**Unloop Full-Stack Web Dev**

**Lab: Beginning JavaScript on the Web**

In this lab you will be writing some basic JavaScript (JS) to make your webpages interactive.

We will cover the following JavaScript concepts in this lab:

* The <script> tag
* alert() function
* onclick() function
* Manipulating elements

The w3Schools JavaScript reference is an excellent resource with lots of examples! Use it if you get stuck or want more information.

[**http://www.w3schools.com/js/default.html**](http://www.w3schools.com/js/default.html)

# **Where to put the JavaScript code?**

# **<script>** <http://www.w3schools.com/js/js_whereto.html>

The browser treats anything in your HTML file as HTML unless otherwise specified. Therefore, we need to let the browser know that the code it’s about to read is not HTML.

Similar to internal CSS that uses the <style> tag, JS must either be referenced to in another file (this is “external JavaScript”) or enclosed in a <script> tag inside of your HTML file. Script tags can occur in the <head> or <body> section in an HTML file.

When to use internal versus external?

Like CSS, external JavaScript is generally preferred. By keeping our scripts in another file, we keep our HTML focused on the page content and structure and our script file focused on the page interactions.

To reference an external script file you simply specify the filename as the src attribute value in the tag:

<script src=”myExternalFile.js”></script>

The browser will import and execute the scripts on page load once the script tag is read in the HTML file.

# **alert()** <http://www.w3schools.com/js/js_output.html>

## **Follow along**

The first interaction we will be learning is the alert function, which displays a message to the user. Its syntax looks like this:

alert(“message to the user”);

The alert is a ***function***, which means that we can ***call*** on it to perform a specific task. In this case show an alert to the user. The message that we ***pass*** into the function between the parentheses is called the ***argument***. We can pass a number of “things” to the alert function for it to display; plain text wrapped in either double or single quotes is a frequent use case.

Because the browser treats anything in your HTML file as HTML unless otherwise specified, JS must be either referenced or enclosed in a <script> tag:

<script>

alert(“message to the user”);

</script>

Place the above example code into your HTML file’s body like this (JavaScript is commonly found at the end of the body section:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<h1>This page displays an alert!</h1>

**<script>**

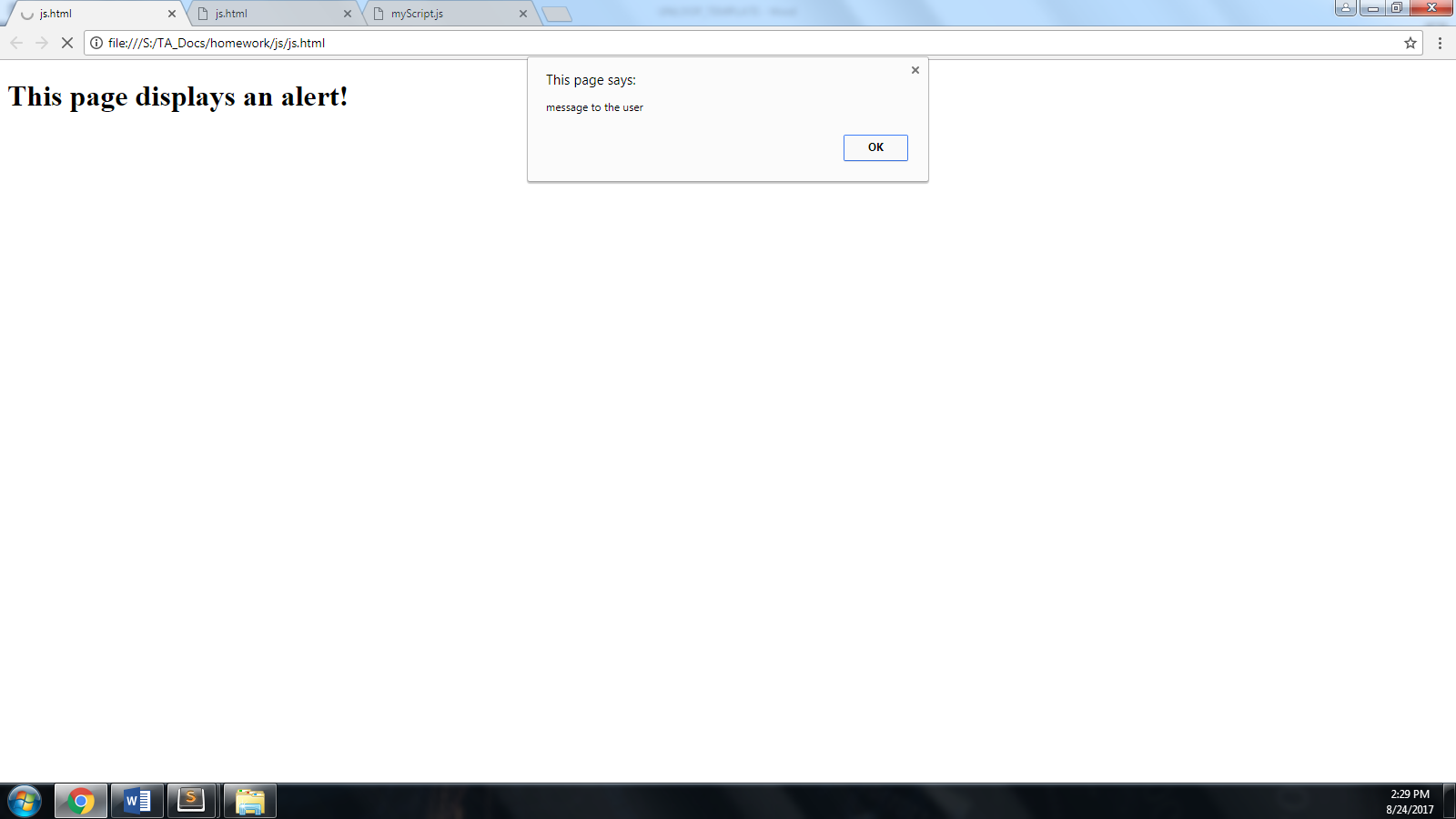
**alert(“message to the user”);**

**</script>**

</body>

</html>

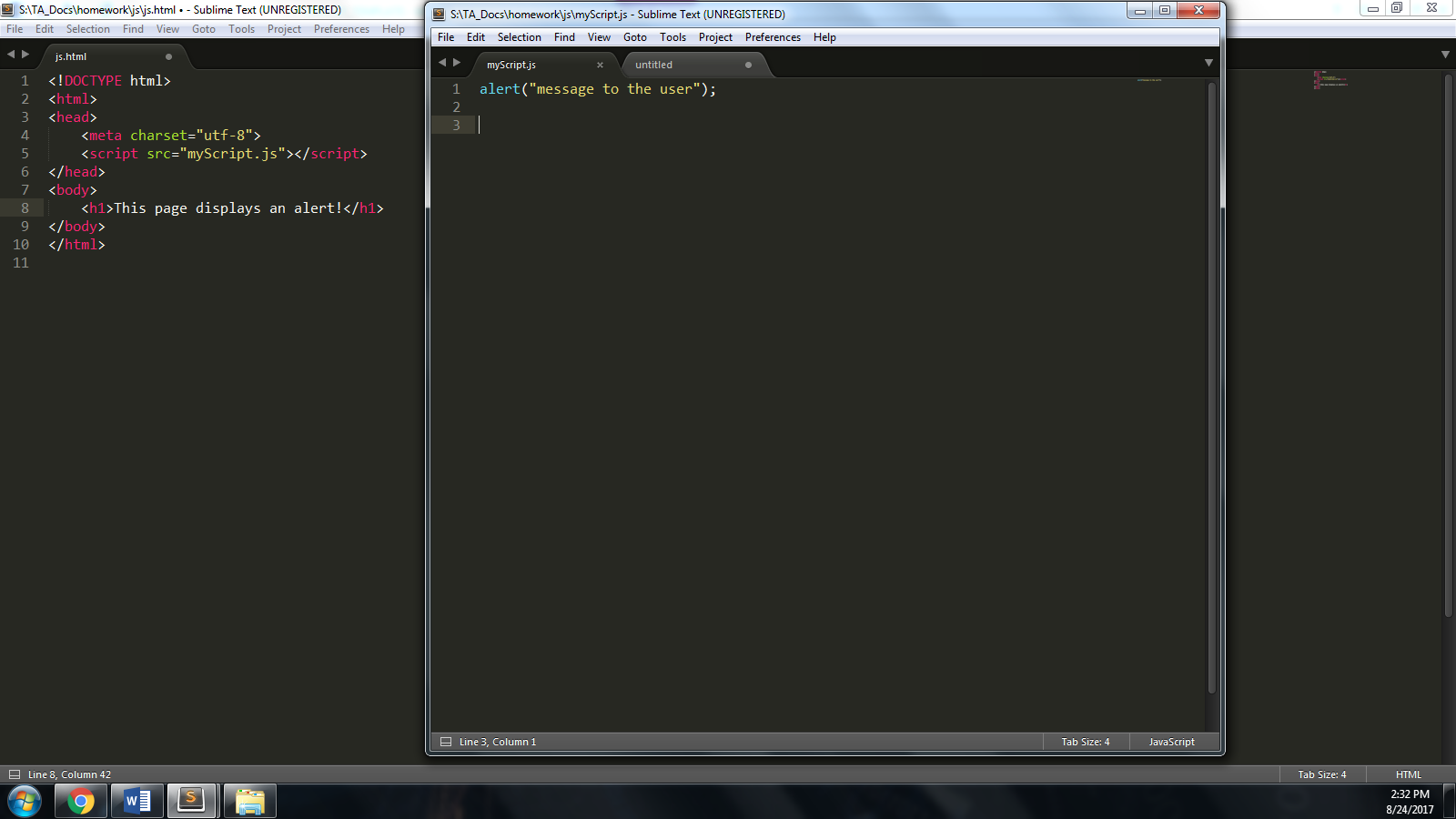
Your page should look/act like this:



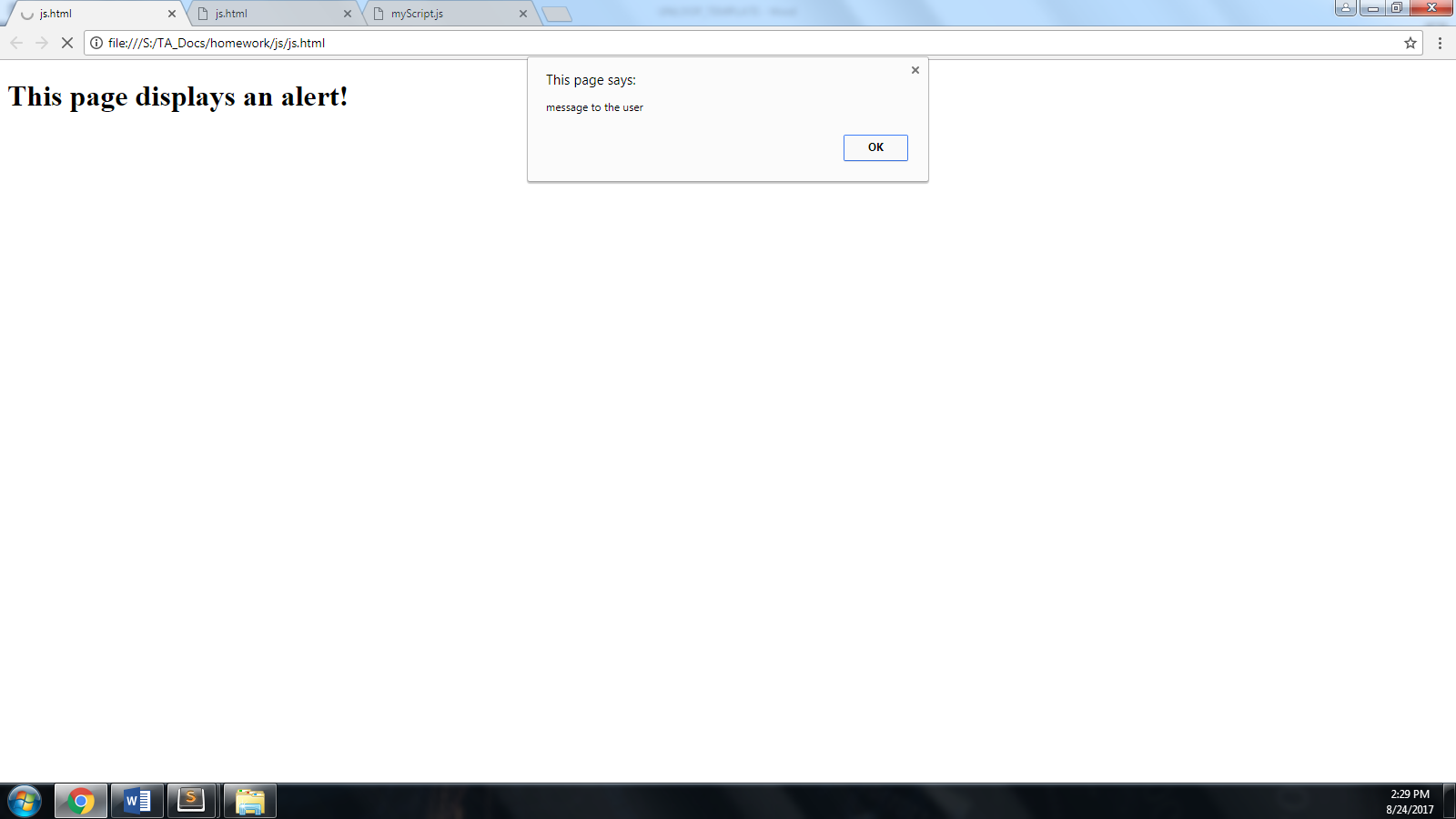
## Using alert in an external file:

The best practice (just like with CSS ☺) is to write JS in an external file and reference the file in the HTML file.

Change your file to include the following contents. Your JavaScript function should go in a separate external file named “myScript.js” Reference your script in your <head> (see line 5):



Because the external JS file executes when the page loads, you can expect an alert as soon as the page finishes rendering!



## **Challenge**

Use **both** internal and external JS to create two alerts in one HTML page.

* Create an internal JavaScript function to alert the user of your name.
* Create an external JavaScript function to display your favorite hobby.

*Did both of these alerts pop up at the same time or did one appear after the other???*

# **onclick** <http://www.w3schools.com/js/js_events.html>

## **Follow along**

Every HTML element has a corresponding **onclick** event that can be tied to some JavaScript code. This includes things like <p> and <img> tags. **onclick is an HTML *attribute* that creates an “HTML event”.**

The onlick syntax looks like this:

<[element] onclick=”JS call goes here”></…

A <p>’s onclick could look like this: (like inline CSS, this is called **inline** JavaScript)

<p onclick=”alert(‘I got clicked’);”> Click Me. Now.</p>

(We use single quotes to surround the message since double quotes are being used on the onclick attribute. Feel free to try using double quotes for both.)

Here is the code in a simple example page:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

</head>

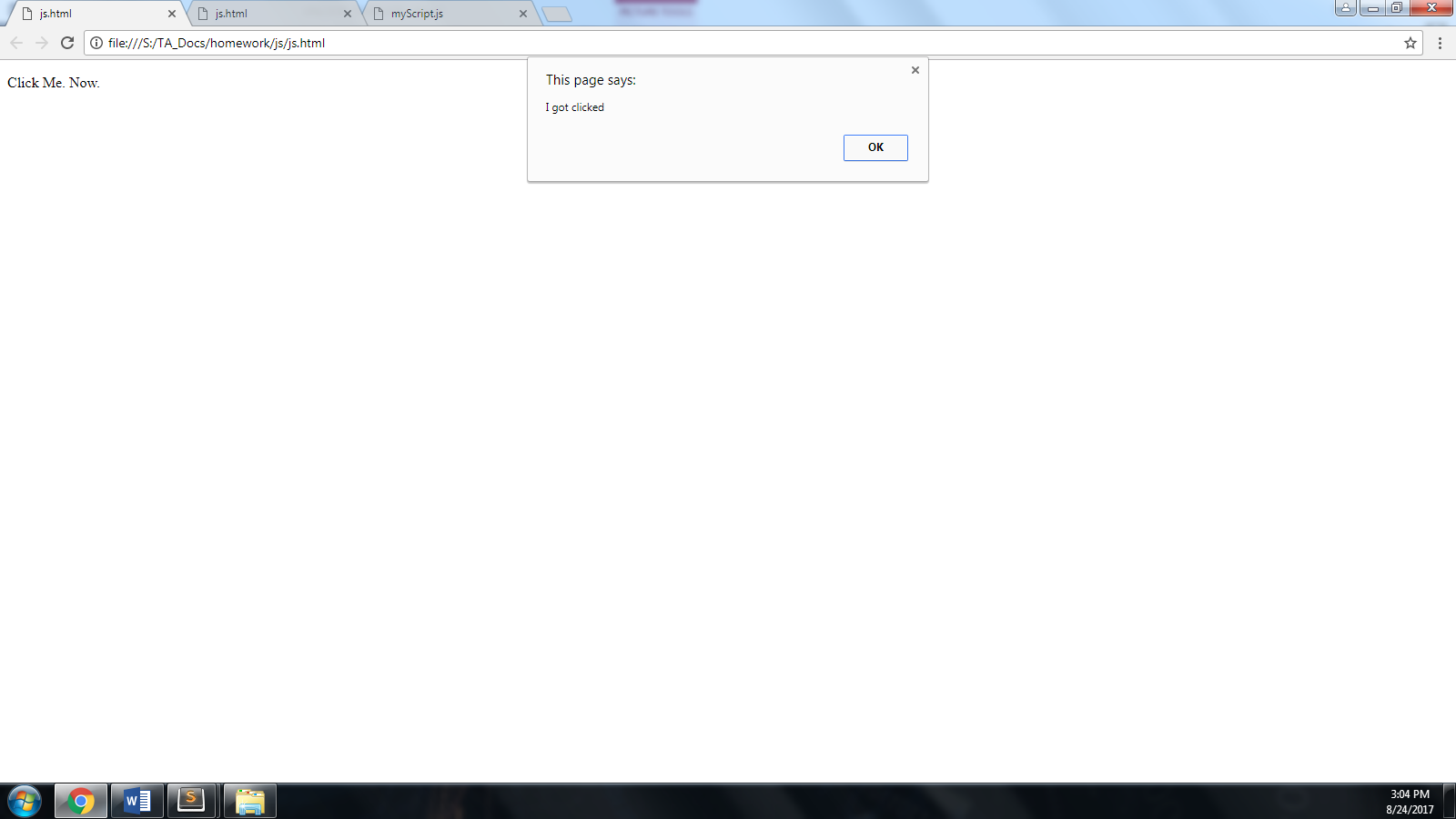
<body>

<p onclick="alert('I got clicked');">Click Me. Now.</p>

</body>

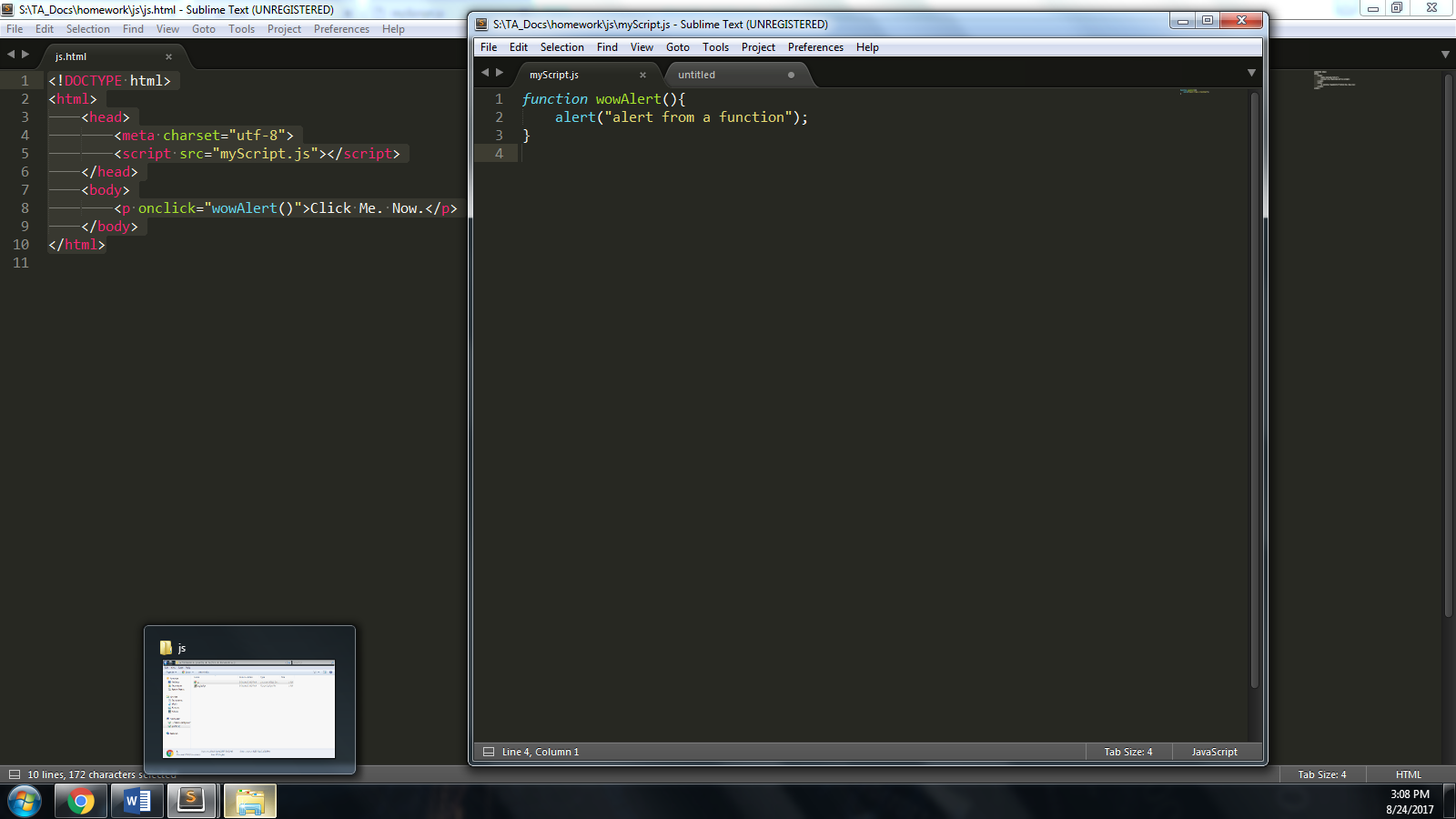
</html>

Clicking the paragraph in the browser displays the alert:

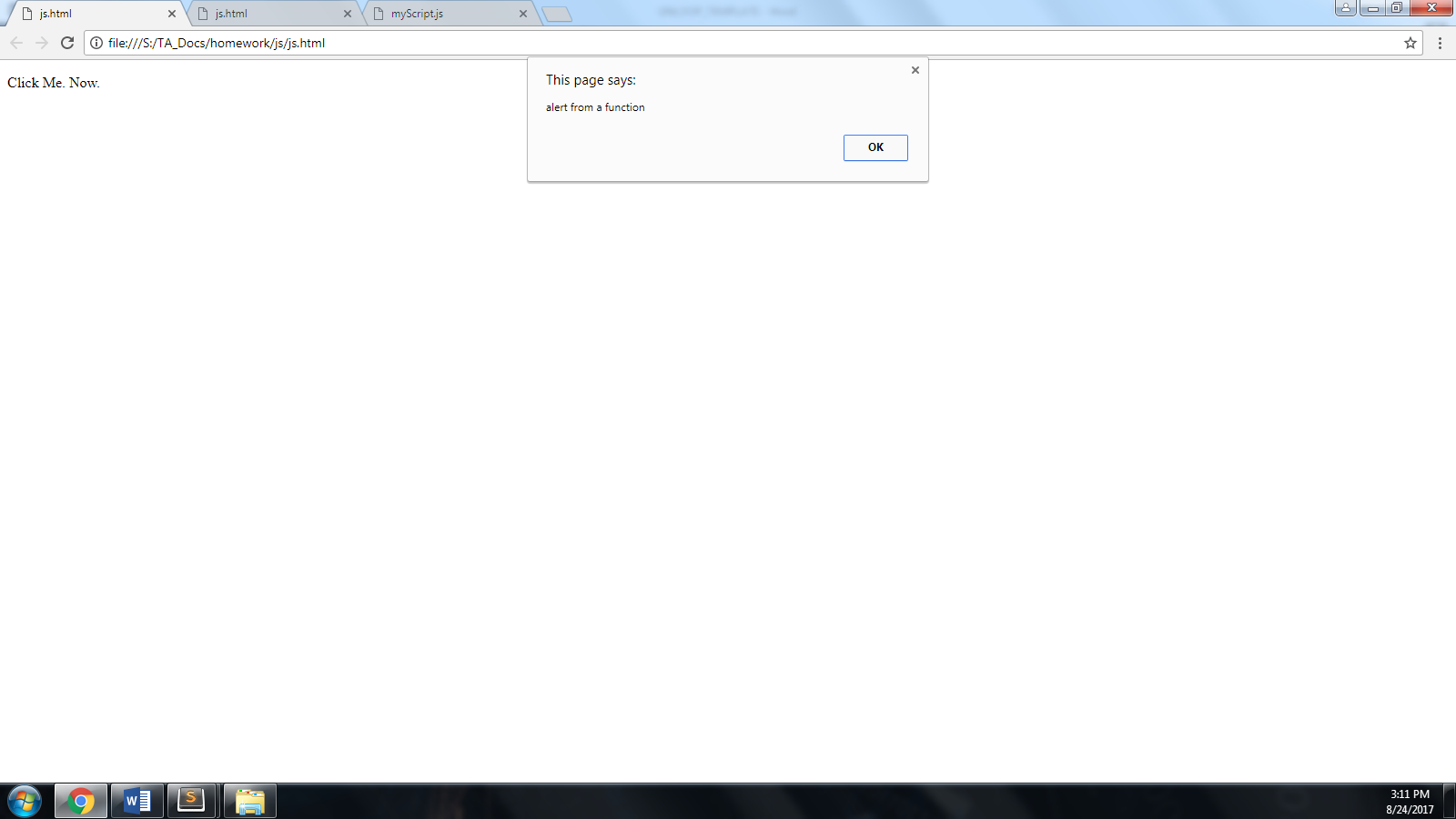


Again, this is to show you that you can write raw JavaScript within an attribute value. **However, a reference to a function in an external file is preferred (keeps your code more readable & modular.)**

To accomplish this, wrap your alert in an external JavaScript function, arbitrarily named “wowAlert”, and then call the function through the onclick event in your HTML file. Remember, our external file must be referenced through a script tag (see line 5).



And…It works!



## **Challenge**

Write your own **onclick** functionality for:

* one element using an **inline** alert.
* one element referencing a **function** in an external file (like the example above).

# **Manipulating Elements with JavaScript** <http://www.w3schools.com/js/js_statements.html>

JavaScript is also very useful for manipulating elements *programmatically,* i.e., using your JavaScript code to make changes to your website’s HTML or CSS.

In this section we will be manipulating both the text of an element (its **innerHTML**) and its CSS properties. This is achieved using the page’s document object model (DOM).

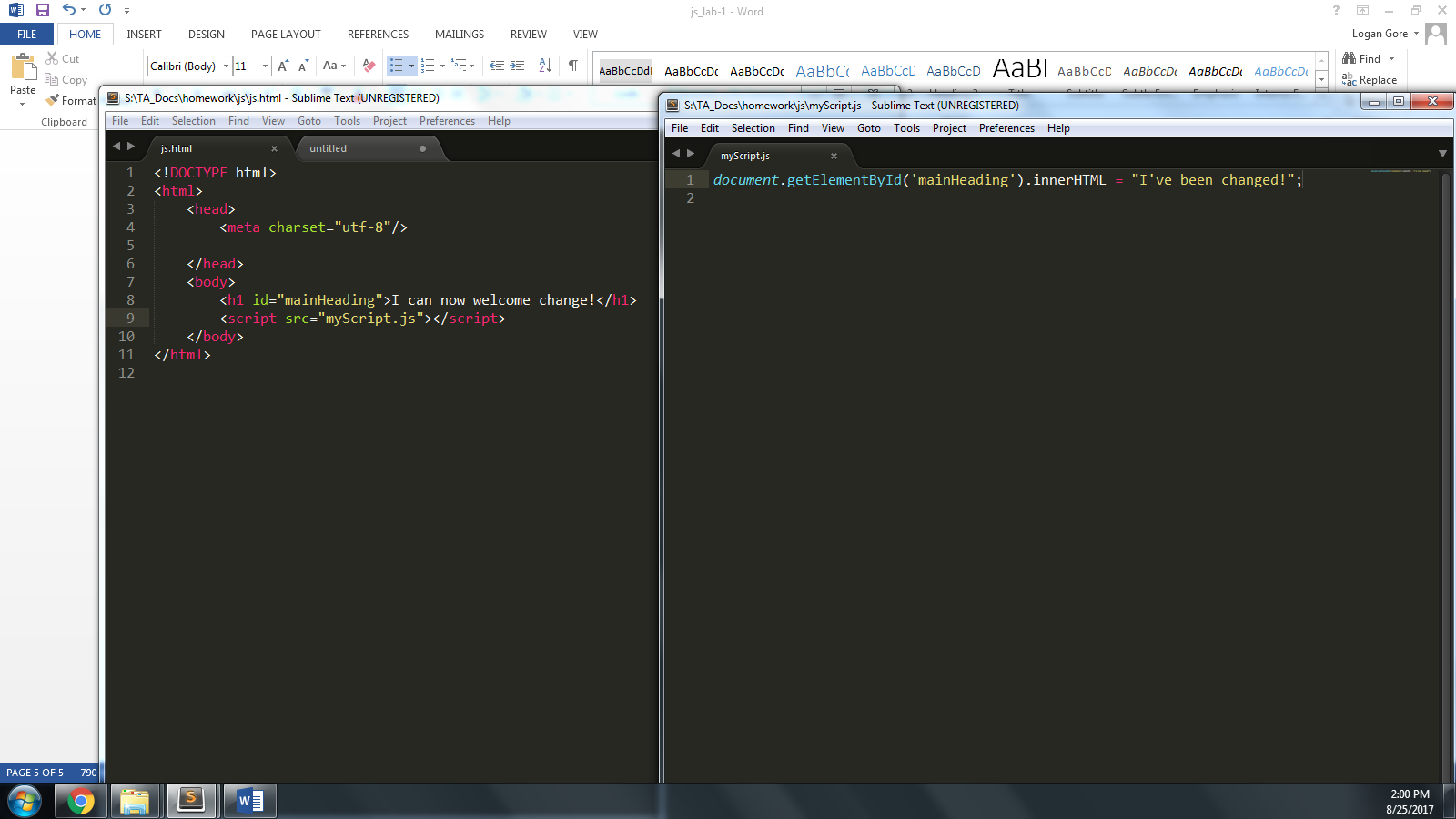
## **Follow Along**

We will only use external JS files in this example, unlike the previous examples where we explored the less-than-preferred ways of writing JavaScript (inline or internal).

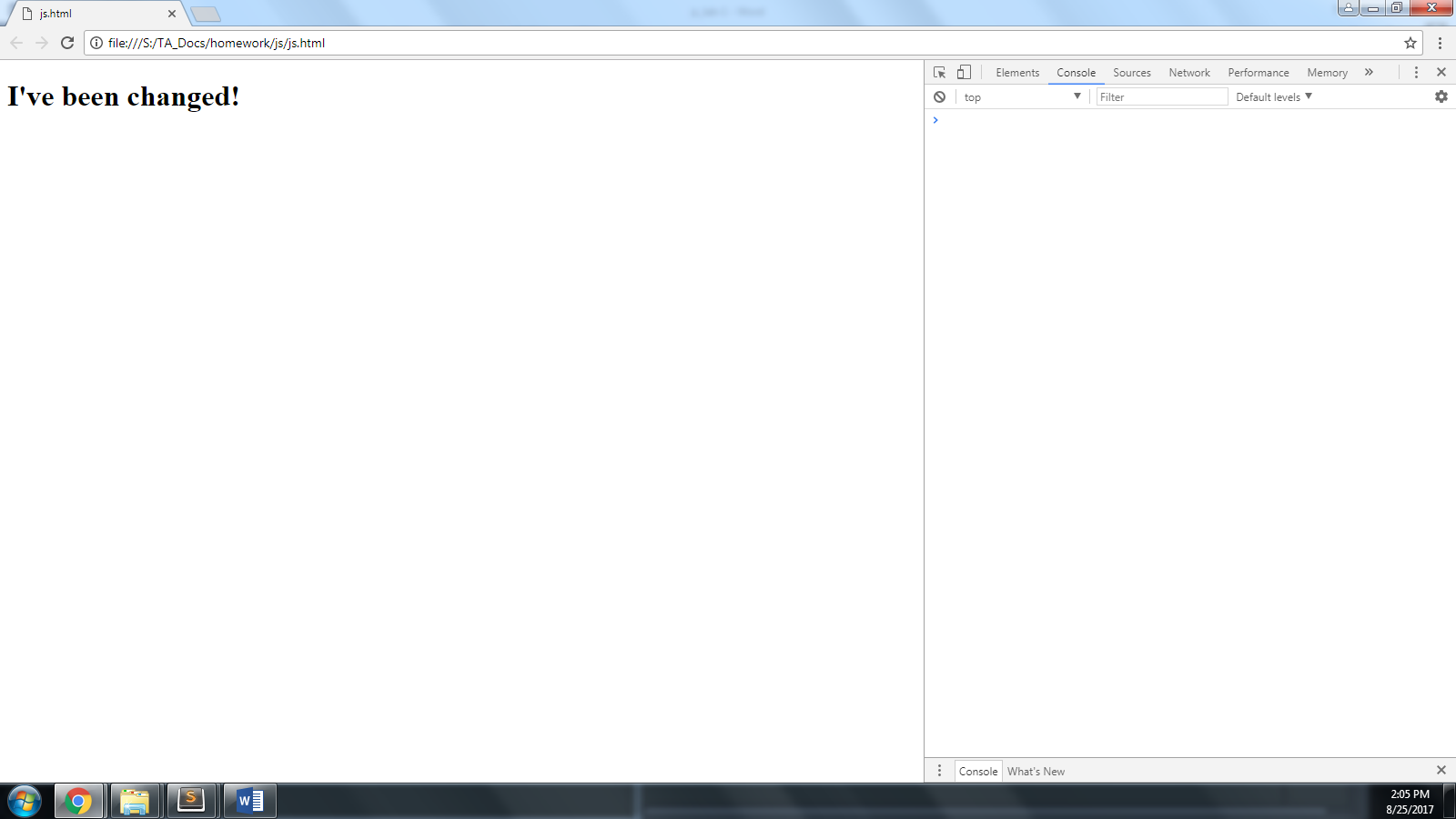
In an external JS file we’ll:

* Use the **document**.**getElementById** function to retrieve an element
* Then access the **innerHTML** property of the element and set its text to text of our choice

Then we’ll link to this external file using a <script> tag placed at the bottom of our body.



Our browser loads the JS file and changes the text in the browser:

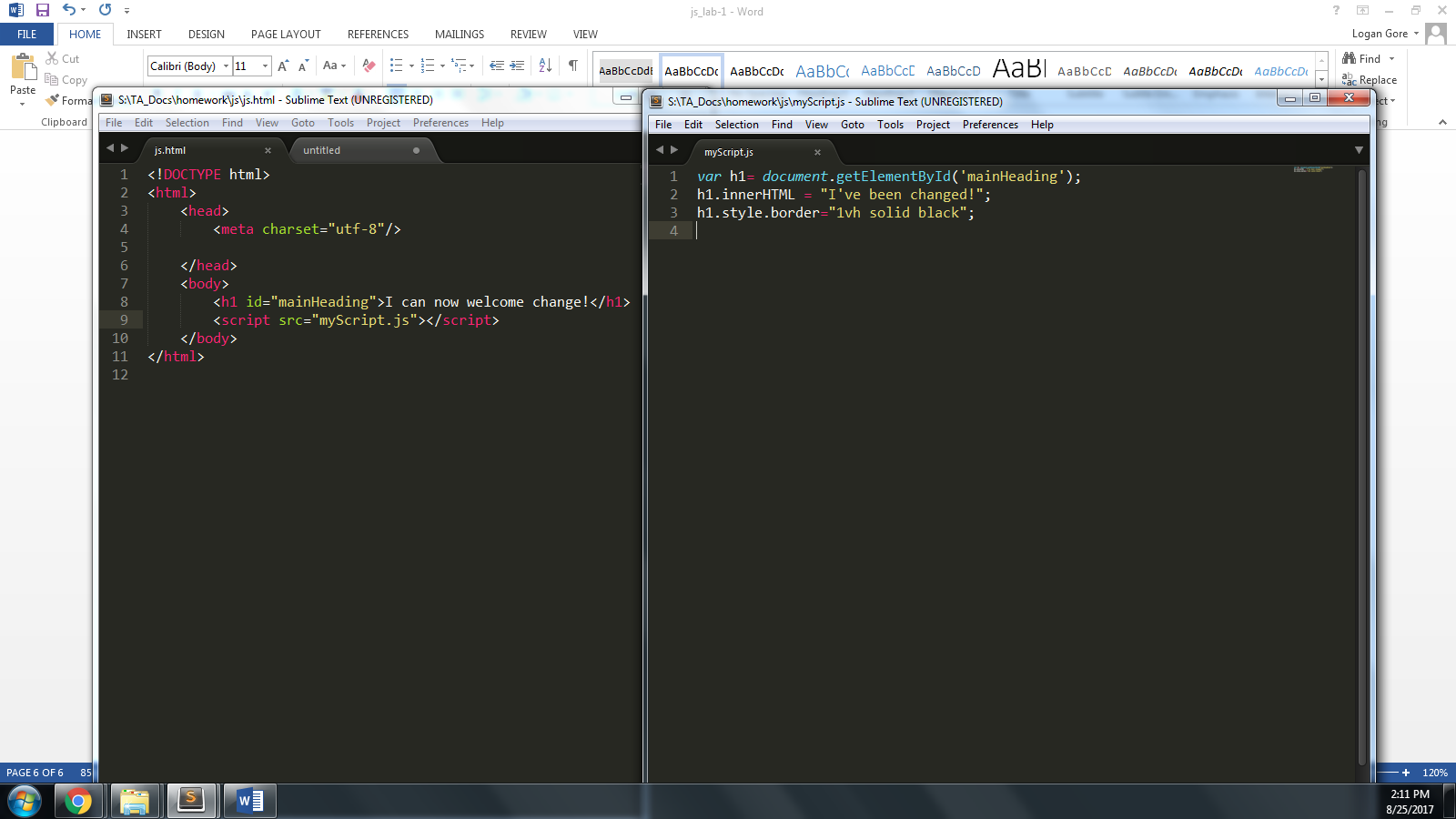


Now, we’ll do some CSS manipulation to the same page!

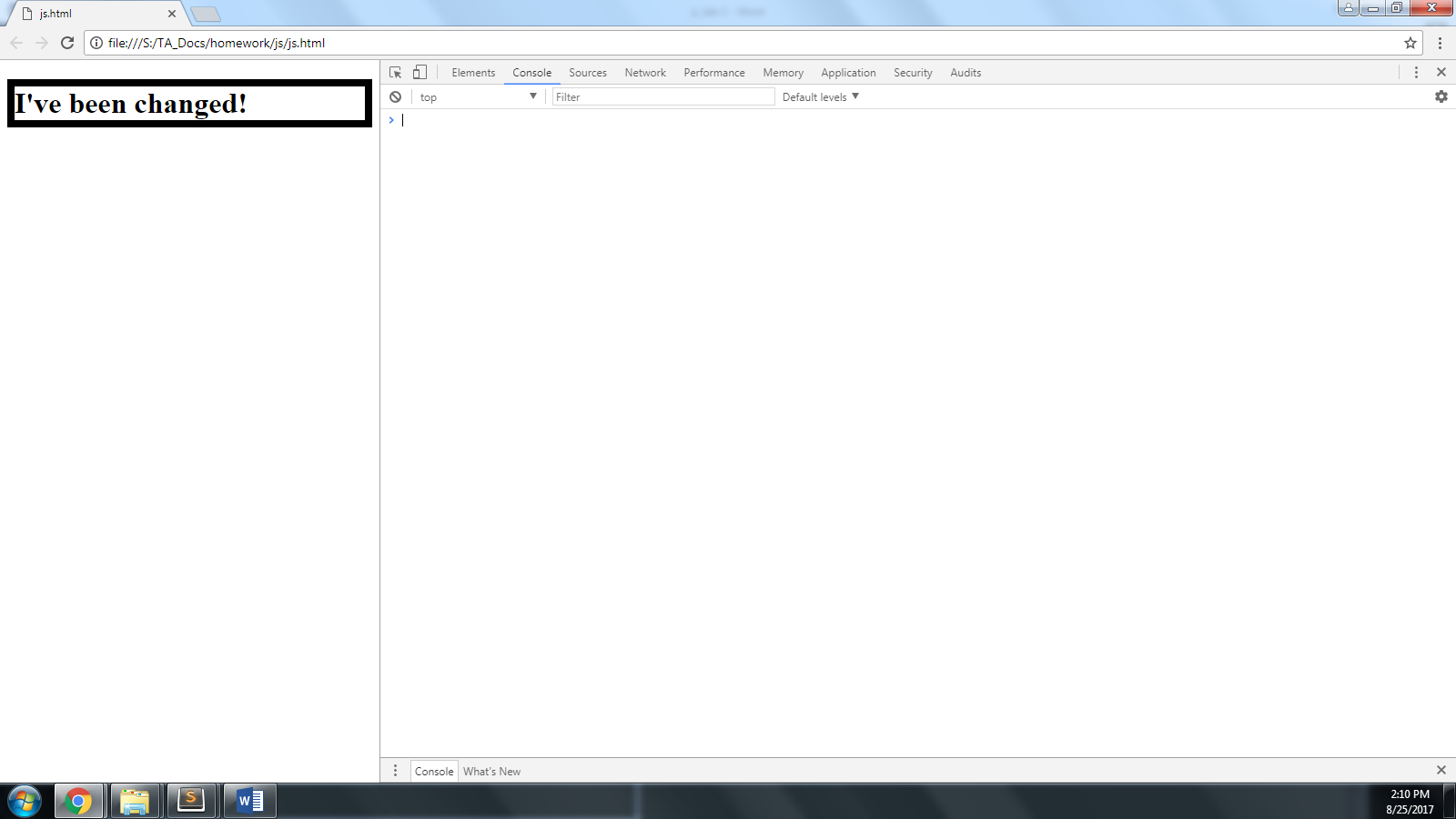
We’re going to change up the code a little also.

First, create a variable to put the “mainHeading” element in so we can reference its properties more than once with simpler syntax.

Then we’ll set the text again, **and add a CSS border style**:



Save the file and update your browser. It should look like this:



## **Challenge**

Create more elements and:

* Create external JavaScript functions to modify their text or styling.

# *Fin.*