**Unloop Full-Stack Web Dev**

**Lab: Hangman Game** ;)-|-<

Frequently in software development you will have to leverage or modify an existing code base or library to solve the problem at hand. In this lab you will be utilizing prebuilt code, containing global variables and functions to make a hangman game!

# Before we start…

**How do we play hangman?**

The game of hangman has a user guess letters contained in a hidden word. If the guess is correct the letter is revealed in the hidden word. If the guess is not correct, a body part is added to the person; our variation uses six body parts. If all letters are guessed before the body is complete, the player wins. But if all body parts are on the body, the player has lost.

**What’s included in this lab?**

Two files: *index.html*, and *hangman.js*.

Make sure you have these files before proceeding.

Lab Overview

You will be given two files, one bare-bones HTML file to use as a user interface (UI), and a JavaScript file that provides the underlying *game logic* via global variables and functions.

**Your assignment is to write your own JavaScript code to use the Document Object Model (DOM) to bridge the gap between the provided HTML and JS files. In other words, you’re gluing together—communicating between—two layers of code with your own.**

HTML (UI) Front End

(Index.html)

External JavaScript file

(hangman.js)

<script>

Your internal JS/DOM code talks to the HTML UI and the external JS file.

</script>

# Let’s Look at the Code

**Index.html**

|  |  |
| --- | --- |
|  |  |

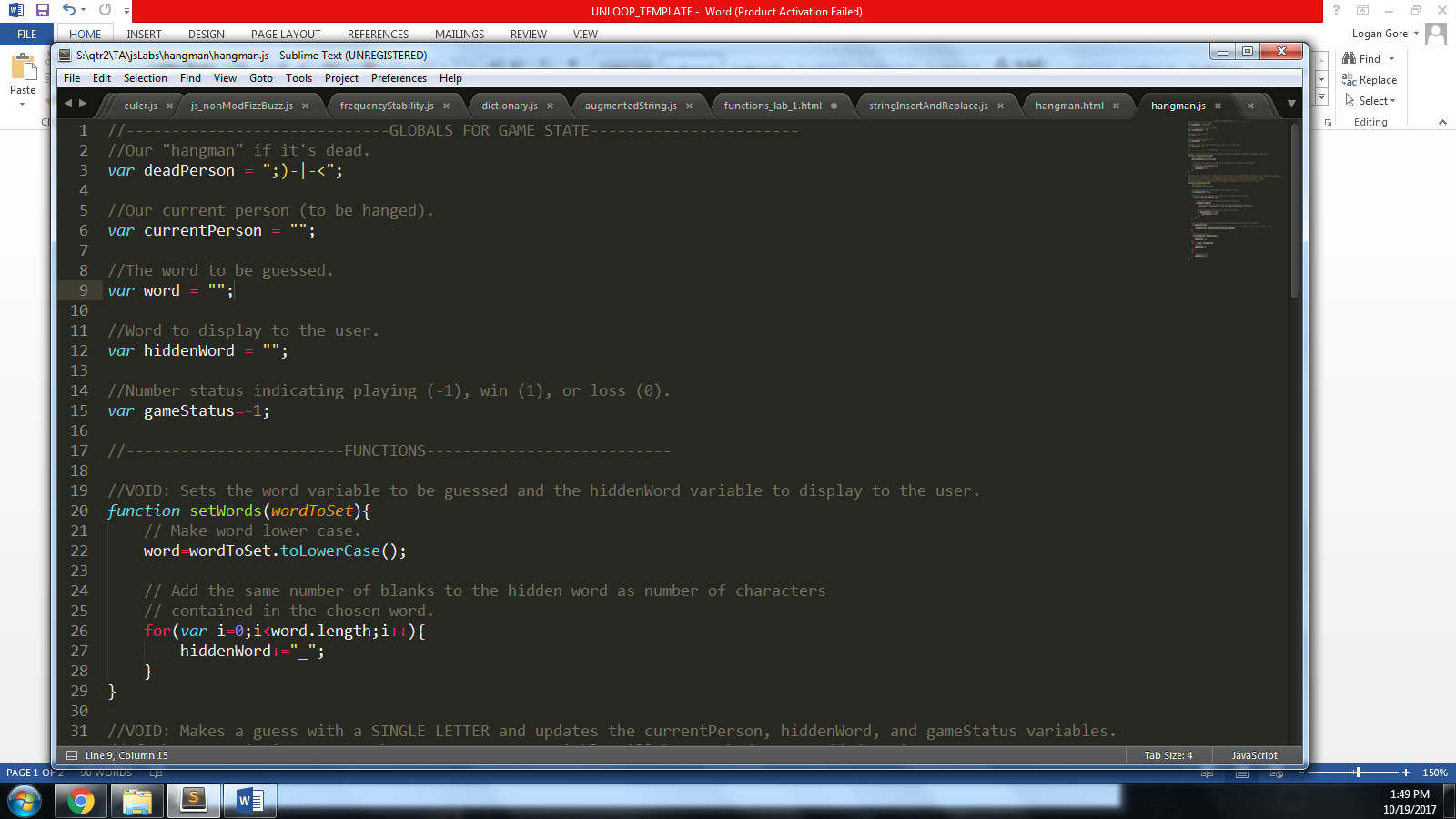
HINT: Every element with an *ID* is expected (though not required) to be used in the game.

**hangman.js**

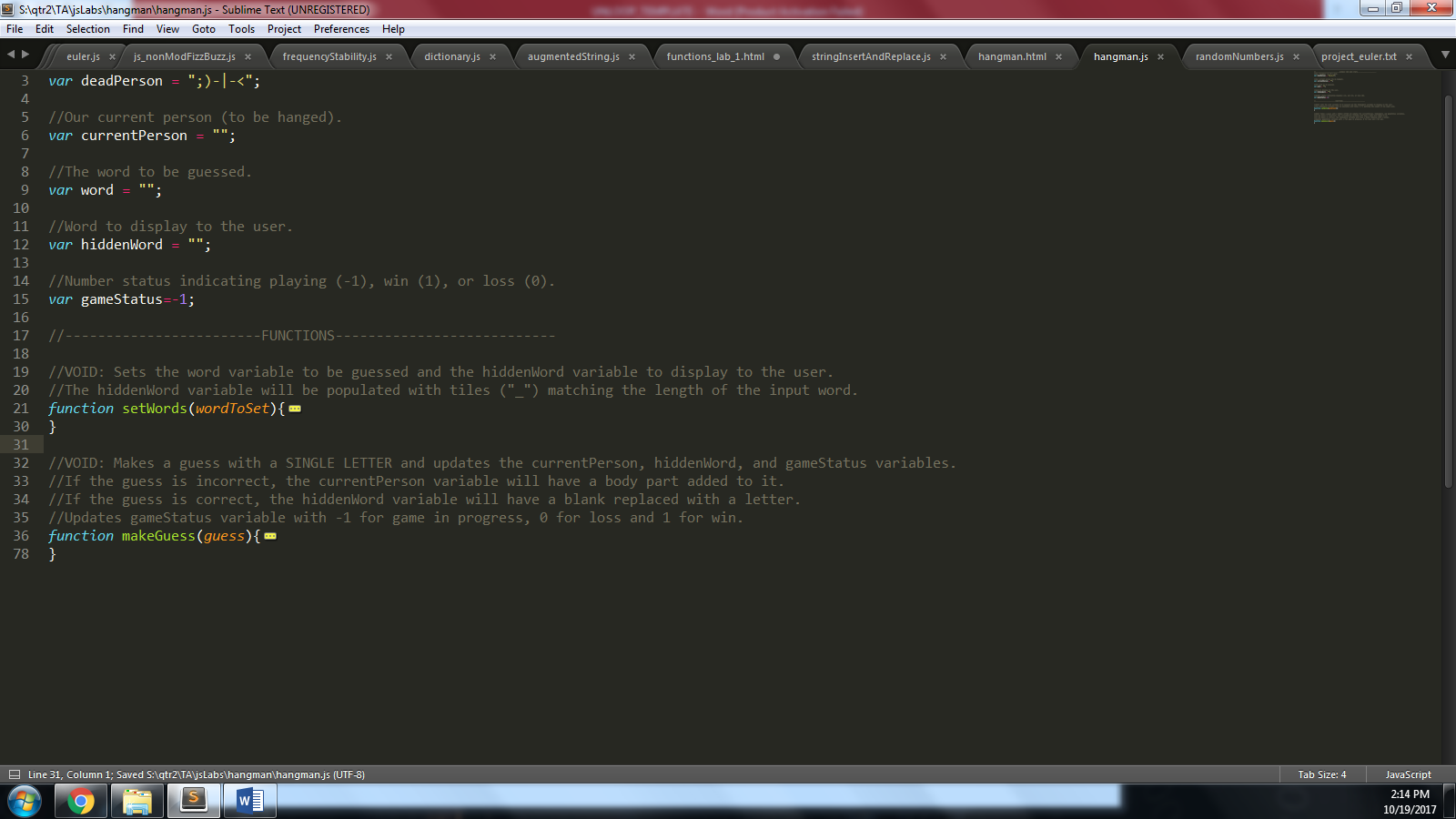
The game code is composed as follows:

Global variables are used to track the *game state*.

Game state refers to the how we keep track of our game when it’s being played. Generally, these are things like player positions, hit points, levels, and so on. In this case, we’re referring to whether we’re currently playing our hangman game, the status of the hangman itself (its body parts), the word we’re trying to guess, and guesses/blank tiles to display to the user. These are “global” so they can be accessed by the functions and still maintain their values between function calls, otherwise, if they were “local”, they would be overwritten on each function invocation and it would be difficult to keep track of our game.



And two functions to operate the game itself (their bodies are hidden in this screenshot):



“VOID” in the comments above the functions means that the function invocation does not *return* anything to the function caller (in JavaScript this means it returns “undefined.”) This term “void” is used often in other programming languages.

You will have full access to this code in the JS file but you CANNOT edit it.

# Lab Objectives

**Game Behavior**

*The word “somehow” is used in the following objectives, implying that you will need to figure out your own solution using JS/HTML/DOM to achieve the particular game behavior.*

1. Allow a user to enter a word to guess:
   1. Using the **word** input box and the **wordBtn** button in the UI, *somehow* invoke the **setWords()** function in hangman.js, which sets the global variables in the game.
2. Allow a user to enter a letter to guess.
   1. Using the **guess** input box and the **guessBtn** in the UI, *somehow* invoke the **guessLetter()** function in hangman.js, which updates the global variables in hangman.js (**hiddenWord**, **currentPerson**, and **gameStatus**).
   2. If the player wins or loses, see #4.
   3. If the game is still in play, see #3.
3. Display the game information if the game is being played, i.e. display the correctly guessed letters and the current person(the body of the hangman) after each guess.
   1. *Somehow* display the **hiddenWord** and **currentPerson** global variables in hangman.js through the **hiddenWord** and **currentPerson** <h2> elements in the UI.
4. Display whether the player has won or lost.
   1. *Somehow* display the outcome of the game with the **status** <h2> in the UI, e.g. “YOU WIN” or “YOU LOSE”, *if* the **gameStatus** global variable in the hangman.js file is either 1 (win) or 0 (lose).

**Technical Constraints**

1. Use the DOM to solicit user input and display output through HTML tags.
2. Do not use prompt() or alert() functions.
3. Do not edit the provided JS file. Put your own JS code inside <script> tags in the provided HTML file.

**Additional**

HINT: **Read the code comments**, they will explain how to use the variables and functions.

**Prior to completing this lab, read the following documentation pages on the JS DOM.**

1. [DOM Methods](http://www.w3schools.com/js/js_htmldom_methods.html)
2. [DOM Elements](http://www.w3schools.com/js/js_htmldom_elements.html)
3. [DOM Events](http://www.w3schools.com/js/js_htmldom_events.html)

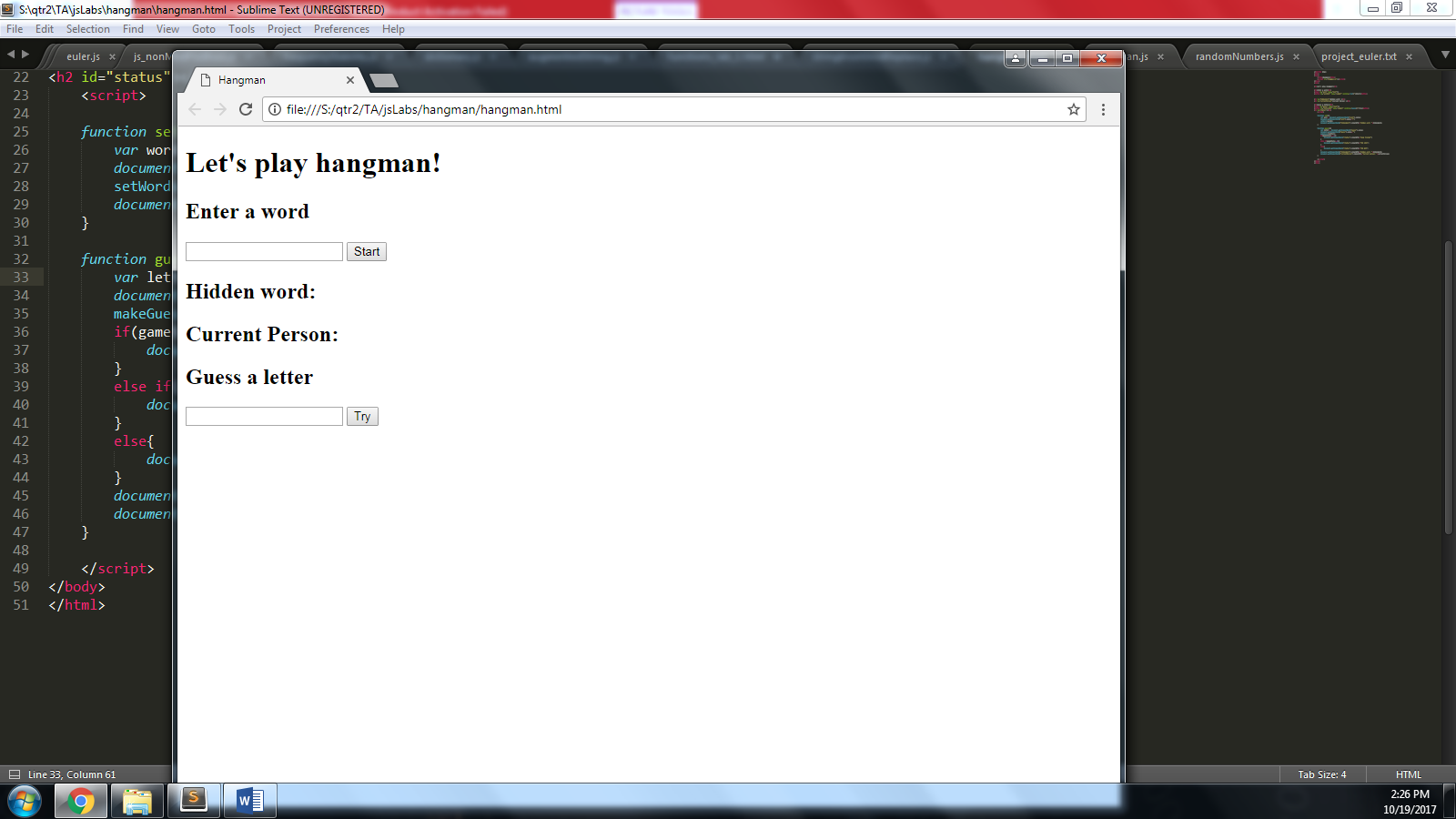
**You will need to utilize the follow DOM methods/properties:**

* document.getElementById()
* document.getElementById().value
* document.getElementById().innerHTML
* The **onclick** attribute for HTML elements.

# Example Hangman

**Here’s an example using the exact code file you have:**

1. We’ll start with a page that looks like this (this bare bones HTML file is provided to you):



1. Next we’ll enter a word into the input box and “display” our hidden word (a series of underscores.) (We also set the text box to a “?” to make it a little more exciting):

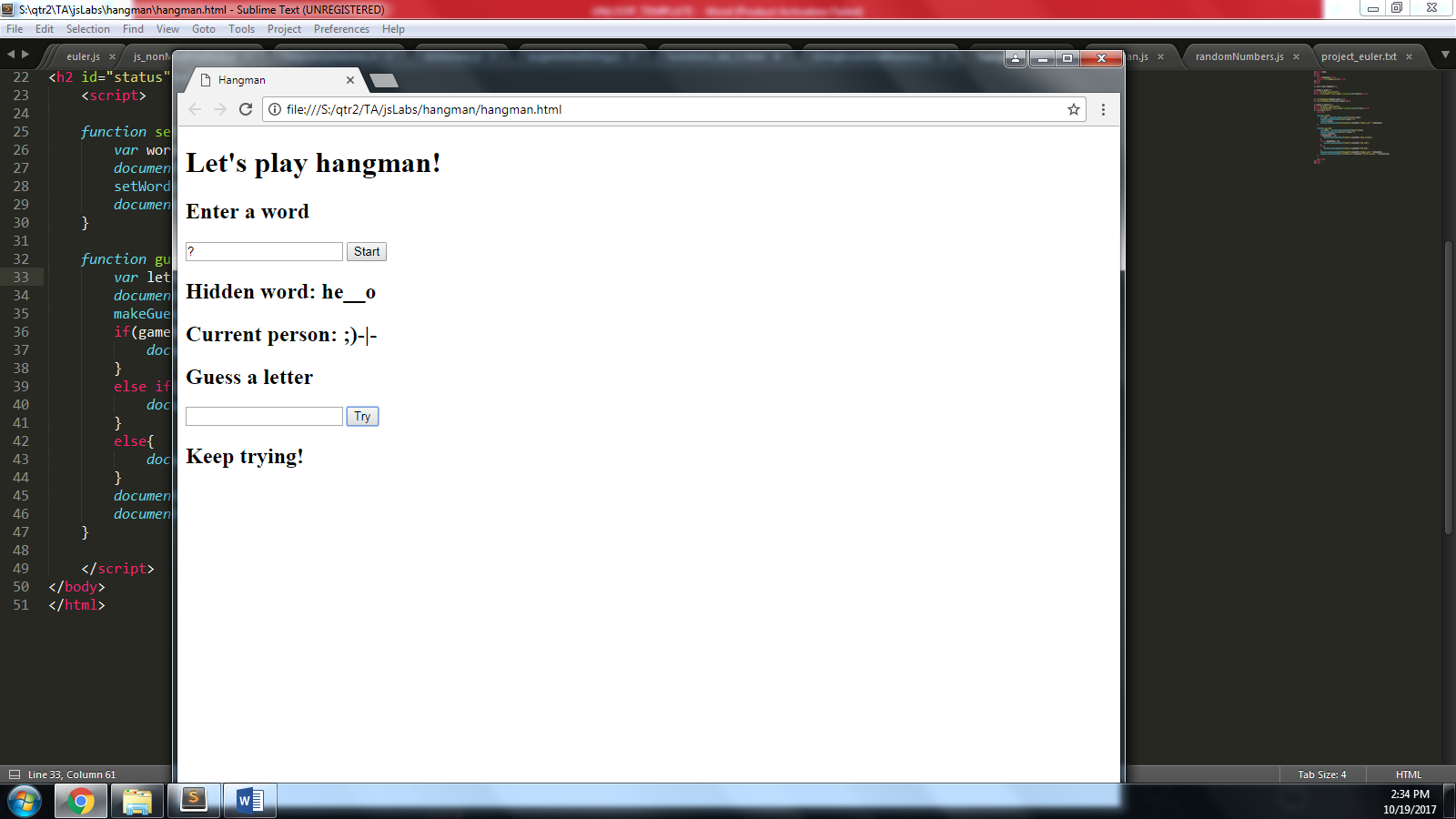
The “start” button should call the **setWords()** function when clicked. The setWords() function should take in the text box input as an argument. This initializes the game state and assigns values to the *word* and *hiddenWord* variables*.*

|  |  |
| --- | --- |
|  | *After* entering our word and invoking **setWords()** the global variable, *hiddenWord*, should display to the user. |

1. Enter letters to guess into the other input box. On the lefthand side below, we have entered our letter guess of “h.” On the right, we see that ‘h’ was found in our hidden word. The “interface” is updated accordingly.

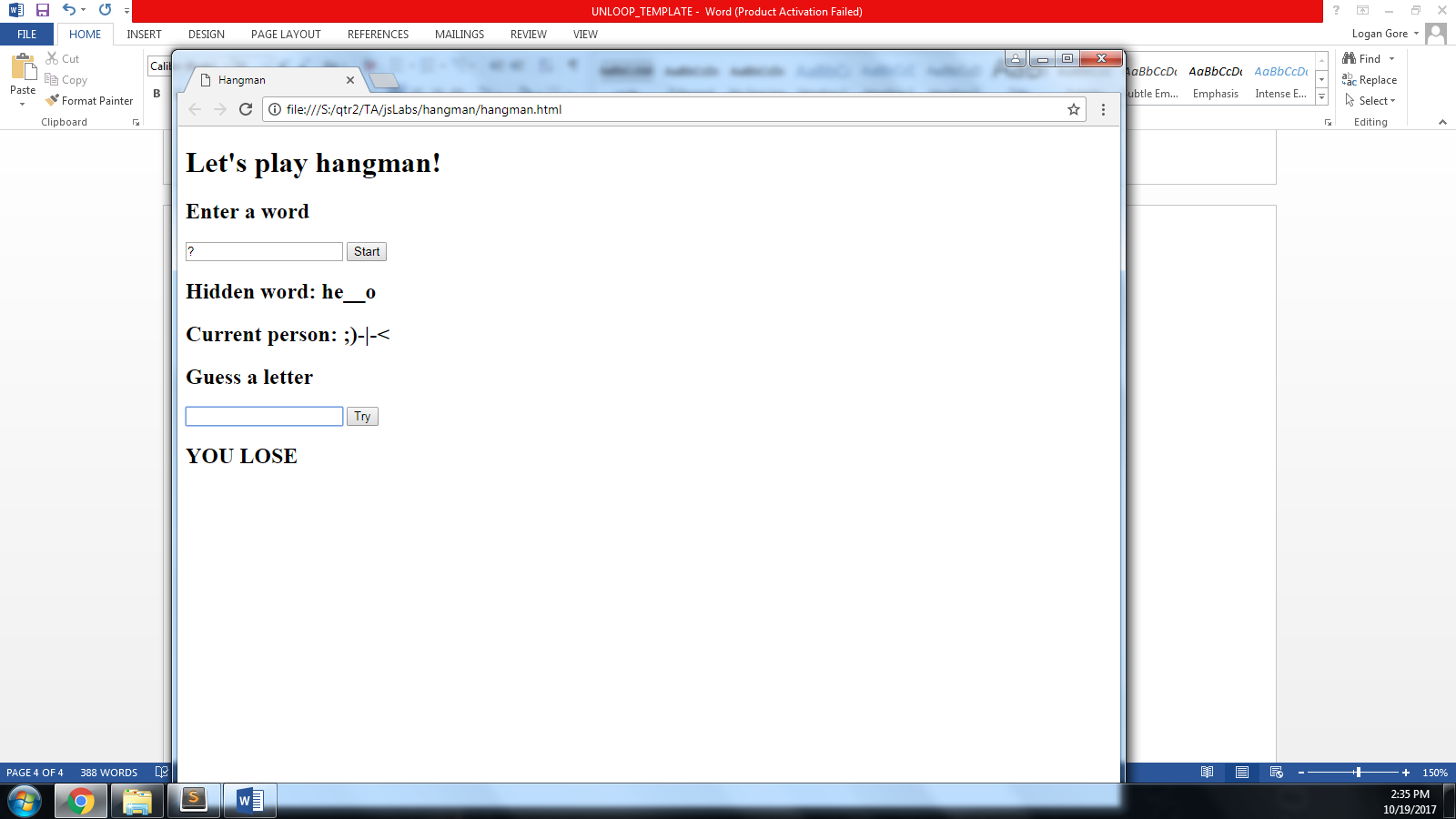
|  |  |
| --- | --- |
|  | Invoking **makeGuess()** will update our *hiddenWord*, *currentPerson* and *gameStatus* variables. Here we display our hidden word *if* our game status reflects a running game (status==-1)  The “try” button calls the **makeGuess()** function and passes it the letter in the text box as an argument. |

And here’s a few incorrect and correct guesses. Every time the user guesses incorrectly, a “body part” of the hangman is displayed. As letters are guessed correctly, the hidden word updates:



Again, invoking **makeGuess()** will update our *global* variables. Here we display our hidden word and current person *if* our game status reflects a running game (*gameStatus*==-1).

Keep in mind, *if* gameStatus is 1, we need to reflect a win, otherwise if our *gameStatus* is 0, we need to reflect a loss.



Finally, invoking **makeGuess()** with a sixth bad guess updates our *global* variables and sets *gameStatus* to 0 (a loss). In which case, we display the totally hanged person and notify the user of their misfortune.

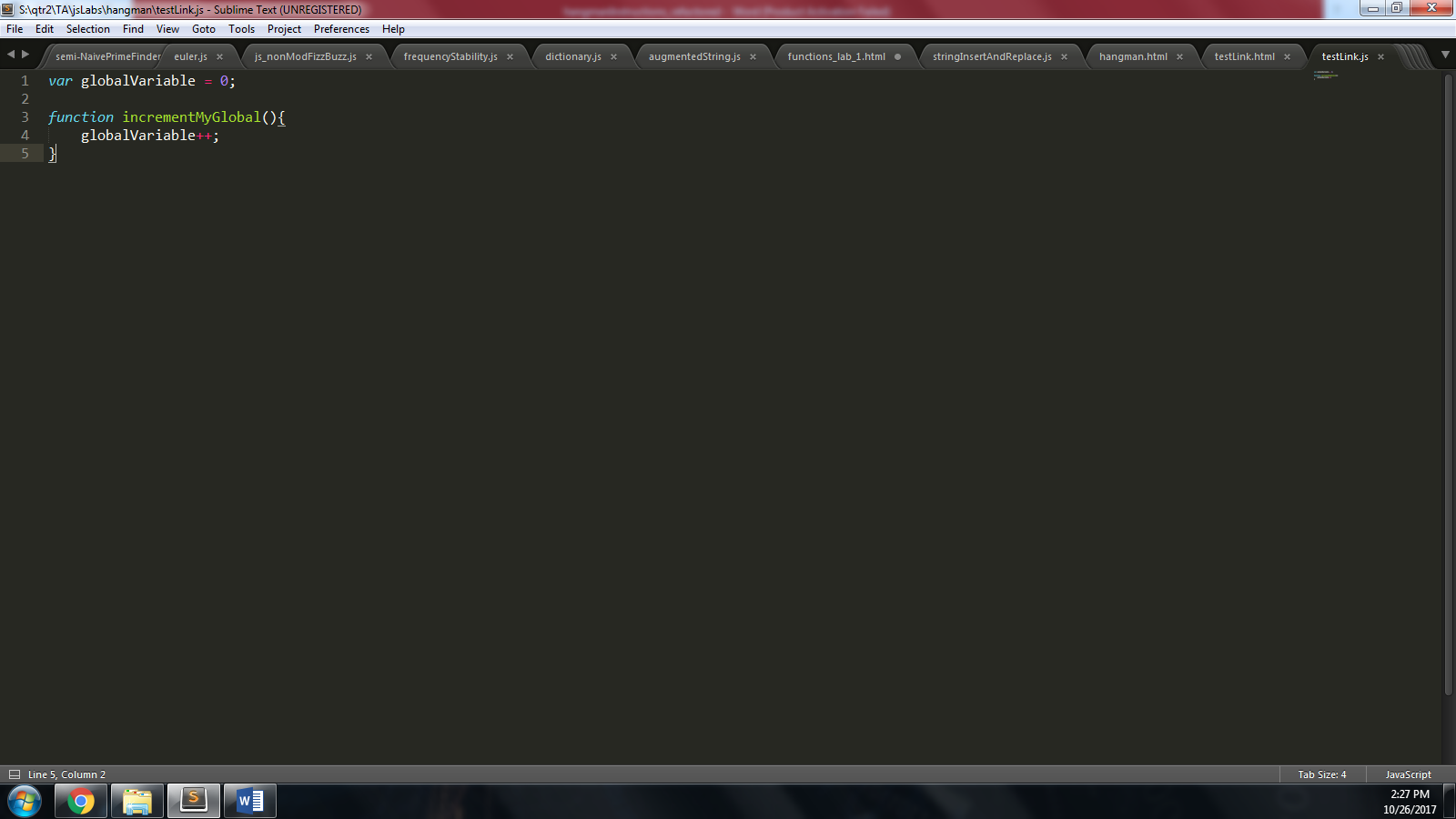
Okay, now that you have an example of what this *could* look like, leverage the existing code to build your game.

# Example: Linking to and leveraging an external script

Okay, we’ll show really quick how an HTML file would link to and variables and functions in an external JS file. You’ll need to do something similar in your own solution.

**Linking to the external file**

Here’s the external js file, named *testLink.js*

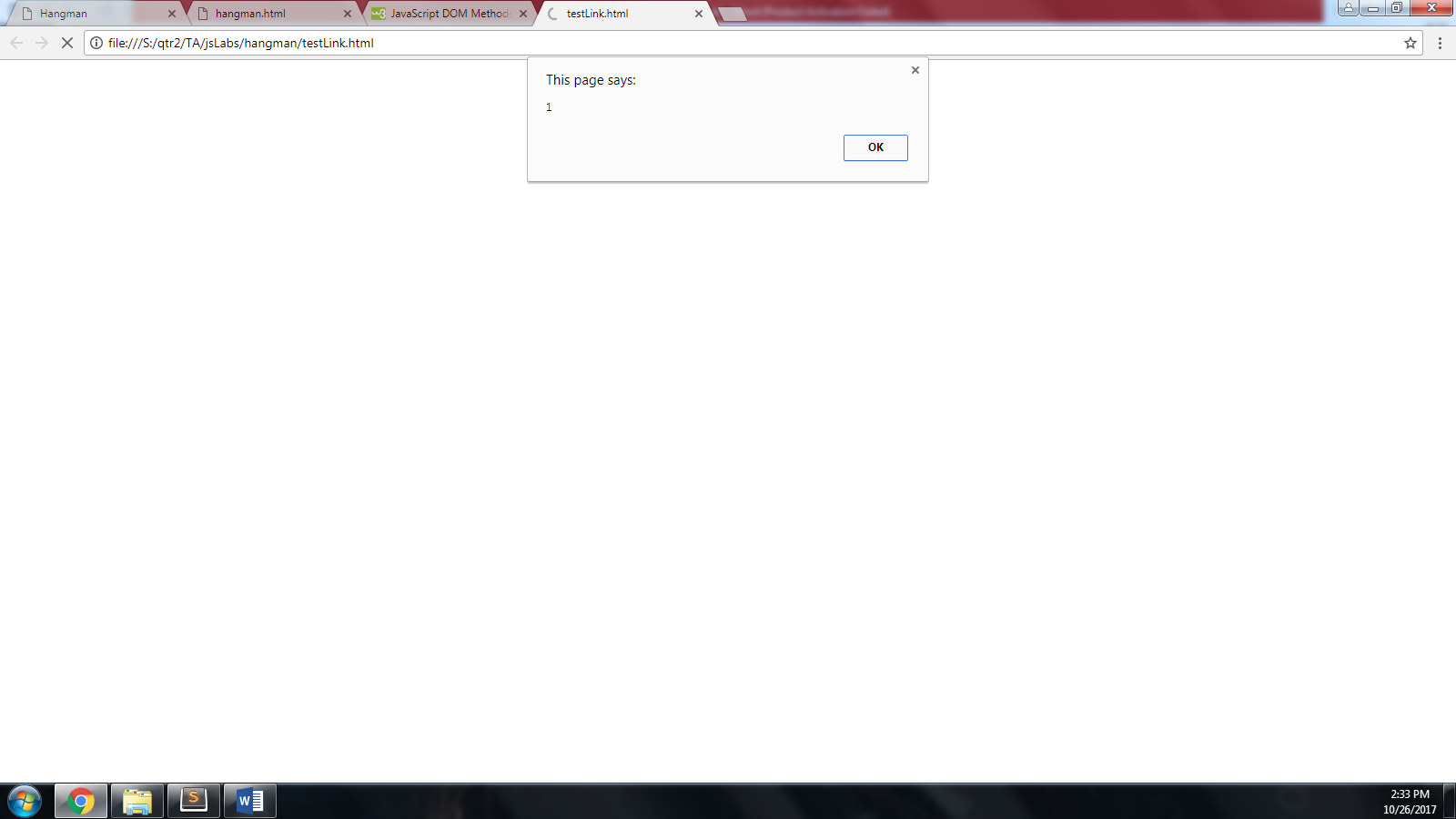


**Leveraging the referenced JS in an internal script tag**

Here’s our HTML. We link to the external script prior to including internal JavaScript code.



Here’s how it displays in the browser:



Notice that the page displays 1, even though the value of the variable is initially set to 0 in the external file. This is because the function call to **incrementMyGlobal()** incremented **globalVariable** whose state/value exists while our page is loaded. Any changes made to **globalVariable** will remain until the page is closed.