**Unloop Full-Stack Web Dev**

**Functions Primer Part 1**

## Prior to completing this Practice Exercise:

1. Watch the Udacity videos on [JavaScript Functions](file:///R:\Full%20Stack%20Web%20Development%20-%20Quarter%201\Videos\Udacity%20Videos\JavaScript%20Videos\Functions%20Videos).
2. Read the documentation on and complete the exercises on the [w3schools Functions page](http://www.w3schools.com/js/js_functions.html).

Before you start, retrieve the **functions\_lab\_1.html** companion file for this primer.

## Start Here!

Functions are a fundamental aspect of programming, perhaps even more so to JavaScript than other languages. Functions bring new adjectives to our programs: *modular*, *portable*, and *maintainable*. These translate to powerful programs and productive development.

In this handout you will learn how to create and use functions.

## What is a function?

A function is a way to ***encapsulate*** a piece of code so it can be reused.

Familiar examples of functions include **alert()** and **console.log().** The code for **alert()** is written once, but we are able to use it whenever we want simply by *calling* or *invoking* it. Notice, also, that we don’t concern ourselves with the code *inside* of the alert() function; it’s encapsulated through the use of the function and we only become concerned with what the function does for us rather than the specific code inside.

## Understanding the problem—the need for functions

Step 1: Open **functions\_lab\_1.html**

Let’s look at the <body> code:

<body>

<p id="p1" onclick='document.getElementById("p1").style.color="green";'>Click 1</p>

<p id="p2" onclick='document.getElementById("p2").style.color="green";'>Click 1</p>

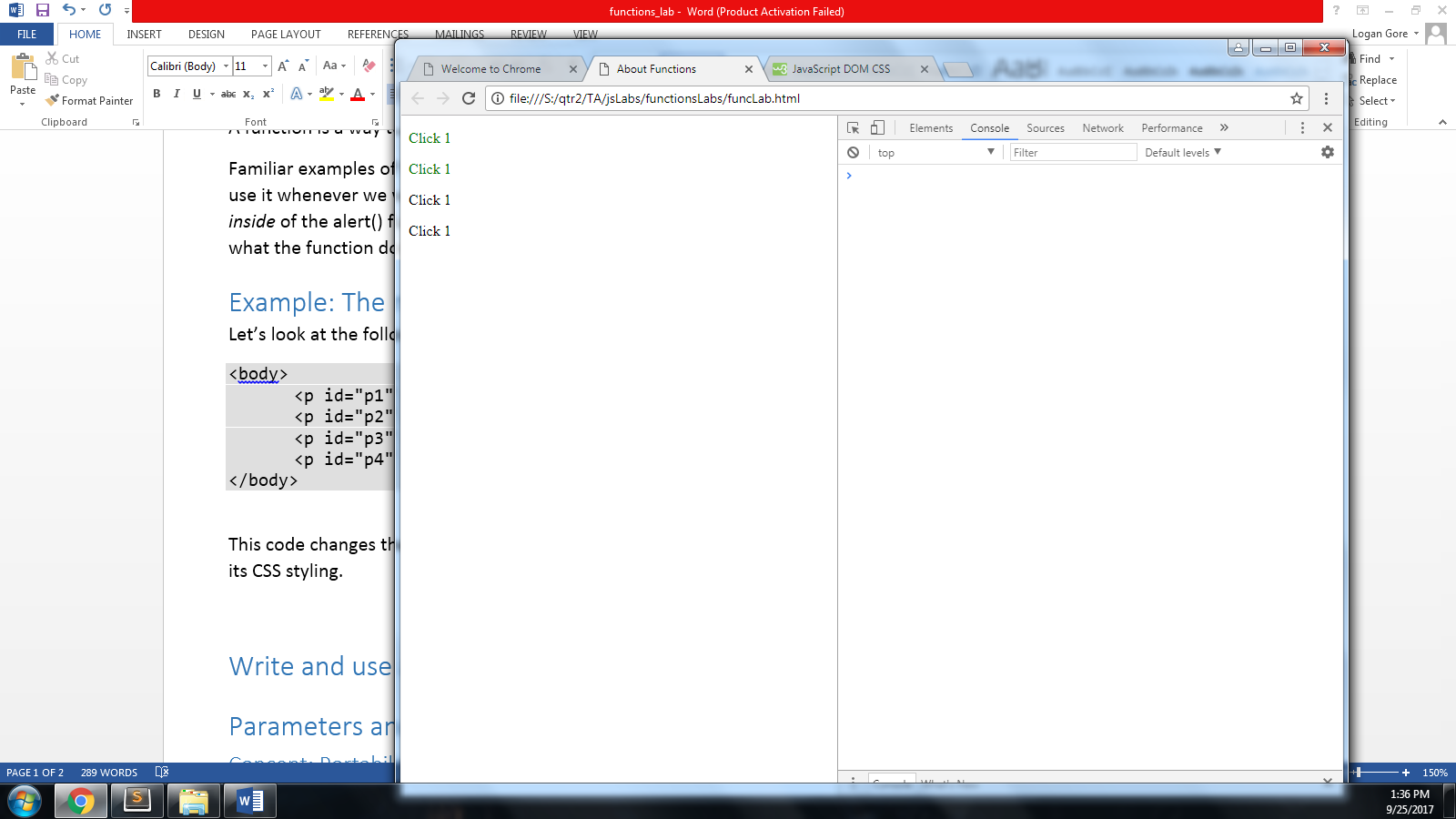
<p id="p3" onclick='document.getElementById("p3").style.color="green";'>Click 1</p>

<p id="p4" onclick='document.getElementById("p4").style.color="green";'>Click 1</p>

</body>

This code changes the color of a <p> when it’s clicked. Each <p> has an attached click event that uses the DOM to modify its CSS styling through the *document.getElementById()* function.

After clicking a couple of our <p>’s, we can see it works:



But, we’ve now decided that the example is super boring, we want to *spice* it up a little. We want to really push the limits…*we want to modify the text.* So, we’ll modify our click functionality:

Step 2: Change your body code to match the following:

<body>

<p id="p1" onclick='document.getElementById("p1").innerHTML="I was clicked";

document.getElementById("p1").style.color="green";'>

Click 1

</p>

<p id="p2" onclick='document.getElementById("p2").innerHTML="I was clicked";

document.getElementById("p2").style.color="green";'>

Click 1

</p>

<p id="p3" onclick='document.getElementById("p3").innerHTML="I was clicked";

document.getElementById("p3").style.color="green";'>

Click 1

</p>

<p id="p4" onclick='document.getElementById("p4").innerHTML="I was clicked";

document.getElementById("p4").style.color="green";'>

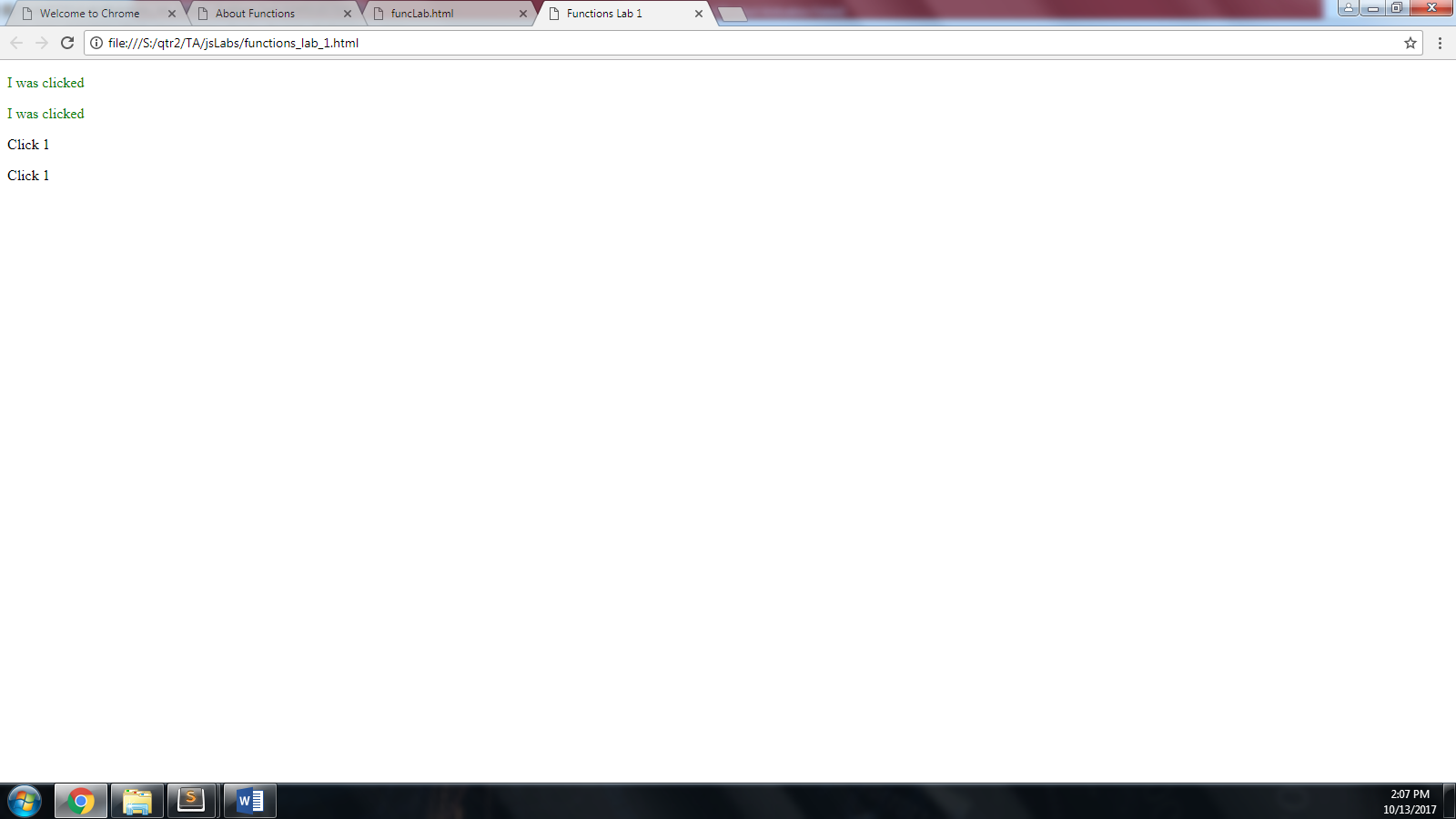
Click 1

</p>

</body>

Awesome!

Try out your click events:



We can see that our “codebase” has really grown! But what else do you notice? (Actually think about this for a second…)

An obvious pattern to an experienced programmer would be the repetitiveness of the code—there’s lots of duplication.

Now, imagine what YOU would have to do, as a developer, if there were 100 of these <p>’s and you had to continue to *maintain* them.

This is a great example of why we need functions.

Let’s see how much a function could help us in this case.

Step 3: Make the following changes to your <body> and add this JavaScript (don’t forget the <script> tags):

<body>

<p id="p1" onclick="pChanger('p1');">Click 1</p>

<p id="p2" onclick="pChanger('p2');">Click 1</p>

<p id="p3" onclick="pChanger('p3');">Click 1</p>

<p id="p4" onclick="pChanger('p4');">Click 1</p>

</body>

<script>

function pChanger(idName){

document.getElementById(idName).innerHTML="I was clicked";

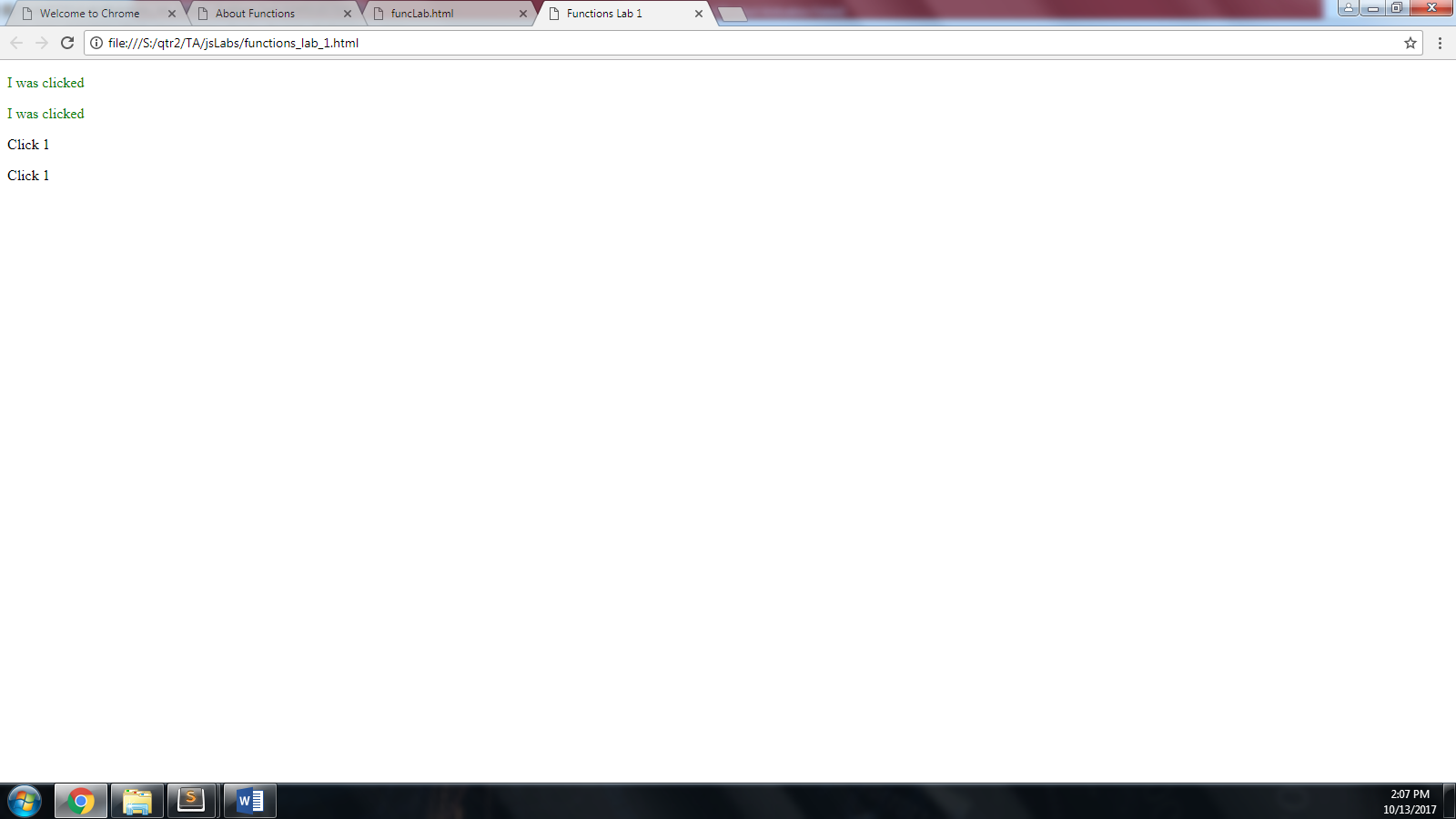
document.getElementById(idName).style.color="green";

}

</script>

Test your code in the browser:

You should still get the same functionality as before:



See how nice and neat our code is! The lines of our file has been reduced by almost half! But there’s more to it…

The real benefit here is that we have only written once what we had four times before. With this comes maintainability, meaning that we can add to this function and not have to touch the rest of the code.

We can easily add an alert to each <p> that displays its ID with only one line of code:

Step 4: Make the modifications in **bold** to your pChanger function:

function pChanger(idName){

document.getElementById(idName).innerHTML="I was clicked";

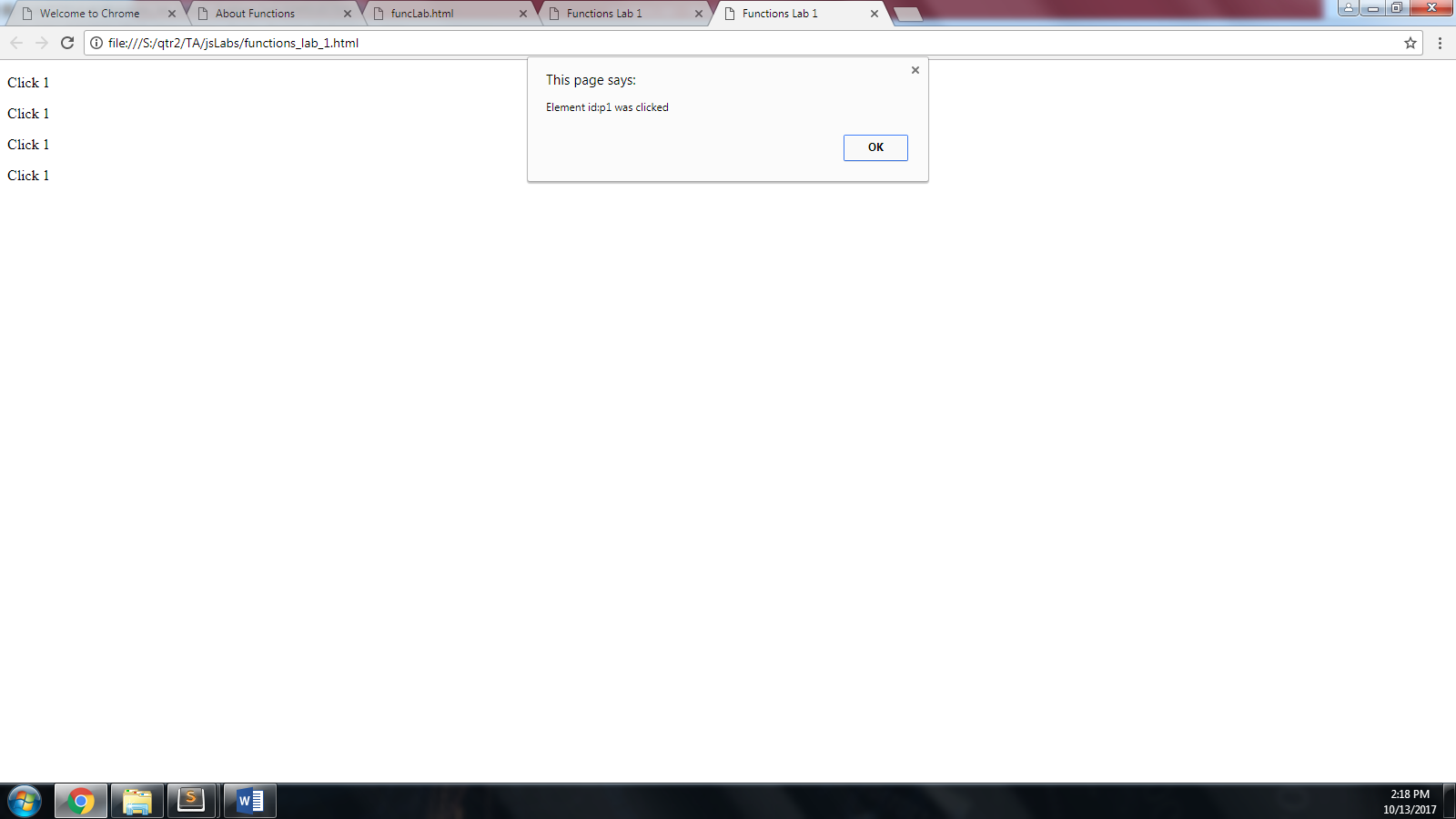
document.getElementById(idName).style.color="green";

**alert("Element id:"+idName+" was clicked");**

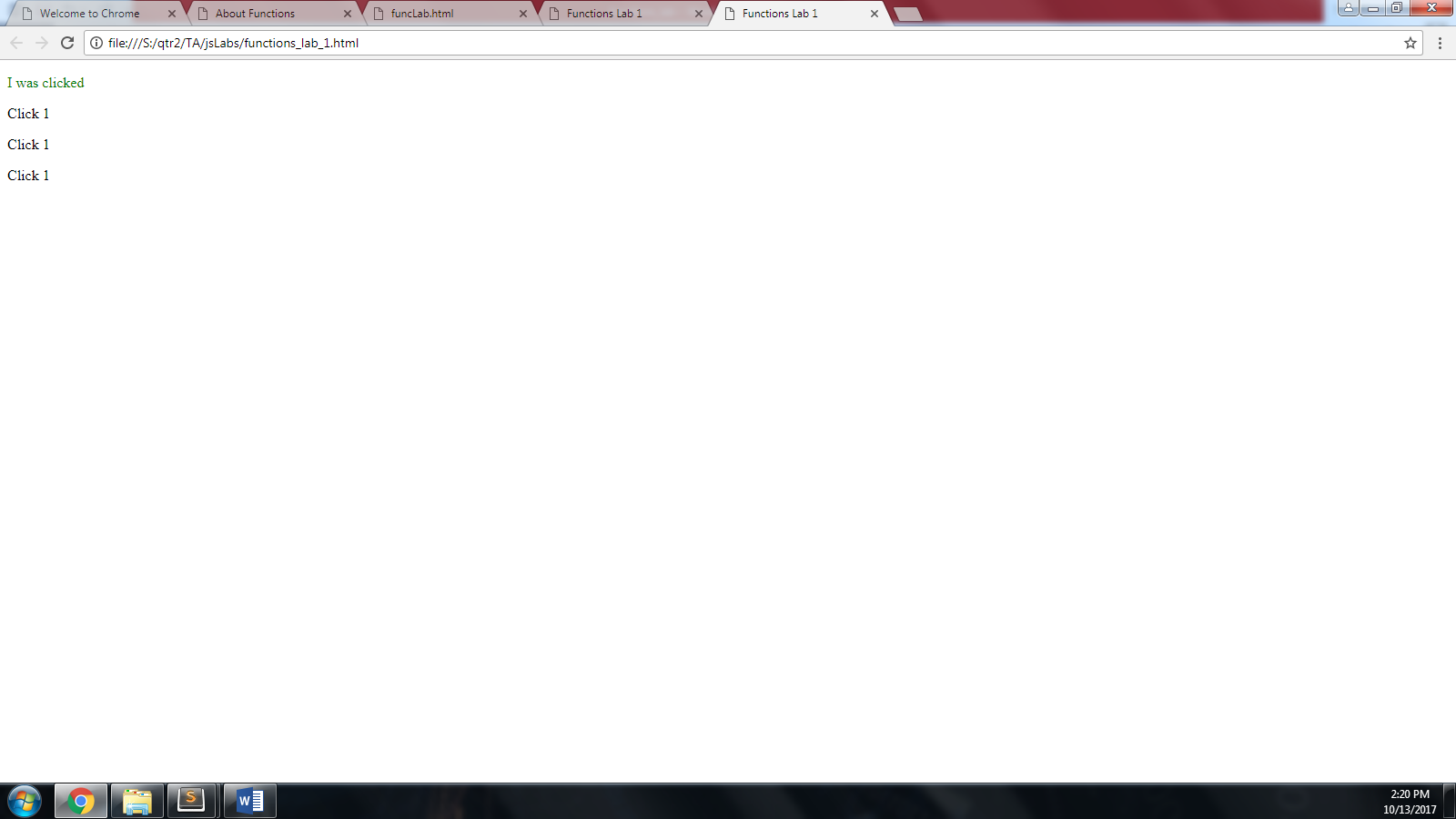
}

Let’s test it out:

You should get an alert showing the id of the element that was clicked:



Then the innerHTML will change after clicking ok.



Okay, we’re finished with the html file. The rest of the primer will use the console.

Now let’s move on and dissect how functions really work.

## The parts of a function

A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:  
**(*parameter1, parameter2, ...*)**

The code to be executed by the function is placed inside curly brackets (this is also known as the function body): **{}**

function name(parameter1, parameter2, parameter3) {  
    code to be executed (the function body)  
}

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables (meaning that they only exist inside of the function body.)

A function is much the same as a Procedure or a Subroutine, in other programming languages.

Let’s dissect our function—using these new terms—that we used in our last code example:

Function signature, function name, parameter(s), function body

function pChanger (idName)

{

document.getElementById(idName).innerHTML="I was clicked";

document.getElementById(idName).style.color="green";

alert("Element id:"+idName+" was clicked");

}

## Return values

You can decide whether your function returns any data using the *return* keyword. The return keyword is typically placed at the end of the function or in a place where the execution is expected to terminate

Here’s an example of a function that adds two numbers and returns the sum of the two:

function add(num1,num2){

var total = num1+num2;

return total;

}

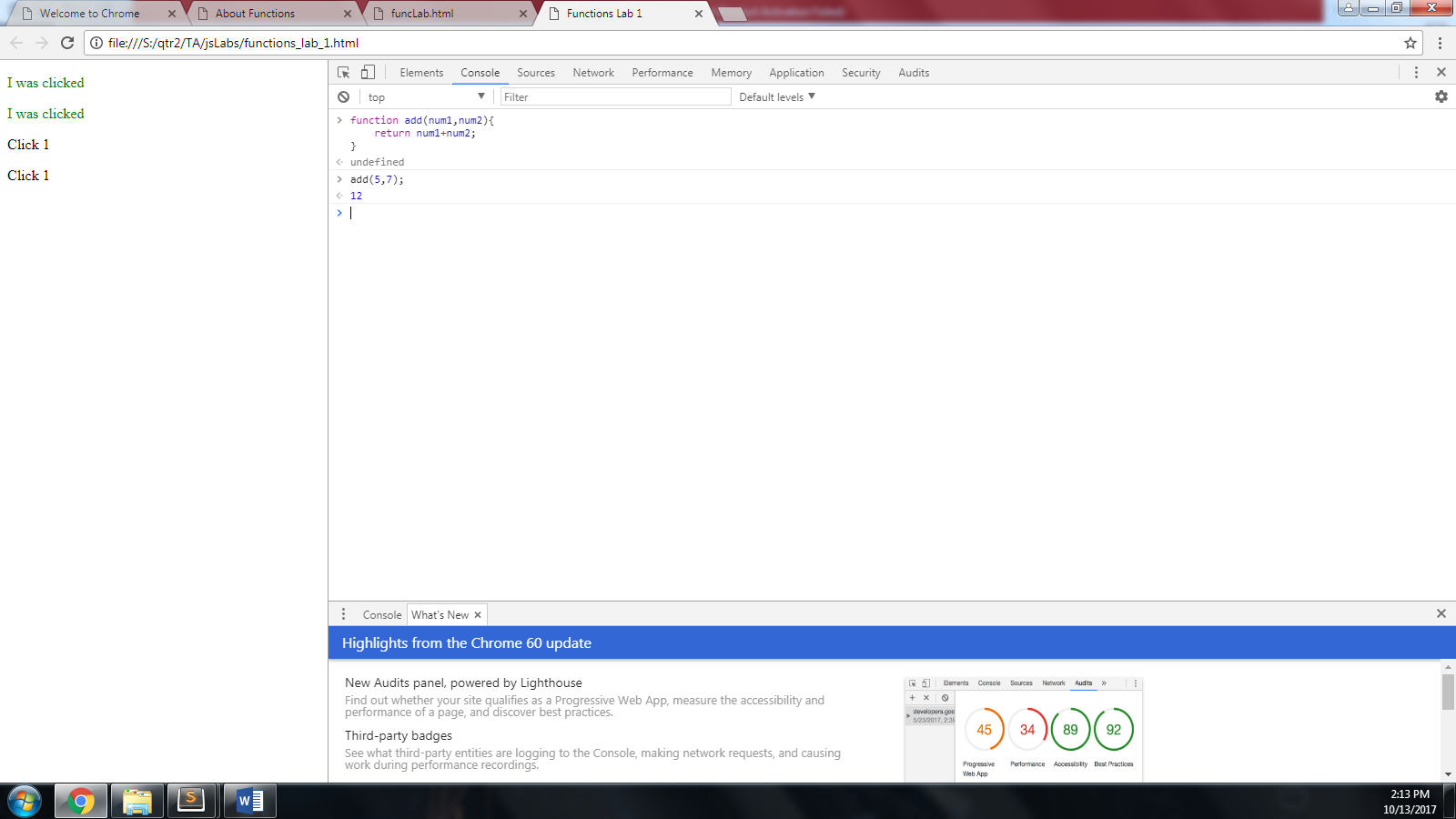
We could further shorten it like so:

function add(num1,num2){

return num1+num2;

}

Let’s test it in the console:



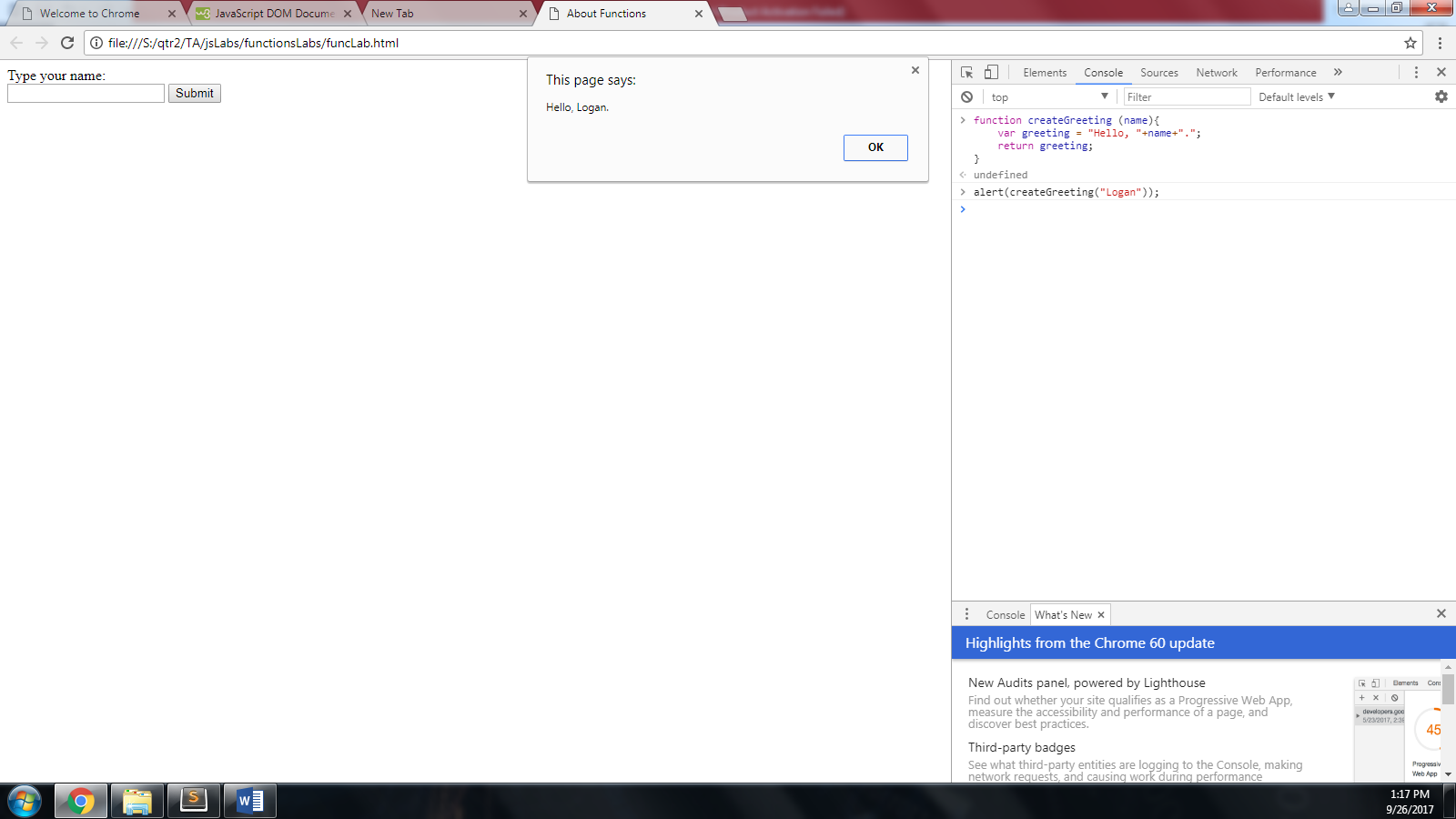
Here’s a function that returns a custom greeting:

function createGreeting (name){

var greeting = "Hello, "+name+".";

return greeting;

}

We can test this function in the Chrome console:

Notice that our *argument* for the alert function was the invocation of createGreeting. In other words, the createGreeting function returned its String variable “greeting” *into* the alert function. Therefore the value of the String “greeting” is displayed in the alert window.

When functions are invoked as arguments, the *deepest nested invocation is evaluated first*. In the example above, the createGreeting function’s code is executed and its value returned **before** the alert’s code executes. The alert doesn’t know that it’s waiting on a function to run, all it knows is its getting a value as its argument, which it is—the greeting (a string).

Using your name, try out the *createGreeting* with *alert* in the console as shown above.