The

DOM

Document Object Model

Implementing the DOM API for HTML Reconstruction

By Chad Jordan – March 24th 2009

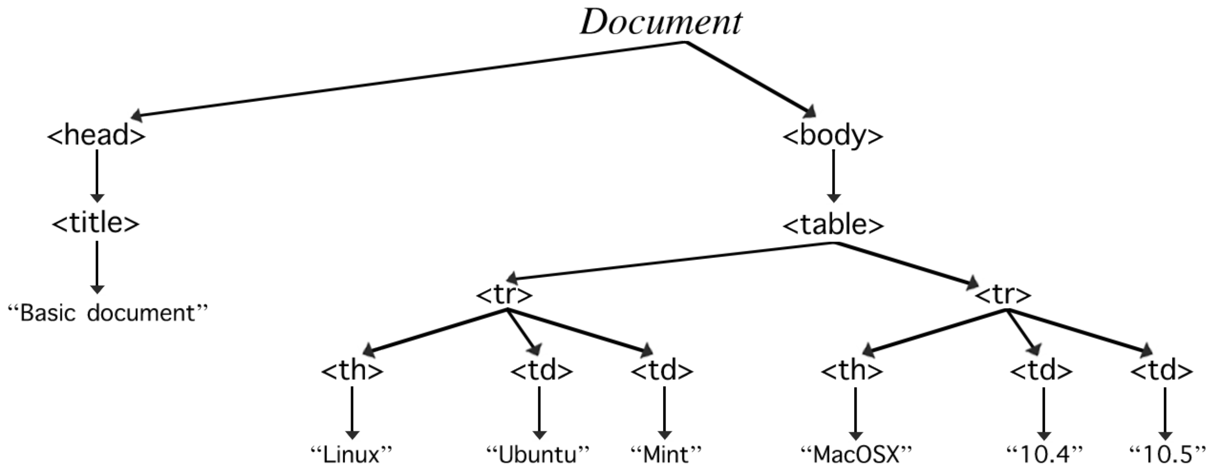
Introduction

In this guide you will learn:

1. A fundamental definition of what the DOM is, and how to interact with it
2. Understanding the difference between HTML and the DOM
3. Implementing JavaScript to instantiate the HTML document object

The DOM *(Document Object Model)* is an API *(Application Programming Interface)* that defines an interface between XHTML documents and application programs. It is an abstract model because it must apply to a variety of application programming languages such as Java and JavaScript. The DOM plays an essential role in making websites interactive. It is an interface that allows a programming language to manipulate the content, structure, and style of a website. Each technology *(language)* that interfaces with the DOM must define a binding to that interface. Documents in the DOM have a treelike structure, but there can be more than one tree in a document. Because the DOM is an abstract interface, it does not dictate that documents must be implemented as trees or collections of trees. Therefore, during implementation, the relationships among the elements of a document can be represented in any number of different ways. For example, the following XHTML document and its corresponding DOM tree illustrate their relationship.

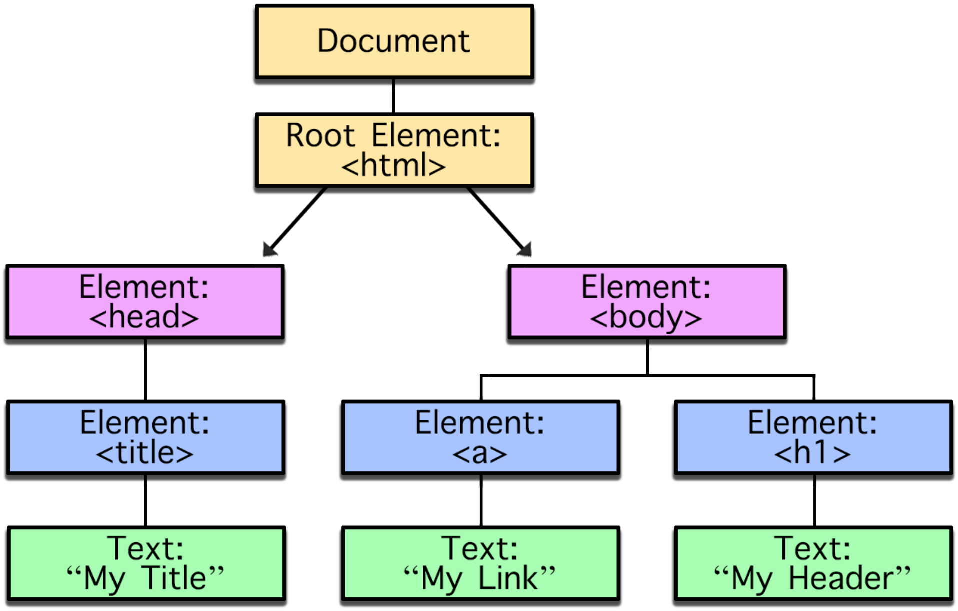
```
<html xmlns = “http://www.w3.org/1999/xhtml”>
   <head> <title> A DOM Demonstration </title> </head>
   <body>
     <table>
        <tr>
          <th> Linux </th>
          <td> Ubuntu </td>
          <td> Mint </td>
        </tr>
        <tr>
          <th> MacOSX </th>
          <td> 10.4 </td>
          <td> 10.5 </td>
        </tr>
     </table>
   </body>
</html>
```



A language that is designed to support the DOM must have a binding to the DOM constructs. This binding amounts to a correspondence between constructs in the language and elements in the DOM. In the JavaScript binding to the DOM, the elements of a document are objects, with both data and operations. This data is referred to as *properties*, and the operations are naturally called *methods*. In JavaScript, a lot of elements are *initialized*, but in the sense of working with objects in the DOM, we are *instantiating* the objects. Even though this API is technically language agnostic, I'll be instantiating it with JavaScript and altering my 2nd assignment for my COS 264 class. Just like my previous guides, my code examples will be displayed using Vim in Linux.

HTML VS The DOM

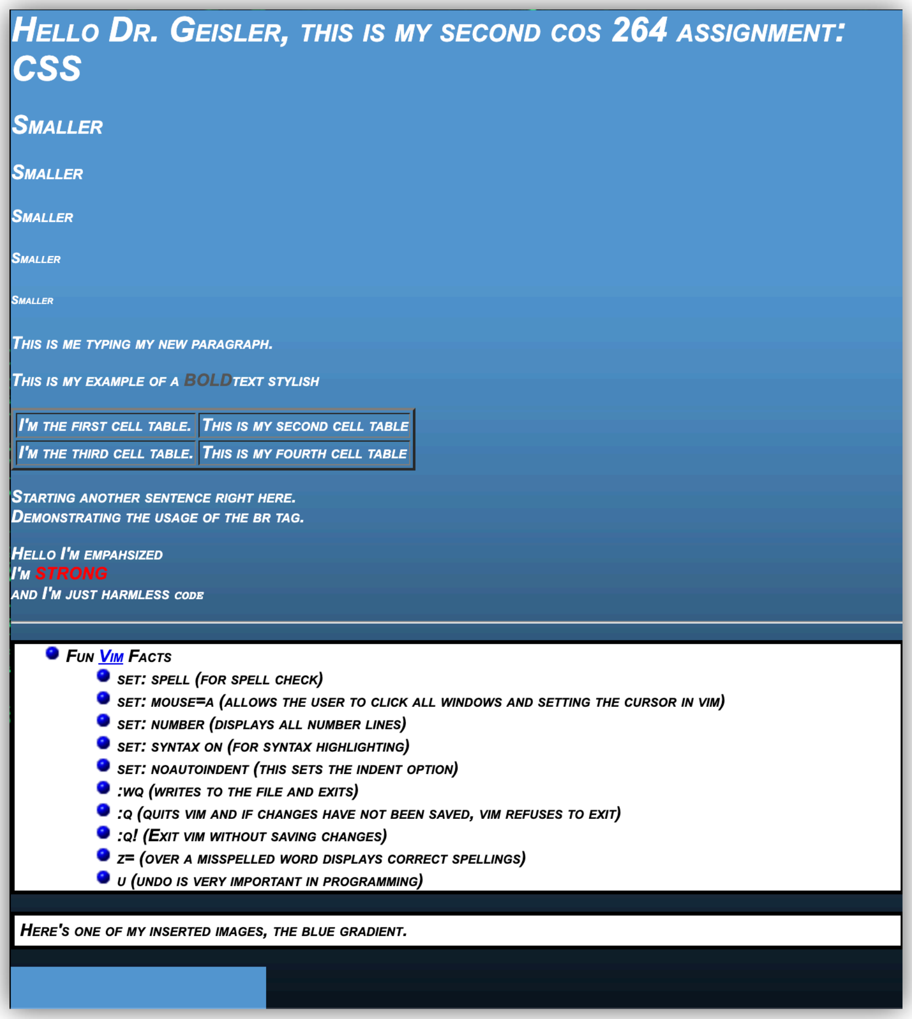
Anyone who is familiar with HTML *(Hypertext Markup Language)* knows that it is a markup language created from a plaintext document structured by elements specifically for building and manipulating text on webpages. As mentioned in my previous *Coding A Basic Website* guide, these elements are surrounded by matching opening and closing tags. Each tag begins and ends with angle brackets (<>). As mentioned earlier, the DOM represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page. A web page can either be displayed in the browser window or as the HTML source. In both cases, it is the same document but the DOM allows it to be manipulated. As an object-oriented representation of the web page, it can be modified with a programming language such as JavaScript. The DOM is built using multiple APIs that work together. The core DOM defines the entities describing any document and the objects within it. This is expanded upon as needed by other APIs that add new features and capabilities to the DOM. For example, the HTML DOM API adds support for representing HTML documents to the core DOM, and the SVG API adds support for representing SVG documents. The following diagram represents the interconnected file structure between HTML and the DOM.



The elements of an XHTML document have corresponding objects that are visible to an embedded script written in JavaScript. The addresses of these objects are required, both by event handling and by the code to make dynamic changes to documents.

Implementing the DOM API

JavaScript is one of the most commonly used programming languages with the DOM. As mentioned earlier, I'm essentially inverting all of the values from my old CSS assignment. This is a fairly straightforward program that can be executed using one large function in JavaScript. If you haven't gathered so far, the whole point of this technology is to dynamically update the content within an existing web page or site for content management. Typically, a program like this would be used for managing websites on a much larger scale, but for this example, I'm only using one web page. For this assignment, I have to invert the case-sensitivity of the text, the bolded font, place incorrect URLs in place of the existing hyperlinks, delete the middle index of the page, swap the footer to the top of the page, and finally append new strings of text all dynamically within a JavaScript file. This will transform my existing CSS page:



Hello Dr. Geisler, this is my second cos 264 assignment: CSS

Smaller

Smaller

Smaller

Smaller

Smaller

This is me typing my new paragraph.

This is my example of a **BOLD**text stylish

I'm the first cell table.	This is my second cell table
I'm the third cell table.	This is my fourth cell table

Starting another sentence right here.
Demonstrating the usage of the br tag.

Hello *I'm empahsized*
I'm **STRONG**
and I'm just harmless code

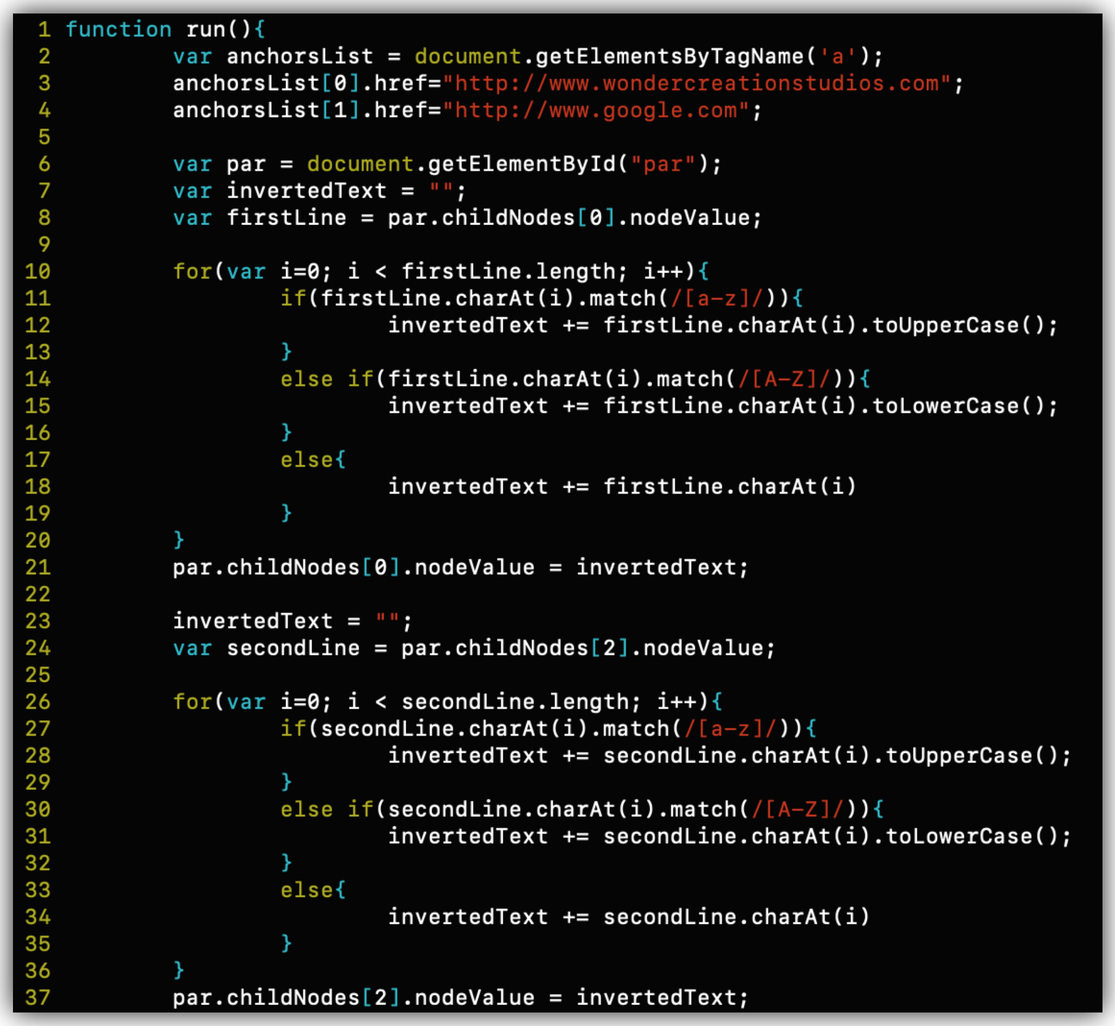
- Fun Vim Facts
  - set: spell (for spell check)
  - set: mouse=a (allows the user to click all windows and setting the cursor in vim)
  - set: number (displays all number lines)
  - set: syntax on (for syntax highlighting)
  - set: noautoindent (this sets the indent option)
  - :wq (writes to the file and exits)
  - :q (quits vim and if changes have not been saved, vim refuses to exit)
  - :q! (Exit vim without saving changes)
  - z= (over a misspelled word displays correct spellings)
  - u (undo is very important in programming)

Here's one of my inserted images, the blue gradient.

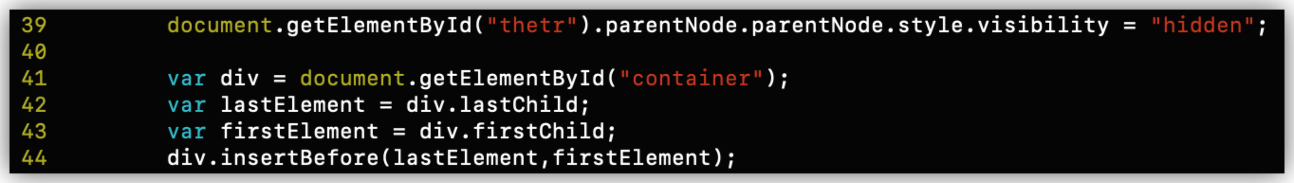
At the beginning of this document I start by getting the URLs and setting them to the *anchorList* attribute for nodes 0, and 1. This means anytime I call those attributes and the nodes connected to them, the specified URLs will be used in place of the original website.

Next, beginning on **line 6** these variables are going to be used to invert the text. Beginning on **line 10** you can see within the for loop, I have a nested if statement that makes a call to the first line characters of every sentence using simple regular expressions. This is the method I put in place for inverting the case-sensitive characters.

This covers the first line, and starting on **line 24** the *secondLine* attribute is given to the second tree node value *(childNode[2]).* Again, the same iterative search is performed using nested conditional statements, a simple regex search, and then we give a character call to the *secondLine* attribute. Starting on **line 39** we need to make the element of the *parentNode* invisible if the child node fails to load. Next, from **lines 41** to **44** I declare my variables and make *lastElement* the first. In this case, it's going to be taking my footer, and swapping it all the way to the top of the page as the first element when the page loads.

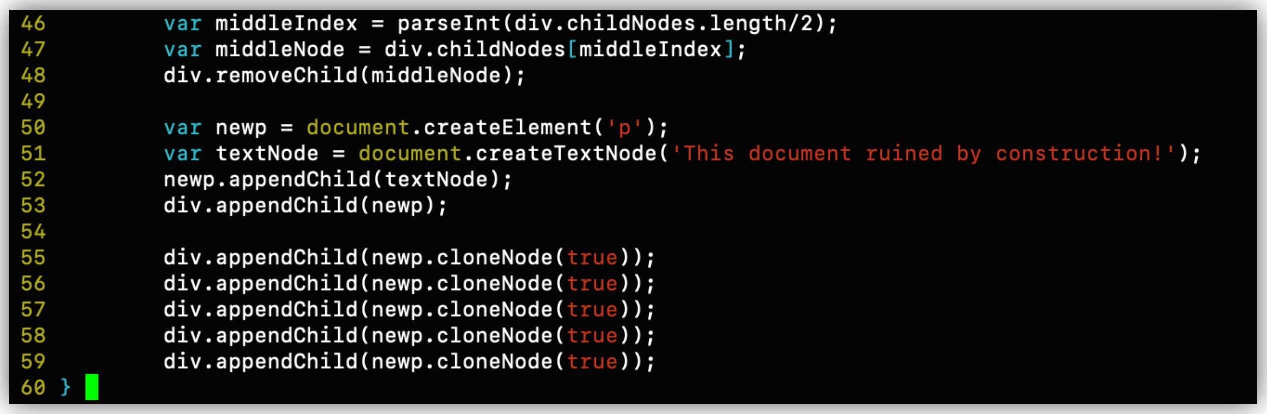


```
 1 function run(){
 2         var anchorsList = document.getElementsByTagName('a');
 3         anchorsList[0].href="http://www.wondercreationstudios.com";
 4         anchorsList[1].href="http://www.google.com";
 5
 6         var par = document.getElementById("par");
 7         var invertedText = "";
 8         var firstLine = par.childNodes[0].nodeValue;
 9
10         for(var i=0; i < firstLine.length; i++){
11                 if(firstLine.charAt(i).match(/[a-z]/)){
12                         invertedText += firstLine.charAt(i).toUpperCase();
13                 }
14                 else if(firstLine.charAt(i).match(/[A-Z]/)){
15                         invertedText += firstLine.charAt(i).toLowerCase();
16                 }
17                 else{
18                         invertedText += firstLine.charAt(i)
19                 }
20         }
21         par.childNodes[0].nodeValue = invertedText;
22
23         invertedText = "";
24         var secondLine = par.childNodes[2].nodeValue;
25
26         for(var i=0; i < secondLine.length; i++){
27                 if(secondLine.charAt(i).match(/[a-z]/)){
28                         invertedText += secondLine.charAt(i).toUpperCase();
29                 }
30                 else if(secondLine.charAt(i).match(/[A-Z]/)){
31                         invertedText += secondLine.charAt(i).toLowerCase();
32                 }
33                 else{
34                         invertedText += secondLine.charAt(i)
35                 }
36         }
37         par.childNodes[2].nodeValue = invertedText;
```



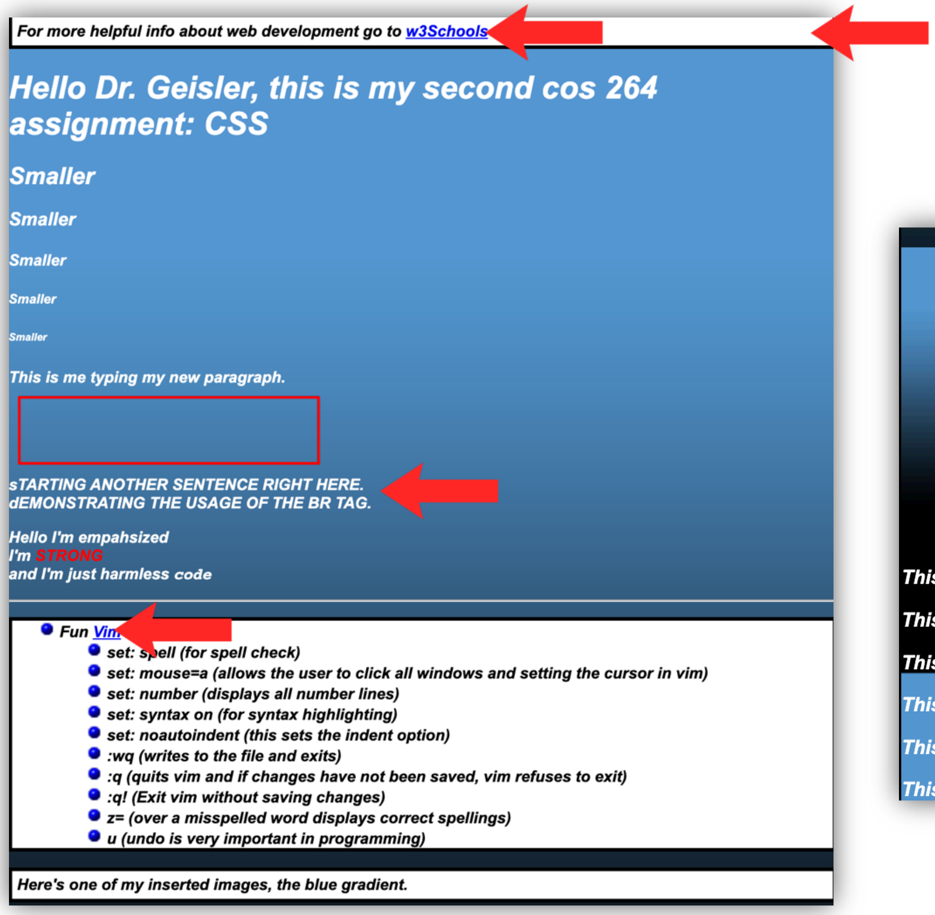
```
39         document.getElementById("thetr").parentNode.parentNode.style.visibility = "hidden";
40
41         var div = document.getElementById("container");
42         var lastElement = div.lastChild;
43         var firstElement = div.firstChild;
44         div.insertBefore(lastElement,firstElement);
```

Starting on **line 46** I set up a block of code to remove the centered content from the page. This means the bolded text and four cell tables section will be completely deleted from the page.



```
46          var middleIndex = parseInt(div.childNodes.length/2);
47          var middleNode = div.childNodes[middleIndex];
48          div.removeChild(middleNode);
49
50          var newp = document.createElement('p');
51          var textNode = document.createTextNode('This document ruined by construction!');
52          newp.appendChild(textNode);
53          div.appendChild(newp);
54
55          div.appendChild(newp.cloneNode(true));
56          div.appendChild(newp.cloneNode(true));
57          div.appendChild(newp.cloneNode(true));
58          div.appendChild(newp.cloneNode(true));
59          div.appendChild(newp.cloneNode(true));
60  }
```

Starting on **line 50** the last section consists of appending the string to the end of the document regex search, and then we give a character call to the *secondLine* attribute. By passing its parameter to the function *createTextNode*. This will print the message, *"This document ruined by construction"* and from **lines 55** to **59** duplicates it five more times.



For more helpful info about web development go to w3Schools

Hello Dr. Geisler, this is my second cos 264 assignment: CSS

Smaller

Smaller

Smaller

Smaller

Smaller

This is me typing my new paragraph.

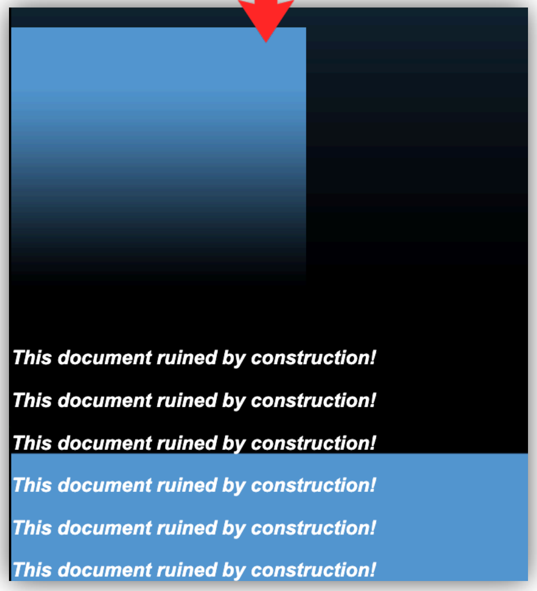
sTARTING ANOTHER SENTENCE RIGHT HERE.
dEMONSTRATING THE USAGE OF THE BR TAG.

Hello I'm empahsized
I'm STRONG
and I'm just harmless code

- Fun Vim
  - set: spell (for spell check)
  - set: mouse=a (allows the user to click all windows and setting the cursor in vim)
  - set: number (displays all number lines)
  - set: syntax on (for syntax highlighting)
  - set: noautoindent (this sets the indent option)
  - :wq (writes to the file and exits)
  - :q (quits vim and if changes have not been saved, vim refuses to exit)
  - :q! (Exit vim without saving changes)
  - z= (over a misspelled word displays correct spellings)
  - u (undo is very important in programming)

Here's one of my inserted images, the blue gradient.

Bottom of the page



This document ruined by construction!

This document ruined by construction!

This document ruined by construction!

This document ruined by construction!

This document ruined by construction!

This document ruined by construction!

Conclusion

This concludes the JavaScript programming side of the DOM, and when I run the original HTML file, the altered results meet all of the requirements of the assignment. As you could see the page was chopped up and modified in such a way to demonstrate how the DOM can allow us to dynamically alter web pages and sites from multiple programming/scripting languages in any way we wish. Aside from being fully editable and dynamic, a few more advantages to using the DOM are:

1. Through DOM manipulation, the possibilities of content management are limitless.
2. You can create applications that update the data of the page without refreshing the page.
3. The data is traversable so the user can move back and forth in the tree.
4. The user can create applications that are customizable by the user and then change the layout of the page without the need of refreshing the page.
5. Language and platform independent

My hope is that this basic DOM API guide is helpful to any who may read it. All diagrams and code in this guide were created, written, and provided by Chad Jordan. 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:

- Sebesta, W. Robert - *Programming the World Wide Web – 4th Edition* – 2008
- W3schools.com