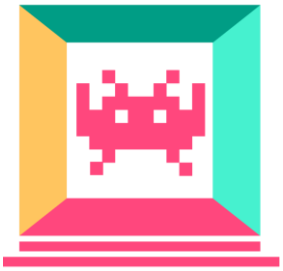


A Guide to HTML5 and CSS3

By

Ashley Menhennett

Professional Web Designer and Developer

Pablo Farias Navarro

Web & Mobile Application Developer and Founder of Zenva

About Ashley

Ashley Menhennett

Ashley is a web designer + developer, programmer and technical author from Australia.

Ashley holds certification in IT + Web Based Technologies and specializes in modern mobile and responsive web design, working with bleeding-edge web based technologies—including HTML5 and CSS3.

About Pablo

Pablo Farias Navarro

Pablo is a web + mobile application developer, entrepreneur and owner of ZENVA from Chile.

Pablo holds a Master in Information Technology (Management) degree from the University of Queensland (Australia) and a Master of Science in Engineering degree from the Catholic University of Chile. Specialized in web and mobile HTML5, his preferred technologies for the development of web and mobile apps are PHP, MySQL, JavaScript and NodeJS.

Before diving into this eBook, why not check out some resources that will supercharge your coding skills:

Table of Contents

Opacity

Alpha Color Space

Box Shadow and Border Radius

Exercise fifteen:

Part 1: The Basics of HTML and HTML5

Requirements

What do I need to start developing with HTML?

> There are absolutely no requirements to start learning HTML, but you will need some tools to help you along the way. There are two tools that are essential to becoming an efficient and professional Web Developer.

Firstly, you will need a Text Editor.

> Windows users, you can get an awesome text editor from notepad-plus-plus.org. As you have probably guessed from the name of the URL, this text editor is Notepad ++ and includes some cool syntax highlighting!
>
> Mac fans, you can get a text editor from Bare Bones. This handy text editor- Text wrangler also supports syntax highlighting for a range of programming languages. Personally I use Text Wrangler.
>
> Linux lovers, you can use the default gnome text editor- gedit. If it's not already installed, you can get it over here at gnome.org.

> *Do I need syntax highlighting? What is syntax highlighting?*
>
> Syntax highlighting allows you to easily see certain elements of your code in a designated color. For example, when using a WYSIWYG (what you see is what you get) editor with syntax highlighting some editors may render the HTML structure as blue, comments as grey and attributes and values as different colors. This allows you to easily distinguish between the sections of code, elements and comments.

Secondly, you will need a browser to render your code. *I recommend Firefox.*

> For the purpose of this eBook, go ahead and download and install Mozilla's Firefox (Cross-platform) Browser and the Firebug add-on for Firefox. The

Firebug add-on will be your new best friend as a learned web developer.

The Fire-bug Firefox add-on provides the ability to closely "inspect" the elements of an HTML document and see what's going on behind the scenes- this will play a big part in your web development career!

Alternatively you can use any other browser that supports HTML5- including Safari, Google's Chrome and Opera.

Definition of HTML

HTML stands for Hyper Text Mark-up Language. HTML is the basis for all things Web and is a necessary skill for any Web Developer. Almost every website is comprised of HTML whether that is a variation of HTML or plain old HTML.

Structure

Let's start with a basic HTML5 structure:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>

    </body>
</html>
```

Exercise one:

1. Type out the above code into a new text file.

2. Save the file with an .html extension (i.e. structure.html).

3. Open the html file with your Browser.

4. What do you see?

No Output?

The Browser didn't seem to show us any content did it?

The browser would not show us any content (output), as we have not yet told the html document to output anything to the browser. All we have told the browser is that we have an HTML document.

Validation

Even though all we have created is a HTML structure and we are not seeing any results (yet), the HTML document we have should validate with W3C's online HTML validation tool.

W3C is the World Wide Web Consortium- they set the standards for HTML (and many other Web Based Languages) to provide a similar cross-browser experience; meaning that web browsers will be more inclined to output (or render) data in a similar fashion.

Exercise two:

1. Go to W3C's online HTML validation tool,

2. Select the third tab along- "Validate by Direct Input",

3. Copy and paste your HTML5 document code into the window and click 'Check'.

4. Notice how the document passed validation with three warnings.

Let's fix these warnings:

1) The warning- "Using Experimental Feature- HTML5 Conformance Checker" is basically telling us that all major browsers do not officially support HTML5 yet.

2) The next warning- 'No character encoding declared at document level'- this is because we haven't declared our character encoding within the HTML structure.

3) The final warning is telling us that no matter what our character encoding is set to within our HTML document that we are validating, it is going to assume and treat is as UTF-8. The logical way to overcome this last warning is to use the file upload tool, rather than the Direct Input.

Now, let's try and use the W3C validation tool to upload our HTML document, by selecting the second tab on the W3C's online HTML validation tool page and uploading our HTML document.

More Warnings

If you notice we still have our '*Using Experimental Feature- HTML conformance*

checker' warning (which is perfectly fine, as we are using HTML5 and it is not yet fully supported by all browsers), the other 2 warnings are related to our character encoding and so is the Error we are now seeing.

Let's fix this, by declaring our Character Set. We will be using the UTF-8 Character encoding and we can do this by adding some simple mark-up to our head section.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>

    </body>
</html>
```

We have just added our first HTML Meta tag!

This Meta tag lets the browser know that we are using UTF-8 character encoding.

UTF-8 is the most commonly used character encoding, basically it provides a standard format (encoding) for text (code) that will assist against the problems of endianness, which could result in incorrect or invalid characters displaying (http://en.wikipedia.org/wiki/Endianness).

Tags inside the head element of a HTML document are often used to tell the browser information about the HTML document that we don't need to output as part of our content, such as our HTML title and character encoding.

Comments

In every programming language comments are widely used to help remind other developers what is happening in the code, to make note of extra code that will be added to the web application at a later date and notes to others who may be working

on the same project.

In HTML, comments are easily added to the document, by adding the opening and closing comment tags.

```
<!--
    This text will not be rendered by the browser
-->
```

Our browser will not render anything placed inside the comment tag.

Example of Comments

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <!--
            This is a comment
        -->
    </body>
</html>
```

HTML5 Structure revisited

Let's go through our HTML document and talk about the structure step-by-step, using comments.

```
<!DOCTYPE html>
<!-- declaring the document type -->
<html>
<!-- opening html tag -->
    <head>
    <!-- opening head tag -->
        <meta charset="UTF-8">
        <!-- declaring our character set -->
        <title>Our Title</title>
        <!-- opening and closing title tags -->
    </head>
    <!-- closing head tag -->
    <body>
    <!-- opening body tag -->
        <!--
            This is where we put our content
        -->
    </body>
    <!-- closing body tag -->
<!-- closing html tag -->
</html>
```

Exercise three (Part 1):

1. Type the above code into a new file.

2. Save the file with an .html extension (i.e. template.html).

3. Open the html file with your Browser.

4. What do you see?

Nothing has been output by the Browser, as we have used comments to explain the structure and have not yet added any "*real*" content.

Inspecting with Fire-bug

Exercise three (Part 2):

1. Open up the Fire-bug Firefox add-on. You can do this by right clicking on the Firefox Browser window and selecting “Inspect Element with Fire Bug”.

2. Notice how we can see exactly what we have typed into our HTML document on the left-hand side of the Firebug window.

So how can we tell the browser to output some data?

It's actually quite simple. But before we add any content to the document, let's talk about the title tag. The title tag allows us to specify the name of the website- more specifically, the web page. It is good practice to be as relevant as you can when giving your web page a title.

As you can see in the previous examples, the title tag is inserted into the head section of the HTML document.

```
<title>Title of the Webpage</title>
```

If we load up this HTML document in our Browser now, we won’t see any changes to the webpage. But, have a look at the top of your browser window or current tab- this is where the Title of your HTML document is shown.

Adding Content

We can add our content in-between our body tags like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Hello World</title>
    </head>
    <body>
        Hello World in HTML5
    </body>
</html>
```

Exercise four:

1. Type the above code into a new file with your text editor.

2. Save the file with an .html extension (i.e. hello_world.html)

3. Open the html file with your Browser and see what the browser is rendering.

4. Remove the “Hello World” text and replace it with a sentence about your favourite season and start to get comfortable with coding in HTML.

5. Make sure you view your html document in your browser and validate it.

HTML Notes:

- All versions of HTML require the basic HTML structure, but the version of HTML depicts a vast difference in the required elements and doc type declarations. HTML4.01, XHTML and HTML5.

- Notice how every tag is closed in order of which they were opened? This is a

very important element to valid HTML.

- HTML5 is not currently supported by all major browsers, but provides plenty of extra features for us to work with and stay ahead of the curve. Although all major browsers do not support HTML5, Google's Chrome, Opera and FireFox are currently the most useful tools for Modern Web Development.

- If you are not seeing this "Inspect Element with Fire-bug" option in the drop-down menu, when you right 'click' on your browsers main area- Take a look at this helpful documentation at mozillaZine.

Part 2: HTML Elements

The Paragraph tag

In HTML, the paragraph tag is part of the block level elements group.

Block level elements will generally start on a new line and any Mark-up under or after the block level element will also start on a new line.

Here is an example of a paragraph tag in HTML, followed by some text after the ending paragraph tag. Even though all of the text is on one line, the paragraph tag (block level element) will place the text after the closing paragraph tag on a new line.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Block Level Elements</title>
    </head>
    <body>
        <p>I am a paragraph</p>I am directly after the paragraph
    </body>
</html>
```

Output:

I am a paragraph

I am directly after the paragraph

Other Block Level Elements

There are a variety of other block level elements available in HTML; including Headings, logical (or document) divisions, horizontal rules, ordered lists and unordered lists.

So, let's check out some of these block level elements in action.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Block Level Elements</title>
    </head>
    <body>
        <h1>Level 1 Heading</h1>
        <h2>Level 2 Heading</h2>
        <div>This is a logical division</div>
        <hr>
        I will have a horizontal rule above me
        <ol>
            <li>I am part of an ordered list</li>
            <li>li stands for list item</li>
            <li>You can add as many as you want!</li>
        </ol>
        <ul>
            <li>I am still a list item</li>
            <li>This time I am inside an un-ordered list</li>
            <li>Notice how I'm rendered in the browser</li>
        </ul>
    </body>
</html>
```

Output:

Level 1 Heading

Level 2 Heading

This is a logical division

I will have a horizontal rule above me

1. I am part of an ordered list
2. li stands for list item
3. You can add as many as you want!

- I am still a list item
- This time I am inside an un-ordered list
- Notice how I'm rendered in the browser

Exercise five:

1. Take note of the code above.

2. Type out the code into a new html document.

3. Change the content of the code to be about your favourite dessert.

4. Add an extra ordered list (containing 7 list items) and a logical division (containing a paragraph element) to the end of the page.

5. Save the html document as “block_level_elements.html”.

6. Open the document with your browser and make sure it appears as intended.

7. Upload the html document to the online validator and correct any warnings using the skills you have learned thus far.

Line Breaks vs. Paragraphs

The break element or tag in HTML (as you can guess) provides a line break (or new line). Some people may like to 'over-use' this element, but I suggest using the paragraph element when dealing with text (where possible), to provide formatting and appropriate spacing.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>The Break Tag</title>
    </head>
    <body>
        This text is on one line
        <br>
        This text is on another line
    </body>
</html>
```

We could do the same thing with the paragraph tag, but with a better format.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Block Level Elements</title>
    </head>
    <body>
        <p>This text is on one line</p>
        <p>This text is another line</p>
    </body>
</html>
```

Exercise six:

1. Type out the above code into a new html document.

2. Add some additional break tags and paragraph elements containing text.

3. Save the file with as “break.html”.

4. Open the html file with your Browser and take note of how each tag performs *almost* the same task.

Inline Elements

Now we have an understanding of what block level elements are, it’s time to move on to some inline elements.

Text Modifiers- Introducing the strong and em tags.

As you may have guessed, the strong tag is used to define important text and will render text as bold.

The em tag is a little harder to guess. The em tag renders text as *Italic* and is used to *'emphasize'* text. The Strong and *em* tags are both part of the Text Modifiers group.

Example:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Text Modifiers</title>
    </head>
    <body>
        The <strong>strong tag</strong> will render bold text.
        <br>
        The <em>em tag</em> will render italic text.
    </body>
</html>
```

Text modifiers can be a simple way to make certain text stand out or add character to a document. Just *like* **this**!

Images

Images are an important part of any form of content, especially websites. As a web developer, you will find it very helpful and necessary to be able to place images onto a web page

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Images</title>
    </head>
    <body>
        <img src="" alt="">
    </body>
</html>
```

We would then put the URL for our image inside the src (source) attribute. The URL could be relative or absolute.

Here is an example of using an absolute URL with the src attribute of the img tag:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Images</title>
    </head>
    <body>
        <img src="http://glorified.me/image.png" alt="">
    </body>
</html>
```

When working in a Live or Development environment it is good Practice to use relative file Paths, rather than absolute or full file Paths.

- A relative file Path can be defined as being a localised link; using the current directory Structure as a means of navigation between files.

- An absolute file Path is a direct link or URL to a file.

If we had an image titled 'logo.png' in the same folder or directory as our Current html file, we could simply link to that file just by using the files name:

```
<img src="logo.png" alt="">
```

If our image or file were in a directory titled "images" inside our Current folder or directory we would then link to The Image using

```
<img src="images/logo.png" alt="">
```

Sometimes we need to navigate downwards (as opposed to upwards) in our directory Structure. If our directory Structure looked something like:

/home/html/Public/Current/

And our Current html document is in our "current" folder; we could link to our Image (which Could be located at /home/html/Public/Images/) by using:

```
<img src="../images/logo.png" alt="">
```

../images/ basically tells the browser to navigate one directory down and then into our Images directory.

Alternate Text

When an image is unable to be displayed by a browser we need a fallback method.

So the alt (alternate text) can be used as our fallback method- meaning we will have some descriptive text to display if the image itself is unable to be displayed for any reason.

An example of an image not displaying could be a HTML email (Gmail will, by default hide any images and ask the user if they want to show images) or the results in a search engine. Search Engines cannot “read” images, so they can only render “alternate” text in Search Engine Result Pages (SERPs).

It is good to be descriptive and short with the alt attribute, like so:

```
<img src="../images/logo.png" alt="HTML5 Logo">
```

Displaying an Image

So let’s try out our image tag with a real image!

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Images</title>
    </head>
    <body>
        <img src="html5-logo.png" alt="HTML5 Logo">
    </body>
</html>
```

Output:

I feel as though it would be a better idea to have the image a little smaller, wouldn't you agree?

Specifying the Size of an Image

We can specify both height and width attributes inside our image tag like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Images</title>
    </head>
    <body>
        <img src="html5-logo.png" width="100" height="100" alt="HTML5 Logo">
    </body>
</html>
```

Note: The height and width values are in px (pixels).

So, let's try out this image with the height and width attributes specified!

I am going to guess... It looks a little more appropriate now!

Exercise seven:

1. Create a new HTML5 document.

2. Using the skills you have learned so far, add an image (of your choice) with a width and height of your choosing to your HTML5 document.

3. Load your HTML5 document in your browser and make sure it renders as expected.

4. Head over to the W3C's HTML Validator and validate your HTML5 Document. Correct any errors you receive.

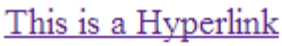
Anchor Tags/ Hyperlinks

Now, let's move on to the ever-so important anchor tags.

We use an anchor tags like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="http://glorified.me">This is a Hyperlink</a>
    </body>
</html>
```

The above code will render as:



Let's try a simple link to Google:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="http://google.com">Go to Google?</a>
    </body>
</html>
```

The above example will render:

Type out the above code and Try it out, notice that the browser will now load Google's homepage when you click on the link.

We have covered how to link to a page, but what if we want our users to go to our

linked page, but in a new window (so they don't leave our interesting website)?

We can do just this by using the target attribute of the anchor tag and passing in a value of '_blank'.

Opening in a New Window:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="http://google.com" target="_blank">Go to Google?</a>
    </body>
</html>
```

The above example will render:

Go to Google?

Type out the above code and Try it out!

Notice that the browser will now load Google's homepage in a new window when you click on the link.

Hypertext Reference

The most important attribute for the anchor tag is the href attribute. The href (Hypertext Reference) attribute will tell the anchor tag where to link to, or where to send the user once clicked on.

This example is slightly different to the previous examples:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="#">Where to?</a>
    </body>
</html>
```

Linking inside a HTML document

Basically, the '#' symbol can also act as a page anchor, when nothing is assigned to the '#' in the href attribute of an anchor tag- when the user clicks on it, as the link does not have a specific location- it will generally go nowhere.

Now, we can assign id attributes to some of our elements, to use our anchor tags to link to specific parts in our HTML, rather than just having the '#' symbol, we would use:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags and Ids</title>
    </head>
    <body>
        <div id="top">Top of the Page!</div>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin ultricies ligula eget lorem
malesuada euismod. Phasellus ac interdum mauris. Vivamus hendrerit auctor arcu, vitae iaculis sem
fringilla eget. Pellentesque adipiscing erat at mollis pellentesque. Aenean fringilla lectus et massa
laoreet hendrerit. Proin vehicula ante non mi molestie semper. Vivamus rutrum elit at sem mattis,
quis mollis neque dapibus. Aliquam sit amet justo nunc. Nunc vel pharetra quam. Morbi accumsan velit
at aliquam lobortis. In luctus, ipsum et accumsan viverra, nisi ipsum tristique enim, ut pellentesque
dolor leo quis nisi. Pellentesque porttitor ultrices sollicitudin. Cras ut odio erat. Duis gravida
pretium tellus, lacinia rutrum sapien vestibulum at.

        Proin vestibulum ante mattis, rutrum quam vel, congue nisl. Ut ornare dignissim urna,
consequat sollicitudin urna. Proin condimentum quis risus nec ultricies. Nulla tincidunt massa
ligula, vel volutpat magna consequat eget. Praesent viverra metus a venenatis mollis. Donec laoreet
fermentum lacinia. Donec vitae sodales mauris.
        <a href="#top">Top of Page?</a>
    </body>
</html>
```

I have added a logical or document division at the start of the above example with an id of top.

Ids are a very useful feature of HTML, but for now we are just going to use it for an anchor link (something to link to). We could assign our div with any id value, as long as we reference it in our href attribute of our anchor tag.

Exercise eight:

1. Create a new HTML document and save it as anchors#.html.

2. Add a logical division with an id of ‘top’ and an anchor tag with a href value of ‘#top’.

3. Head on over to lipsum.com and generate some ‘dummy text’, copy it and paste it in between the logical division and the anchor tag you have placed in your HTML document. Save your document.

4. Open your anchors#.html document in your browser, scroll down to the bottom of the page and click on the link you have created.

3. Notice how the link takes you to the top of the page (The logical division with an id of ‘top’).

Note: Make sure that there is enough text to actually cover the height of the page.

Email Links

In HTML we can create a matilto: link, when the user clicks on this link their email client will open and the mailto: value (our email address) will be added to the TO:

field.

```
<a href="mailto:admin@glorified.me?Subject=Email">Contact Us</a>
```

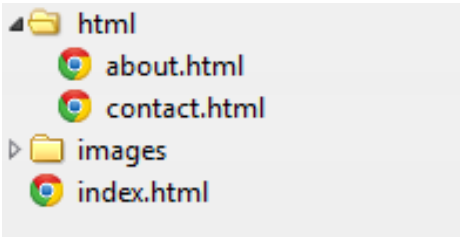
This makes it easy and enticing for a user to quickly send us some email, regarding our web page.

You may have noticed that I have added '?Subject=Email', this will add “Email” to the subject field within the email. You can change the mailto and Subject values to suit your needs.

Directories

You will often be using a folder structure. The folder structure is a crucial part of good coding practice and helps to tidy up our files (an images folder and an html folder).

A basic html folder structure would look similar to this:



Note: When working with folder in Web Development, we refer to folders as directories.

As you can see we have a few html files in different locations. We have our index.html file, which; when working in a server environment will automatically load once we navigate or load the specific folder that index.html resides in. As we are not working in a server environment this is not required knowledge at this point in time.

We have an about.html and a contact.html file in our “html” directory. The content of these two files are irrelevant at this point in time. The purpose of the following

exercise is to make sure you have a grasp of 'navigating the directory structure'.

Example of linking to the about.html page from index.html:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="html/about.html">About Us</a>
    </body>
</html>
```

Exercise nine:

1. Create three new html documents named; about.html, contact.html and index.html.

2. Create the same directory structure as shown above and place your new html files in the appropriate directories.

3. Give each html file an appropriate title tag and a level 1 heading (h1).

4. Add a paragraph of appropriate text to each of our html pages and a mailto: link to the contact page.

5. Now that we have some text and a heading for each html document, we need place an image (you can use any image you would like) on each page.

6. Using you skills, you can add an image under the paragraph tag to each html document and a link to each other html document in the directory structure under the image.

7. Test out your html documents by saving them with the same names as detailed above. Make sure the images in your html documents are appearing in the browser and when you click the links under the images, you are taken to the respective html document (i.e. a link to about.html should take you to the about.html page when you click on the link) in your browser.

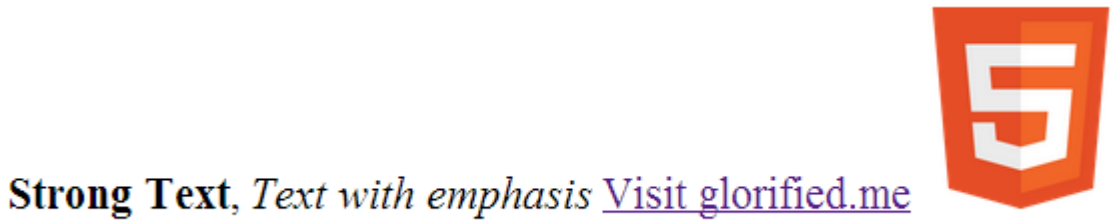
8. Now that you have completed your three html pages, go ahead and validate them with the online validator and fix any errors that appear.

Inline Elements in Action

Let's check out these inline elements in action.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Inline Elements</title>
    </head>
    <body>
        <strong>Strong text</strong>, <em>Text with emphasis</em>
        <a href="http://glorified.me">Visit glorified.me</a>
        <img src="html5-logo.png" width="100" height="100" alt="HTML5 Logo">
    </body>
</html>
```

Output:



Notice how the contents of the strong, em, anchor and image tags are being

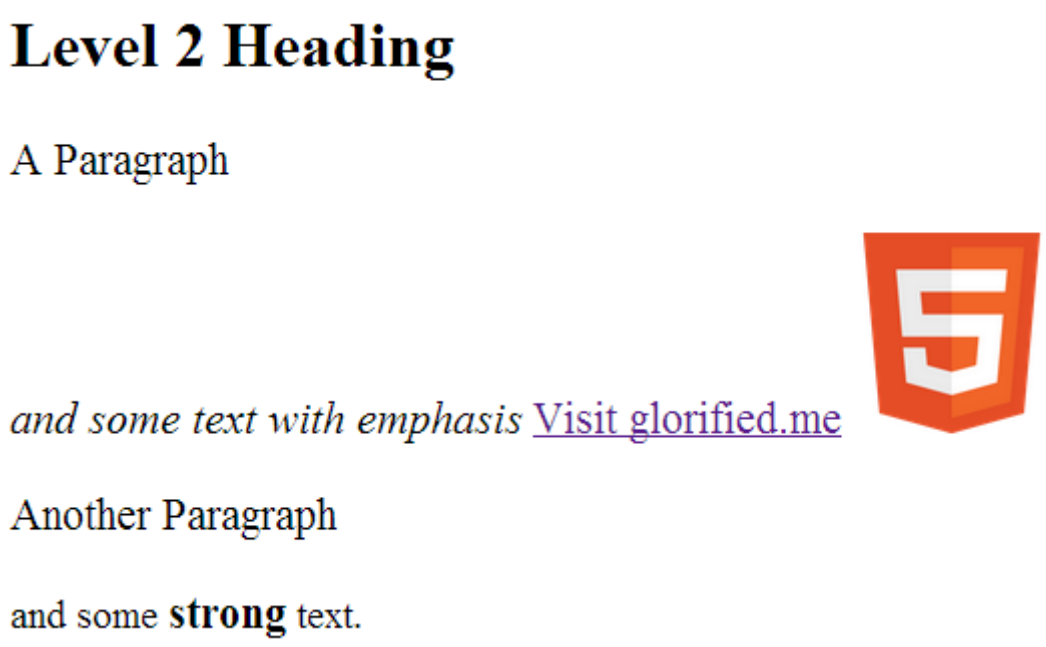
displayed on the same line, rather than being displayed on separate lines- like the block level elements.

Inline & Block Elements in Action

So, let's try some block level and inline elements together.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Block Level and Inline Elements</title>
    </head>
    <body>
        <h2>Level 2 Heading</h2><p>A Paragraph</p><em>and some text with emphasis</em>
        <a href="http://glorified.me">Visit glorified.me</a>
        <img src="html5-logo.png" width="100" height="100" alt="HTML5 Logo">
        <p>Another paragraph</p> and some <strong>strong</strong> text.
    </body>
</html>
```

Output:



Tables

Sometimes when we have some information to display on our web page, it makes sense to display that information or data in a table. Tables in HTML are relatively simple. We have the Opening Table Tag and The closing Table tag. Inside of our Table element, we have table rows. Inside the table rows we have tabular data or cells. But, for our table headers, we will be using the table header tags inside our table row.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Tables</title>
    </head>
    <body>
        <table>
            <caption>My HTML5 Table</caption>
            <!-- the caption tag is new in HTML5 -->
            <tr>
                <th>Color</th><th>Width</th><th>Height</th>
            </tr>
            <tr>
                <td>Black</td><td>55</td><td>99</td>
            </tr>
            <tr>
                <td>Green</td><td>85</td><td>45</td>
            </tr>
        </table>
    </body>
</html>
```

Forms

One of the many things you may have noticed on a web page is a contact form for example. We can create a form in HTML by using the form opening and closing tags. Inside of our Form element we have inputs and a submit button. Some of our inputs will be of type text and our submit button will actually be an input of type submit. When we work with HTML forms in the real world, there are two attributes that we need to add to our opening form tag.

Method and Action:

> The scope of this eBook will not be covering server-side processing of HTML data, but it is helpful to know what these attributes do.

For the action attribute, you will enter in the destination of the Form data. The method will take either a POST or GET value. POST and GET are used with server-side processing.

For the action attribute, you would enter in the destination of the Form data. The method will take either a POST or GET value. You would generally be using POST to submit most forms of data. The main difference between POST and GET is that POST sends data to the server 'behind the scenes', whereas GET will form a query string. A query string consists of name attribute values and the values input by the user. As you can imagine, if we were submitting sensitive data to the server we would definitely not use GET; in the case that we did, all information input by the user would be clearly visible in the URL address bar.

For the purpose of this eBook, we will be taking a look at forming a query string, using GET.

Type out the following code and select some values from the form and click the submit button. Take a look in your URL Address bar; this is called a query string- the part following the '?'.

```
<form action="#" method="GET">
    <label for="option1G1">Radio Button 1:</label>
    <input id="option1G1" type="radio" name="radio1" value="option1G1">
    <br>
    <label for="option2G1">Radio Button 2:</label>
    <input id="option2G1" type="radio" name="radio1" value="option2G1">
    <br>
    <label for="option1G2">Radio Button 3:</label>
    <input id="option1G2" type="radio" name="radio2" value="option1G2">
    <br>
    <label for="option2G2">Radio Button 4:</label>
    <input id="option2G2" type="radio" name="radio2" value="option2G2">
    <br>
    <label for="cb1">Checkbox 1:</label>
    <input id="cb1" type="checkbox" name="cb1" value="cb1">
    <br>
    <label for="cb2">Checkbox 2:</label>
    <input id="cb2" type="checkbox" name="cb2" value="cb2">
    <br>
    <input type="submit">
    <input type="reset">
</form>
```

Again, type out the following code, load it up in your browser, enter some text into the inputs in the form and click the submit button. Take a look in your URL Address bar.

```
<form action="#" method="GET">
    <label for="first">First Name: </label>
    <input id="first" type="text" name="first">
    <br>
    <label for="last">Last Name: </label>
    <input id="last" type="text" name="last">
    <br>
    <label for="password">Password: </label>
    <input id="password" type="password" name="password">
    <br>
    <input type="submit">
    <input type="reset">
</form>
```

The label tag allows us to add descriptive text regarding the input expected from the user and when the user clicks the label text, focus will be drawn to the input.

To use labels properly, you must give the label a ‘for’ attribute and a value of the ‘for’ attribute that corresponds to the input’s id.

When you use an input of type password, you may think that because the input entered renders as dots that it must be secured. This is a common misconception. The dots that render for password inputs are only a masking mechanism. Meaning that the input is actually still just the plain text that the user enters, but the password field will mask this to prevent prying eyes when a user is entering in a password. By using GET, you have just seen (in the query string) that the input entered into a password field remains plain text.

Exercise ten:

1. Create a new .html file titled tables-forms.html.

2. Add a table (4x7) and type in 4 of your favourite bands and seven of their best songs, to fill out the table. By doing this simple exercise, you will learn how to create a HTML table of any size.

3. Under the new table that you have created; add a form.

4. This form will have 4 radio inputs, 3 Check boxes, 1 text and 1 password input field.

5. Appropriately name each input and test that you have done so correctly by entering in text/ making selections and submitting the form (clicking 'submit'). Pay close attention to the query string that has formed in your URL address bar.

6. Create some new inputs of type: color, number, URL, date and email and test them out in your browser!

PART 3: Main HTML5 Specific Elements

So far we have only really learned about general HTML tags and elements, now it's time to get into some HTML5 specific elements. As you may notice when you start to work with HTML and build some of your own web pages, it would be really helpful if there was a logical way to define the header and footer of our web page With HTML5, we can just that with the header and footer tags:

Header & Footer

```
<header>This is the Header Element</header>
```

```
<footer>This is the Footer Element</footer>
```

The header element defines a header for our html document. We could also have multiple headers in our html document, to define headers for different parts of our html.

The footer element defines a footer for our html document; just as we can with the header element we can also have multiple footer elements within our html document- defining footers for different parts of our html.

Along with these new header and footer elements, there are also a few more important elements that have been introduced with HTML5.

Navigation

We can now define a navigational element with the nav tag:

```
<nav>This is the Nav Element</nav>
```

We would use this element to hold our **main** navigational content.

Section

We can now define a ‘section’ of our HTML document. Do keep in mind that the contents of a section should be related.

```
<section>This is the Section Element</section>
```

Article

We can now define an 'article' within our HTML document. Within a section, we could have several articl

```
<section>
    <article>
        This is the Article Element
    </article>
    <article>
        This is another Article Element
    </article>
</section>
```

Aside

We can now define an 'aside'; a part of our HTML content that is 'aside' from our main content, but is still related:

```
<article>
    This is another Article Element
    <aside>
        This is the Aside Element
    </aside>
</article>
```

The Meter Element

One of the really cool things with HTML5 is the ability to add meters. We define the meter element with the opening and closing meter tags.

The HTML5 meter tag will take a few attributes, min, max and value.

The Min and Max values will define a range in which our value will be compared. For example, with a min of 0 and a max of 100, a value of 50 will show a ‘gauge’ that is 50% complete:

```
<meter min="0" max="100" value="50">This is the Meter Element</meter>
```

The text that I have entered in between the opening and closing meter tag is fallback text. This fallback text will be rendered in older browsers that do not yet support the meter tag.

The new HTML5 Elements in Action

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>HTML5</title>
    </head>
    <body>
        <header>This is the header element</header>
        <nav>This is the nav element</nav>
        <section>
            <article>
                This is the article element
                <aside>
                    This is the aside element
                </aside>
            </article>
            <article>
                This is another article
            </article>
            <article>
                And, yet another article
            </article>
        </section>
        <footer>This is the footer element</footer>
    </body>
</html>
```

Exercise eleven:

1. Using only the new HTML5 tags, create a Valid HTML5 document with some content about your favourite holiday.

2. Validate this HTML5 specific document with the online validator and fix any errors.

Video

One of the greatest features of HTML5 is the ability to implement a fully featured Video with Controls.

We start out with the opening Video tag, passing in a height & width and the controls attribute, so the video player will have controls. Next we need to add the source tag and actually add the Video URL to the src attribute of the source tag. Again this can be an absolute or relative URL.

Along with our actual video src, we need to tell the browser the mime-type. A mime-type basically lets the browser know what kind of format or media type to expect.

To provide a similar cross-browser experience you will need to supply multiple formats, contained within separate source tags.

Here we have the video Element set to 640x480px with controls displayed, some fallback text (to display some text in a browser that doesn't support the video element) and two sources- each have a different format and mime-type.

```
<video width="640" height="480" controls>
    <source src="video.mp4" type="video/mp4">
    <source src="video.webm" type="video/webm">
    Fallback Text
</video>
```

Audio

As you have just learned about HTML5 Video, HTML5 Audio isn't going to be all that difficult to grasp. HTML5 audio isn't all that different from HTML5 Video (apart from the fact that it's audio and not video).

With the audio element, we do not need to set the size, but it is ALWAYS good practise to use the controls attribute so your users can actually operate the Audio Player.

Just as we can have multiple sources in our video element, we do the same thing with our audio element; passing in audio files and appropriate mime-types.

```
<audio controls>
    <source src="audio.mp3" type="audio/mpeg">
    <source src="audio.ogg" type="audio/ogg">
    Fallback Text
</audio>
```

Part 4: The Basics of CSS

Definition of CSS

CSS stands for Cascading Style Sheets and provides HTML with layout and design. Along with making things pretty and aesthetically pleasing, CSS also provides a general structure to HTML.

Some of the most important CSS properties (in my opinion) are (in no order):

- Color - specifying text color.
- Font-family - specifying font type.
- Font-size - specifying font size.
- Text-decoration - specifying text decorations, such as underline.
- Font-style - specifying font styling, such as italics.
- Font-weight - specifying font weight, such as bold.
- Width - specifying the width of an element.
- Height - specifying the height of an element.
- Background - specifying the background.
- Border - specifying a border.
- Text-shadow - specifying a shadow for our text.
- Float - specifying the float of an element, such as left or right.
- Position - specifying the position of an element, such as absolute or relative.
- Z-index - specifying the z-index of an element, such as 999; which would put that styled element 'on-top' of all other elements that either have a negative z-index specified or no z-index specified.
- Padding - specifying padding inside an element, such as padding around text.
- Margin - specifying the margin between elements.
-

CSS can be implemented in three different ways to our HTML:

1. **Inline**

2. **Internal**

3. **External**

Using Inline CSS

So, let's use some inline CSS to change a few things in our HTML structure.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Inline CSS</title>
    </head>
    <body>
        <header>
            <h1 style="color:red">My Heading</h1>
        </header>
        <section>
            <article style="color:blue">
                My Article of Content
            </article>
        </section>
        <footer>
            <strong style="color:green">My Bold Text</strong>
            <br>
            <em style="font-weight:100">My Company</em>
        </footer>
    </body>
</html>
```

Output:

Color

As you have probably noticed, we have used the English word for the color that we

want to use. But there are two other ways we can define colors in CSS; the rgb color values and something called Hexadecimal.

All three types of defining colors in CSS are acceptable; you can read more about colors in CSS here: http://www.w3schools.com/cssref/css_colors.asp

We will be using a mixture of the three different ways to define colors in CSS.

Using Internal CSS

As you can see, our inline CSS is very effective, but perhaps not very efficient. Inline CSS is good for adding slight changes or specifying colors for different text elements, but it starts to get a little 'wordy' and messy.

When we add CSS to HTML either; externally or in the head section, we can use selectors.

Selectors allow us to 'select' or 'point' to a specific element of our HTML. CSS can use HTML elements as selectors, such as the paragraph, anchor, em and strong tags. If we referred to these elements as selectors in our CSS we would be styling every paragraph, anchor, em and strong element in our HTML.

Let's try the same thing, but this time adding our CSS to the head section of our HTML document and using selectors.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Internal CSS</title>
        <style>
            h1{
                color: #ff0000;
            }
            strong{
                color: #00ff00;
            }
            em.bottom{
                font-weight: 100;
                font-size: 1.1em;
            }
        </style>
    </head>
    <body>
        <header>
            <h1>My Heading</h1>
        </header>
        <footer>
            <em>My Italic Text</em>
            <strong>My Bold Text</strong>
            <br>
            <em class="bottom">My Company</em>
        </footer>
    </body>
</html>
```

I have added an additional em tag, to demonstrate using classes as selectors in CSS.

Using ids in CSS

As you may have guessed, we are using a class to identify our bottom em tag. The dot notation before the class name allows us to select or target an element in our HTML by its class name.

We can use ids like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Internal CSS</title>
        <style>
            h1{
                color: #ff0000;
            }
            strong{
                color: #00ff00;
            }
            em#bottom{
                font-weight: 100;
                font-size: 1.1em;
            }
        </style>
    </head>
    <body>
        <header>
            <h1>My Heading</h1>
        </header>
        <footer>
            <em>My Italic Text</em>
            <strong>My Bold Text</strong>
            <br>
            <em id="bottom">My Company</em>
        </footer>
    </body>
</html>
```

All we have to do is change 'class' to 'id' for the element we are referring to, and change the '.' in front of our CSS selector (in the head element) to the '#' symbol.

The # symbol (when not used as href attribute) is generally used to signify an id within the HTML. The major different between using classes and id's is; classes can be re-used time and time again in the same HTML document, whereas id's can only be used once in a single HTML document. You can think of a class as a group or multiple items, and an id as a single identification.

The output of the above code is the same (we have also set a font-size for the #bottom em tag) as the output for the previous code example, we are getting the

same results as we are basically telling the browser the same thing, just in a different way.

There are several ways to make selectors 'unique' or point to only 'some' parts of the HTML.

A class is an effective way of referencing a specific part of our HTML; we can basically pinpoint the section of our code that contains the content we wish to style.

Note: When using em in CSS it's slightly different to the em tag in HTML. In HTML the em tag renders italic text. In CSS the em value can be used as a unit of measurement. A font size with a value greater than 1em will generate text larger than the default for that web page or User Agent, but does not render text as italic.

Ids must be unique, we can only use the same id only once in our HTML page.

With classes, we can 'reuse' the class several times in our HTML page.

Creating External CSS

To add an external CSS to our HTML, we need to tell the HTML all about it- what relation it has to our HTML, the type of file it is and its location and name.

Remember the Meta tags from before?

Well this is implemented in the same way (completely different concepts), by adding a line of code into our head section of our HTML document. We use a rel value to tell HTML what the CSS file's relation is to the HTML, a type value to tell the HTML the type of file it is and a href value telling the HTML where the file is located and its name.

Note: CSS files have a file extension of .CSS

We can add an external style sheet to our HTML by using link tag.

So, let's create a small CSS file, to use externally.

```
header{
    color: rgb(255,0,0);
    font-size: 2em;
    text-decoration: underline;
}
section article{
    font-weight: 200;
    font-size: 1.1em;
}
footer{
    font-size: 0.8em;
}
```

Go ahead and save the above styling into a new CSS file, titled style.css.

Linking to External CSS

If our style sheet (CSS) were located in the same directory (or folder) as our HTML file, we would add the tag to the head section of the HTML document, like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>External CSS</title>
        <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <body>
        <header>
            <h1>My Heading</h1>
        </header>
        <section>
            <article>
                My Article of Content
            </article>
        </section>
        <footer>
            <em>My Footer</em>
        </footer>
    </body>
</html>
```

Just as our CSS example before, no matter of its location (inline, internal and external), the CSS will tell the browser to render the styles for our HTML in the same way.

Inefficient Selectors

Let's have a look at some inefficient CSS selectors with some external CSS:

```
header{
    color: red;
    font-size: 2em;
    text-decoration: underline;
}

section article{
    font-weight:200;
    font-size: 1.1em;
    color: red;
}

footer{
    font-size: 0.8em;
    font-weight: 200;
    color: red;
}
```

The advantages of using external CSS include the ability to completely separate the HTML from the CSS, to reduce individual file size and length, make things more readable and use effective selectors; meaning selectors that target multiple elements where necessary. Rather than having a large portion of CSS repeating and applying the same styling to different elements, we could 'join' the selectors together to create a smaller file size or to simply be more efficient with our use of CSS.

We can target or select multiple elements by separating the selectors with a comma in the CSS.

Let's fix this up!

Efficient Selectors

```
header{
    font-size: 2em;
    text-decoration: underline;
}

section article{
    font-size: 1.1em;
}

footer{
    font-size: 0.8em;
}

header, section article, footer{
    color: red;
}

section article, footer{
    font-weight: 200;
}
```

Exercise twelve:

1. Create your HTML5 document structure and create a file titled style.css

2. Add the new HTML5 elements and style the HTML accordingly:

    - Articles within a section will have a font-size of 2 times the default font-size.

    - The Footer and Header will have a text-decoration of underline and a color

of red.

- The aside will have a font-weight of bolder and a color of blue.

3. Save your CSS and HTML files and load them into your browser, checking that they render as expected.

HTML Element State

With our new CSS abilities, we are able to style a HTML element, based on its 'state'. HTML Element state refers to the 'state' that the elements are in; some of these include: Hover and Active.

You may have noticed that when you hover over a link on a web page, that the link will change color (among other aspects). We can do this with almost any HTML elements.

```
article:hover, article:hover a{
    background-color: #ff0000;
    color: rgb(255,255,255);
}
```

In the above example we have styled all 'hovered' over articles and the anchor tags, when the article is being 'hovered' over with a background of red and a text color of white.

Along with defining hover style, we can define active style. Active is defined by an element that is 'actively' being clicked on, i.e. if you are clicking a button, that button's state is now active.

```
article:active{
    background-color: #111111;
}
```

In the above example, an article that is 'active' will have a background color of almost black.

Exercise thirteen:

1. Create a new HTML document with an external Style sheet.

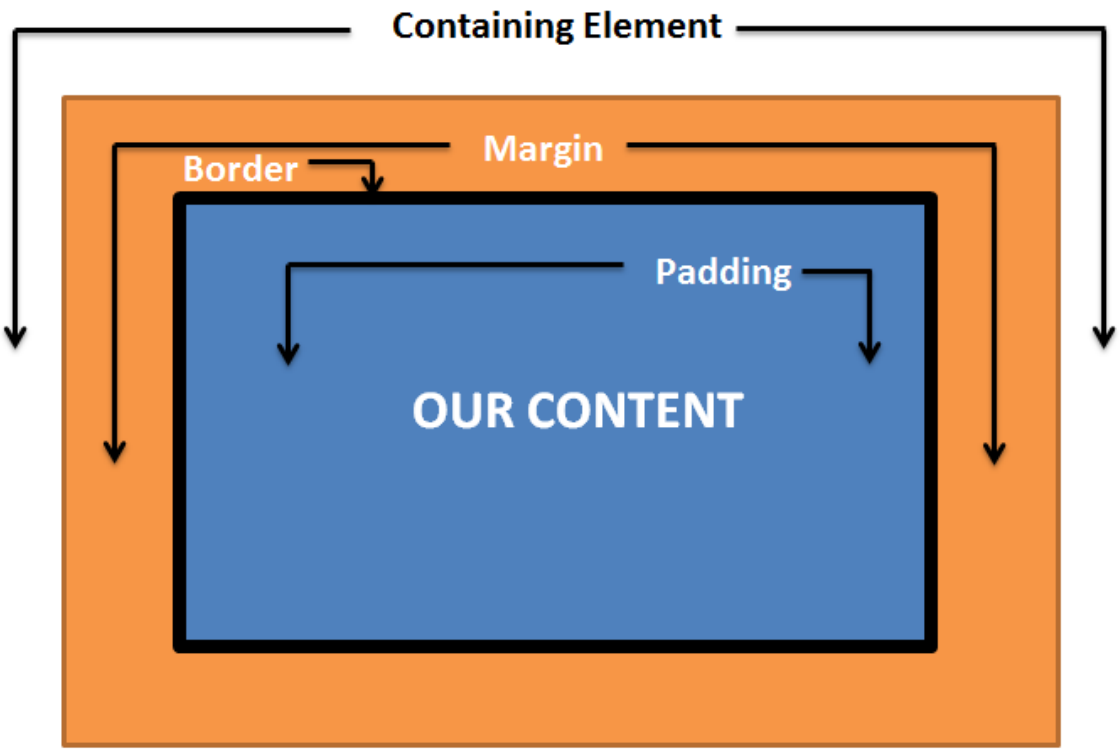
2. Add some styles that will target specific elements, based on their 'state'.

3. Make sure to make use of the rgb and hexadecimal color values.

4. Test your work in your browser and make sure it renders as expected.

The CSS Box Model

One of the fundamental understandings of CSS is the Box Model. The Box model helps us to understand the layout and design of HTML and CSS.



The CSS Box model is made up of content, Padding, Borders and Margins.

So, what are Padding, Margins and Borders?

As you can see, padding is the space that surrounds our content; borders are what surround the padding and margins are what surround the borders.

By definition:

-The padding is the area that separates the content from the border.

-The border is the area that separates the padding from the margin.

-The margin is the area that separates our box from surrounding elements.

How do we define these?

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>The Box Model</title>
        <style>
            section article{
                height: 50px;
                width: 500px;
                border: 1px solid #000000;
                padding: 50px 30px;
                margin: 0 auto;
            }
        </style>
    </head>
    <body>
        <section>
            <article>
                The CSS Box Model
            </article>
        </section>
    </body>
</html>
```

In the above example, we have set the padding for the top and bottom of the element to 50px and the left and right to 30px. The margin has been set to 0px for the top and bottom and the left and right margin is set to auto. When we set a value of auto in our margins, it will basically 'centre' the element within the containing or parent element.

As you may have noticed, we have also set our border. Our border is 1px wide for each side, is solid and has a color of black.

The above code renders the following output:

Note that the rendered output will be 152px in height (top and bottom padding, plus the width of our borders and the specified height of our element) and 562px in width (left and right padding, plus the width of our borders and the specified width of our element).

Exercise fourteen:

1. Create a new HTML file and an external Style sheet.

2. Using your knowledge of the CSS Box Model, create a ‘box’ that has a height of 60px, a width of 260px, a solid border of 3px (you can pick any color you wish), top and bottom padding of 55px, left and right padding of 25px and a top and bottom margin of 0px and a left and right margin set to auto.

3. Save, test your work in your browser and calculate the exact size of your box.

Fonts

When using CSS we can change the font-family of our text. We can specify multiple font-families for any given element. If the user has the first specified font on their system; that is the font that will be used. If the user does not have our first specified font on their system, the browser will attempt to render the next font and so on until one of the fonts are located on the users system. These font-families are separated with a comma and the proceeding fonts are referred to as fallback fonts.

```
header{
    font-family: "Helvetica Neue", Helvetica, sans-serif;
}
article, footer{
    font-family: Arial, sans-serif;
}
```

In the above example we are saying that our header should have a font-family of Helvetica Neue, if that is not located on the users system, we will try Helvetica. If Helvetica is not located on the user's system, we will 'fall-back' to a generic sans-serif font.

You can find out more about sans-serif fonts here: http://en.wikipedia.org/wiki/Sans-serif.

Part 5: Main CSS3.0 Specific Properties

Opacity

In CSS3.0 we can easily specify opacity for an element:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Opacity</title>
        <style>
            section article{
                height: 100px;
                width: 200px;
                opactity: 0.5;
            }
        </style>
    </head>
    <body>
        <section>
            <article>
                We are setting the opactity for the article element.
            </article>
        </section>
    </body>
</html>
```

In the above code we are saying that every article within a section should have opacity of 50%. This will make all of our articles within a section appear transparent. When we set the opacity of an element, we are also setting the opacity of the contents as well and this can have undesired effects.

Alpha Color Space

Instead of using the opacity property in CSS, we can set an Alpha Color Space. We can do this by specifying the color or background-color of an element using the rgb color values.

Now, to specify the Alpha Color Space, we simply add an extra value to our rgb color values; what I mean by that is we change rgb to rgba and have four values for the colors, instead of the usual three.

The Alpha Color Space value will take a value of between 0 and 1. The Alpha Color

Space will specify the opacity of that element.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Alpha Color Space</title>
        <style>
            section article{
                height: 100px;
                width: 200px;
                background-color: rgb(255,0,0);
                box-shadow: 2px 2px 3px #000000;
                border-radius: 8px;
                padding: 8px;
            }

            section article:hover{
                background-color: rgba(255,0,0,0.5);
            }
        </style>
    </head>
    <body>
        <section>
            <article>
                This is our article of content.
            </article>
        </section>
    </body>
</html>
```

In the above example we have set all of our articles within a section to have a background-color of red. When these articles are hovered over, we are setting using the same color, but with an Alpha Color Space value of 0.5. In other words, when we hover over our article elements we will have a background-color that will be slightly transparent.

In the above example you may have noticed the border-radius and box-shadow properties.

Box Shadow and Border Radius

We can specify a box-shadow for most of our HTML elements and this will literally give our 'box' (or element) a box-shadow; giving the element a 3D like appearance. The box-shadow property can take 4 arguments: the h-shadow value, the v-shadow value, the blur-distance and the color of the shadow.

The border-radius property will set a 'border radius' for each of our element's corners; resulting in rounded corners.

This will be the resulting output when we hover over our element.

As you can see we have rounded corners, a shadow and we have an almost pink background-color. The background-color is set to red, but when we hover over it, we are setting the background-color to be 50% transparent.

To learn more about CSS3.0, check out the CSS3.0 Roadmap: http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/

Exercise fifteen:

1. Create a new HTML file and an external Style sheet.

2. Using all of the techniques, concepts and code that you have learned; create your first Website! It doesn't have to be glamorous! The point of this final exercise is to get you started with HTML5 and CSS3.0- in the real world!

3. Good Luck!