Hello GEOG461 Students!

In order to make dynamic web maps, you need to know some basics of HTML, CSS and JavaScript. No programming experience before? No worries. All the things you need to get through this class are included in this document.

I. Intro to HTML

Approximate time: 1h

Please read the following HTML tutorial sessions from W3schools.com: Don't miss the fun parts of Try it yourself >>

- HTML Introduction **TO** HTML Attributes (5 sessions here)
- HTML Comments
- HTML CSS
- HTML Blocks
- HTML Layout
- HTML JavaScript
- HTML Charset
- HTML Event Attributes

Other sessions and quizzes are optional. You are welcome to learn as much as you want!

II. Intro to JavaScript

Take a brief look at this page for a flyby introduction to JavaScript. A lot of the following notes are summarized from the book *Coding with JavaScript for Dummies,* which I think are most related to the contents covered in this class. Copyright belongs to original authors. if you have extra time and energy, and you're new to JavaScript (JS), you're highly encouraged to read the original book, especially Part I. Also please read JS Debugging here on 3swchools.com**.** You can check out other sections during your spare time.

Tools needed

(~25 min)

Download and install accordingly

- **Web browser**
  Google chrome is recommended since all the instructions below are based on Chrome. Also it has an excellent Developer Tool to work with Javascript. However, you can also use Mozilla Firefox and get the Firebug add on for development and debugging purposes.

To open the Developer Tool, find the Chrome menu at the right-upper corner of your browser, ☰ , More Tools -> Developer Tools. The interface looks like **Figure 2.1**:

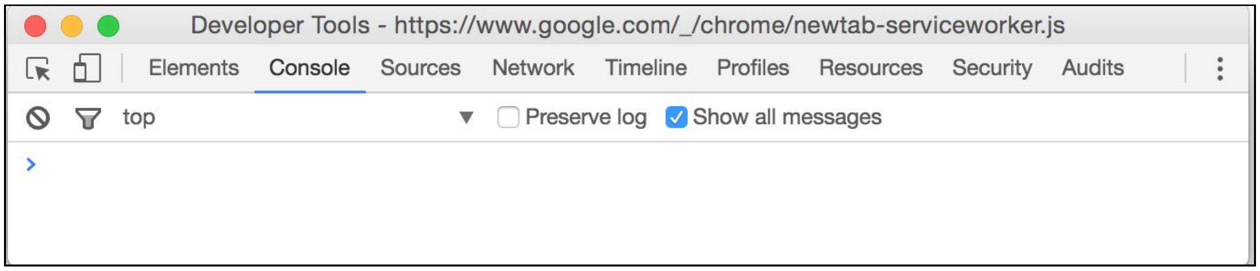


Figure 2.1. Developer Tools of Chrome with the display of console panel

The developer tool can appear in three ways (see **Figure 2.2**), on a separate window, bottom of your browser window or right side of your browser window. Click on the three dots symbol ⋮ at the right-upper corner to choose your preferred appearance.

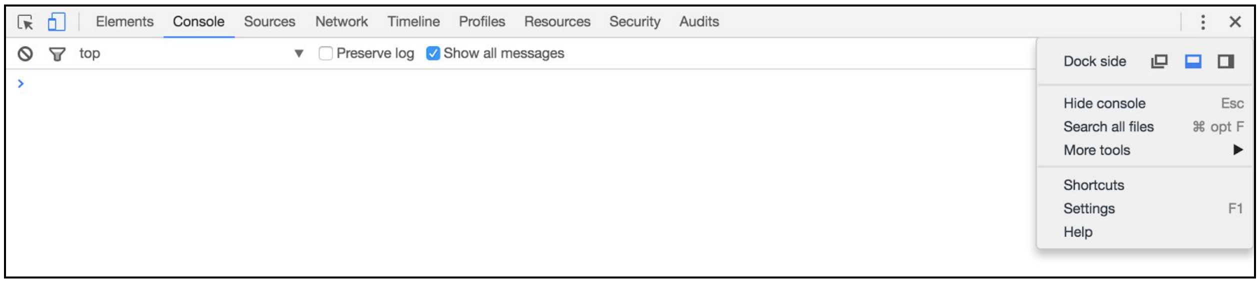


Figure 2.2 Placements of Chrome Developer Tools

If your web page doesn't show up as you expected, you can find error messages in the **Console** tab of this developer tool. Also, JavaScript programmers like to use console.log("messages you want to show"); to debug their code, which we will cover in our lab tutorials.

- **Code Editor**
  Sublime Text is recommended. Not only because it's very popular among JavaScript programmers, but also it provides a lot of powerful plugins with a simple user interface.

  *It doesn't matter if you use Sublime Text 2 or Sublime Text 3, they both can be downloaded for evaluation purpose for free. The developers expect users who continue to use the software to buy the license ($70). Specifically, the web site explains future purchase expectations after an evaluation as follows: "Sublime Text may be downloaded and evaluated for free, however a license must be purchased for continued use. There is no enforced time limit for the evaluation.".

Run JS in Browser Window

(~20 min)

Here are three ways of using JS:

✓Put it directly in an HTML event attribute (example 01)
✓Put it between an opening and closing script tag (example 02)
✓Put it in a separate document and reference it in your HTML document (example 03)

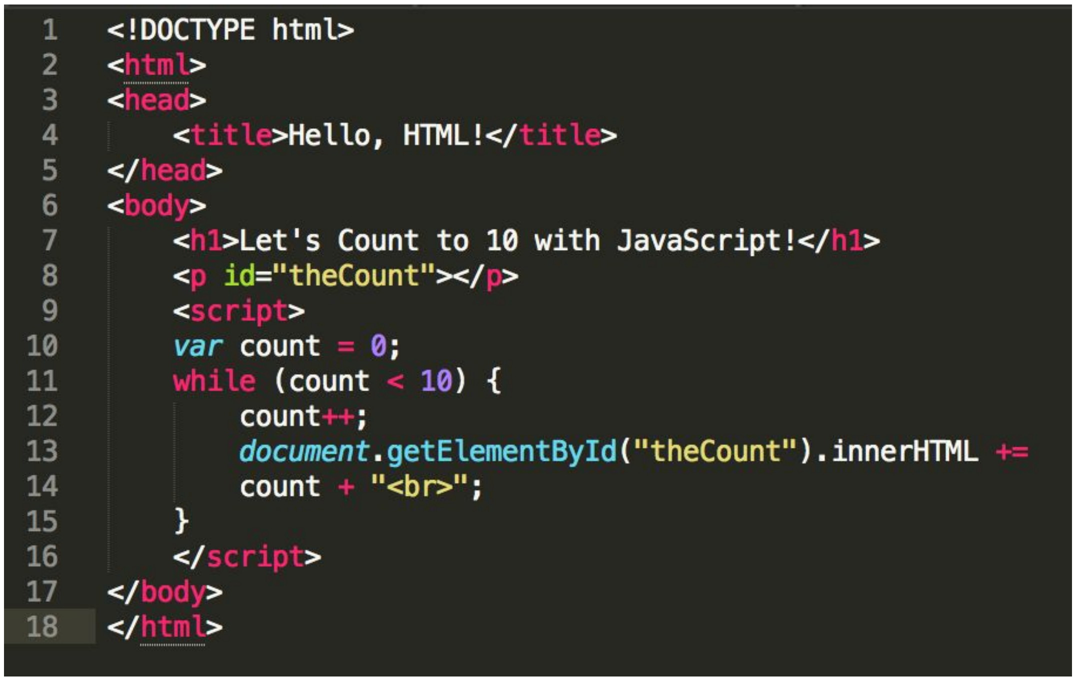
Example 01 As An HTML Event Attribute

```
<button id="bigButton" onclick="alert('Hello World!');">Click
Here</button>
```

In this case, when a user clicks on the button created by this HTML element, a popup will appear with the words “Hello World!”. The string (marked by double quotes), alert('Hello World!'); is actually a javascript function. The JavaScript will be executed after the button is clicked.

Example 02 Within <script> and </script> Tags

Insert the code within script tags, like line 10 to line 15 below in Figure 2.3:



```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>Hello, HTML!</title>
5   </head>
6   <body>
7       <h1>Let's Count to 10 with JavaScript!</h1>
8       <p id="theCount"></p>
9       <script>
10      var count = 0;
11      while (count < 10) {
12          count++;
13          document.getElementById("theCount").innerHTML +=
14          count + "<br>";
15      }
16      </script>
17  </body>
18  </html>
```

Figure 2.3 Put JS codes inside script tags

Example 03 As External JS File

```
<script src="myScript.js"></script>
```

In this example, rather than having the Javascript within the HTML, it can be in a separate file. Specifically, you can extract line 10 to line 15 in example 02 and make it a separate file (named myScript.js), and then put this js file in the same folder as your html document. That file can be called and executed using just one line of code, as shown above. This one works exactly as example 02, but by separating the js file with html, it gives more flexibility of reusing and modifying the js file (This is how we use leaflet js library throughout this class)

The Placement of JavaScript in a HTML Document

Sometimes you'll see the script tags are placed within the head and sometimes in the body of the HTML document. It actually is not the exact place that matters but rather the order. The browser will read and execute the webpage from the top down. So if you want to wait to execute your code until all contents are loaded, you can do something like example 01, onclick = <your javascript> or like example 02, simply put your code at the very bottom of your page.

Run the Script

Okay. Now the most important part. How to run the script in a web browser? Here are two ways:

1) Double click your html file. Your default browser will show the html to you.
2) Chrome -> File -> Open file and select the html file.

It's really easy, right? However, knowing what happened behind the click is more important and helping you do so is the goal of this tutorial and labs that follow.

Key Rules and Concepts
(~35 min)

Now let's take in a few key rules of writing javascripts which can be easily missed by beginners and cause most of errors. The key concepts will help you see the forest for the trees.

A few rules of JavaScript (JS):

- JS is **case-sensitive**. So Ben and ben would indicate a different "person" (we call it an object) in the JS world.
- Watch out for **reserved words**. Some words are reserved in JS world and you cannot use them to name your variables, like *function*, *var*, *while* and so on. These reserved

words usually are explicitly colored in sublime text to make their reserved status easy to recognize.

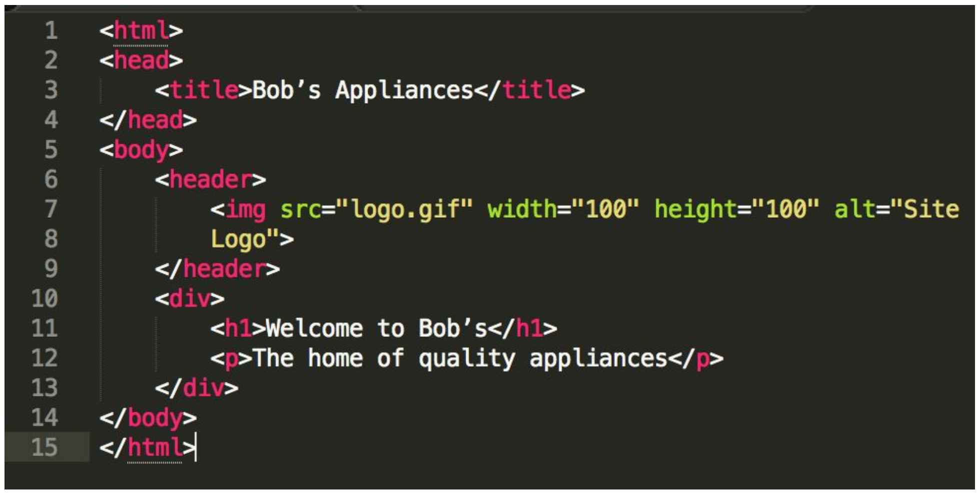
- A statement is like sentence of a paragraph. It's the fundamental building block of JS programs. Statements are ended with **semicolons** in JS.

Key concepts:
(~30 min)

1) **JS Connector to HTML-- DOM.**

   DOM (Document Object Model): The Document Object Model is **the interface for JavaScript to talk to and work with HTML documents** inside of browser windows. Think of JavaScript as a set of commands to control all the components in the webpage, thus commands for how to get, change, add, or delete any HTML elements. In order to send out a command to the right object, JavaScript has to know how to identify and call the target component. This is where the DOM (a World Wide Web Consortium standard for accessing documents) comes in.

   The DOM can be visualized as an inverted tree, see example below in **Figure 2.4** (sample document code) and **Figure 2.5** (a diagram illustrating how the HTML document's components are organized):



```html
<html>
<head>
    <title>Bob's Appliances</title>
</head>
<body>
    <header>
        <img src="logo.gif" width="100" height="100" alt="Site
        Logo">
    </header>
    <div>
        <h1>Welcome to Bob's</h1>
        <p>The home of quality appliances</p>
    </div>
</body>
</html>
```

Figure 2.4 A simple html document

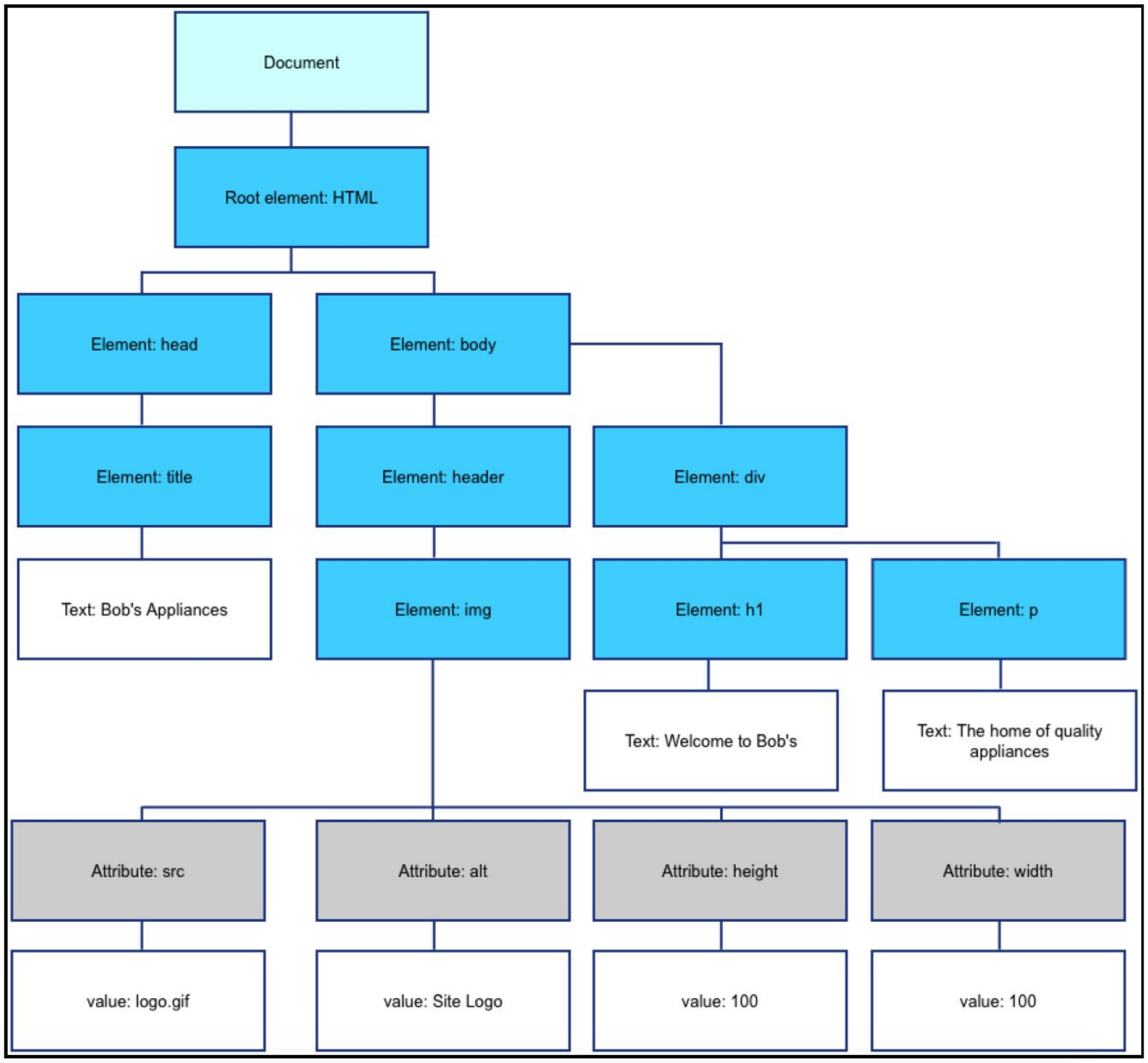


Figure 2.5 Corresponding DOM tree

2) Connector to JS Plugins -- API.

API: Think of the standard for plugs and sockets.See **Figure 2.6** below. You have to know how many holes exist in each socket and how they are organized in order to design or choose plugs. Just like the standard for plugs and sockets. APIs, or Application Programming Interfaces, are sets of software routines and standards that guide programmers to interact with each other's application. When a web browser allows a JavaScript program to interact with it, it does so by using APIs.

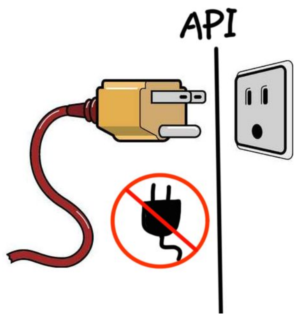
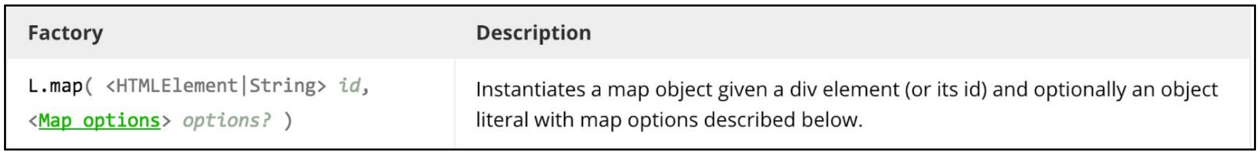


Figure 2.6 A Metaphor of API

Typically, an API is written up as a specification that tells what properties and methods are available to programmers, what arguments can be passed to the methods, and what sorts of values are returned by the properties. Let's take Leaflet's API for example. See **Figure 2.7.**

Leaflet API:

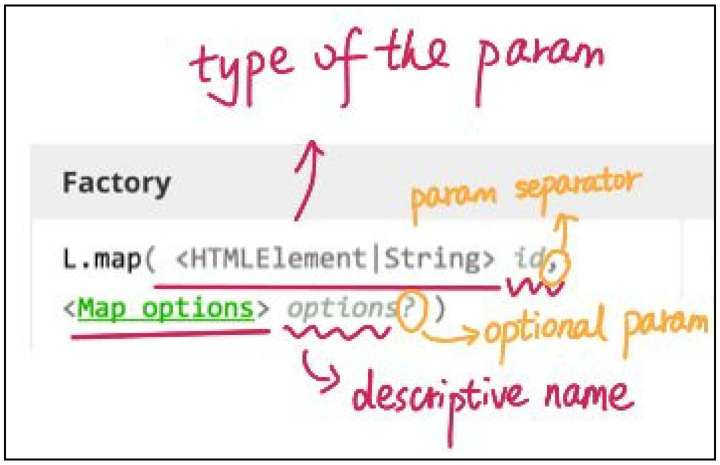
Map creation



Factory	Description
`L.map( <HTMLElement|String> id, <Map options> options? )`	Instantiates a map object given a div element (or its id) and optionally an object literal with map options described below.

Figure 2.7 Leaflet API for L.map function

When using Leaflet, you create a map by using the L.map() function in JavaScript. What about the stuff within the parenthesis? These are parameters of this function. They are separated by commas. <HTMLElement | String> indicates the type of the first parameter. And id is the descriptive name of the parameter. The question mark ? after options means this options' parameter is optional. See **Figure 2.8** for an annotated version:



```
L.map( <HTMLElement|String> id,
<Map options> options? )
```

Figure 2.8 Annotated version of L.map function API.

It seems a bit complex. While let's have a look on a sample code below:

Sample code 01:

```
// initialize the map on the "map" div with a given center and
zoom
var map = L.map('map', {
    center: [51.505, -0.09],
    zoom: 13
});
```

In sample code 01, 'map' is the id of the HTML DIV element that we want to hold the map. Center and zoom are all map options. However, when you call the L.map() function and pass arguments to it, you don't have to specify the options parameter, like the sample code 02 showing below .It only gives a 'mapid' to the L.map function.

Sample code 02:

```
<div id="mapid"></div>
<script>
      var mymap = L.map('mapid').setView([51.505, -0.09], 13);
</script>
```

Okay! Let's wrap up this bootcamp. Take the W3Schools HTML quiz. Remember, you can take the quiz as many times as you want, and take as long as you want to take the quiz. You can also look up W3Schools resources while taking the quiz, so treat it as an open book exam. Once you are able to get more than 15 out of 25 questions right, submit a screenshot of your completion page. That earns you the full grade for this lab.