blame for a new generation of young ones that walk around staring at their cell phone all day. Maybe that is simply a testament to the quality that CSS can produce. A vital ingredient to the success of the modern tech industries boom. CSS is possibly one of the most under-rated assets any developer could possess. Sure everyone needs to know it to even have a chance at getting hired in the field today. The difference between knowing CSS and thriving with your CSS creations may be the difference from getting by and getting ahead of the competition. This is the true glory of CSS. No matter what kind of website you want to make. Everyone wants to produce something that will look good in a way that makes it easy to use and understand. Anyone can create a list of links and throw it in an HTML document. Fewer can say that they know how to style links to look like buttons so that their users will feel more at home. The most famous of this type of need is the call to action. Any marketing expert will fill you in on why you need a call to action. I will fill you in on what parts actually make a call to action. A call to action is simply a link to a form that collects user meta data and gets them to sign up for something or buy something. However to increase the odds that anyone will click a link and spend their valuable time giving you resources at a minimum it needs to look good and by easy to find. Adding a box around the link and giving it a background color that stands out from the text inside and the rest of its containing element is a great way to do that. CSS is what makes that all possible for you.

This book will not only contain the fundamental know how that will guide you in your quest of learning CSS. It will also provide extensive examples that will show you the code in use. Chapter Two contains a long list of valuable resources which includes all of the available properties as well as the accepted values for each. While reading through the material and studying the examples it is highly recommended that you try to decipher why each set of code works the way it does. It is also a great idea to plug the

code into an HTLM document of your own and play around with different values just to see what it does.

Like anything in life, mastering CSS takes practice. A lot of practice. All coding in general becomes easier as time goes on and you have gained more experience. CSS has grown up from a newly invented language to its more adult version of wider more impressive capabilities quickly. For that reason it can reasonably be assumed it will continue to do so. It is very important therefor to do your best to stay up to date with the changes in new releases as they roll out.

While the rules, selectors, and values you use are completely up to you as you seek to pursue your agenda. It is also advised to abide by World Wide Web Consortium standards. The easiest way to code to these standards that we teach in this book is to practice them in a habit forming manner. These standards are designed to save you and other programmer's time by making your code easier to read. This also makes your code easier for other programmers to may pick up the project after you to decipher. The World Wide Web Consortium also has a list of many other appropriate standards for many web development languages such as HTML. If you are using CSS to style an HTML document it is also advised to do your best to try and stay within the habit of using these beneficial and common coding standards. They will also make you less prone to mistakes and help you prevent frequent confusion. The more standardized the code you get used to writing is the quicker you will be able to pick up code written by foreign authors and understand what they are intending to have happen.

Clean code is also easier to check for bugs. While many consider this phase of coding in CSS to be boring it can pay off big in the long run. Whether they are small errors that are hard to notice or errors that alter the functionality of the intended project, the size of the error is often less important than the psychological response it triggers in your users. Things that look hacked or put together without effort can make your customers or users think your work is not important to you. It erodes your credibility and undermines the reliability of information you are trying to get across or receive with you HTML document. This makes it important to take enough pride in your work to test it thoroughly.

Testing your own work for errors, bugs, or any unintended behavior will also provide a great opportunity to learn your trade of coding in CSS far more in depth. Regardless of what level programmer you are currently you know that mistakes will be made. It is not an if they will be made but more of a when. Make sure things are placed where they are intended. Make sure they look like they are the right size. Make sure they are layered in the intended order. Make sure the intended appearance does not hide or complicate functionality. Make sure that the code is not doing anything redundant or needlessly

overlapping itself. This is just a short and brief list of things to look for in testing your code. You will like find a how-to guide that will walk you through testing your code more in depth. It is wise to use a guide or some form of checklist to make sure you don't accidentally look over anything.

While this book will give you everything you need to become a CSS expert if used properly and frequently. How much of it you take in and make it a point to practice it will be completely up to you. As such it should be clarified that this is a guide and not an all inclusive course. Meaning that the results you receive are dependent only on yourself.

I sincerely hope you will find this book helpful and fun enough to continue enjoying the possibilities offered by CSS. If you are studying this for career related reasons than I am confident you are well on your way to a bright and rewarding future. If not for a career, but instead a hobby this book will certainly guide you into a great experience full of challenges to overcome and experiences to grow from. Through CSS we can all fulfill accomplishments, achieve goals, make things to be proud of and show off. With that being said, I would like to leave it at that and wish you the best of luck on your quest to become a CSS expert.

Table of Contents

Part I – Beginner

1. Introduction to CSS
    - 1.1 What is CSS
    - 1.2 Why is CSS Needed
    - 1.3 What Does CSS Do
    - 1.4 How to use Style Sheets
    - 1.5 Cascading Orders
2. Selectors, Properties, and Values
    - 2.1 Defining HTML Elements with Selectors, Properties, Values, and Units
    - 2.2 Saving Time With the Universal Selector
    - 2.3 The Child Selector
    - 2.4 The Adjacent Selector
    - 2.5 The Attribute Selector
    - 2.6 Properties and Values
    - 2.7 Units
3. Colors, Text, and Backgrounds
    - 3.1 Colors
    - 3.2 Hexadecimal Notation
    - 3.3 RGB
    - 3.4 HSL and Alpha Transparency
    - 3.5 Text
4. Images, Padding, Margins, and Borders
    - 4.1 Images
    - 4.2 The Box Model
    - 4.3 Shorthand
    - 4.4 Padding
    - 4.5 Borders
    - 4.6 Margins

Part II – Intermediate

5. Class, ID Selectors, with Grouping, Nesting, and Pseudo Elements
    - 5.1 Classes and ID Selectors

5.2 Grouping
5.3 Nesting
5.4 Pseudo Classes
6. Shorthand Properties, Specificity, Pseudo Elements
6.1 Shorthand Properties
6.2 Specificity
6.3 Pseudo Elements
7. Box Display, Rounded Corners, Shadows, and Page Layout
7.1 Box Displays
7.2 Rounded Corners
7.3 Shadows
7.4 Page Layout

Part III - Advanced

8. Importing Style Sheets
8.1 The Link Tag
8.2 The @import Rule
8.3 The @media Rule
8.4 Embedding Fonts
9. Transitions and Gradients
9.1 Transitions
9.2 Selecting Multiple Properties
9.3 Gradients
9.4 Linear Gradients
9.5 Radial Gradients
10. Media Queries
10.1 Media Queries
10.2 Browser-Size Specific CSS
10.3 Orientations and Device Specific CSS

PART I - BEGINNERS

CHAPTER 1 – INTRODUCTION TO CSS

Chapter Objectives

- Learn who created CSS.
- Learn what CSS stands for.
- Learn what CSS is and why it is used.
- Learn what CSS can do.
- Learn the three different types of style sheets.
- Learn how to incorporate each type of stylesheet.
- Understand the opportune time to use each type of style sheet.
- What does cascading mean.

1.1 What is CSS?

CSS is a language created by the World Wide Web Consortium (W3C) that was designed specifically to format HTML documents. More specifically CSS is an acronym for Cascading Style Sheets. A style sheet is a type of template file that contains font, styling, formatting, and display settings that allow a file to give a standardized look to the intended document. We'll get back to these style sheets soon as there are various ways to incorporate them into your HTML document. CSS first appeared in HTML 4.0 in order to solve a big problem with earlier versions on HTML.

1.2 Why is CSS Needed?

So what is this big problem that CSS solved? Original versions of HTML never intended to use the tags that are needed for formatting a document. That's because HTML was designed to define the content of HTML documents. This was done using body, paragraph, and heading tags to name a few. These may be familiar to you, if not they would look like this:

```
<html>
<header>
```

```
</header>
<body>
<p> Hello World! </p>
<h1> This is an H1 heading font size. </h1>
  </body>
  <footer>
  </footer>
</html>
```

The problem with using these content tags to style a document is the same reason that most experienced CSS programmers don't like to use inline style sheets to style their documents. It is a very long and extensive process. Albeit even this basic form of styling using <font> and color attributes did not come out until HTML 3.2. As you can deduce from the amount of version upgrades it took to fix, this form of styling was an absolute nightmare for the programmer. This was because every content item on every page had to be restyled individually one at a time. It also created room for frequent errors that eroded the uniformity of a site. Without uniformity an HTML document can look unprofessional and quickly take great strides away from being user friendly.

1.3 What Does CSS Do?

CSS is what makes HTML websites look good, easy to use, and even disabled user friendly. This is done by allowing the programmer to customize fonts, content items sizes, spacing, colors, background, borders, placement, overlays, margins, boxes, formatting, link effects, buttons, and the list goes on. In newer versions like CSS3 you can even play with gradients, transitions, and other advanced features like transparencies.

1.4 How to Use Style Sheets

There are three types of style sheet you can use. Which one you use will be up to you, although there are situations which it would be advised to use a particular type. The three types of style sheets are inline style sheets, internal style sheets, and external style sheets. The difference between each being simply where the code is placed in relation to the HTML document that it will be styling.

We already covered why inline style sheets are discouraged as they basically defeat the entire purpose of even using CSS in the first place. Meaning they are inconvenient and tedious to use or modify on a large scale. That being said, in the rare case of a small HTML document with only one file and not many repetitive content pieces it could be faster and easier to just style the content tags as you write them. In this form of style sheet the style attributes are placed directly in the tag of the element they are effecting. This would look like:

```
<html>
<header>
</header>
<body> <p style=”color: red ; font: Calibri ;”>This will create red text using the Calibri font. </p>
  </body>
  <footer>
  </footer>
</html>
```

The second type of style sheet is an internal style sheet. In this type of style sheet the CSS code is placed directly inside the HTML file. The CSS is rendered by using the <style> </style> tags. While they can be placed in numerous areas of the document it is not only faster to load but also easiest to find and therefor advised to use these tags in one specific place. Then you can put all of the needed CSS inside those tags. Typically when using an internal style sheet the <style> tags are placed within the header section of the document. Placing yours there would be good practice and make it easier for other coders to find later on. This would look like:

```
<html>
<header>
<style>
p {
font: calibri ;
```

```
color: red ;
align: right ; }

.logo-image {
align: left ;
min-height: 120px ;
border: 2px solid #fff ;
}
</style>
</header>
<body>
  </body>
  <footer>
  </footer>
</html>
```

The third type of style sheet is an external style sheet. These are the most common due to being the most convenient especially on a document that needs to style multiple files each containing multiple uses of the same content tags. An external style sheet is placed external to the HTML document in its own file. Additionally, these types of style sheets should not include any HTML. External style sheet files always end in a .css file extension. The most common file name being simply style.css. However, custom.css, mobile.css, bootstrap.css, or anything else you make up will all work just the same as long as it is called from within the HTML document correctly. You can call the external file sheet by placing any of the following links or calls in the HTML documents <header> section:

```
  <html>
  <header>
<link rel=”stylesheet” type=”text/css” href=“…style.css” />
  </header>
```

```
    <body>
    </body>
    <footer>
    </footer>
</html>
```

You can also use the @import method as shown below:

```
    <html>
    <header>
    <style type="text/css"> @import url(…style.css) </style>
    </header>
    <body>
    </body>
    <footer>
    </footer>
    </html>
```

1.5 Cascading Orders

Cascading order is the system used to sort rules by which declarations are sorted out and given priority to prevent conflicts from affecting the presentation of content. The rules are always sorted in this specific order so as to remain predictable. Using a Boolean (true or false) method the sorting goes in order so if rule 1 is not met then it sorts by rule two if it is met the search ends, if not it sorts by rule three and so on. The rules are as follows:

1. Search for all declarations that are relevant to a certain attribute. Declare the specific attribute style if the attribute matches the element, if there isn't any let the element inherit the parent property if there is not an initial value.
2. Sort by weight (! Important) weight attributes will take priority over normal weight attributes.
3. Original rules with normal weight attributes declared in the origin author's style

sheet will override rules with normal weight declared in the user's personal style sheets. Rules with increased weight in the user's personal style sheet will override rules with normal weight declared in the author's style sheet. Rules with increased weight declared in the author's sheet will override rules with increased weight from the user's personal sheets. Both the author's and user's rules will override the default style sheet.

4. Sort by an attributes specificity. More specific selectors will override less specific selectors. ID-selectors are the most specific. The second most specific are classified contextual selectors. Third are class selectors. Contextual selectors gain more and more weight as more of them are used.
5. Sort by the order specified. If two rules have the same weight, the latter will override the earlier rule. Style sheet types are sorted as follows, inline style sheets override all other styles. Internal style sheets will override linked and imported style sheets. The linked stylesheet will override a sheet called with the @import statement. If multiple external style sheets are used they will cascade in the same order that they are imported. It is also possible to use inline, internal, and external style sheets all at once. They will simply cascade in order of priority.

Practical Use Exercise Questions

1. Who created CSS?
2. What does the acronym CSS stand for?
3. In what version of HTML did CSS first appear?
4. What was the big problem that CSS solved?
5. What types of content can CSS customize?
6. What do newer versions of CSS allow to modify?
7. What are the three types of style sheet?
8. Name two ways of calling an external style sheet?
9. Describe what is meant by cascading?
10. Which type of style sheet overrides all others?
11. Which type of style sheet is overridden by any other?

Chapter Summary and Exercise Solutions

- CSS was created by the World Wide Web Consortium.
- CSS stands for Cascading Style Sheets.
- CSS first appeared in HTML 4.0

- CSS made formatting and standardizing large projects quicker and easier by allowing programmers to style every item of similar attribute across entire sites from one file with the same statement.
- CSS can style any HTML content.
- CSS3 allows gradients, transitions, and transparencies.
- The three types of style sheets are inline style sheets, internal style sheets, and external style sheets.
- External style sheets can be called using a link statement or an @import statement.
- Cascading refers to the order in which style sheets and their statements take priority in order to eliminate confusion in the case of conflicting statements.
- Inline style sheets override all other style sheets.
- External style sheets are overridden by any other type of style sheet.

CHAPTER 2 – SELECTORS, PROPERTIES, AND VALUES.

Chapter Objectives

- Learn what selectors can name.
- Learn how to use a Selector.
- Learn how to use a Universal Selector.
- Learn how to use a Child Selector.
- Learn how to use Adjacent Selectors.
- Learn how to use Attribute Selectors.
- Learn how to use a Property.
- Learn some common Property Units.
- Learn how to use a Value and a unit.
- Learn the purpose of the z index.

2.1 Defining HTML Components with Selectors, Properties, Values, and Units

A selector can be used to name any specific HTML item or component that you want to style. To go more in depth selectors are modified by defining their properties with values which are defined in units. So one more time since that was a long string on explanation. Selectors are defined with properties. Properties are defined with values. Values are defined with units. All of these combined make a statement.

Now what do you do with all these selectors, properties, values, and units? Fortunately all CSS statements follow the same format. The selector is placed first in the CSS statement. Then after the selector inside the opening curly brace { the property is followed with a colon, after the colon the value is stated in units followed with a semi-colon and the closing curly brace. Altogether it would look like this:

```
button {width: 100 px;}
```

There are many properties, values, and units that will modify a given selector. You can also save more time by adding as many properties as you need to a selector by just

putting the next property after the closing semi-colon of the previous property. Don't worry as usual we will show you how this should look. In the example below the tag <h1> will be the selector we have chosen to style. This will be done by using the properties font-family, color, border, and border-width. While these properties will be defined using the values times new roman, blue, solid, and 2px respectively. The HTML for this example may look like:

```
<html>
<header>
</header>
<body>
<h1> Lets make this heading blue times-new-roman with a solid border of 2px. </h1>
</body>
<footer>
</footer>
</html>
```

To style the above HTML we can use the CSS below inserted into the style sheet with a .css file extension:

```
h1 {font-family: times new roman; color: blue; border: solid; border-width: 2px;}
```

Also in CSS, spacing is not used by the parser so it is treated like it is not even there. This means the above example will work the exact same as either of the statements below, although you may note that the third is a lot easier on the eyes to read. This is the standard format used by most CSS programmers for that reason.

```
h1{font-family: times new roman ; color: blue ; border: solid ; border-width: 2px ; }
```

or

```
h1{
        font-family: times new roman ;
        color: blue ;
        border: solid ;
        border-width: 2px ;
}
```

2.2 Saving Time with the Universal Selector

In CSS you can often save time by using the universal selector. The universal selector is an asterisk (“ * “). This is hugely convenient when you want to target everything without having to go through all the trouble of writing them all out one by one. Using the asterisk by itself as a selector will set the properties you place within the curly braces globally throughout the whole page.

This is rarely needed but extremely convenient when it is. Typically the universal selector is only used to reset browser display settings in order to insure your file will display the same in all browsers. Thanks to cascading order you can reset your browser then override the reset whenever you want by placing this statement at the very top of your style sheet. Then everything underneath it will have a higher priority. While a complete browser reset would be more inclusive a basic reset statement using an asterisk could be as simple as the one below. Also notice that when a values defining unit is zero, units are not needed to define the value. This is because the value zero is a universal measurement.

```
* {
Margin: 0 ;
Padding: 0 ;
}
```

You can also use the asterisk as a descendant of a selector by placing it after the selector but before the curly braces. This allows you to define everything within that selector. In some cases where you may have many descendants of a similar selector that you want to all have common traits it would be fastest to style them all at once. Links for example often are styled with descendants such as active, hover, focus, and visited. Since a link in HTML uses the tag <a> links are defined in CSS using the selector (“ a “). To visualize this being used to save keystrokes we will show you below:

```
a visited, focus, hover, active {style: underlined ; }
```

Or you could use:

```
a * {style: underlined ; }
```

Either of the above examples will underline anchor selectors which in HTL will look like:

```
<html>
<header>
</header>
<body>
<a href=”domainname.com” title=”domain-link”> Click this Link </a>
</body>
<footer>
</footer>
</html>
```

Now you could use the first statement and it will work. However, the second statement does the same thing and will work, but it saves you some typing. Trust me, I know you’re thinking, “Ok this saved four words big deal!” The thing is some style sheets can be thousands of lines of code, which equates to tens of thousands of words. That is a lot of typing and in your career you may make thousands of stylesheets. So work smarter, not harder. A few words here and there add up to a lot of time saved in the long run of the big picture.

2.3 The Child Selector

Anything that is nested within something else would be considered a child. In CSS a child is shown by putting it after a greater than sign (“ > “). People often mix up child selectors with descendant selectors. The first noticeable difference being that a

descendant selector does not have a greater than sign. The in depth difference is that a child selector will only match with an item if it is a direct child of the first selector, or basically it will only go one level deep in a nested statement. A descendant selector will match the item regardless of how many levels deep it is as long as it is nested anywhere inside the first selector.

A great example of when the child selector would be useful is in a list. You may want to style the list's headers without allowing the larger fonts styling to cascade down to other list items. The HTML of this list might look like:

```
<html>
<header>
</header>
<body>
<ul id="types_of_selectors">
        <li>Child Selectors
                <ul>Descendant Selectors</ul>
                <ul>Adjacent Selectors</ul>
        </li>
</body>
<footer>
</footer>
</html>
```

Now the fun part. First let's style the list! Using CSS let's make only the list item "Child Selectors" show up in red 16pt font while the rest will remain un-styled. To do this it will look like:

```
#types_of_selectors > li {font-color: red; font-size: 16pt ; }
```

Without the greater than sign our statement would style the entire list to use red 16pt font instead of just the header. This statement would simply look like:

```
#types_of_selectors li {font-color: red; font-size: 16pt ; }
```

2.4 The Adjacent Selector

Using an adjacent selector you can select an element that follows directly after another element. This is done by using the plus sign (“ + “). The adjacent selector is most commonly used to style depending on context. Maybe you have most of your pages arranged in the order of <h1> tags followed by an image. However, in a few instances you may have your <h1> tag followed by a list or a <p> tag which might throw off your spacing. So to correct these specific elements without messing up everything else we will simply use the adjacent selector. In the example below only paragraphs the come directly after the <h1> tag will be styled blue. The HTML is an example of a paragraph that would be affected by this CSS.

```
<html>
<header>
</header>
<body>
<h1> This H1 text is the parent of the Selector. </h1>
<p> This paragraph will be styled by using an Adjacent Selector. </p>
</body>
<footer>
</footer>
</html>
```

The CSS to style the above paragraph is below:

```
h1 + p {font-color: blue ; }
```

Also considered a form of adjacent selector is the sibling selector. The sibling selector functions the same way as a descendant selector does to the child selector, it will select

the element more than one level deep and style it as long as it comes after the first element. The sibling selector is specified with the tilde accent mark (“ ~ “). In the example below we will use a sibling selector to style a paragraph that does not come directly after the <h1> tag.

```
<html>
<header>
</header>
<body>
<h1> This H1 text is the parent of both selectors. </h1>
<h2> This H2 subheading would be the adjacent selector. <h2>
<p> This is the paragraph we will style using the sibling selector. </P>
</body>
<footer>
</footer>
</html>
```

The CSS to style the above HTML will look like:

```
h1 ~ p {text-color: blue ; }
```

2.5 The Attribute Selector

Using an attribute selector you can style elements that have a specified attribute value. This is commonly used in styling boxes, buttons, and input forms. Attributes are placed in the square brackets [ ]. However, there are 7 ways to use the attribute selector for us to go over. By placing only the name of the attribute in the square brackets you can style that certain attribute. This will look like:

```
button[submit] { background-color: red ; }
```

The HTML for the above CSS may look like:

```
<html>
<header>
</header>
<body>
<input type-“submit” value=”Submit”>
</body>
<footer>
</footer>
</html>
```

The second way to use the [attribute=value] selector is to style an attribute with a value. An example of this would look like:

```
a[target=”_blank”] { background-color: red ; }
```

The HTML that the above example would target may look like:

```
<html>
<header>
</header>
<body>
<a href=”website.com” target=”blank” title=”This will open in a black page”>
Click here</a>
</body>
<footer>
</footer>
</html>
```

The third way to use the attribute selector is to use the [attribute~=value] selector. This is used to style an attribute with a specific word as a value. This would look like:

```
[title~=”value”] { border: 1px solid yellow ; }
```

The HTML for this example may look like:

```
<html>
<header>
</header>
<body>
<a href=”/value.html” title=”value”> Save Now! </a>
</body>
<footer>
</footer>
</html>
```

The fourth way to use the attribute selector is to use the [attribute|=value] selector. This is used to style elements that have a specified attribute beginning with a specified value. Be sure to use a value that is a whole word either as one word or use a hyphen (“ – “) to split two words up. The example below will select elements with a class attribute value that begins with “featured”:

```
[class|=”featured”] { background-color: blue ; }
```

The HTML that the above rule would target could be:

```
<html>
<header>
```

```
</header>
<body>
<div class=”featured’’>
<p> This Paragraph will be colored blue since it is in the featured div. </p>
</div>
</body>
<footer>
</footer>
</html>
```

The fifth way to use an attribute selector is with the [attribute^=value] selector. This is used to select elements whose attribute value begins with a specific value. In this example we will select all of the elements with a class attribute that begin with “feat”. In this type of selector the value does not have to be a full word.

```
[class^=”feat”] { background-color: blue ; }
```

The example above will style either of the below HTML div statements:

```
<html>
<header>
</header>
<body>
<div class=”featured_intro”> </div>
<div class=”featured_paragraph”> </div>
</body>
<footer>
</footer>
</html>
```

The sixth way to use the attribute selector is with the [attribute$=value] selector. This selector is used to style an element when the attribute value ends with a specified value. The value does not have to be a whole word for this selector to work.

```
[class$="contact"] {background: yellow ; }
```

An example of the HTML that the above rule would style could look like:

```
<html>
<header>
</header>
<body>
<form id="form_contact" action="#" method="POST"
enctype="multipart/form-data">
<div id="row"> <label for="name"> Your Name: </label> <br />
<input id="name" class="input" name="name" type="text" value="name"
size="30"  /> <br /> </div>
<div id="row"> <label for="email"> Your Email: </label><br />
<input id="email" class="input" name="email" type="text" value="email"
size="30"  /> <br /> </div> <br />
<input type="submit" value="Submit" />
</body>
<footer>
</footer>
</html>
```

The last way to use an attribute selector is with the [attribute*=value] selector. This selector is used to choose an element when its attribute value contains a specified value. The following example will style all elements with a class attribute that contains "rwp", with this selector the value does not have to be a whole word.

```
[class*=”rwp”] {background: green ; }
```

The example above will style both HTML divs below since they both contain the value of rwp.

```
<html>
<header>
</header>
<body>
<div class=”rwp-portfolio”> <p> This paragraph contains the portfolio. </p>
</div>
<div class=”rwp-resume”> <p> This paragraph contains the resume. </p>
</div>
</body>
<footer>
</footer>
</html>
```

2.6 Properties and Values

Properties define which attribute of the given selector will be modified. These are used to change just about any aspect of an HTML document that you can think of. When using properties in your code they are always typed with all lower case and any time there is a space between the words of a property they are separated by a hyphen “ – “ and not a space. However, in the interest of keeping the list below readable the keywords are properly capitalized and separated with a space. After the colon you will find a brief description of each definition as well as the accepted values for each property inside parenthesis. If the value is numerical then the accepted units will be included in parenthesis instead. The inclusive list of CSS properties are listed below:

Align content: This property aligns a flexible container’s items vertically. Values can be left, right, or center. (stretch, center, flex start, flex end, space between, space around, initial, inherit)

Align items: This property sets the default alignment for items that are inside flexible

containers. (stretch, center, flex start, flex end, baseline, initial, inherit)

Align self: This property sets the alignment for the selected item inside of a flexible container. (auto, stretch, center, flex start, flex end, space between, space around, initial, inherit)

Animation: This property is shorthand for the six animation properties name, duration, timing-function, delay, iteration-count, and direction.

Animation delay: This property sets a delay for the start of an animation. (time, initial, inherit)

Animation direction: This property defines whether an animation will play in reverse direction or in alternate cycles. (normal, reverse, alternate, alternate reverse, initial, inherit)

Animation duration: This property defines how long an animation takes to complete one cycle in seconds or milliseconds. (time, initial, inherit)

Animation fill mode: This property sets a style for an element when the animation is not playing. (none, forwards, backwards, both, initial, inherit)

Animation iteration count: This property sets the number of time an animation should be played. (number, infinite, initial, inherit)

Animation name: This property sets a name for the @keyframes animation. (keyframename, none, initial, inherit)

Animation play state: This property sets whether an animation is running or paused. (paused, running, initial, inherit)

Animation timing function: This property sets the speed curve or the rate at which the speed of an animation changes. (linear, ease, ease in, ease out, ease in out, cubic bezier(n,n,n,n), initial, inherit)

Backface visibility: This property defines whether or not an element should be visible when it is not facing the screen. (visible, hidden, initial, inherit)

Background: This property sets all background declarations.

Background attachment: This property sets whether a background image is fixed or scrolls with the page. (scroll, fixed, local, initial, inherit)

Background clip: This property sets the painting area of the background. (border box, padding box, content box, initial, inherit)

Background color: This property sets the background color. (color, transparent, initial, inherit)

Background image: This property sets the background image. (url('url'), none, initial, inherit)

Background origin: This property sets where the background image is positioned. (border box, padding box, content box, initial, inherit)

Background position: This property sets the position of the background. (left top, left center, left bottom, right top, right center, right bottom, center top, center center, center bottom, x% y%, xpos ypos, initial, inherit)

Background repeat: This property sets how a background image will be repeated. (repeat, repeat x, repeat y, no repeat, initial, inherit)

Background size: This property sets the size of the background image. (auto, length, percentage, cover, contain, initial, inherit)

Border: This property sets all border properties.

Border bottom: This property sets all the bottom border properties at once. (border bottom width, border bottom style, border bottom color, initial, inherit)

Border bottom color: This property sets the color of the bottom border. (color, transparent, initial, inherit)

Border bottom left radius: This property decides the shape of the borders bottom left corner. (length, %, initial, inherit)

Border bottom right radius: This property decides the shape of the borders bottom right corner. (length, %, initial, inherit)

Border bottom style: This property sets the style of the bottom border. (none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, initial, inherit)

Border bottom width: This property sets the width of the bottom border. (medium, thin, thick, length, initial, inherit)

Border collapse: This property sets whether or not the table borders are collapsed into a single border or detached. (separate, collapse, initial, inherit)

Border color: This property sets the color of the border. (color, transparent, initial, inherit)

Border image: This property is a shorthand property that sets all the border-image properties.

Border image outset: This property sets the amount that the border image area extends beyond the border box. (length, number, initial, inherit)

Border image repeat: This property sets whether the border image should get

repeated, rounded, or stretched. (stretch, repeat, round, space, initial, inherit)

Border image slice: This property sets how to slice the border image. (number, %, fill, initial, inherit)

Border image source: This property declares the path for the image to be used as a border. (none, image, initial, inherit)

Border image width: This property sets the width of the border image. (length, number, %, auto, initial, inherit)

Border left: This property sets all the left border properties at once. (border left width, border left style, border left color, initial, inherit)

Border left color: This property sets the color of the left border. (color, transparent, initial, inherit)

Border left style: This property sets the style of the left border. (none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, initial, inherit)

Border left width: This property sets the width of the left border. (medium, thin, thick, length, initial, inherit)

Border radius: This property sets the borders radius properties. (length, %, initial, inherit)

Border right: This property sets all the right border properties at once. (border right width, border right style, border right color, initial, inherit)

Border right color: This property sets the color of the right border. (color, transparent, initial, inherit)

Border right style: This property sets the style of the right border. (none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, initial, inherit)

Border right-width: This property sets the width of the right border. (medium, thin, thick, length, initial, inherit)

Border style: This property sets the style of all four borders. (none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, initial, inherit)

Border top: This property sets all the top border properties at once. (border top width, border top style, border top color, initial, inherit)

Border top color: This property sets the color of the top border. (color, transparent, initial, inherit)

Border top left radius: This property decides the shape of the borders top left corner. (length, %, initial, inherit)

Border top right radius: This property decides the shape of the borders top right corner. (length, %, initial, inherit)

Border top style: This property sets the style of the top border. (none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, initial, inherit)

Border top width: This property sets the width of the top border. (medium, thin, thick, length, initial, inherit)

Border width: This property sets the width of the four borders. (medium, thin, thick, length, initial, inherit)

Box decoration-break: This property sets the behavior of the background and the element of a page break or line break.

Box shadow: This property attaches one or more shadows to an element. (none, h-shadow, v-shadow, blur, spread, color, inset, initial, inherit)

Box sizing: This property tells the browser the boxes height and width. (content box, border box, initial, inherit)

Bottom: This property sets the bottom position of a positioned element. (auto, length, %, initial, inherit)

Caption side: This property sets the placement of a table caption. (top, bottom, initial, inherit)

Clear: This property sets which side or sides of an element where there are other elements need to be kept clear. (none, left, right, both, initial, inherit)

Clip: This property clips an absolutely positioned element. (auto, shape, initial, inherit)

Color: This property sets the color of text. (color, initial, inherit)

Column count: This property sets the number of columns an element should be cut up into. (number, auto, initial, inherit)

Column fill: This property sets how to fill columns, whether they should be balanced or not. (balance, auto, initial, inherit)

Column gap: This property sets the gap between columns. (length, normal, initial, inherit)

Column rule: This property is used as a shorthand property for setting all of the column rules at once. (column rule width, column rule style, column rule color, initial, inherit)

Column rule color: This property sets the color of the rule between columns. (color, initial, inherit)

Column rule style: This property sets the style of the rule between columns. (none,

hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, initial, inherit)

Column rule width: This property sets the width of the rule between columns. (medium, thin, thick, length, initial, inherit)

Column span: This property sets how many columns an element should span across. (1, all, initial, inherit)

Column width: This property sets a suggested optimal width for the columns. (auto, length, initial, inherit)

Columns: This property sets column width and count. (auto, column width, column count, initial, inherit)

Content: This property is used with before and after elements to insert content. (normal, none, counter, attribute, string, open-quote, close-quote, no open quote, no cluse quote, url(url)initial, inherit)

Counter increment: This property increments one or more counter values. (none, id number, initial, inherit)

Counter reset: This property creates or resets one or more counters. (none, name, number, initial, inherit)

Curser: This property sets the type of cursor that will be displayed when hovering over an element. (alias, all scroll, auto, cell, context menu, col resize, copy crosshair default e resize, ew resize, grab, grabbing, help, move, n resize, ne resize, nesw resize, ns resize, nw resize, nwse resize, no drop, none, not allowed, pointer, progress, row resize, s resize, se resize, sw resize, text, URL, vertical text, w resize, wait, zoom in, zoom out, initial, inherit)

Direction: This property sets the text direction. (ltr, rtl, initial, inherit)

Display: This property sets how a specific HTML element should be displayed. (inline, block, flex, inline block, inline flex, inline table, list item, run in, table, table caption, table column group, table header group, table footer group, table row group, table cell, table column, table row, none, initial, inherit)

Empty cells: This property sets borders and background on empty cells in a table. (show, hide, initial, inherit)

Flex: This property sets the length of the item relative to the rest of the flexible items in the same container. (flex grow, flex shrink, flex basis, auto, initial, none, inherit)

Flex basis: This property sets the initial length of a flexible item. (number, auto, initial, inherit)

Flex direction: This property sets the direction of the flexible items. (row, row reverse,

column, column reverse, initial, inherit)

Flex flow: This property is a shorthand property for the flex-direction and the flex-wrap properties. (flex direction, flex wrap, initial, inherit)

Flex grow: This property sets how much the item will grow relative to the rest of the flexible items inside the same container. (number, initial, inherit)

Flex shrink: This property sets how much the item will shrink relative to the rest of the flexible items inside the same container. (number, initial, inherit)

Flex wrap: This property sets whether the flexible items should wrap or not. (nowrap, wrap, wrap reverse, initial, inherit)

Float: This property sets whether or not a box should float. (none, left, right, initial, inherit)

Font: This property sets all the font properties. (font style, font variant, font weight, font size, line height, font family, caption, icon, menu, message box, small caption, status bar, initial, inherit)

@font face: This property lets you use fonts other than the "web safe" fonts. You just define a name for the font and then point to the font file URL. (font family, src, font stretch, font style, font weight, Unicode range)

Font family: This property sets the font for an element. (family name, initial, inherit)

Font size: This property sets the size of a font. (medium, xx small, x small, small, large, x large, xx large, smaller, larger, length, %, initial, inherit)

Font size adjust: This property lets you set a second font to use as a backup in case the first font is not available. (number, none, initial, inherit)

Font stretch: This property allows you to make text wider or narrower. (ultra condensed, extra condensed, condensed, semi condensed, normal, semi expanded, expanded, extra expanded, ultra expanded, initial, inherit)

Font style: This property lets you set the style of a font. (normal, italic, oblique, initial, inherit)

Font variant: This property converts all lowercase letters to uppercase in a smaller font size. (normal, small caps, initial, inherit)

Font weight: This property lets you set how thick or thin the font is displayed. (normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900, initial, inherit)

Hanging punctuation: This property sets if a punctuation mark can be placed outside the line box at the start or end of a line of text. (none, first, last, allow end, force end,

initial, inherit)

Height: This property sets the height of an element. (auto, length, %, initial, inherit)

Justify content: This property aligns a flexible containers items horizontally when the items are different sizes. (flex start, flex end, center, space between space around, initial, inherit)

@keyframes: This rule sets the animation code. This is done by gradually changing one set of css to another. (animationname, keyframes selector, css styles)

Left: This property sets the left edge of an absolutely positioned element to the left or right on the left edge of its containing element. (auto, length, %, initial, inherit)

Letter spacing: This property increases or decreases the space between characters in a text. (normal, length, initial, inherit)

Line height: This property sets the line height. (normal, number, length, %, initial, inherit)

List style: This property sets all list properties. (list style type, list style position, list style image, initial, inherit)

List style image: This property replaces the list-item marker with an image. (none, url, initial, inherit)

List style position: This property sets whether or not the list item markers should appear inside or outside the content flow. (inside, outside, initial, inherit)

List style type: This property sets the type of list item marker in a list. (disc, Armenian, circle, cjk ideographic, decimal, decimal leading zero, Georgian, Hebrew, hiragana, hiragana iroha, lower alpha, lower greek, lower latin lower roman, none, square, upper alpha, upper latin, upper roman, initial, inherit)

Margin: This property sets all the margin properties. (length, %, auto, initial, inherit)

Margin bottom: This property sets the bottom margin of an element. (length, %, auto, initial, inherit)

Margin left: This property sets the left margin of an element. (length, %, auto, initial, inherit)

Margin right: This property sets the right margin of an element. (length, %, auto, initial, inherit)

Margin top: This property sets the top margin of an element. (length, %, auto, initial, inherit)

Max height: This property is used to set the maximum height of an element. (none,

length, %, initial, inherit)

Max width: This property is used to set the maximum width of an element. (none, length, %, initial, inherit)

@media: This rule defines different style rules for different media types/devices. (all, aural, braille, embossed, handheld, print, projection, screen, speech, tty, tv, aspect ratio, color, color index, device aspect ratio, device height, device width, grid, height, max aspect ratio, max color, max color index, max device aspect ratio, max device height, max device width, max height, max monochrome, max resolution, max width, min aspect ratio, min color, min color index, min device width, min device height, min height, min monochrome, min resolution, min width, monochrome, orientation, resolution, scan width)

Min height: This property sets the minimum height of an element. (length, %, initial, inherit)

Min width: This property sets the minimum width of an element. (length, %, initial, inherit)

Nav down: This property tells where to navigate when using the down arrow key. (auto, id, target name, initial, inherit)

Nava index: This property tells the sequential navigation order or tabbing order of an element. (auto, number, initial, inherit)

Nav left: This property sets where to navigate when using the arrow left navigation key. (auto, id, target name, initial, inherit)

Nav right: This property sets where to navigate when using the arrow right navigation key. (auto, id, target name, initial, inherit)

Nav up: This property sets where to navigate when using the arrow up navigation key. (auto, id, target name, initial, inherit)

Opacity: This property sets the opacity level for an element. (number, initial, inherit)

Order: This property sets the order of a flexible item relative to the rest of the flexible items. (number, initial, inherit)

Outline: This property draws a line around elements. (outline color, outline style, outline width, initial, inherit)

Outline color: This property sets the color of an outline. (invert, color, initial, inherit)

Outline offset: This property adds space between an outline and the edge or border of an element. (length, initial, inherit)

Outline style: This property sets the style of an outline. (none, hidden, dotted, dashed, solid, double, groove, ridge, inset, outset, initial, inherit)

Outline width: This property sets the width of an outline. (medium, thin, thick, length, initial, inherit)

Overflow: This property sets what happens when content overflows an elements box. (visible, hidden, scroll, auto, initial, inherit)

Overflow x: This property sets what will be done with the left or right edges of the content if and when it overflows outside of the elements content area. (visible, hidden, scroll, auto, initial, inherit)

Overflow y: This property sets what will be done with the top or bottom edges of the content if and when it overflows outside of the elements content area. (visible, hidden, scroll, auto, initial, inherit)

Padding: This property sets all the padding properties. (length, %, initial, inherit)

Padding bottom: This property sets the bottom padding space of an element. (length, %, initial, inherit)

Padding left: This property sets the left padding space of an element. (length, %, initial, inherit)

Padding right: This property sets the right padding space of an element. (length, %, initial, inherit)

Padding top: This property sets the top padding space of an element. (length, %, initial, inherit)

Page break after: This property sets whether a page break should occur after a specific element. (auto, always, avoid, left, right, initial, inherit)

Page break before: This property sets whether a page break should occur before a specific element. (auto, always, avoid, left, right, initial, inherit)

Page break inside: This property sets whether a page break should occur inside a specific element. (auto, avoid, initial, inherit)

Perspective: This property defines how many pixels a 3D element is placed away from the view which changes how the 3D element is viewed. (length, none, initial, inherit)

Perspective origin: This property defines where a 3D element is based on the x-axis and y-axis which changes the bottom position of 3D elements. (x axis, y axis, initial, inherit)

Position: This property sets the type of positioning method used for an element which

can be static, relative, absolute, or fixed. (static, absolute, fixed, relative, initial, inherit)

Quotes: This property sets the type of quotation marks that are used in quotations. (none, string string string string, initial, inherit)

Resize: This property sets whether or not an element will allow the user to resize the element. (none, both, horizontal, vertical, initial, inherit)

Right: This property sets the right edge of an absolutely positioned element to the left or right of the right edge of its containing element. (auto, length, %, initial, inherit)

Tab size: This property sets the length of the space used for the tab character. (number, length, initial, inherit)

Table layout: This property sets the table layout algorithm that is used for a table. (auto, fixed, initial, inherit)

Text align: This property sets the horizontal alignment of text. (left, right, center, justify, initial, inherit)

Text align last: This property sets how to align the last line of a text. (auto, left, right, center, justify, start, end, initial, inherit)

Text decoration: This property sets the decoration that is added to the text. (none, underline, overline, line through, initial, inherit)

Text decoration color: This property sets the color of the decoration that is added to the text. (color, initial, inherit)

Text decoration line: This property sets what type of line the text decoration will have if there is a line. (none, underline, overline, line through, initial, inherit)

Text decoration style: This property specifies how the line will display if there is a line. (solid, double, dotted, dashed, wavy, initial, inherit)

Text indent: This property sets the indentation of the first line in a block of text. (length, %, initial, inherit)

Text justify: This property sets the justification method to use when text-align is set to justify. (auto, inter word, inter ideograph, inter cluster, distribute, kashida, trim, none, initial, inherit)

Text overflow: This property sets how overflowed content that is not displayed should be signaled to the user. (clip, ellipsis, string, initial, inherit)

Text shadow: This property adds a shadow to the text. (h shadow, v shadow, blur radius, color, none, initial, inherit)

Text transform: This property sets the capitalization of text. (none, capitalize, uppercase, lowercase, initial, inherit)

Top: This property sets the top edge of an absolutely positioned element above or below the top edge of its containing element. (auto, length, %, initial, inherit)

Transform: This property applies 2D or 3D transformation to an element. (none, matrix(n,n,n,n,n,n), matrix3d(n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n), translate(x,y), translate3d(x,y,z), translateX(x), translate(y), translate(z), scale(x,y), scale3d(x,y,z), scaleX(x), scaleY(y), scaleZ(z), rotate(angle), rotate3D(x,y,z,angle), rotateX(angle), rotateY(angle), skew(x angle, y angle), skewX(angle), skewY(angle), perspective(n), initial, inherit)

Transform origin: This property allows you to change the position of transformed elements. 2D transformations can change the x-axis and y-axis of an element. 3D transformations can change the z-axis of an element as well. (x axis, y axis, z axis, initial, inherit)

Transform style: This property sets how nested elements are rendered in 3D space. This can only be used with the transform property. (flat, preserve 3d, initial, inherit)

Transition: This property allows you to set the transition properties of duration, timing-function, and transition delay. (transition property, transition duration, transition timing function, transition delay, initial, inherit)

Transition delay: This property sets when the transition effect will start. (time, initial, inherit)

Transition duration: This property sets how long it takes a transition to complete in seconds or milliseconds. (time, initial, inherit)

Transition property: This property sets the name of the CSS property the transition effect will happen to. (none, all, property, initial, inherit)

Transition timing function: This property sets the speed curve of the transition effect. (ease, linear, ease in, ease out, ease in out, cubic bezier(n,n,n,n), initial, inherit)

Unicode bidi: This property is used together with the direction property to set or return whether the text should be overridden to support multiple languages in the same document. (normal, embed, bidi override, initial, inherit)

Vertical align: This property sets the vertical alignment of an element. (baseline, length, %, sub, super, top, text-top, middle, bottom, text bottom, initial, inherit)

Visibility: This property sets whether or not an element is visible. (visible, hidden, collapse, initial, inherit)

White space: This property sets how the white space inside an element is handled. (normal, nowrap, pre, pre line, pre wrap, initial, inherit)

Width: This property sets the width of an element. (auto, length, %, initial, inherit)

Word break: This property sets line breaking rules for non-CJK scripts. (normal, break all, keep all, initial, inherit)

Word spacing: This property increases or decreases the white space between words. (normal, length, initial, inherit)

Word wrap: This property allows long words to be able to be broken down and wrapped into the next line of text. (normal, break word, initial, inherit)

Z index: This property sets the stack order of an element. (auto, number, initial, inherit)

2.7 Units

In CSS you can often describe values using units. Most properties that affect size or distance or any other type of measurement will require using a unit. There are two types of length units they are relative and absolute. Some properties will even allow negative lengths. The most common units are % (percent), px (pixels), cm (centimeters), mm (millimeters), in (inch), pt (points).

Practical Use Exercise Questions

1. What can a selector do?
2. What symbol is used as the Universal Selector?
3. What symbol is used as the Child Selector?
4. What symbol is used as the Adjacent Selector?
5. What are Attribute Selectors commonly used for?
6. What property sets text color?
7. What does the unit px stand for?
8. What does the z index property do?

Chapter Summary and Exercise Solutions

- A selector can select any HTML component or item.
- The asterisk (“ * “) is used as the Universal Selector.
- The greater than sign (“ > “) is used as the Child Selector.

- The plus sign (“ + “) is used as the Adjacent Selector.
- Attribute Selectors are commonly used to style boxes, buttons, and input forms.
- The color property sets text color.
- The px unit stands for pixels.
- The z index property sets the stack order of an element.

CHAPTER 3 – COLORS, TEXT, AND BACKGROUNDS

Chapter Objectives

- Learn hexadecimal notation.
- Learn the format of RGB and HSL values.
- Learn how to add Alpha Transparency to RGB and HSL statements.
- Learn how to set the background color.
- Learn how to alter the size and shape of text.
- Learn how to set a block-level background image.
- Learn the placement of properties in the box model.

3.1 Colors

With CSS you have 16,777,216 colors at your fingertips to play with. In the old days developers were limited to 216 web safe colors since computer monitors of the time could only display a max of 256 colors. This limitation led to the use of web safe colors. Now day's web safe colors aren't an issue since display technology has improved so much that you now have the freedom to choose from over 16 million colors. These colors are called by using the color property followed by any one of either their name, an RGB value, or a hex code. You can also alter the background color using any one of the same methods simply by placing them after the background-color property.

Color is displayed on a computer screen using red, green, and blue light just like in the rest of the world. CSS does allow you to choose a color simply by its name as long as it is one of the predefined color names. The predefined color names are aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, yellow, and transparent. Other than black and white, these names are rarely used in modern development because they are so specific and limiting.

3.2 Hexadecimal Notation

One way that the rest of the colors can be called is by assigning hexadecimal notations

to each colors RGB value in order to confer its red, green, and blue color values. A hexadecimal is a base-16 number system. I am sure you are used to using a normal decimal system which is known as base-10 meaning the digits go from 0 to 9. The difference is that in a hexadecimal system digits go from 0 to f. This may be a little confusing but think of it as 0 through 9 then instead of 10 we switch to alphabet characters a through f. Hex values range from 0 up to 255. The lowest value 0 is known as hex 00 while the highest 255 is known as hex ff. Hex values are typically written as three double digit numbers that start with a # sign. However, they can also be written as three-digit versions which are just a compressed version of the 6 digit. Therefor the normal hexadecimal format would look like:

```
p { background-color: #ff0099 ; }
```

While the compressed version would simply be:

```
p { background-color: #f09 ; }
```

The HTML that would be styled by the above rules could look like:

```
<html>
<header>
</header>
<body>
<p> This paragraph will have a background color of #f09. </p>
</body>
<footer>
</footer>
</html>
```

Many consider the three digit version easier to read. That is because the digits act like the RGB values. The first digit is the red value, the second digit is the green value, and the third digit is the blue value. Despite the three digit version being easier to use, the

six digit version offers more control over the specific shade of the color you choose.

3.3 RGB

RGB is basically the hex code just un-encrypted so that it is written out in a regular number format. It is also important to mention the instead of a pound sign RGB uses a lowercase rgb with the red, green, and blue values following inside of parenthesis. Similar to hex values rgb values range from 0 up to 255 with 0 signifying none of that color added and 255 meaning the highest level of that color added. To create a red color we could then easily set the red value to 255 and the other green and blue values to 0. This would look like:

```
#title {color: rgb(255, 0, 0) }
```

The HTML the example above would turn red could look like:

```
<html>
<header>
</header>
<body>
<div id=”title”> <h1> The Red Title. </h1> </div>
</body>
<footer>
</footer>
</html>
```

To do the same with green we would set the first number to 0, the second number to 255, and the third to zero. Likewise blue could be called by setting the first two numbers to 0 and the third number to 255. The same example as previous but using blue instead of red would then be:

```
#title {color: rgb(0, 0, 255) }
```

The HTML the example above would turn blue could look like:

```
<html>
<header>
</header>
<body>
<div id=”title”> <h1> The Blue Title. </h1> </div>
</body>
<footer>
</footer>
</html>
```

A very convenient alternative to using the hex number between 0 and 255 that RGB allows is using percentages instead. It is written in the same format but instead of digital values it can read percentage values. This makes RGB even easier to use. To create a green title we simply set the green value to 100% and the others to 0%. An example of this can be found below:

```
#title {
color: rgb( 0%, 100%, 0%)
}
```

The HTML the example above would turn green could look like:

```
<html>
<header>
</header>
<body>
<div id=”title”> <h1>The Green Title. </h1> </div>
```

```
  </body>
  <footer>
  </footer>
</html>
```

3.4 HSL and Alpha Transparency

Since the release of the newest version of CSS which is currently CSS3 the capabilities of the color system have been extended. The new color properties in the arena are called HSL which is short for hue, saturation, and lightness. The other addition is alpha transparency which affects a colors transparency, as in how much you are able to see through the color.

Let's start with HSL which also can be used as HSLa if you want to add the alpha transparency effect as well. Like we covered earlier in the chapter, CSS selects colors by their red, blue, and green values. However, unless you're a lifelong artist finding the right color by altering different amounts of red, blue, and green using numbers instead of a visual of the color you are creating can be time consuming. To make this process slightly more intuitive for the rest of us. HSL was created to allow programmers to alter a color by modifying aspects of the colors shade rather than its logical color ingredients. HSL is used similarly to the way RGB is used. In a stylesheet it may look like:

```
#title { color: hsl(24, 65%, 50%) }
```

Like RGB, HSL values are listed in the order the name describes. In the above example the number 24 designates the colors hue. The number set for hue can represent any angle from 0 to 360 that would be represented on a standard color wheel. If you don't have the color wheel memorized, 0 and 360 would be red, 120 would be green, and 240 would be blue. The 65% from the above example refers to the colors saturation. 0% would be none while 100% would be complete saturation. Finally, the 50% refers to the colors lightness with 0% being black, 50% being normal, and 100% being white.

To add alpha transparency to either HSL or RGB a fourth value is added inside the parenthesis. This value can range from 0 to 1 and is a base-10 digit, or in other words a normal decimal. RGBa is also a common way to use alpha transparency. With RGBa the alpha value is also added into the fourth position inside the brackets and is also a base-10 digit. The same example as above but with transparency looks like:

```
#title { color: hsla(24, 65%, 50%, 0.5) }
```

3.5 Text

CSS can also be used to change the appearance of text in both its size and shape according to many variables. Specifically the variables that can be modified are the font itself, the size, the weight, the style, the decoration, the spacing, and you can even transform the case of the text.

The type of font the text appears in is known as its font-family. Examples of this you are probably already familiar with would be Times New Roman and Arial. While you can add fonts that no one has ever heard of, for it to display the user would have to have the same font installed on their computer. This can get risky quick since most average users don't have the long list of custom text fonts that we developers do. Luckily we can use multiple fonts by separating them with commas. When this technique is used, if the browser cannot find the first font it automatically checks if it can use the second or third font listed. It is a good idea to set multiple fonts with at least a few near the end being common and well used fonts that would be assumed to be on any computer. Fonts with names that are more than one word long should be added inside quotation marks. An example of a safe font selection could be:

```
h1 { font-family: “My Made Up Font”, Arial, serif, “Times New Roman” ; }
```

An example of the HTML document that would be styled by the above rule to display in the My Made Up Font may look like:

```
<html>
<header>
</header>
<body>
<h1> This title will load in My Made Up Font. </h1>
</body>
<footer>
</footer>
</html>
```

The font-size property changes the size. Although it is possible to create a paragraph

with font larger than an h1 heading. It is advised that you still use h1, h6, p headings in your HTML and style them accordingly. Font-size values can be declared using several units. You can choose either px, pt, or em to describe your font size. An example of this is below:

```
h1 { font-size: 24pt ; }
```

An example of the HTML document that would be styled by the above rule to display in a 24 pt. font size may look like:

```
<html>
<header>
</header>
<body>
<h1> This title will be displayed in 24 pt. font. </h1>
</body>
<footer>
</footer>
</html>
```

The font-weight property tells the HTML element how bold to make the text. While you can simply state the font-weight value as normal, or bold. These are the only font-weights most older browsers will be able to pick up and is thus the most common way to use font-weight. You will find that using numerical designations will give you more control. Font-weight numerical values range from 100 to 900 and can be called in increments of 100. As weight implies, the higher the number the fatter the text, the lower the weight the lighter it is. Therefor 100 is the lightest and 900 is the boldest. A statement will look like:

```
p {font-weight: 900 ; }
```

An example of the HTML document that would be styled by the above rule to display with a font weight of 900 may look like:

```
<html>
```

```
<header>
</header>
<body>
<h1> This title will be displayed in a 900 weight font. </h1>
</body>
<footer>
</footer>
</html>
```

Thanks to Microsoft Word the whole world now confuses font-weight with font-style. That is because Microsoft Word considers bold, italics, and underline to all be font styles. In CSS however these are all separate properties. Here we will recognize font-style as the property that decides if the text is italic or not. The font-style property only has two values, italic and normal. A statement creating italic text will look like:

```
h6 {font-style: italic ; }
```

An example of the HTML document that would be styled by the above rule to display in an italic font style may look like:

```
<html>
<header>
</header>
<body>
<h1> This title will be displayed in an italic font style. </h1>
</body>
<footer>
</footer>
</html>
```

Onto the last of the three most commonly mixed up properties. Yes, we are talking about underlining text. In CSS this property is called text-decoration. This property has four potential values being underline, overline, line-through, and none. It is advised underlines only be used for links since that is a commonly known web convention that users expect and understand. However, if you are a CSS ninja that likes to style links to look like buttons you may want to remove the underline from links. That is exactly what the none value is for. An example of its use is shown below:

```
a { text-decoration: none ; }
```

An example of the HTML document that would be styled by the above rule to display an anchor tag without an underlined link may look like:

```
<html>
<header>
</header>
<body>
<a href="website.com" title="no_underline"> This link will display without an underline. </a>
</body>
<footer>
</footer>
</html>
```

Another common way of styling text is to alter the spacing. There are five properties that you can use to adjust the spacing of text. They are letter-spacing, word-spacing, line-height, text-align, and text-indent. With the exception of text-align which uses the values left, right, and center. All of the rest of these properties use the other common measurement units pt, px, and em to define their values. An example of all five being used to style a paragraph with an intro ID is below:

```
p.intro {
letter-spacing: 0.5em ;
word-spacing: 1.5em ;
line-height: 1.2em ;
text-indent: 5em ;
```

```
text-align: right ;
        }
```

An example of the HTML document that would be styled by the above rule to display with the given spacing values could look like:

```
<html>
<header>
</header>
<body>
<p id="intro"> This paragraph will display with 0.5em letter spacing, 1.5em word spacing, 1.2em line height, 5em text indent, and will all be aligned to the right of the page. </p>
</body>
<footer>
</footer>
</html>
```

The last way to modify text is the text-transform property. This allows you to transform the case of the text. This is most frequently used to modify titles and sub-titles. There are four values that can used with this property and they are capitalize, uppercase, lowercase, and none. The capitalize value turns only the first letter of every word into uppercase. The uppercase and lowercase values turn everything into upper or lowercase respectively. The none value has no effect. The example below will transform both h1 and h2 elements into uppercase:

```
h1, h2 { text-transform: uppercase ; }
```

An example of the HTML document that would be styled by the above rule to display all text regardless of how they are typed in uppercase may look like:

```
<html>
```

```
<header>
</header>
<body>
<h1> This text will all be shown in uppercase even though it is not written that way. </h1>
<h2> This text will also all be shown in uppercase even though it is not written that way. </h2>
</body>
<footer>
</footer>
</html>
```

CHAPTER 4 – IMAGES, PADDING, MARGINS, AND BORDERS

Chapter Objectives

- Learn some elements that are block level elements by default.
- Learn how to turn an element into a block level image
- Learn how to set a block-level background image.
- Learn the placement of properties in the box model.
- Learn the order to place values in when using shorthand with four values.

4.1 Images

While you can use the HTML tag <img> to link an image to a web page. Once it is there you may want to format it so that it appears in a specific place and that is done using CSS. Additionally CSS allows all inline and block-level elements to have background images inserted into them. The tags <p> and are block-level elements by default. However both of the codes below can turn any element into a block-level element.

```
Display: block ;
Display: inline ;
```

Once an element is a block-level element inserting the background image is done with the background-image property followed by url then a set of parenthesis with the file location of the chosen image inside them. This would look like:

```
#featured {
display: block ;
background-image: url(filename.jpg) ;
        }
```

The HTML for this example might look like:

```
<html>
<header>
</header>
<body> <div = “featured”>
<p>
This paragraph is a block-level element named featured. It will show the background image filename.jpg.
</p>
</div>
</body>
<footer>
</footer>
</html>
```

4.2 The Box Model

After you get your backgrounds set you may want to adjust the look, layout, and spacing of the other images or items on the page. While the box model is most frequently used to set the spacing for images, it is important to note that it can be applied to any element. Other uses may be placing widgets, creating grids, placing buttons, even search forms. No matter which element you are working with it is done the same way. The box model consists of a spacing system that uses padding, borders, and margins to create the spacing and look you want. While each of these parts of the box model affect different sections of area around a box ultimately they all simply create space around the chosen item. If all the sides of the property are to be set to the same size then only one value and unit are needed in the statement. However, each of these properties can also be declared to specifically set the top, right, left, or bottom measurements if desired. On a border his would look like:

```
p {
   border-top: 2px ;
```

```
    border-right: 4px ;
    border-bottom: 2px ;
    border-left: 6px ;
        }
```

An example of the HTML that could be styled using the rule shown above may look like:

```
    <html>
    <header>
    </header>
    <body>
    <p>
This paragraph will have equal 2px wide borders on the top and bottom. It will
    have a 4px wide border on the right side of the box, and a 6px wide border on
the left side of the box. </p>
    </body>
    <footer>
    </footer>
    </html>
```

4.3 Shorthand

You can also use shorthand to do the same thing but faster and with less typing. The standard format for shorthand is the property name followed by a colon and then up to four values if you want the top, right, bottom, and left sides to be unique sizes. If both top and bottom will be the same size and right and left spacing will be the same then you can use two values in the shorthand statement. Let's start with an example of a four value statement. The values are placed in the order of top first, right second, bottom third, and left fourth. Each value is separated only by a space but be sure to include the units unless the value is zero. Remember from earlier units are not needed when the value is zero. Using this technique to set the margin could look like:

```
p { padding: 2px 5px 7px 5px ; }
```

Similarly if we were using the two value shorthand it could look like:

```
p {padding: 5px 4px ; }
```

4.4 Padding

I am sure you have seen an image with a border around it, in some cases the border may not have actually been touching the image but was slightly larger than the image creating a frame type of appearance. This space between the element and the border is called the padding. Padding can be set using the familiar units of px and pt. As you saw in the last example in section 4.3 padding can be set quickly using shorthand.

4.5 Borders

Around the padding is the border which is also set using units of px and pt. Borders can also be styled by setting the border-style property to a value such as solid, groove, hidden, dotted, dashed, double, ridge, inset, outset, initial, inherit, or none. To set only the style this will look like:

```
#avatar {
border-style: dashed ;
        {
```

An example of the HTML containing the image with the ID selector “avatar” that will show with a dashed border due to the rule shown above may look like:

```
<html>
<header>
</header>
```

```
<body> <img src=”/images/profile.jpg” id=”avatar” title=”Avatar Image”/>
</body>
<footer>
</footer>
</html>
```

You can also modify a borders color, width, or radius. When equal width is used all the way around the border then the border-width, border-style, and border-color can be set in shorthand all in one line using the border property. You know how I love saving time right? Good then I don’t even have to tell you how nice this is to use. Let’s say we want a solid, 2px wide, green border, the statement would first state the width, then the style, and lastly the color. An example of this border around a sign-up form could be:

```
#signupform {
border: 2px solid green ;
        }
```

Border radius is what makes the corners of the border rounded or not and also sets how round they are. Border-radius uses the px unit. A basic example of these properties may look like:

```
#contactform {
border-radius: 5px ;
        }
```

Border radius can also be used in a shorthand form that allows you to set the radius of individual corners. Again there is also a certain order that the values need to be placed in to code in border-radius shorthand. If all four corners will have a unique value then the order is such that the top-left corner is placed first, top-right corner goes second, bottom-right corner would go third, and the bottom-left corner is last. In cases with three values the first value goes to the top-left, the second value sets the top-right and bottom-left, while the third sets the bottom right. When there are two values the first value will set the top-left and bottom-right, and the second value will set the top-right and bottom-left. When there is only one value used all the corners will be set to the same radius.

4.6 Margins

Lastly the spacing around the outside of the border is the margin. The margins measurement values also use the px and pt units. The margin will not have a background color and is transparent. It only clears the space for the border, it does not create any new appearance. The margins value like other shorthand formats can set in the order of top, right, bottom, and then left. You can also set the values of all the sides by using only one value. You can also set each of these to a unique values using the same shorthand system used in the padding example above.

By combining the use of all the above properties you can effectively set the spacing of any block-level element. Although they are most commonly used for images, they will occasionally be needed in other situations as well.

Practical Use Exercise Questions

1. Which HTML elements are block level elements by default?
2. Which property allows you to turn an element into a block level element?
3. What is the order of the box model spacing from the inner element out?
4. What type of units can be used to set the padding and margin?
5. When setting margin, border, or padding to unique sizes on each side of the box in which order are the values placed inside a four value shorthand statement?
6. Out of all the properties in the box model, which has no background color and is transparent?

Chapter Summary and Exercise Solutions

- The <p> and elements are block level elements by default.
- The display property allows you to turn an element into a block-level element.
- The inner most item in the box model is the element itself. The first layer outside of the element is the padding. Around the outside of the padding is the border. The margin goes outside of the border.
- The padding and margin properties can be set using pt and px units.
- In a shorthand statement with four values the values are placed in the order of top, right, bottom, left.

- A margin has no background color and is transparent.

PART II - INTERMEDIATE

CHAPTER 5 – CLASS, ID SELECTORS WITH GROUPING, NESTING, AND PSEUDO ELEMENTS

Chapter Objectives

- Learn how many HTML elements a class can identify.
- Learn what character identifies a class.
- Learn how many HTML elements an ID selector can identify.
- Learn what character identifies an ID selector.
- Learn how to separate selectors when grouping.
- Learn how many levels deep selectors can be nested.
- Learn to identify Pseudo Characters.

5.1 Classes and ID Selectors

Classes and ID selectors allow you to identify a particular block of code in order to style either that one instance of it or anytime that block of code is used. While they function the same the main difference from a class and an id selector is that an id can only identify one element. Contrary to this limitation a class can be used to id multiple elements. In CSS a class uses the full stop (“ . “) which you may know as a period before the class selector. The class selector is the name used to identify a class. The HTML labeling a class can be as simple as:

```
<html>
<header>
</header>
<body>
<p class=”headline”> Todays Headline is About Classes</h1>
</body>
<footer>
```

```
</footer>
</html>
```

The CSS used to style the headline class shown above would look like:

```
.headline {
        color: blue ;
        font-size: 18px ;
        font-style: none ;
                }
```

ID selectors can be identified by the use of a hash character (“ # “). If you grew up in the 90’s like I did you may know this as the pound sign. If you are younger then that, I am talking about the hashtag. In the example below I will show you an HTML declaration of an ID selector followed by the CSS that could be used to style the given selector.

```
<html>
<header>
</header>
<body>
<div id=”top”>
<h1>Site Title</h1>
</div>
</body>
<footer>
</footer>
</html>
```

Now that the id top has been given to the area of the page containing the sites title we

can style it by using the following CSS:

```
#top {
background-color: #fff ;
padding: 5px ;
border: 2px groove green ;
margin: 10px ;
}
```

Selectors can also be set to a certain HTML element by putting the HTML sector before the full stop or hash character. To use this capability on the first example above we could a p right before the .headline code of the CSS to make that code style not just any .headline class but only apply the statements to paragraphs that fall within the .headline class. This would look like:

```
p.headline {
color: blue ;
font-size: 18px ;
font-style: none ;
        }
```

5.2 Grouping

Grouping allows you to give the same properties to any number of selectors without having to repeat the same property for each selector. I know I'm beating the same drum again, but efficiency is the name of the game if you couldn't tell yet. Think of all the selectors the like names of people in a group or clique of friends and the properties describe all the traits that they have in common. Let us say for example the sub-headings in h2 font, the paragraph font, and the comment class font will all use blue font. You could write a statement using each of the above selectors then set each selector to use the color property set to a blue value. This would make you write the "color: blue;" line three times. The alternative is that you simply list all the selectors you want to have blue font at once separated by commas. Then apply the color property to all three at

once. The proper way to Group these three tags will look like:

```
h2, p, .comment {
Color: blue ;
}
```

As you can see that is a bit less typing then not grouping these selectors with the common property, which would look like:

```
h2 { color: blue ; }
p { color: blue ; }
.comment { color: blue ; }
```

5.3 Nesting

Since we can specify properties to selectors within other selectors, there shouldn't be a need for very many classes or ID selectors in properly structured CSS. To nest selectors we just separate the selectors with a space. Using this technique we can nest selectors multiple levels deep. Just put the class or id selector first then a space then the property you specifically are referring to. An example of an HTML segment this could be used in would look like:

```
<html>
<header>
</header>
<body>
<div id="heading">
<h1> The Heading </h1>
<h2> Subheading </h2>
<p> A paragraph. </p>
</div>
```

```
</body>
<footer>
</footer>
</html>
```

Properly nested CSS for the heading div could be:

```
#heading {
background-color: white ;
padding: 5px ;
}
#heading h1 {
color: blue ;
}
#heading p {
color: black ;
font-size: 14pt ;
}
```

In the above example because we separated the selectors with spaces the CSS will understand it to mean that the h1 inside the ID heading will have blue font while the paragraph inside the heading ID will have black 14pt font.

5.4 Pseudo Classes

Pseudo classes are added onto selectors in order to specify a relation to the selector. This is done by adding a colon in between the selector and the pseudo class. Pseudo classes can be very convenient in many situations, although they are most commonly used when styling links. Another time when pseudo classes are commonly used is when making lists. However, since there is not very many pseudo classes we will just go over all of them for you.

As I mentioned links are the most common pseudo classes. So let's start with the link-related pseudo class selectors. Since a link in HTML uses the <a> tag we will proceed each of the following selectors with an <a> tag before the colon and chosen selector. The available link pseudo classes are link, visited, hover, and active. The most confusing part of these links are that many people think all <a> tags are links. An <a> tag is only a link if it includes the href property. An example that would turn all visited links blue can be:

```
a:visited {
color: blue ;
font-style: underlined ;
}
```

Input related pseudo selectors are also common. One property of input related pseudo classes that is often confused with the link related hover is the focus property. Hover is great but does not help users who use keyboard navigation to get to the link. Focus selects the link, input, or text area that is the current focus of the keyboard. Many programmers choose to define a focus pseudo class on any element that has a hover pseudo class for this reason.

Another input related pseudo class is the target class. Target is used with IDs and match when the current URL hash tag matches that ID. If for example you were at the URL:

http://yoursite.com/#portfolio

Then you could use the CSS selector:

```
#portfolio:target
```

This may seem like no big deal but can prove to be very powerful. It is also a great way to create a table of contents at the top of a page with links that automatically scroll to the proper area of the page that that section is referring to. It will also allow you to create a tabbed area where the tabs link to hash tags and then makes panels activate by matching target selectors and even using z-index to move them to the top.

The next two pseudo classes enabled and disabled set an inputs default state to either enabled or disabled. If enabled it will be ready to be used. If disabled it will make the

input a faded out gray color. This allows the user to easily know where to begin. For example if you were making a website that sells car parts you may allow users to search for parts based on the make, and model of the vehicle which could be selected with a drop down menu. In this scenario the user would need to input the make of the vehicle in order for the proper models to display. You would therefor want to enable the make field and disable the model field until the make had been chosen.

The last input related pseudo classes are checked and indeterminate. The checked pseudo class sets if a checkbox is checked or not. The indeterminate pseudo class puts radio buttons in a state of neither chosen nor unchosen. This allows radio buttons to load with no default setting. You may have noticed that every time you sign up for a new service or company's product on the internet you always have to check a box agreeing to the terms and conditions. Some thoughtful companies that have an interest in saving their users time in order to make the registration process faster in hopes of realizing higher success rates have begun to use the checked pseudo class to have the terms and conditions checked by default. In my opinion this is a good idea since more often than not if you do not check the terms and conditions box you can not use the product. As a result I generally go with the train of thought that if you can do something that will save your users time and eliminate redundant extra clicks it is a good thing.

Another type of pseudo class are either position or number based pseudo classes. There are a few more of these than the other types. Let's begin with the root and work our way down in priority. The root pseudo element selects the element that is at the root of the document, almost always being the <html> tag. XML and CSS are the rare exemptions.

First-child and last-child pseudo classes select the first and last element inside a parent. This is typically used to denote list titles or add a big "Sale Price" at the end of a list of a products features. The nth-child(N) selects an element based on the provided algebraic expression. This can be used to select even or odd elements even every third and others. The nth-of-type(N) pseudo class also works like the nth-child but is used when the elements at the same level are different types. This is extra convenient when working in lists and alternating their elements such as background. An example of this would look like:

```
li:nth-of-type(odd) { background: white ; }
li:nth-of-type(even) { background: grey ; }
```

First-of-type and last-of-type both do exactly what the name implies they do. That is select the first or last element of that type inside any parent. An example of this being

used could be to select the first image in a div with multiple paragraphs, the appropriate CSS to accomplish this would be:

```
div img:first-of-type { padding: 2px ; }
```

Nth-last-of-type (N) works like nth-of-type other than it counts from the bottom of the element instead of the top. Nth-last-child works just like nth-child except that it also counts from the bottom of the element instead of the top. Only-of-type selects an element if it is the only one of its type within the parent element. This is used for the same types of things you could use nth-of-type for, it just can be easier for people's brain to work in the reverse order when it comes to certain types of things. In a case like that it may save a lot of time to keep the work flow going the same direction and the flow of information being put into the work.

There are two relational pseudo classes. The empty pseudo class selects elements that contain no text or child elements. This is often used with the property visibility: hidden in order to hide things like a comments box if there are no applicable comments. If you do use the empty pseudo class in this way just make sure you only hide the field that comments that were already added would show up in if it is empty and do not hide the comment input area as this would effectively hide your ability to add comments altogether. The pseudo class not(S) can be used to remove an element from a matched set that matches the selector placed in the parenthesis. The not(S) pseudo class can't be nested but it can be chained. An example of using this pseudo class to style all divs except for contact so that nothing has a background image other than a contact form would look like:

```
div:not(.contact) { background-image: none ; }
```

There are six text related pseudo classes. These can also be chained but not nested. The first-letter pseudo class selects only the first letter of text in an element. Similarly the first-line pseudo class selects the first line of text in an element. The lang pseudo class works only in CSS3 and was only introduced in IE in versions after 8. The lang pseudo class will match any descendant of an element that matches the lang attribute. An example of a CSS rule that would make the text in a paragraph show in 12 point font, while making the first letter show in 18 point font may look like:

p { font-size: 12pt ; }

p:first {font-size: 18pt ; }

The HTML that would be affected by these rules may look like:

<html>

<header>

</header>

<body>

The first letter T in the word “The” of this paragraph will be displayed in a larger 18 point font while the rest of the letters will be displayed in the 12 point font.

</body>

<footer>

</footer>

</html>

Practical Use Exercise Questions

1. How many HTML elements can a class identify?
2. What character is used to identify a class?
3. How many HTML elements can an ID selector identify?
4. What character is used to identify an ID selector?
5. What character is used to separate selectors when grouping?
6. How many levels deep can selectors be nested?
7. What character is used in a pseudo class?
8. What is one way to use the empty pseudo class?
9. What is the difference from the pseudo classes first and first line?

Chapter Summary and Exercise Solutions

- A class can be used to identify multiple HTML elements without a specific limit.
- A full stop also known as a period identifies a class.
- An ID selector can only identify one HTML element.
- The hash character also known as a hashtag identifies an ID selector.

- A comma is used to separate selectors when grouping.
- Selectors can be nested multiple levels deep.
- Pseudo classes separate the ID from the selector with a colon.
- The empty pseudo class can be used to hide dynamically filled text elements that have no content added to them yet, such as hiding a box that shows comments when there are not yet any comments.
- The pseudo class first will style only the first letter of an element while the pseudo class first line will style the entire first line of an element.

CHAPTER 6 – SHORTHAND PROPERTIES, SPECIFICITY, AND PSEUDO ELEMENTS

Chapter Objectives

- Learn the purpose of the shorthand properties.
- Learn the order of width values used by shorthand margins, padding, and borders.
- Learn what properties can be set using the border properties shorthand in order.
- Learn what properties can be set using the font properties shorthand in order.
- Learn what kind of problems specificity can solve.
- Learn the specificity value of ID selectors, class selectors, and HTML selectors that are used to calculate specificity priority.

6.1 Shorthand Properties

CSS is all about saving time. So it should come as no surprise that it allows shorthand properties so make things even faster. We have already covered a few examples of this as it pertained to specific selectors as we went along. Now let's go further in depth to allow you to maximize your usage of shorthand properties whenever you can. The purpose of a shorthand property is to allow you to modify multiple properties of a single selector all at once without having to retype each selector and property. This is accomplished by using a string of values instead of multiple statements. All you have to remember is to put the property values in the correct order. The four types of selectors that allow shorthand are margins, padding, borders, and text.

When using shorthand selectors with margins and padding you will be able to set the values of the width on all four sides of the box at once. As usual the values will be stringed out in the order of top, right, bottom, then left. To separate the values you only need to use a space since the CSS will already know where each value goes and what it should look like. This allows you to get rid of the four separate selector's padding-top,

padding-right, padding-bottom, padding-left and merge them into the single selector statement using the padding selector. An example of a H1 heading using the four separate statements would look like:

```
h1 {
padding-top: 4px ;
padding-right: 5px ;
padding-bottom: 6px ;
padding-left: 7px ;
}
```

Using the shorthand property of margin saves a considerable amount of key strokes as you can see in the correct way to set these values below:

```
h1 {
margin: 4px 5px 6px 7px ;
}
```

Borders can also have the shorthand properties applied in the same way and order as the margins and patterns to set the width of individual sides. Additionally, the border property can also be combined to set the border-width, border-color, and border-style properties. When setting this combination of properties the values are again separated by a space. The values are placed in the order of width, color, and then style. The long drawn out way of setting these properties for a paragraph may look something like the example below:

```
p {
border-width: 2px ;
border-color: black ;
border-style: double ;
}
```

Instead of using all these properties we can save time by setting them all using only the border selector. An example of this same command using the proper shorthand variation will look like:

```
p {
border: 2px black double ;
}
```

The last shorthand type is fonts. While fonts obviously don't have a top, right, bottom, or left property like margins, padding, or borders. They can still use shorthand to set the font-style, font-weight, font-size, line-height, and font-family properties all at once using the font selector. The font-style, font-weight, and font-size are also separated only by a space. However the font-size and line-height properties are separated by a backwards slash. Also the font-family property allows for multiple values to be listed and each font-family in the list is separated by a comma. Again styling a paragraph using the long version could look like:

```
p {
font-style: italic ;
font-weight: bold ;
font-size: 14px ;
line-height: 2 ;
font-family: "Times New Roman", Arial, serif ;
}
```

As you can see that was a longer command. However, using shorthand we can save a lot of time setting font properties. Especially if different elements of your HTML are going to be using different font styles. The proper way to write out the same information as above would look like:

```
p {
```

```
font: italic bold 14px/2 “Times New Roman”, Arial, serif ;
}
```

6.2 Specificity

Specificity is the way used to solve problems when you have multiple, at least two, CSS rules pointing to the same element in your HTML document. In this case your browser will determine which rule to obey depending on each elements specificity. The more specific rule will be obeyed and the other(s) ignored. Starting out you will probably rarely ever see a need to use specificity. However, as your CSS files become larger, more precise, and maybe even using multiple CSS files at once then you should expect to start needing to understand specificity as conflicts will naturally start appearing.

The general guideline to follow is that the more specific a rule is the more precedence it will have. If the selectors and number of values are the same then the last one loaded will be used. An example of this happening to an H1 heading would look like:

```
h1 { color: green ; }
h1 { color: blue ; }
```

In the above set of rules the H1 text will appear blue since it came last. While this example is unlikely to ever happen since it would be pointless to use the same selector and property twice. This is more common and easier to accidentally make happen when you are using nested selectors. Say for example we are setting a block level div elements paragraph font size. While then trying to set a regular paragraphs font size after. We may try to use something like:

```
div p { font-size: 14px ; }
p { font-size: 12px ; }
```

This is where most programmers get confused. You would think that the above rules would make a regular paragraph that is not in the div show in 12px font since it comes last. However, since the div paragraph has higher specificity the paragraph will instead show in a 14px font size. The more specific selectors take higher preference so since

there are two paragraph rules the browser selects the first rule.

Now onto the math part of specificity. At times when we have large groups of nested selectors it will become necessary to calculate the specificity. In order to calculate specificity we assign arbitrary values to different types of selectors. ID selectors such as #intro have a value of 100. Class selectors such as .featured get a value of 10. All HTML selectors such as button get a value of 1. You then add up all the values from each group of nested selectors and whichever group has the higher value will over-ride the others. To explain better I will provide several example statements below with total values explained in the comments section of each statement.

body #intro .featured p //** Specificity is 112, since this rule consists of 2 HTML selectors body and p = 2, 1 class selector = 10, and 1 ID selector = 100. **//

#intro //** Specificity is 100, since this rule consists of 1 ID selector. **//

div p.featured //** Specificity is 12, since this rule consists of 2 HTML selectors = 2, and 1 class selector = 10. **//

.featured //** Specificity is 10, since this rule consists of 1 class selector. **//

div button //** Specificity is 2, since this rule consists of 2 HTML selectors. **//

button //** Specificity is 1, since this rule consists of 1 HTML selector. **//

Now that we know the priority of all these examples, the question is which would over-ride the others? The rule div p.featured with a specificity of 12 would over-ride div p that had a specificity of 2. Also the body #intro .featured p would over-ride all the other commands regardless of which order they are in.

6.3 Pseudo Elements

Similar to the Pseudo classes covered in chapter 5, there are also content-related pseudo elements. Pseudo elements are added onto selectors almost just like pseudo elements. Pseudo elements are also written in the same form of selector:pseudoelement { property:value; }. While using only one colon between the selector and pseudo element was the standard and CSS and CSS2. In CSS3 this was switched to two colons in order to identify a pseudo element. This update is backwards compatible so if you use two colons older browsers will still read it as one colon. For that reason it is a good idea to just use the more modern two colon standard. These are considered pseudo elements and not pseudo classes because they do not select any real elements on the page. There are four pseudo elements in total. They are first-letter, first-line, before, and after.

The first-letter and first-line pseudo elements allow you create drop caps to the first letter in an element or bold the first-line of paragraphs. An example of this would look like:

```
p {
font-size: 14px ;
}
p::first-letter {
font-size: 18px ;
float: left ;
color: red ;
}
p::first-line {
font-weight: bold ;
}
```

The before element can add content before a certain element. This is most frequently used to add blockquotes before a certain type of paragraph such as a testimony. Following that up, the after property can add content after a content element. This is the only time you should use CSS and not HTML to add content to your HTML document because CSS is for styling and HTML is for content. The content property can use open-quote, close-quote, any string enclosed in quotation marks, or any image using URL(imagename.jpg) as a value. An example of that will add quotation marks before and after a testimonial paragraph may look like:

```
p.testimonial::before {
content: “\201C” ;
}
p.testimonial::after {
content: close-quote ;
}
```

You can also use the content property to create another box to modify and style presentational content. One use of this could be to add the word Sale! In bold red font next to a list item. This would be executed using the following rule:

```
li::before {
content: “Sale!” ;
color: red ;
font-weight: bold ;
}
```

Practical Use Exercise Questions

1. What is the purpose of the shorthand properties?
2. What is the order of the width values used in shorthand margins, padding, and borders?
3. What are the properties that can be set using the border properties shorthand in order?
4. What are the properties that can be set using the font properties shorthand in order?
5. What kind of problems specificity can solve?
6. What is the specificity value of ID selectors, class selectors, and HTML selectors that are used to calculate specificity priority?

Chapter Summary and Exercise Solutions

- The shorthand property is used to modify multiple properties of a single selector all at once.
- When using shorthand to set width values of margins, padding, or borders the values go in the order of top, right, bottom, left.
- In order the border selectors’ shorthand properties are width, color, and style.
- In order the font selector’s shorthand properties are font-style, font-weight, font-size, line-height, and font-family.
- Specificity is used to solve problems that arise when you have multiple, at least two, CSS rules pointing to the same element in your HTML document.

- The specificity value of an ID selector is 100, the value of a class selector is 10, and the value of an HTML selector is 1.

CHAPTER 7 – BOX DISPLAY, ROUNDED CORNERS, SHADOWS, AND PAGE LAYOUT

Chapter Objectives

- Learn the three types of box displays.
- Learn the benefits of CSS tables.
- Learn the dangers of over using CSS tables.
- Learn what kind of elements can have rounded corners.
- Learn the order of shadow values for boxes.
- Learn the order of shadow values for text.
- Learn the values of the position property.

7.1 Box displays

The display property is one of the most important tricks that exists in the manipulation of HTML elements. As I'm sure you have noticed by now there is nothing special about how elements work. The HTML documents visual representation generally consists of different display types. There are six display types in total but there are only three fundamental display types which are inline, block, and none.

The inline display type sets boxes to follow the flow of a horizontal line. Emphasis and links are a few of the elements that display inline by default. If for example we wanted to make the list of links in the footer navigation menu appear on a single line next to each other instead of putting each link in the list on its own line we could use the rule below:

```
li {
display: inline ;
}
```

The block display type makes a box standalone and fits the entire width of it's containing box. It also adds a line break before and after the box. Block boxes are often used to manipulate height, margins, and padding in ways not available when using inline blocks. Headings and paragraphs are displayed as blocks by default. If we wanted to make the main navigation menu appear as clickable blocks we could use the example below:

```
#nav a {
display: block ;
padding: 5px 4px ;
)
```

You can also combine the two using the inline-block property to keep a box inline while allowing more of the block formatting abilities such as placing margins to the sides of the box. This could look like:

```
#widget a {
display: inline-block ;
margin: 0 10px ;
}
```

On the other side of the spectrum is the none property which does not display a box at all. This is most commonly used to help with dynamic effects such as alternative stylesheets. The difference from display: none and visibility: hidden is that display: none removes the box while display hidden keeps the flow created by the box but just hides it visually. The none property is often used to create a printable version of a page by turning off style sheets display elements that would make the printout more confusing. An example of using this property to remove the nav blocks from the previous example would look like:

```
#nav {
display: none ;
```

```
}
```

Tables are another display property, however they are rarely used because they are similar to the HTML tables. The table property is the initial display and you can copy the tr and td elements with the table-row and table-cell property values respectively. The advantage to using the table display is that it also offers table-column, table-row-group, table-column-group, table-header-group, table-footer-group, and table-caption as well. The purpose of these values are all pretty self-explanatory. The benefit in these values are that it allows you to construct a table using columns as opposed to the row method used in HTML. You can also use the display property inline-table to set a table without the line breaks before and after. The down side to using CSS tables is that older browsers rarely support them and over using CSS tables can be hazardous to your accessibility.

The last two display types are list-item and run-in. List-item displays a box like the HTML element li creates. To work properly elements displayed in this display type should be nested in a ul or ol element. The run-in display makes a either block or inline depending on its parent element.

7.2 Rounded Corners

Rounded corners used to be a nightmare of limitations before CSS. Rounded corners can be added to any box element using the border-radius property. Despite its name you do not have to have a border to use the border-radius property. Border-radius uses px as its values unit and higher values create more rounded appearances. As an example we will add a 10px border-radius to our logo image by using the rule shown below:

```
#logo {
padding: 5px ;
border-radius: 10px ;
}
```

That example will round all the logos corners by a radius of 10px. You can also round only specific corners that you choose by using the more specific properties. These properties are border-top-left-radius, border-top-right-radius, border-bottom-right-radius, and border-bottom-left-radius. I would strongly advise using the border radius

shorthand method with lets you put the values in the order listed above and separated by a space in order to save time. An example of giving the same logo as the last example a more speech balloon shaped appearance we could use the rule below:

```
#logo {
padding: 5px ;
border-radius: 10px 10px 0px 10px ;
}
```

You can also use just two values instead of four to select the top-left and bottom-right with the first length or percentage and the bottom-right and top-left with the second value. Yes, you can also use three values. The three values in order would style the top-left, then the top-right and bottom-left, then the bottom-right.

The biggest drawback to the border-radius property is that it is not supported by internet explorer versions 8 and below.

Border-radius will also allow you to use ellipses instead of circles. This is done by specifying different horizontal and vertical radii by splitting the values with a backslash instead of a space. This would look like:

```
#logo {
width: 100px ;
height: 100px ;
background: white ;
border-radius: 50px/100px ;
border-bottom-left-radius: 50px ;
border-bottom-right-radius: 50px ;
}
```

7.3 Shadows

Shadows add the appearance of light being shined onto your page in order to give it extra pop. You can apply shadows to both boxes and text. To do so you can use the box-

shadow and text-shadow properties.

The box-shadow property consists of several parts and is written in the shorthand method separated by spaces like always. The first value is the horizontal offset, this is how far the shadow is goes out to the right or left if you choose to use a negative value. The second value is the vertical offset, this is how far the shadow extends downwards or upwards if you use a negative value. The third value is the blur radius, the higher this value is the less sharp the shadow will appear with 0 being the sharpest value. The fourth value is the spread distance, the higher this value is the larger the shadow with 0 being the inherited size of the box. The fifth value is the color, this value is optional. An example of a yellow shadow will look like:

```
#contactTable {
box-shadow: 4px 4px 3px 1px yellow ;
}
```

If you want you could also add the shadow to the inside of the box by adding inset before the first value. To add the same shadow as defined in the rule above but this time to the inside of the box the rule will simply look like:

```
#contactTable {
box-shadow: 4px 4px 3px 1px yellow ;
}
```

Similar to the box-shadow property you can use the text-shadow property to add shadow to text. The only difference in the order of values is that the text-shadow omits the fourth box-shadow value of spread distance. Unfortunately browsers took a little longer to figure out text shadow so you won't be able to replicate any of that cool word art you saw in office on browsers below and including Internet Explorer 9. For that reason shadows are usually reserved for situations of low importance.

7.4 Page Layout

Before CSS HTML tables were used to layout web pages. This system offered very limited capabilities and could get confusing. CSS made this much easier. With CSS you

just define a piece of your page then put it where ever you want it to go.

These pieces can be placed absolutely or relative to another chunk. This is accomplished with the position property. The position property defines whether a box is absolute, relative, static, or fixed. Static is the default value and this value renders a box in the normal order of things as they are set up in the HTML. Relative is a lot like static only the box is offset from its original position with the properties top, right, bottom, and left. The absolute property pulls a box out of the normal HTML flow and puts it into a set place anywhere on the page using top, right, bottom, and left. The fixed property behaves just like the absolute property. The difference being that fixed sets the box in reference to the browser window while absolute sets a box relative to the web page. As a result fixed boxes will stay exactly where they are on the screen even when the page is scrolled. As an example let's create a two column layout using absolute positioning. The HTML of this example could be a page section that looks like:

```
<html>
<header>
</header>
<body>
<div id="nav">
<ul>
<li> <a href="home.html"> Home </a></li>
<li><a href="about.html">About</a></li>
</ul>
</div>
<div id="content">
<h1>Title</h1>
<p>Paragraph 1</p>
</div> </body> </html>
```

Now to style the above HTML we will use the CSS example that is below:

```
#nav {
position: absolute ;
top: 0 ;
```

```
left: 0 ;
width: 400px ;
}
#content {
margin-left: 400px ;
}
```

In the examples above you will notice that what we did was set the navigation or as we called it the nav div to the left with a width of 400 pixels. Since this section is absolute it has nothing to do with the rest of the page so all that is needed is to set the left margin of the content area equal to the width of the navigation div. The downside to using the absolute property is that since they are not in the HTML flow there is no way to tell where they will end up on different size screens.

One solution to that problem is to float the pieces where you want. Floating a box will shift it to the right or left of a line while letting other content flow around it. The float property is usually used to shift around smaller elements of the page but can and is sometimes also used to move around larger pieces. The float property has two values left and right. To use the same HTML example we previously used we could use the CSS below to push the navigation to the left.

```
#nav {
float: left ;
}
```

If you do not want the next box to wrap around the floating objects you can use the clear property. The clear property has three values left, right, and both. The left value will clear floated boxes to the left, right will clear floated boxes to the right, and both will clear all the floated boxes. This can be helpful when trying to make sure that the footer will appear at the bottom of any other box regardless of how long a box might be. To do this lets use the following HTML as an example:

```
<div id=”footer”>
<p>Footer text</p>
```

```
</div>
```

To clear boxes from each side of the footer we will use the both property as shown below:

```
#footer {
clear: both ;
}
```

Practical Use Exercise Questions

1. What are all the types of box display?
2. What is a benefit to using a CSS table?
3. Why are CSS tables dangerous to overuse?
4. Does border-radius require using a border?
5. What kind of elements can border-radius be used with?
6. Why are shadows not used in important situations?
7. What is the order of values for box-shadow property?
8. What is the order of values for the text-shadow property?
9. What values can be used with the position property?

Chapter Summary and Exercise Solutions

- The three types of box display are inline, block, none, table, list-item, and run-in.
- CSS tables allow you to construct a table using columns instead of the row method used by HTML.
- CSS tables are not supported well by older browsers and can cause accessibility issues.
- The border-radius property can be used with or without a border.
- The border-radius property can be used with any box element.
- Shadows are not supported by many of the older versions of Internet Explorer.
- The box-shadow property values are horizontal offset, vertical offset, blur radius, spread distance, and color.
- The text-shadow property values are horizontal offset, vertical offset, blur radius, and color.

- The values that work with the position property are static, relative, absolute, and fixed.

PART III - ADVANCED

CHAPTER 8 – IMPORTING STYLE SHEETS

Chapter Objectives

- Learn what the <link> tag is used to accomplish.
- Learn where to place the @import rule.
- Learn what the @media rule is used to accomplish.
- Learn the advantages of the @font-face rule.

8.1 The Link Tag

The link tag looks like <link> and is the first method for attaching an external stylesheet to your websites. The link tag is added to the head section of your HTML document. This does just what it sounds like, it links the stylesheet to your document. An example of this looks like:

<html> <header>

<link href=”styles.css” type=”text/css”>

</header> <body> </body> <html>

The main reason to use linked style sheets is to provide alternate style sheets for your users. Browsers like Firefox, Safari, and Opera support the rel=”alternate stylesheet” attribute. When this attribute is available is allows users to switch between the different style sheets. This can also be accomplished using a JavaScript switcher to switch between style sheets in Internet Explorer. This is commonly used with Zoom Layouts for accessibility purposes.

Additionally the link tag is preferred by some because if you have a simple head section in your HTML document with only the @import rule it may display a flash of unstyled content as it is loading. This can look bad to viewers. However it can be fixed simply by adding at least one extra <link> or <script> element within the <head>

section.

8.2 The @import Rule

The @import rule is used to include external style sheets into another stylesheet or an HTML document. Each at rule begins with an apetail which is more commonly known as the at sign. This should be used if a site needs long complex stylesheets that might be easier to manage when they are broken down into smaller files. The @import rules must be placed at the top of the stylesheet before any rules. In an HTML document the @import rule must be placed within the style tag. An example of this is below:

```
<html> <header>
<style type=”text/css”>
@import url(stylesheet.css);
@import url “custom.css” ;
</style> </header>
</body> </html>
```

As you may have noticed the second @import statement in the example above did not have parenthesis around the double quotes and no it’s not a typo. The parenthesis after the url are not required, only the double quotations are required for valid XHTML. It should also be mentioned that most browsers that support the url with parenthesis also support the url without them.

You may have also noticed in the example that the @import rule was placed in style tags. You can also place styles rules within the style tag with the @import rule. This might look like:

```
<html> <header>
<style type=”text/css”>
@import url(“styles.css”)
p {color: black ; }
</style> </header>
</body> </html>
```

The most common reason programmers use @import instead of or along with the <link> tag is because older browsers did not recognize @import so you could hide styles from them. More precisely you can hide styles from Netscape 4, Internet Explorer 3 and 4 by using the statement below:

```
@import url(../style.css) ;
```

You can also add the double quotations within the parenthesis of the example above to additionally hide the styles from Konqueror 2, and Amaya.

Another use for the @import method is to use multiple style sheets on a page while only having one <link> in the head section. An example of when this would be needed is if a corporation had a global style sheet for every page on a site, but also had subsections each of which had their own style that needed to apply to that subsection. The only requirement is that all of the @import rules are placed above any other style rules in your page. Just be careful as this is a common cause of inheritance problems.

8.3 The @media Rule

The @media rule is used to apply styles to a specific media. This is commonly used to create printer friendly versions of websites. The @media rule can include the values screen, print, projection, handheld, and all. You may also use a comma separated list of values if you need to use more than one. An example of this could look like:

```
@media print, handheld {
font-size: 2em ;
}
```

In CSS3 the @media not only allows you to target specific media but also variables that relate to media like screen size. This is the holy grail of the responsive web site design techniques. However, you will have to wait until chapter 10 to learn more about media queries.

8.4 Embedding Fonts

The @font-face rule was until CSS3 pretty much useless. Now that is has been fixed up by CSS3 it has become a widely used technique for embedding fonts into web pages. This allows a typeface to be used regardless of if it is on the user's computer or not. Basically this is what ended the mandatory use of web safe fonts. The @font-face rule is followed by brackets which include the font-family followed by the name of the font and then a src property with a url to the font files location on the server. The src property is an abbreviated form of source, this just tells the document where to find the font you are referring to. Let's use the example below to set the font called our made up font and its location within a @font-face rule:

```
@font-face {
font-family: "our made up font" ;
src: url(ourmadeupfont.woff) ;
}
```

As a result of this embedded font rule being applied we are now free to go throughout the the style sheet where ever it is wanted. An example of using this font on an h1 title would look like:

```
h1 { font-family: "our made up font", arial, sans-serif ; }
```

When the document loads the above rule, the font is downloaded from the location listed in the @font-face rule then applied to H1 text. If the browser for some reason can not work with the loaded font it will still load the other listed fonts.

If you would like you can also check if the font already exists on the user's computer in hopes of preventing the need to download it. This is done by replacing the url in the src value with local. Additionally most text files will contain tons of weights or styles of the same font. That makes it a good idea to specify which one you are looking to use. As a result of all these things a well written @font-face rule could end up looking more like:

```
@ font-face {
```

```
font-family: “our made up font” ;
src: local(“our made up font”), url(ourmadeupfont.woff) ;
font-weight: 300 ;
font-style: bold ;
}
```

There are plenty of free web fonts out there that I would suggest you download and use just to be safe. I say that for legal reasons as many fonts that you will come across can have copyright and usage terms. That is in addition to the optimization and compatibility issues. Another safe option is to find and use a web font provider such as Adobes Typekit or Google Web Fonts. The Google Web Fonts are by far the easiest to use as all you need to use them is a @font-face rule to import their font.

Practical Use Exercise Questions

1. Where do you place the @import rule?
2. What can you use the link tag for?
3. What is a benefit of the @font-face rule?

Chapter Summary and Exercise Solutions

- The @import rule must be placed at the top of the stylesheet before any other rules.
- The link tag attached multiple style sheets to your document, which allows users to change style sheets.
- The @font-face rules main benefit is that it allows you to use fonts that are not already on a user’s computer.

CHAPTER 9 – TRANSITIONS AND GRADIENTS

Chapter Objectives

- Learn the two mandatory properties that need to be declared in order for a transition to take place.
- Learn the four transition properties.
- Learn how to target multiple properties.
- Learn what a linear gradient is.
- Learn what a radial gradient is.

9.1 Transitions

Transitions in a way seeks to replace the need for JavaScript in your code on a limited scale. With CSS transitions you can quickly animate parts of your HTML document without needing to run a script. Not only is this route faster for the programmer to write, it also helps to load the page faster since it lowers demand on the scripting server. On the most basic level you could think of this effect as what happens when you hover the mouse on top of a link and it changes color. That is known as a binary transition.

However, like I said this is the most basic level of a transition. The transition property is however capable of far more than this. It will allow for smooth animation as a value transitions into a new value over a set period of time. The only two things you must declare in a transition rule are the CSS property you want to have a transition effect on and the duration it will take for that transition to occur. As opposed to jumping from on value to another like it would in a binary transition.

The transition property is also a shorthand property. The shorthand property combines the following four properties that will be explained in further detail later in this section of the chapter. These can all be used at once or used individually if that is what is needed. The transition shorthand property is transition-property. This can be used to set which property will transition or as the shorthand property. The order of properties in the shorthand form would be transition-property, then transition-duration, then transition-timing-function, and

lastly the transition-delay. When using shorthand the value to each of these properties is simple separated by a space. An example of a rule containing all four properties may look like:

```
#div {
transition: font-size .8s ease-in 0.3 ;
}
```

The value for the transition-property can be all, none, or any other property that you choose. An example of a rule containing this property may look like:

```
a:link {
transition-property { color: red ; }
```

The second transition shorthand property is transition-duration, this sets how long the transition will take to complete. The value should be a number but does not need to be a whole number as decimals are often required. The units for the value is typically s, which means seconds. An example of a rule containing this property that when hovered over changes the link in the example above from red to blue over a period of one half second may look like:

```
a:hover {
transition-property { color: blue; transition-duration: .5s ; }
```

The third transition shorthand property is transition-timing-function. This property sets the rate of change of the transition. It may be set to a constant speed, a speed that accelerates, or decelerates. The values that are available to be used with this property are ease, linear, ease-in, ease-out, ease-in-out, initial, cubic-bezier, and inherit. An example of the transition-timing-function may look like:

```
a:visited {
transition-property: color ;
transition-duration: 1s ;
transition-timing-function: linear ;
}
```

To go a little further in detail regarding each of these property values behaviors I will start with ease. Ease specifies a transition effect that has a slow start, then

gets faster, and then ends slowly. This is the equivalent to using the cubic-bezier property with values (0.25, 0.1, 0.25,1). The linear value specifies a transition effect that has the same speed from start to end. This is the equivalent to using the cubic-bezier property with values (0, 0, 1,1). The ease-in value specifies a transition that starts with a slow effect and then becomes faster. This is the equivalent to using the cubic-bezier property with values (0.4, 0, 1, 1). The ease-out value specifies a transition that starts with a faster effect and then becomes slower. This is the equivalent to using the cubic-bezier property with values (0, 0, 0.5, 1). The cubic-bezier function has four values that can range from 0 to 1. These four values represent the rate of change that will occur at 4 different places during the transition. The cubic-bezier is fairly unique in that it has a numerical value other than zero without any unit descriptor. The initial value allows you to set a CSS property to its default value. You can actually use this value to set the default value of any HTML element, not just transitions. However, it is most commonly used with transitions so that is why it's covered in this section of book. The inherit property specifies that a property will inherit its value from its parent element. Just like initial, this value can also be used with any HTML element. Now that we have covered all of the transition-timing-function values let's move onto the next transition property.

The transition-delay property specifies how long of a wait there will be until the transition occurs. This is one of the two only mandatory properties that are needed for a transition to occur. The other being the property that you want a transition to happen to. An example of a transition-delay taking 1 second to change the font size of an H1 element may look like:

```
#div {
transition-property: h1 ;
transition-delay: 1s ;
}
```

In order for a transition to occur the transition-duration property is the only property that is required. If any of the others are not used they will be automatically set to their default values. The default value for the transition-property is all. The default value for the transition-timing-function is ease. The default value for the transition-delay is 0. As a result of these defaults a complete shorthand transition statement could be as simple as:

```
a:link{ transition: .5s ; color: blue ; }
```

9.2 Selecting Multiple Properties

Transitions can be applied to multiple properties in the same rule. In order to save time you can list multiple properties at once after the transition-property declaration just by separating them each by a comma. Of course technically using the all value also selects more than one property, in this example we will work with selectively choosing which elements you would like to transition without having to transition all elements. In the example below we will transition the color, font-size, and text-decoration of an anchor link to red, 14px, and none in the period of one half second:

```
a:link {
color: blue ;
font-size: 12px ;
text-decoration: underline ;
transition-property: color .5s, font-size .5s, text-decoration .5s ;
}
a.hover {
color: red ;
font-size: 14px ;
text-decoration: none ;
}
```

9.3 Gradients

A gradient is are the smooth change from one color to the other that looks perfectly blended. I am sure you have seen these all over the web. In the old days we used to have to create an image file and blend these by hand. Needless to say it took a really long time to make look right. Thanks to CSS those days are long gone and now we can create perfect gradients for our designs with no art skills at all and without even needing an image file. Although I should mention you can also overlay a gradient over an image file with CSS if you would like to do so.

One trait that makes gradients fairly unique is that they do not have a special property for this task. Rather it is a value that can be added to many properties. Properties like

background and background-image will both allow you to play with gradients and define the gradient with a value. The value can be either linear or radial. You can also fill a box with a repeating gradient to build on color stops and make a pattern. This is done simply by putting repeating- before whichever type of gradient you are using, such as repeating-linear-gradient or repeating-radial-gradient. I will go into more detail about the difference between linear and radial gradients below.

9.4 Linear Gradients

Linear gradients create an even spread of two colors that fade into each other one from the top and one from the bottom. An example of how to make a background change from green to yellow may look like:

```
background: linear-gradient(green, yellow) ;
```

You can also use a linear gradient to transition to as many colors as you want, or from right to left, or on any angle that you choose simply by to right, or to left, or even to bottom left inside the parenthesis before the first of the colors that you are transitioning. Likewise, you can also set a specific degree by typing any digit from 1-360 in the first place inside the parenthesis with the abbreviation deg following the digits but before the colors. This would also be separated by a comma. To make an example background image gradient that changes from white to transparent at a 45 degree angle we would simply create a rule that said:

```
background: linear gradient(45deg, white, transparent) ;
```

In order for gradients to work on as many browsers as possible it is recommended that use –webkit-linear-gradient to cover Safari and older Chrome versions. Unfortunately there is still not a fix for Internet Explorer versions 9 and below. If you simply must have a gradient functioning on older browsers then try using an image file with the CSS gradient as a backup. This way of problem solving may look more or less like:

```
body {
background: #fff url(bg-gradient.jpg) 0 0 repeat-y ;
```

```
background: -webkit-linear-gradient(right, #fff, #444) ;
background: linear-gradient(to right, #fff, #444) ;
}
```

9.5 Radial Gradients

Radial gradients fade into other colors similarly to linear gradients. The difference is that radial gradients do not fade out along a straight line. Instead radial gradients fade with their start at a central point and then fade outward into other colors. The syntax is similar to linear gradients except that it is called a radial-gradient instead. An example of a radial gradient fading from red to blue can be accomplished by using:

```
background: radial-gradient(red, blue) ;
```

One other option you can specifiy is the shape that the gradient should form. By default it will form an ellipse, however, you can also make it circular. While using the same gradient from the above example but to make the gradient form a circle instead of an ellipse we could just use the rule shown below:

```
background: radial-gradient(circle, red, blue) ;
```

Naturally all of these gradients will fall within some type of box element. You do not have to leave the gradient radiating out from the center either. You can also use the values closest-corner, closest-side, farthest-side, and farthest-corner. If you decide to use one of these more specific descriptors they would be placed immediately before the colors and separated from everything else with a comma.

You also are not stuck with those perfectly blended gradients either. If you would like you can also specify exactly where in the gradient you want each color to begin or end. This value can range from 0 up to 100% depending on how large of a length you would like the fade to take up. To replicate the previous red to blue circle but this time with sharp lines instead of a balanced fade we will use the example shown below:

```
background: radial-gradient(circle, red 0%, blue 100%) ;
```

Practical Use Exercise Questions

1. What two properties at a minimum must be declared for a transition statement?

2. What are the four transition properties?

3. How can you transition multiple properties?

4. If you need a color gradient to occur along a straight line which type of gradient would you use?

5. If you need a color gradient to occur in a circle or ellipse which type of gradient would you use?

Chapter Summary and Exercise Solutions

- Transition duration and the property which you are transitioning must be used at a minimum.
- The four transition properties are transition-property, transition-delay, transition-timing-function, and transition-duration.
- To transition multiple properties you put them in a list separated with commas.
- A linear gradient will create a gradient along a straight line.
- A radial gradient will create a gradient in a circle or ellipse.

CHAPTER 10 – MEDIA QUERIES

Chapter Objectives

- Learn the more advanced capabilities of the @media rules.
- Learn the two properties that allow you to target specific screen sizes.
- Learn the two values accepted by the orientation property.
- Learn the six device specific properties.
- Learn about pixel ratios.
- Learn how to define resolution.

10.1 Media Queries

As we discussed earlier, you can use the @media rules to target rules used to style specific forms of media such as screen or print. Using media queries you can push this envelop even further to a higher level of sophistication. This allows you to for example use a different design style depending which screen size the user is viewing the content on. This is the holy grail of responsive design. In the manner we can give the same page a completely unique design for a mobile phone or tablet than you would see viewing the same page in a browser window.

As a review so for example we wanted to apply a certain CSS rule to make font show twice as big when showing only in print or handheld devices. Then we may use a statement with something like the example shown below:

```
@media screen, handheld {
font-size: 2em ;
}
```

10.2 Browser-Size Specific CSS

Now to expand the capabilities of the previous example. Not only can we target media that is shown on a screen, we can also target media that shows on a specific screen size. This is accomplished by using the @media rule with the screen selector followed by the "and" keyword with the max-width in pixels stated inside a set of parenthesis. When it

is all put together you will end up with an example that looks like the example shown below:

```
@media screen and (max-width: 1000px) {
#content {  width: 100% ; }
```

In the example above the rule is telling the browser to apply any code inside that block of CSS whenever the viewport is equal to or less than 1000 pixels. This block can be used to do anything you want. Which may include making the content area the full width of the screen or maybe changing the navigation from horizontally aligned block links like a drop down menu into a single button triggered accordion menu. It is also advised that when using this type of feature to create a mobile responsive design that you stack page sections on top of each other instead of displaying them in columns. In order to be truly mobile responsive you may also apply more than one @media rule in order to give each screen size a quality and user friendly design. When doing so it is common to start with the largest screen then work your way down to smaller screens just for the sake of organization. An example of using this technique to customize particular appearances on multiple screen sizes may look like the example below:

```
@media screen and (max-width: 1000px) {
        #content { width: 100% ; }
        }
@media screen and (max-width: 800px) {
        #navigation { float: right ; }
        }
@media screen and (max-width: 600px) {
        #content aside{ float: none ;
                Display: block ; }
        }
```

You may be wondering what would happen on a small screen when it loads the example above. If the screen is equal to or less than 600 pixels it will in fact apply all three

rules. This is because of max-width values of 1000 pixels, 800 pixels, and 600 pixels are all larger than the 600 pixels or less screen size so they will all be applied. If the screen size is 601 pixels then only the first two would be applied. If the screen size was 801 pixels then only the first rule would be applied.

In addition to the max-width property you can also use the min-width property to do the same thing in the opposite fashion. This property only applies the rules if they load on a screen with at least that many pixels. The min-width property is used when programmers refer to mobile first design. This means they designed the site first to be viewed on a very small screen then used the min-width property to scale it up for larger screens. If you do not have multiple devices to test how your code is appearing on different screen sizes you do not need to worry. Just resize your browser and it should automatically adjust to the different styles.

10.3 Orientation and Device Specific CSS

Perhaps you are worried, as you should be, about what your CSS will look like when a phone for example views it from different orientations. This is where the orientation property comes into play. This property can be used with two values being landscape and portrait. To set the styling of a specific orientation you just set it up the same way as we set up the max-width example above. If you recall we did this by using the @media rule with the screen selector followed by parenthesis filled with the orientation property a colon and then the value you would like to use. When it is all put together you will end up with an example that looks like the example shown below:

```
@media screen and (orientation: landscape) {
#navigation {  float: left ; }
}
@media screen and (orientation: portrait) {
#navigation {  float: none ; }
}
```

Using the @media rule you can also set what is called device specific rules. No I do not mean using different CSS for every brand and model of every desktop, laptop, cell phone, and table. That would be absolutely brutal. However, slightly less horrendous is the idea of setting common devices screen dimensions and pixel ratios.

You could technically accomplish something similar to this by using the @media handheld rule. However, because the industry has grown to include such a wide variety of devices that are arguably handheld with all different screen setups. This is not commonly used because it is just a little too vague.

In place of the @media handheld rule we prefer to set height and width. There are several ways to do so as the properties available include device-width, device-height, min-device-width, max-device-width, min-device-height, and max-device-height. Using a combination of these properties set to the appropriate values we can target the computed resolution of many devices regardless of their exact screen size. You can apply as many conditions as you need as long as they are separated by "and" as well as placed in their own set of parenthesis. A common example may look like:

```
@media screen and (min-device-height: 768px) and (max-device-width: 1024px) {
#navigation { float: right ; display: block ; }    }
```

We can also set the pixel ratio in a similar manner. It is important to recognize that a CSS pixel does not need to be the same exact size as a physical pixel. Often times a screen that is 720 pixels wide physically will display a browser that loads to show a space that is 480 pixels wide. This is actually done intentionally by the browser so that a normal web page will be more likely to fit the screen. This particular example uses a pixel ratio of 1.5:1. This means that there are 1.5 physical pixels for every CSS pixel. A standard desktop monitor would typically have a pixel ratio of 1:1 meaning 1 CSS pixel to every 1 physical pixel. Applying rules to only devices with a specific pixel ratio is pretty straight forward. To apply this technique to an iPhone 4 or newer which have a pixel ratio of 2:1 all you have to do is set the device-pixel-ratio as such. As you can see in the example below:

```
@media (device-pixel-ratio: 2) {
body { background: url(iphonebackground.png) }
-webkit-device-pixel-ratio
        }
```

Just like pixel ratio. CSS also lets us set the aspect ratio in the same way. The aspect

ratio is written out as a fraction. Other than that it is used the exact same as pixel ratio. An example of this could look like:

```
@media screen and (device-aspect-ratio: 16/9) {
body { background: url(sixteenninthsbg.jpg) }
        }
```

Last but definitely not least we can also apply styles depending on the screens resolution. This is done using the resolution and min-resolution properties. Both of these properties use values with a dpi unit. As an example this may look like:

```
@media screen and (resolution: 326dpi) { margins: 5px ; }
@media screen and (min-resolution: 96dpi) { margins: 0 ; }
```

Practical Use Exercise Questions

1. What is an advanced capability of the @media rule?
2. What two properties allow you to target a specific screen size?
3. What two values are accepted by the orientation property?
4. What are the six device specific properties?
5. In a pixel ratio which value goes first, the physical pixel count or the CSS pixel count?
6. What units are used to define resolution or min-resolution properties?

Chapter Summary and Exercise Solutions

- The @media rule allows you to use a different design style depending which screen size the user is viewing the content on.
- The two specific properties that allow you to target a specific screen size are max-width and min-width.
- The two values accepted by the orientation property are landscape and portrait.
- The six device specific properties are device-width, device-height, min-device-width, max-device-width, min-device-height, and max-device-height.

- In a pixel ratio the physical pixel count is the first number and the CSS pixel count is the second number.
- The dpi unit defines the value of resolution properties.

CONCLUSION

This book has found you because you have the ultimate potential.

It may be easy to think and feel that you are limited but the truth is you are more than what you have assumed you are. We have been there. We have been in such a situation: when giving up or settling with what is comfortable feels like the best choice. Luckily, the heart which is the dwelling place for passion has told us otherwise.

It was in 2014 when our team was created. Our compass was this – the dream of coming up with books that can spread knowledge and education about programming. The goal was to reach as many people across the world. For them to learn how to program and in the process, find solutions, perform mathematical calculations, show graphics and images, process and store data and much more. Our whole journey to make such dream come true has been very pivotal in our individual lives. We believe that a dream shared becomes a reality.

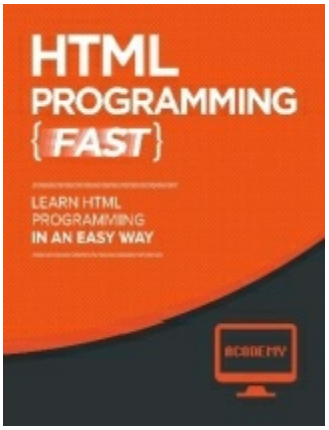
We want you to be part of this journey, of this wonderful reality. We want to make learning programming easy and fun for you. In addition, we want to open your eyes to the truth that programming can be a start-off point for more beautiful things in your life.

Programming may have this usual stereotype of being too geeky and too stressful. We would like to tell you that nowadays, we enjoy this lifestyle: surf-program-read-write-eat. How amazing is that? If you enjoy this kind of life, we assure you that nothing is impossible and that like us, you can also make programming a stepping stone to unlock your potential to solve problems, maximize solutions, and enjoy the life that you truly deserve.

This book has found you because you are at the brink of everything fantastic!

Thanks for reading!

You can be interested in: “**HTML**: Learn HTML FAST!”



Here is our full library: http://amzn.to/1HPABQI

To your success,

Acodemy.