# Front-End Developer - V2.0
# Lab Exercises (5/13/2019)

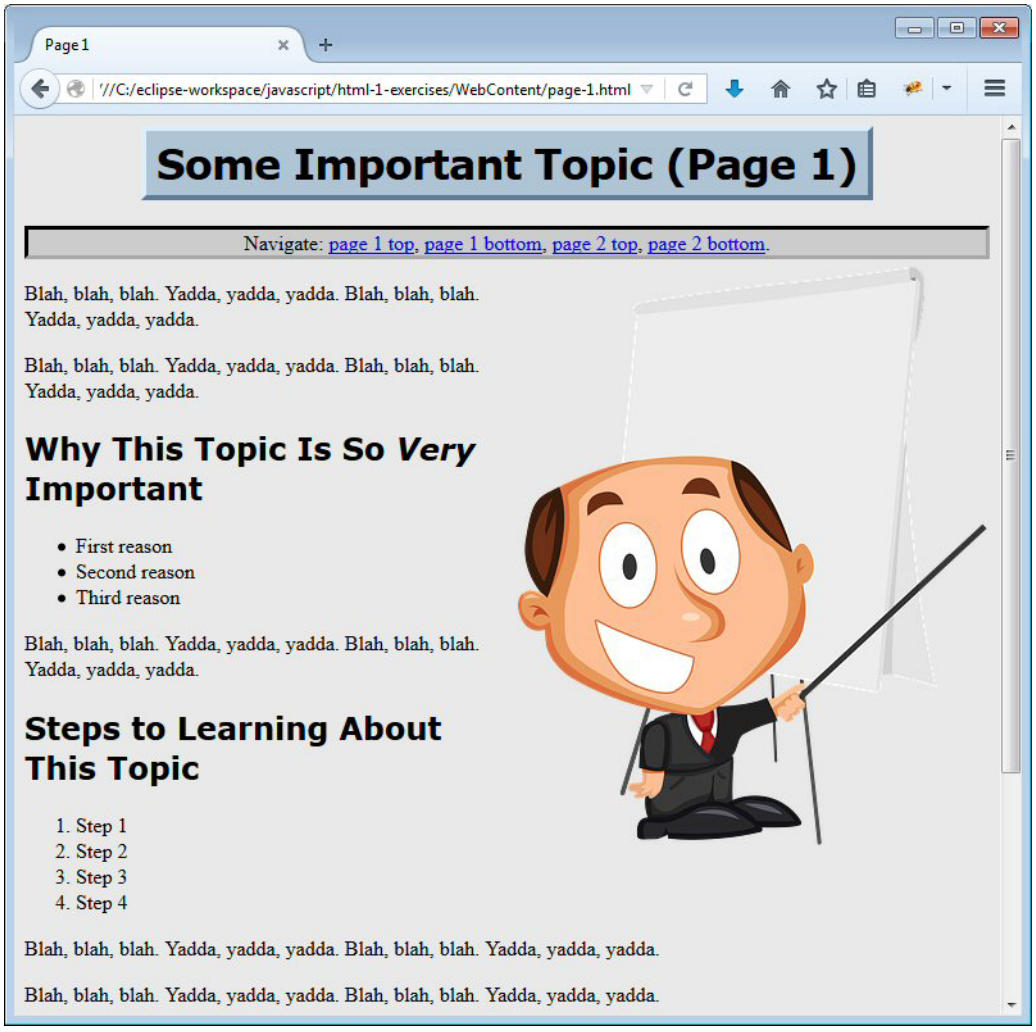
## Table of Contents

# Lab Exercise 1: HTML Basics, Part 1

**1.** Make a page with these capabilities:

- A main heading
- Some subheadings
- Some paragraphs
- An image that floats to the right of the page with text wrapping around it. Note that images found on images.google.com are often not legal for use on your own pages. However, there are many free image sites, one of which is pixabay.com.
- A bulleted list
- A numbered list
- A navigation panel with hypertext links to the top of the current page, bottom of the current page, top of a second page, and bottom of a second page. For the second page, just copy the first page and change the tile.

Here is a representative example, although if you do not know about CSS yet, do not worry about reproducing the page background color or the color and background of the main heading.

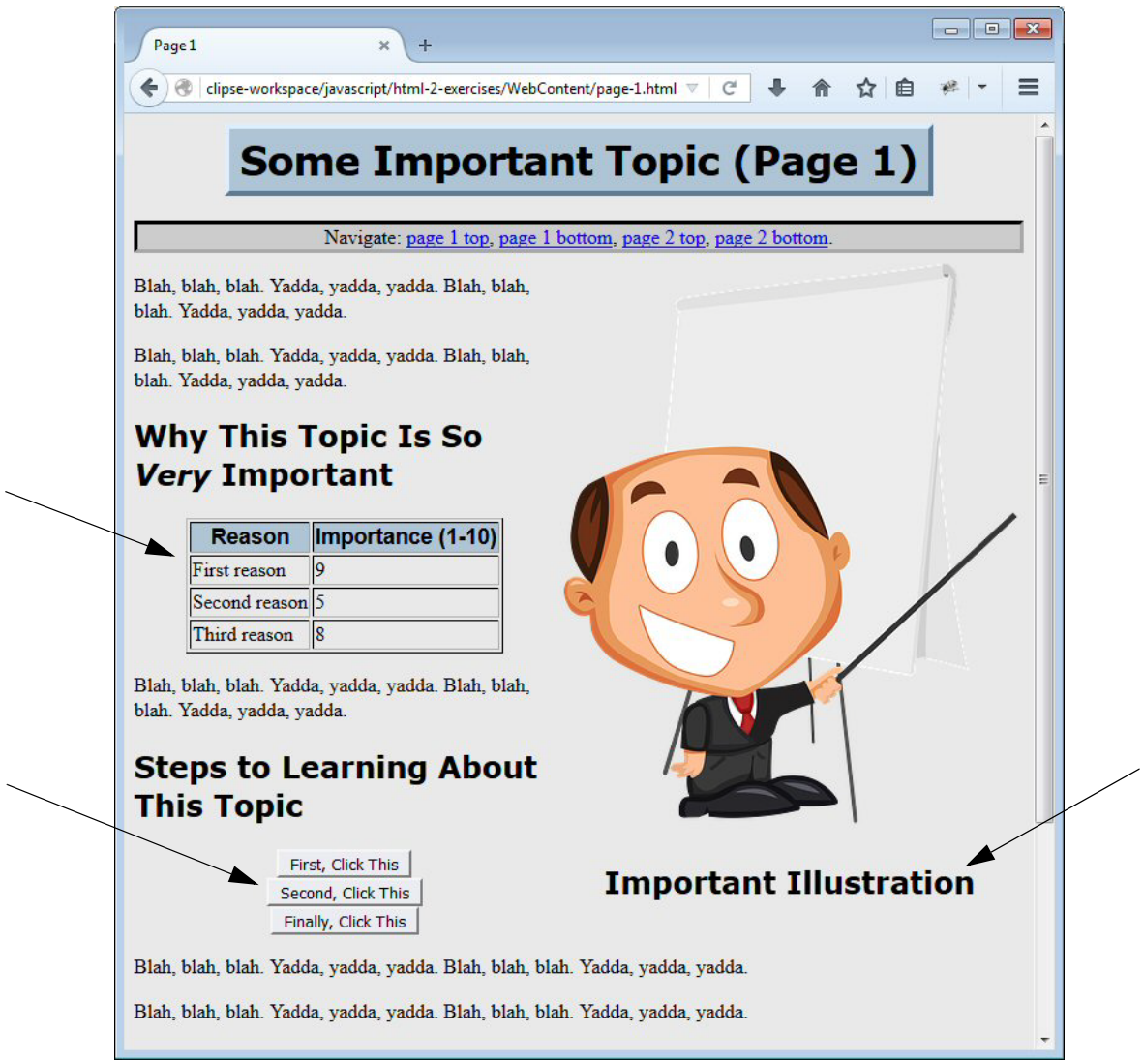


# Lab Exercise 2: HTML Basics, Part 2

**1.** Start by copying your two pages from the previous set of exercises. Then, make the following changes

- Replace the first list with a bordered table that displays some tabular data.
- Replace the second list with a set of buttons. Each of the buttons, when pressed, should pop up a JavaScript dialog (alert) that gives confirmation that the button was pressed.
- Add a subtitle below the right-aligned image.

Here is a representative example, but do not worry about reproducing the fonts and colors.



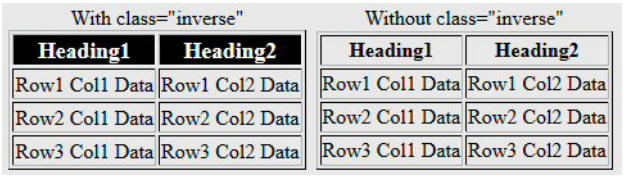
# Lab Exercise 3: CSS Basics

For the HTML on these exercises, you can have separate HTML pages for each problem, or a single HTML page for all, whichever you prefer. For the CSS, however, try to use a single style sheet to accomplish all of the tasks.

1. Have all text in both <ul> and <ol> lists be blue. Do this with only a single entry: i.e., do not style ul and ol separately. Test this by putting in both kinds of lists.

2. Change your style sheet so that ul text is blue and ol text is red. Do not modify your existing entry, but instead add a single additional entry to your style sheet.

3. Make a style sheet entry so that designated tables will have their headings in inverse video (i.e., white text on black background). This should not apply to all tables, but only tables with the appropriate CSS class name. Test by making two tables: one with the style and one without. For slight bonus credit, put the two tables side-by-side instead of on top of each other (this bonus part is really an HTML question, not a CSS question).



With class="inverse"

| Heading1 | Heading2 |
|---|---|
| Row1 Col1 Data | Row1 Col2 Data |
| Row2 Col1 Data | Row2 Col2 Data |
| Row3 Col1 Data | Row3 Col2 Data |

Without class="inverse"

| Heading1 | Heading2 |
|---|---|
| Row1 Col1 Data | Row1 Col2 Data |
| Row2 Col1 Data | Row2 Col2 Data |
| Row3 Col1 Data | Row3 Col2 Data |

4. Make some style sheet entries that will let you make zebra-striped tables, where every other row has a different background color. For bonus credit, style all the rows at once, without having to put "class" entries in any of the rows. (Hint: look at the positional selectors.)

| Year | Apples Sales | Orange Sales |
|---|---|---|
| 2002 | $123.34 | $456.78 |
| 2003 | $223.35 | $556.79 |
| 2004 | $323.36 | $656.79 |
| 2005 | $423.37 | $756.80 |
| 2006 | $523.38 | $856.81 |
| 2007 | $623.39 | $956.82 |
| 2008 | $1,234.56 | $45,567.89 |
| 2009 | $7.56 | $43.25 |
| 2010 | $92.86 | $86.92 |
| 2011 | $100.00 | $100.00 |
| 2012 | $999.99 | $222.22 |
| 2013 | $666.66 | $777.77 |
| 2014 | $888.88 | $555.55 |

# Lab Exercise 4: Getting Started

**1.** Try JavaScript interactively

- If you use Firefox, install Firebug from http://getfirebug.com/. Copy the simple test-page.html from the section's source code archive (javascript-getting-started), load it in Firefox, then launch Firebug by clicking on the Firebug icon or hitting F12. Click on the Console tab and try some interactive JavaScript commands. Customize the console as described in the notes if you wish.
- If you use Chrome, copy the simple test-page.html from the section's source code archive (javascript-getting-started), load it in Chrome, then launch the developer tools with Control-Shift-J. Click on the Console tab and try some interactive JavaScript commands. Most browsers, even Internet Explorer and Microsoft Edge have something similar now.

**2.** Define a variable x and give it a value of 5. Evaluate x and verify it shows the value.

**3.** Enter this function into the console:

```
function half(x) {
  return(x / 2);
}
```

Try to predict what you will get for half(x), half(4), and half(3). It is simple to predict what you will get for half(4), but, depending on your programming background, it might not be so easy to predict what you will get for half(x) and half(3). Call half(x), half(4), and half(3) and see if you were right.

**4.** Try to predict what you will get if you evaluate x in the console. Is it still 5, or is it 3? Try it and see.

**5.** Enter this function into the console:

```
function seven() {
  x = 7;
  return(x);
}
```

Call seven() in the console. Try to predict what you will get if you evaluate x in the console. Try it and see. How do you explain the surprising result?

**6.** Make a function called calculation that, given three numbers a, b, and c, returns (a + b)/c. Try it with a few normal values. Then, try to predict what you will get for calculation(1, 1, 0), calculation(-1, -1, 0), and calculation(1, -1, 0). Try those tests and see if you get what you expected.

**7.** Try the "more powerful practice" approach from the notes, where you load an HTML file that has a script tag pointing at a JavaScript file. Put a function definition in the JS file, load the HTML file, go to the console, and call the function. Then, add a second function definition to the JS file, reload the HTML file, return to the console, and call the new function.

**8.** Write a function called isEven that, given a number, returns true if the number is even and false if the number is odd. This would be simple if we had covered if statements or the ?: ternary operator, but we haven't, so you cannot use either one.

# Lab Exercise 5: Basic Syntax 1

For all of these exercises, use the "more powerful practice" approach from the first lecture, where you load an HTML file that has a script tag pointing at a JavaScript file, test the function calls in Firebug or Chrome, and reload the HTML page each time you change the function definitions.

**1.** Make a function that returns "even" or "odd" depending on the number passed to it.

```
parity(1); --> "odd"
parity(2); --> "even"
```

**2.** The notes showed a simple version of max that took exactly two arguments. Update this to take exactly three arguments. Note that the builtin version of Math.max takes any number of arguments, which is much better, but we don't know how to do variable arguments yet. And, of course, no using Math.max on this exercise.

```
max(1, 2, 3); --> 3
max(1, 3, 2); --> 3
max(3, 2, 1); --> 3
```

**3.** Copy the flipCoin function from the last set of exercises. Now, make a function that, given a number, flips a coin that many times and returns the number of heads.

```
numHeads(10); --> 4
numHeads(10); --> 6
numHeads(10); --> 6
```

**4.** Make a function that takes a number of flips and returns the fraction that were heads.

```
headsRatio(10); --> 0.7
headsRatio(10); --> 0.4
headsRatio(10000); --> 0.5023
headsRatio(10000000); --> 0.4999948
```

**5.** Make a function that takes a number and a short string, and returns the string concatenated that number of times.

```
padChars(5, "x"); --> "xxxxx"
padChars(7, "-"); --> "-------"
```

**6.** Write a function that returns the number of times you have to roll a die to get a 6.
(Minor hint: compare Math.random() to 5/6).

```
numRollsToGetSix(); --> 13
numRollsToGetSix(); --> 2
```

**7.** Update the HTML page so that each time you reload it, you randomly see either a "Have a GOOD day!" or "Have a BAD day!" message. If you know some CSS already, make the font big.

# Lab Exercise 6: Basic Syntax 2

The last two problems are quite a bit more difficult than the first three.

**1.** Create an array containing four random numbers. Use one-step array allocation. Print out the array by evaluating the array variable in Firebug.

```
var fourNums = ...;
fourNums; --> [0.871570877817405, 0.9107447521970577,
               0.743357509580703, 0.6571292972456975]
```

**2.** Create an array containing 100 random numbers. Use two-step array allocation. Print out the array.

```
var hundredNums = ...;
for(...) { ... }
hundredNums; --> [0.8742489161574934, 0.7147785711684753, 0.8062322101495641,
                  ...
                 0.41288219216760613, 0.5113443687277072]
```

**3.** Make a function that given an array of strings, where each string represents a number, returns an array of the corresponding numbers.

```
var strings = ["1.2", "2.3", "3.4"];
var nums = numberArray(strings);
nums; --> [1.2, 2.3, 3.4]
```

**4.** Write a function that, given a string, will return the longest token (consecutive string of characters) that contains neither an a nor a b.

```
longestToken("abab<b>cd</b>abab<b>efg</b>abab<b>hi</b>ab"); --> "efg"
longestToken("aba"); --> ""
```

**5.** Write a function that, given an array of strings, will compute the sum of the lengths of the words that do not contain a “q”. Do not use a loop or the forEach method, only array methods (filter, map, reduce) and string methods/properties (indexOf, length).

```
var test1 = ["stop", "quit", "exit"];
lengthOfNonQWords(test1); --> 8
var test2 = ["queen", "quit"];
lengthOfNonQWords(test2); --> 0
```

# Lab Exercise 7: Functional Programming: Basic Exercises

Even these supposedly basic exercises are a bit tricky if you have never seen functional programming.

**1.** Make a function called composedValue that takes two functions f1 and f2 and a value and returns f1(f2(value)), i.e., the first function called on the result of the second function called on the value.

```
function square(x) { return(x*x); }
function double(x) { return(x*2); }
composedValue(square, double, 5); --> 100 // I.e., square(double(5))
```

**2.** Make a function called compose that takes two functions f1 and f2 and returns a new function that, when called on a value, will return f1(f2(value)). Assume that f1 and f2 each take exactly one argument.

```
var f1 = compose(square, double);
f1(5); --> 100
f1(10); --> 400
var f2 = compose(double, square);
f2(5); --> 50
f2(10); --> 200
```

**3.** Make a function called "find" that takes an array and a test function, and returns the first element of the array that "passes" (returns non-false for) the test. Don't use map, filter, or reduce.

```
function isEven(num) { return(num%2 == 0); }
isEven(3) --> false
isEven(4) --> true
find([1, 3, 5, 4, 2], isEven); --> 4
```

**4.** Recent JavaScript versions added the "map" method of arrays, as we saw in the notes and used in the previous set of exercises. But, in earlier JavaScript versions, you had to write it yourself. Make a function called "map" that takes an array and a function, and returns a new array that is the result of calling the function on each element of the input array. Don't use map, filter, or reduce.

```
map([1, 2, 3, 4, 5], square); --> [1, 4, 9, 16, 25]
map([1, 4, 9, 16, 25], Math.sqrt); --> [1, 2, 3, 4, 5]
```

Hint: remember the push method of arrays.