CS 1100: Web Development: Client-Side Coding / Fall 2016
Lab 9: Loops and Arrays

Goal

- Practice working with loops (while and for)
- Practice working with arrays
- Revisit assignment statement, string concatenation, and conditional statements
- Continue exploring the DOM API

Work in Pairs DIFFERENTLY

This lab is a work-in-pairs exercise. BUT. You won't pair program. Instead, work together, but use 2 laptops. BOTH STUDENTS SHOULD SUBMIT THEIR WORK. Your code can be identical (although it doesn't have to be).

Setup

Create a new folder: lab9. Download the lab9.html and lab9.js files into this folder. Open files lab9.js in your favorite editor. Open the lab9.html in the browser. Open the JavaScript console.

REMINDER: DO NOT COPY AND PASTE CODE FROM THIS PDF: THIS MAY CAUSE CHARACTER ENCODING ISSUES AND YOUR CODE WILL NOT WORK. TYPE IT YOURSELF!

Activity 0: review assignment statements + string concatenation

0.1. Use the console to practice these statements:

```
> var a = "So long";
> var b = "and thanks";
> var c = "the fish!";
```

(there are no spaces around the words; you may have to provide for them in your code)

Now try to display the string "**So long and thanks for all the fish!**" using these 3 variables (**a, b, c**) and any string literals* you might want to add.

*A string literal is just a string of text - i.e. **"dolphins"** is a string literal (because it evaluates to its literal value). If we were to assign it to a variable, like this: **var x = "dolphins"**, then x would be a variable containing the string **"dolphins"**.

0.2. Type this statement:

```
> var a = "The answer is ";
```

Use variable **a** to display the string "**The answer is 42**" (hint: you can concatenate a string and a number - JavaScript will convert the number to the string for you!)

Activity 1: The while loop

We use the while loop when we do not know in advance the number of iterations.

2.1. For this step, use your lab9.js file; make sure not to delete the code provided - you will need it later. Write a loop which keeps asking the user for input. If the user types in "quit", the loop stops; in any other case, it keeps iterating.

Let's try this straightforward implementation. First, read this commented version. Make sure you understand it:

```
//initialize the boolean variable (the sentinel)
var stop = false;

//while the sentinel is true (i.e., not false), keep iterating
while (!stop) {
	//ask the user for their input and assign to a variable
	var input = prompt("type quit to exit loop");
	console.log("you typed " + input); //write this to the console
	//if the user typed "quit"
	if (input == "quit") {
		stop = true;//change the value of the sentinel
		console.log("loop stopping!");
	}
	//if the user typed anything BUT "quit", execute any statements you like
	else {
		console.log("executing stuff...");
	}
}
```

Now type in this code. Feel free to add to it any comments that help you understand the code.

```
var stop = false;

while (!stop) {
	var input = prompt("type quit to exit loop");
	console.log("you typed " + input);
	if (input == "quit") {
		stop = true;
		console.log("loop stopping!");
	}
	else {
		console.log("executing stuff...");
	}
}
```

Save it and refresh your file in the browser. Try to type in some input and watch the console report on what the loop is doing.

Once you are done, comment out all the code using a multiline comment block (slash-star and star-slash) like this:

```
/*
this is
commented out
*/
```

2.2. The while construct you've just used can be very useful. However, for our current task, we can do better!

Whenever you program, try to write your code so that it *reflects what you really want to do* as closely as possible. In our case, what we *really* want to tell the computer is this:

```
- keep doing this until I tell you to stop:
      - ask the user for input
      - if that input is "quit", exit the loop now.
      - otherwise, do whatever each iteration should do
```

Again, first, read this commented version and make sure you understand what it does:

```
//do this forever
while (true) {
      //ask the user for their input and assign to a variable
      var input = prompt("type quit to exit loop");
      console.log("you typed " + input);

      //if the user typed "quit", use the break statement to
      //   literally "break out" of the loop immediately
      if (input == "quit") {
            console.log("loop stopping!");
            break; //exit loop now!
      }
      //YOU DO NOT NEED THE ELSE STATEMENT:
      //  in all other cases execution will proceed to the next line anyway
      console.log("executing stuff...");
}
```

Now type in this code. Feel free to add to it any comments that help you understand the code.

```
while (true) {
      var input = prompt("type quit to exit loop");
      console.log("you typed " + input);
      if (input == "quit") {
            console.log("loop stopping!");
            break;
      }
      console.log("executing stuff...");
}
```

Compare it to your previous version. It should feel more intuitive.
Save it and refresh your file in the browser. Try to type in some input and watch the console report on what the loop is doing. It should be doing exactly the same things your previous code was doing.

Once you are done, comment out all the code using multiline comments.

Activity 2: arrays

An array is a data type that holds a collection of items. Before we move to the cool stuff, let's practice with the console.

Create an array of several strings (use any values you like):

```
> var suspects = ["tinker", "tailor", "soldier", "spy"];
```

Now your variable **suspects** contains an array. An array knows its length; try this:

```
> suspects.length
```

We can access the items in the array using their position index in square brackets. Arrays are zero-based, so we start from zero:

```
>suspects[0]   //will give us "tinker"
>suspects[1]   //will give us "tailor"
>suspects[2]   //will give us "soldier"
>suspects[3]   //will give us "spy"
```

In other words, the array **suspects** at position **0** contains the string "tinker".

We could use the items in the array individually in any way we like. For example, this -

```
var book = suspects[0] + " " + suspects[1] + " " + suspects[2] + " " + suspects[3]
```

- will result the variable **book** containing the string "Tinker Tailor Soldier Spy".

We can change the contents of the array:

`suspects[1] = "George Smiley";` would change the second item in the array **suspects** from "tailor" to "George Smiley"*

Activity 3: The for loop and the DOM

We use the for loop when we know the number of iterations in advance.

3.1. Try this simple loop in your console (remember to hold down the SHIFT key together with ENTER when typing multiline statements into the console):

```
for (var i=0; i<10; i++){
      console.log(i);
}
```

This code will simply print 10 numbers to the console: from 0 to 9..

Now try to modify the loop:
a) make it print numbers from 0 to 20
b) make it print numbers from 20 to 30
c) make it print numbers from 10 to 0
d) make it print even numbers from 0 to 20 *

*Hint: change the increment statement in the "header" part of the loop from i++ to something else. Recall that i++ is shorthand for i += 1, which is shorthand for i = i + 1

3.2. But who needs numbers when we have the DOM to play with!

Your lab9.html web page contains a table which looks boring. Let's spice it up by modifying its cells.

Again, take a look at the following code and read it carefully:

```
//use querySelectorAll("td") to get a collection of ALL elements that match the "td"
//selector
var cells = document.querySelectorAll("td");
//iterate over all elements in the collection
for (var i=0; i < cells.length; i++) {
      //get the i-th element in the collection and assign it to a variable
      var cell = cells[i]; //now variable cell contains the i-th TD element

      //Option 1: change the cell's text
      //cell.innerText = "new text"

      //Option 2: change the cell's text to its number
      //cell.innerText = "cell number " + i;

      //Option 3: change the cell's background color
      //cell.style.backgroundColor = "lightblue";

      //Option 4: call the provided function to change the cell's background color
      //to something completely different!
      //cell.style.backgroundColor = getRandomColor();
}
```

You are provided with four options to modify the cell - do you see what each option does?

Now type in this code. Feel free to add to it any comments that help you understand the code.

```
var cells = document.querySelectorAll("td");
for (var i=0; i < cells.length; i++) {
      var cell = cells[i];
      //cell.innerText = "new text"
      //cell.innerText = "cell number " + i;
      //cell.style.backgroundColor = "lightblue";
      //cell.style.backgroundColor = getRandomColor();
}
```

Uncomment each option, one at a time, save the file and refresh your browser. See the change? Of course you can do all that in HTML and CSS ...until you get to option 4 - which you CANNOT implement in HTML and CSS. When you get to option 4, refresh the browser several times.

Want to experiment with color? Look inside the getRandomColor() function: you modify it by changing 255 to something else (between 0 and 255) - that would create a very different color! Want shades of red? change 255 to 0 for both green and blue!

Submit your work

Save your lab9 folder as a zip file and submit the zip file to eLearning:
https://bb9.uni.edu > log in with CatID > our course > Course Content > Labs > Lab 9

Bonus Activity

Do you have any time left? Here's some bonus fun stuff. Who needs table cells when we can access ANY element on the page! And who needs to use a toy example when we can explore ANY webpage in the world?

Try this page for starters: http://www.cs.uni.edu/~wallingf/blog/archives/cat_6.html

This page is an archive of blog posts going back to 2004, written by a good friend of mine. My friend often links to other resources which are often worth visiting.

Let's extract all these links from this page and display them as a list - that's exactly what a search engine's crawler (bot, spider, etc.) would do; or, in our case, a very curious user! A list of links will make it easier to visit them. This will also show us something we cannot immediately see from the page: what websites my friend tends to link to!

But first, let's practice with our lab9.html page.

To do this, we will slightly amend our previous code:

1. we will select all "a" elements
2. we will access their "href" attribute values
3. we will use those values to build up this simple HTML:
   `<p><a href=" foo">foo</a>` , where **foo** is the extracted URL

Again, take a look at the following code and read it very carefully:

```
//use querySelectorAll("a") to get a collection of ALL elements that match the "a" selector
var links = document.querySelectorAll("a");
//iterate over all elements in the collection
for (var i=0; i < links.length; i++) {
      //get the i-th item in the collection and assign it to a variable
      var link = links[i];
      //get the value of the "href" attribute of that item and assign it to a variable
      var href = link.getAttribute("href");
      //build up a string which displays this value as a hyperlink
      var html ="<p><a href='" + href + "'>" + href + "</a>";
      //write it out to the document
      document.write(html);
}
```

Now type in this code. Do not add any comments: we will be copying this code into the console, so we want it to be compact.

```
var links = document.querySelectorAll("a");
for (var i=0; i < links.length; i++) {
      var link = links[i];
      var href = link.getAttribute("href");
      var html ="<p><a href='" + href + "'>" + href + "</a>";
      document.write(html);
}
```

Save your file and refresh your browser. See the list of links at the bottom of the page?

Now go to my friends page, http://www.cs.uni.edu/~wallingf/blog/archives/cat_6.html , open the console, paste this code and hit ENTER.

You should see a long list of links. You can save the generated HTML by saving the web page from your browser (right-click > save as).

Curious about pages? Go to any page you like and collect the links! Tyy nytimes.com, washingtonpost.com, nypost.com, etc... Think of it this way: ***linking to a resource***, to a certain extent, ***implies endorsing that resource***: Imagine the analysis you could do here: we can look at who endorses who, and we can compare sites using their linking practices. In fact, we can tell a lot about a site by the sites it links to!

You have just build a link extractor. Congratulations!