UNDERSTANDING AND CODING A BASIC WEBSITE FROM SCRATCH

By Chad Jordan - February 13th 2009

In this guide, you will learn:

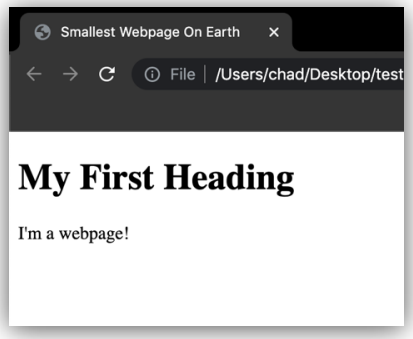
1. The simple differences between coding with markup languages, and programming with programming languages
2. An essential file structure for clean data management practices
3. Creating a basic HTML *(Hypertext Markup Language)* document
4. Understanding how HTML interacts with CSS *(Cascading Style Sheets)* to position and decorate your content within your documents
5. Completing a simple website template complete with header navigation and footer info.

Introduction

When working with any code on the web, there are two parts to understanding how it's implemented and built. These two areas of coding are known as the front-end and the back-end. The front-end consists of **HTML** *(Hypertext Markup Language)*, **CSS** *(Cascading Style Sheets)* **XML** *(Extensible Markup Language),* and *typically* JavaScript. The back-end consists of programming languages such as PHP, Perl, Ruby, Java, *(also)* JavaScript, C#, and MySQL. While there are subset technologies of each of these, this guide is only designed to look at the bare essentials needed to build a simple website. As a result, this guide will only focus on HTML, and CSS. An HTML document is a plaintext document structured with elements. Elements are surrounded by matching opening and closing tags. Each tag begins and ends with angle brackets (<>). There are a few empty or void tags that cannot enclose any text, for instance: <img>.

HTML and CSS are not programming languages, they only play hand in hand for markup purposes. We will not be creating/assigning variables, for loops, or if conditional statements. Markup is strictly used for visual purposes and navigating between web pages. To start things off, I'm using a simple code editor in Linux called Vim, but you can use any blank text document to start. As I progress through this guide, I will provide a side by side 'Code to Web View' result to help the reader visualize what is happening. This example provides the first steps we take in creating our index file using XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
 lang="en">
    <head>
        <title>Smallest Webpage On Earth</title>
    </head>
    <body>
        <h1>My First Heading</h1>
            <p>I'm a webpage!</p>
    </body>
</html>
```



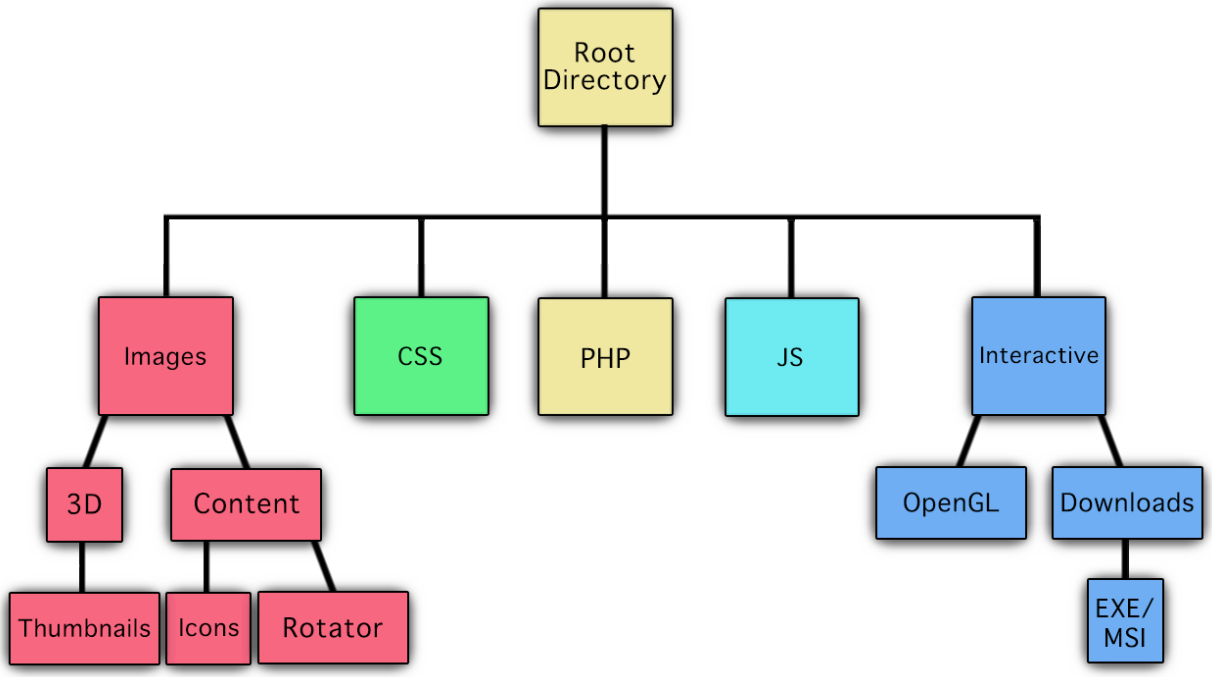
The initial HTML tag is declared as XHTML DTD *(Document Type Definition)* because it describes the allowed syntax and grammar of XHTML markup. While Chrome, Firefox, and Opera will likely run with no issues, this declaration will allow the browser to ignore any potential errors that Internet Explorer may throw us down the road. The document can also be simply declared as a regular HTML tag, but XHTML allows for more versatility when XML data is used. However, when using XHTML this type of notation in my code is mandatory.

With every open tag there is a closed tag indicating that the section of code being implemented has been completed. This is represented as **<html>** and **</html>**. The *closing* tag will always have a forward slash as shown. All content for your website is wrapped within these two tags as you can see from the above example. Our website also includes an opening and closing <head> tag, followed by a set of <title> tags and then the main <body> tags where all main content is placed.

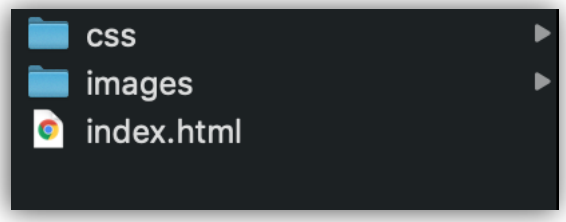
File Structure/Data Management

Once we have the essentials of our document in a good place, it's time to allow the content to be properly read through the system. You will want to save your progress in your blank text document, but not with a .txt file extension. In order for your system and the web to read your file, you must save it as a .html extension so save your file as **index.html**
The screenshot example above on the right displays the result of what displays on your HTML document after you've saved the proper filename and run the file.

The more you build upon your website, the more content will be required for you to keep track of. Maintaining clean and structural data practices is absolutely vital for referencing your content in data form as you progress the implementation of your site. As such, the following hierarchical chart is a visual representation of how I store my content for my website:



The basics of what you should be starting with will look like this on your computer:



HTML & CSS Interaction

No matter what kind of website you are wanting to build, you will learn how important it is to keep everything organized not only for later changes or removal, but also as a point of reference in your code. When building your website, you need to become familiar with other web technologies that allow you to add some structure and style to your pages.

CSS is the technology we want to harness for that stylizing structure. The tag allows us to define a section in an HTML document where we create a container for HTML elements. These elements are defined as ID's and classes. Just like your text document was made into an HTML file, you will do the same thing again for the stylesheet. Rename the text document to style.css and make sure the file is placed in the 'css' directory.

Using the <meta> tag with this string:
`<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>`
will tell the browser to use the utf-8 character encoding when translating machine code into human-readable text and vice versa to be displayed in the browser. Type this into the index.html file underneath the <title> tag inside the <head> section. We need to link in the style.css file so that every element we create in the stylesheet will be applied to the HTML file. Do this by typing the following string underneath the <meta> tag:
`<link rel="stylesheet" type="text/css" href="css/style.css"/>`
At this point, your index file should look like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>Smallest Webpage On Earth</title>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <link rel="stylesheet" type="text/css" href="css/style.css"/>
    </head>
    <body>
        <h1>My First Heading</h1>
            <p>I'm a webpage!</p>
    </body>
</html>
```

Notice we use the 'href' data attribute to pass the string to reference the location and link our CSS file that we created and placed earlier. At this stage of the guide, it's time to start taking more control of our content and customizing the page the way we want. I'm making changes to the index.html file and replacing portions of the syntax to reflect the new divs that I'm making in the CSS file.

First, decide what kind of font and color scheme you're wanting to use on your site. For consistency, I'm going to use a very standard and widely used font. The content that we'll be making changes to occurs within the body of the webpages. Ergo, it's time to write some CSS elements that we can use within the main body across the entire site no matter how many different pages we wish to add. Think about how you want your site to look. Do you want the

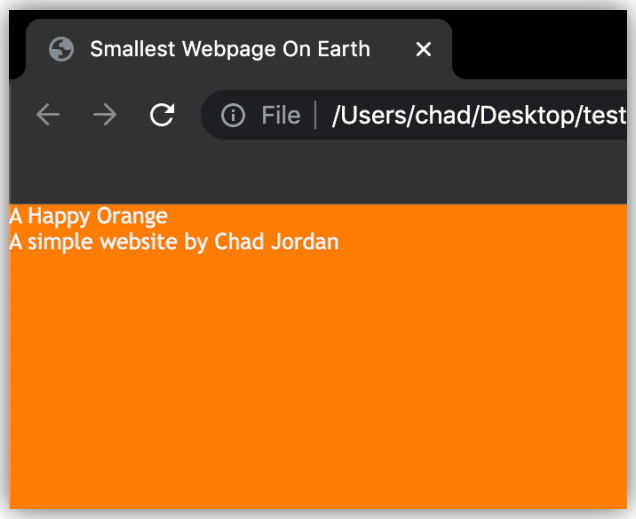
text to float to the left, and have other content off to the right, or perhaps you want the text to float to the right, and have other content off to the left. I typically like to center my content. As such, this piece of CSS code is what I'm starting with in my stylesheet:

```css
html, body {
text-align: center;
}
p {text-align: left;}

body {
        margin: 0;
        padding: 0;
        background: #333333 url(images/img01.gif) repeat-x;
        text-align: justify;
        font-family: "Trebuchet MS", Arial, Helvetica, sans-serif;
        font-size: 13px;
        color: #F1F5F8;
        background-color:#FF8000;
}
```

Everything created in your CSS file is referenced in your main index file and all other webpages that you decide to add to your site. In this file the body attribute will contain all of the following data for my page. Setting margin and padding to 0 removes all default margin and padding for all objects on the page, regardless of browser. Imagine placing a box into another box where the inner box is just slightly smaller so it fits perfectly. Ergo, there is no space between the inner *(content)* box and the outer *(border)* box because there is zero padding. In terms of applying colors to your site, CSS understands all colors are represented as hexadecimal values so it's just a matter of setting the values to the data properties that you're wanting to use. Text-align is a property that specifies the horizontal alignment of text in an element. When you justify text, the property inherits added blocks *(spaces)* between words so that both edges of each line are aligned with both margins.

Once saved and then applied to the index file, we see the following results at runtime:



Creating the Header

As you can see, I went with a vibrant orange for the background of the body and the font has been altered. At this stage of the guide it's time to start accelerating the process of the site by adding div-containers to center our content, implementing a header, customized fonts, and more colors to visually separate sections of the page.

```
#header {
        position:relative;
        width: 680px;
        height:99px;
        margin-left:29px;
        margin-right:21px;
        background: url(header.png) no-repeat;
}

#page {
        margin: 0 auto 0 auto;
        display: table;
        height: 100%;
        position: relative;
        overflow: hidden;
        background: #252F33 url(background.png) repeat-y;
        width: 730px;
}
```

When setting a position to relative, we offset the tag by nudging it in the opposite direction from the specified location so the pixels margin-left and right are moving it into the reverse location for us. The background property in the header element sets the header image at the position based on the data properties. Setting margin to '0 auto 0 auto' where 0 is top-bottom and auto for left and right at X and Y positions. This means the left and right margins will take auto margins according to the width of the element and the width of the container. Setting the display property to a table value makes the element behave like a table. This allows you to make a replica of an HTML table without using the tr *(table row)* and td *(table data)* elements. Setting a 'hidden' value to the overflow property is clipped and the remaining content is hidden. What this means is you can use the overflow property when you want to take more control over your layout. This allows us to specify what happens if content overflow's an elements box. The background property takes the background.png image file seen here:

and repeats it in the Y-direction creating a 'drop-shadow' affect around the page for as long as the developer wants to keep extending more content on their pages. Since the length of content on any given website is never predefined, this method provides an ultra-lightweight and simple approach to loading pages faster, and extending content with less code. Creating your own images and manipulating them in this manner is an effective way to build the pages on your website. The same is said for linking CSS code to an HTML document. If you think about what we've done so far, you can just as easily write all of the CSS within the HTML document. This can be done in 3 ways: **Inline**, **Internal**, and **External**. Inline uses the style attribute inside of HTML elements. Internal is a <style> tag in the <head> portion of the index file, and external is the linking method I've been using in this guide to connect to an outside file.

As I add more CSS elements to my stylesheet, you can see where I now define classes that are applied to the data attributes 'title' and 'subText' globally instead of the ID's that I specified with the 'header' and 'page' attributes. ID's are applied to one specific element on a page whereas classes are applied to multiple elements. We do this with ID's when an element has a particular style applied to it, and a class is more effective with building your overall layout since it applies to more than one element.

```
.title {
        position:relative;
        left:30px;
        top:22px;
        text-align:left;
        font-family:Georgia, "Times New Roman", Times, serif;
        font-size:32px;
        font-weight:bold;
        color:#192B33;
}

.subText {
        position:relative;
        left:35px;
        top:26px;
        text-align:left;
        font-family:Georgia, "Times New Roman", Times, serif;
        font-size:15px;
        font-weight:bold;
        color:#452f22;
}
```

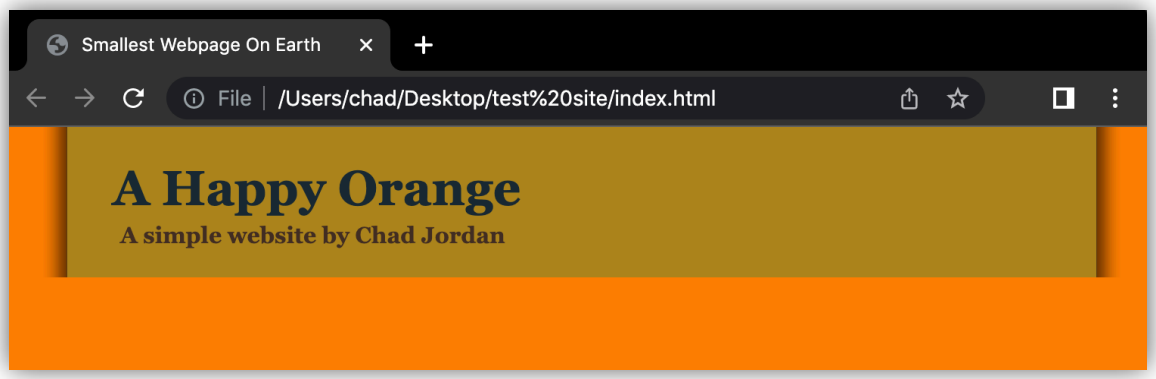
The 'Times New Roman' font family is fairly standard and elegant for this example of my header titles. These data attributes are used as tags throughout your site. With the CSS elements in place, we can add the divs to the html as we want them to appear on the webpage:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>Smallest Webpage On Earth</title>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <link rel="stylesheet" type="text/css" href="css/style.css"/>
    </head>
    <body>
        <div id="page">
    <div id="header">
        <div class="title">A Happy Orange</div>
        <div class="subText">A simple website by Chad Jordan</div>
    </div>
    </body>
</html>
```

As shown here, the attributes are called in the html as divs just as we defined them in the stylesheet. In any code, no matter if it's written for software or for the web, it's read in all the same way starting from left to right, and top to bottom of each file we create. Keeping this in mind, we structure our code as we want it to appear on our website. Once this is written in a clean and readable manner, we save our progress and run the index file again.

Adding a Menu Bar

Our page is now centered as I want it and the shadow is looking good around the page border:



Keeping in mind this is only the header, it's time to expand our page with the next CSS element.

```
.rightLinks .linkTitle {
        font-size:13px;
        font-weight:bold;
        margin-top:18px;
        margin-bottom:32px;
        margin-right:1px;
        color:#708B41;
}


#bar {
        position:relative;
        width: 680px;
        height:57px;
        margin-left:29px;
        margin-right:21px;
        background: url(bar.png) no-repeat;
}

.menuLink {
        height:37px;
        width: 105px;
        text-align:center;
        float:left;
        font-family:Geneva, Arial, Helvetica, sans-serif;
        font-size:14px;
        font-weight:bold;
        color:#EAD5A8;
        padding-top:18px;
}
```

What we've learned so far is that CSS has the ability to create padding around objects of our specifications, positioning of content anywhere we wish to place it, as well as applying colors and fonts to text. Creating a menu bar is the same principle. We simply consider where we want to place the menu bar on our page. A general rule of thumb is that menu bars are near the top of the page due to having a drop-down submenu when hovered over, and this concept applies the same way here. For this guide, I've decided to make my own menu bar in

Photoshop. Afterward, I simply position and place it using code. The above CSS elements are the same conceptual processes that we've been using so far, just slightly altered with the new location and dimensions being the big difference.

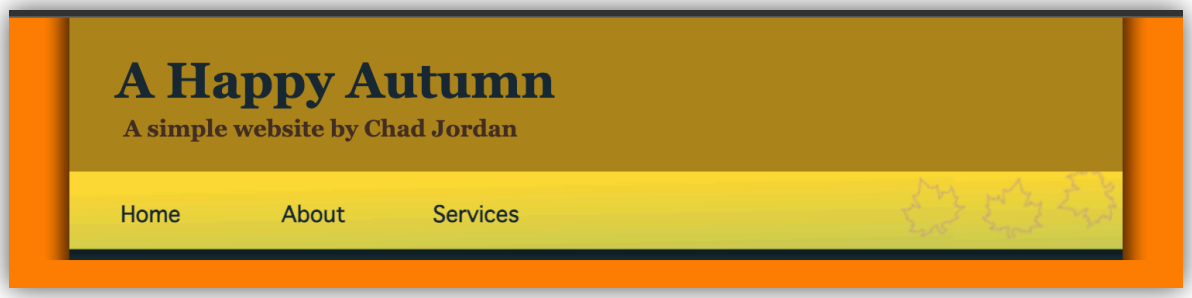
Since I love the fall season, I've decided to evolve the site into more of an autumn theme, keeping with the existing colors of orange and brown. This menu bar that I made in Photoshop will provide a nice yellow to balance out the growing color scheme toward my autumn theme:

Just like previous CSS elements that we've written, we also implement the menu elements over the top of our menu bar using html:

```
<div id="bar">
    <div class="menuLink"><a href="index.html">Home</a></div>
    <div class="menuLink"><a href="about.html">About</a></div>
    <div class="menuLink"><a href="services.html">Services</a></div>
</div>
```

As you reference this screenshot, remember that this guide only represents creating a light-weight website template. Even though I have typed index, about, and services.html as an anchor tag reference, I'm not going to build them. They are simply here as a learning reference to what you would implement on your own website. These elements are 100% valid for real-world situations, but writing out a guide for every webpage is considerably too much.

For now, our focus remains on our website template and how our progress is growing with our newly added menu bar!



Creating the Body

We're coming along well, but now we need to implement a body where our text will go, and placing an additional submenu on the side of our pages. I'm putting my submenu on the right-hand side of the page.

```css
.menuLink {
        height:37px;
        width: 105px;
        text-align:center;
        float:left;
        font-family:Geneva, Arial, Helvetica, sans-serif;
        font-size:14px;
        font-weight:bold;
        color:#EAD5A8;
        padding-top:18px;
}

.menuLink a {
        color:#192B33;
}

.menuLink a:hover {
        color:#FF8000;
}

.menuLink:hover {
        background: url(bar2.png) repeat-x;
}
```

As a visual representation, you can see the bar2.png image just to the right of this code. I've enlarged the image 20 times it's actual size for the sake of reference in this guide. Per our CSS element, this image is repeated in the X-direction so once the browser compiles it on the server, it loads instantly due to being so small. In the case of swifter loading times, and data size we know this to be the most efficient and logical approach to loading content on our site.

Now we implement these elements into our html:

```html
<div id="pageContent">
 <div class="articleTitle">Autumn Is Cool</div>
   <div class="articleContent">
   <div class="rightLinks">
         <div class="linkTitle">Links</div>
           <p class="links">
             <a href="http://www.wondercreationstudios.com">Web Design</a><br />
             <a href="http://www.wondercreationstudios.com">Templates</a><br />
             <a href="http://www.wondercreationstudios.com">Marketing</a><br />
             <a href="http://www.wondercreationstudios.com">SEO</a><br />
             <a href="http://www.wondercreationstudios.com">Programming</a><br />
             <a href="http://www.wondercreationstudios.com">Consulting</a><br />
          </p>
         </div>
   </div>
   </div>
```

Just as we've written earlier, the div id's and classes are assigned to their own div tag and then an anchor tag references the location of the online link or page that you are wanting to navigate to.

This is now the next result of our additional code:

Now we have our submenu on the right side for additional navigating through our site. Any of those elements are also clickable because they are set as hyperlinks. Granted as they are, they will all re-route you to my website, but per the above syntax, you can simply change the location to whatever site/link you are creating on your site. We have a lot of empty space on our page, so let's fill it in with some Lorem Ipsum dummy text.

```
        </div>
      <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Integer mi. Viva
endum sodales. Curabitur elementum. Duis imperdiet. Donec eleifend porttitor sapien
t sed tellus. Suspendisse potenti. Aenean laoreet imperdiet nunc. Donec commodo sus
io. Aliquam lobortis risus ut felis. Sed vehicula pellentesque quam.</p><br />
      <p>Vestibulum augue quam, interdum id, congue semper, convallis non, velit. Q
scelerisque eget, aliquam id, sem. Aenean lorem. Fusce velit nibh, dapibus quis, la
urna. Proin eget elit. Nunc scelerisque venenatis urna. Lorem ipsum dolor sit amet,
s aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himena
 mattis, pede erat fringilla tellus, pulvinar suscipit odio lorem sed pede.</p>
    </div>
    </div>
```

For this example, I'm just creating <p> tags because I'm wanting to fill in our empty area with some text. While paragraph tags have no direct CSS applied to them, we still need to be mindful where we are putting them because inside the current *(articleContent)* they will in fact have the CSS that was written for that particular div applied to them. Remember, as mentioned before, all code is read from left to right, and top to bottom in the order that it was written, so we have to look down the hierarchical flow of our tags to ensure it's being placed inside the 'articleContent' div so that the CSS elements are still properly applied.

Since we have done so, the result is as follows:

The text is properly inside of its container and displaying as we want it to on the page. Now, for the sake of learning by example let's purposefully place the paragraph tags outside of the 'articleContent' div just to see what would happen on our page:

```
            </div>
        </div>
        </div>
    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Integer mi. Vivamus si
dum sodales. Curabitur elementum. Duis imperdiet. Donec eleifend porttitor sapien. Prae
sed tellus. Suspendisse potenti. Aenean laoreet imperdiet nunc. Donec commodo suscipit
. Aliquam lobortis risus ut felis. Sed vehicula pellentesque quam.</p><br />
          <p>Vestibulum augue quam, interdum id, congue semper, convallis non, velit. Q
ac, scelerisque eget, aliquam id, sem. Aenean lorem. Fusce velit nibh, dapibus quis, la
 ac urna. Proin eget elit. Nunc scelerisque venenatis urna. Lorem ipsum dolor sit amet,
Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himena
mmodo mattis, pede erat fringilla tellus, pulvinar suscipit odio lorem sed pede.</p>
    </body>
</html>
```

Notice the <p> *(paragraph)* tags are outside of the 'articleContent' div just before the closing body tag. So what will happen to the text?

Precisely correct! Just as you would expect our text is pushed outside of its container. Therefore we properly ensure the paragraph tags are placed back within the proper div before we add the last touch to our site.

Creating the Footer

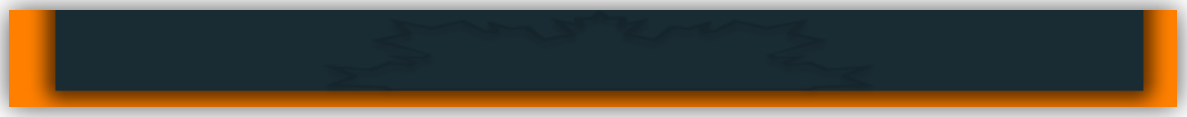
```
#pageContent {
        width: 680px;
        height:auto;
        padding-bottom:20px;
}

#footer {
        width: 730px;
        height:60px;
        background: url(footer.png) no-repeat;
        text-align:center;
        font-size:9px;
        color:#386172;
        padding-top:36px;
}

#footer a {
        font-size:10px;
        color:#386172;
}
```

The closing content of your page is the footer. Every page needs to have a proper header, body, and footer. Since we've come so far with understanding and creating CSS elements, there's not too much to this last process that I haven't already mentioned earlier.

From all previous examples that I've provided, we're sticking with the same general concepts. We're simply creating more CSS elements that are referenced in our html file. Also, just like previous examples and for the sake of learning the quick and easy way to add your footer, here is my footer that I made in Photoshop:



With my last bit of coding for this guide, we're creating the accurate alignment, padding, font size, hexadecimal color values, and image reference to generate our footer before writing it in to our html file. The id footer with an 'a' is nothing more than providing an anchor tag adjustment for mouse interaction. Once we confirm the accuracy of our CSS elements, we add the following syntax to our index file:

```
<div id="footer"><a href="http://www.wondercreationstudios.com"
target="_blank">Chad Jordan</a></div>
```

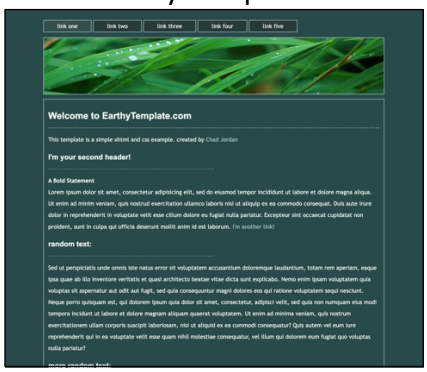
The anchor tag is passed to the footer, and will load the specified website in a new tab when clicked. This is due to the target="_blank" attribute. Your clients as well as prospective clients should be able to see your website. However, I use the target="_blank" attribute because I don't particularly like websites that open external website links in the same window just in case I'm still wanting to navigate the existing site that I'm already on. Per our last example, we will in fact be placing this div outside of all other divs right before the closing body tag because we are connecting the footer directly to the bottom of the page, so it still has to remain inside the body. Doing so will properly connect it to the bottom of your website:

Conclusion

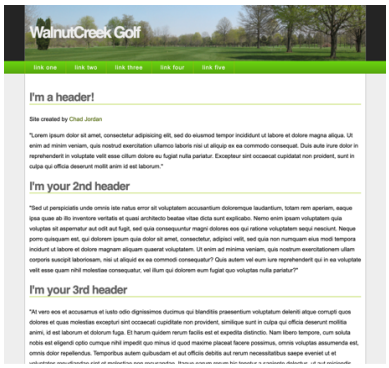
From here, you can expand the website as much as you want! This is what you should consider once the homepage *(index.html)* file is complete. At this stage in development the foundation of your website is well underway and the foundation is one of the most important parts of development. Now, you can create more pages by simply copying and pasting the index file, rename it, and make small, necessary changes to build other pages like 'About Me' 'Services' 'Contact Info' and more. This saves you a lot of time since the foundation of your code is already built. Your index file provides you with all of the code for your other pages so you are that much further ahead.

Overall consistency in your general page layout is a must for websites. Converting your web design into an electronic template is one of the most time-consuming, and conceptually-draining parts of the entire process. This is the case unless you are planning on a lot of additional implementation using Javascript for front-end special effects, animations, or other practical back-end purposes. My hope is that this guide has helped and inspired any who wish to learn the basics of markup languages for basic, front-end web development. If you feel that this guide has inspired you to keep moving forward with markup languages, I would encourage you to start creating your own templates. Doing so will vastly increase your familiarity with these technologies and make you a better coder. The following links are several templates that I've created with most of the same methods from this guide using only HTML and CSS coding.

Earthy Template



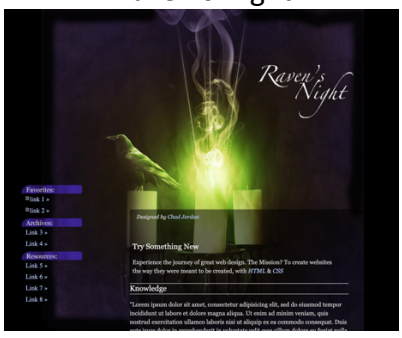
Walnut Creek Golf



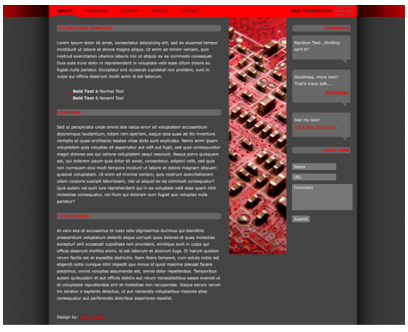
The Beauty of Design

Darkness Falls

Raven's Night



Red Technology



When I was starting my journey in web development even prior to college I used W3Schools to learn nearly all aspects of coding for the web. I highly recommend w3schools.com to anyone

who is wanting to learn and build upon their skills in web development. That site has everything you need for front-end and back-end development. All diagrams and code provided for this guide were created by me. For any possible inquiries such as general questions regarding this guide or other professional inquiries please feel free to email me at cjordan@wondercreationstudios.com

Resources Used:

- W3schools.com