CSCI 341 : HANDS ON

UNIT: Functions

Activities

- Explore **functions** and **scope**.
- Function: A Battle Function - (add text to page element)
- Parameters: Each battle function should accept a page element id, two message fragments, and 1 number
- Returns: none required

Activity One

A JavaScript program is composed of reusable fundamental building blocks called functions. Functions are powerful constructs, which provide the user with the ability to name blocks of code and then instead of having to rewrite the entire block of code, the user can simply use the block by calling its name.

Functions are miniature work-horses, burying detail at a lower level to keep things simple at the calling level. Instead of creating a long program that lists line after line of instructions for getting up and to class on time, imagine a JavaScript program that says: get up, get ready, go to class. Each of those instructions expand to lots of lines of code underneath them when called, but they don't clutter up the main program.

To declare (create) a function, simply type the reserved keyword **function**, followed by the function's name. Then, inside of curly brackets you write out the reusable block of code. Here is an example of a function:

```
/*****
Purpose: displays positive reinforcement
Parameters: N/A
Return: N/A
*****/
function alertpositive() {
  alert("You get an A for Argh!");
  alert("Well done matey.");
}
```

The function can then be called by simply typing the function's name, as such:

```
alertpositive();
```

Notice the **block comment** just above the function. You should comment each function you create (except for the main function). In doing so, be sure to identify the name of the function, describe the function, identify the parameters it takes (if any) and the value it returns (if any). What are "parameters" and "return values" you ask?

Activity Two

Not all functions can perform their jobs by themselves. Some of them require information to be passed in to them. The information passed in goes in the parentheses, separated by commas. This passed information is known as parameters. Here is an example of a function with parameters:

```
/*****
Purpose: displays positive reinforcement
Parameters: strUserName – the user's first name, charScore – the user's letter grade
Return: N/A
*****/
function alertpositive(strUserName, chrScore) {
 alert("You get a " + chrScore + " Argh that better be an A!");
 alert("Well done " + strUserName);
}
```

Data created inside the function will not be in scope from the main program. Scope is the area in which the data may be accessed. If you want to have access to data created within a function, you can pass it back outside the function to it's calling program. You do this with the reserved keyword **return**. The following is an example of a function which returns a value.

```
/*****
Purpose: displays positive reinforcement
Parameters: strUserName – the user's first name, charScore – the user's letter grade
Return: string with user's name and new rank
*****/
function alertpositive(strUserName, chrScore) {
 alert("You get a " + chrScore + " Argh that better be an A!");
 alert("Well done " + strUserName);
 return "First mate " + strUserName;
}
strUserName = alertpositive(strUserName, "B");
```

For more function examples, see pages 88 – 97 of the textbook.

It turns out that where a variable exists determines its visibility to the rest of the program. If you declare a variable INSIDE of a function, it doesn't exist outside of the function (unless passed out with a return). If you declare a variable at the program level (and not the function level), however, any functions inside of that program have visibility (and access) to it. Variables defined at the program level are called Global variables. Variables declared within a function are called Local variables. If you aren't careful, this could cause some problems. You could mean to modify a variable only within a local function, but wind up changing it everywhere. For good defensive programming, use global variables sparingly.

For more scope examples, see pages 98 - 99 of the textbook.

For live demonstrations of functions and scope, follow the links **at the top of this page**.

Lab Instructions

Our pirate ship is going into battle with an enemy ship! In this lab, you will write a function to represent the captain's orders. The function will output text that describes the ship's activities in a concatenated string that includes a number that is passed to the function as a parameter. For example, your function accepts two string fragments and a number as parameters. Within that function, there is code that could output, "The quartermaster steers the ship in route to intercept the enemy, passing within a mere 5 feet." The part of the output that appears before the number would be the first string parameter, the number that is inserted into the string will come from the numeric parameter, and the word "feet" would come from the second string parameter that is passed to the function. Call your function at least 5 times to represent the captain's commands. Instructions are detailed in the Canvas rubric and in the list below.

1. Create a function that accepts 4 parameters: the <div id> for output, and a number, and two strings.
2. The function should concatenate the first string, the number, and the second string into a combined string of original text and add that text to the page.
3. The function should output to the page using textContent - not alert!
4. Comment the function with its identity, parameters, and return value (if any).
5. Call the function at least five times.
6. Follow all published **lab requirements** and important procedures listed below.

You may want to re-read the multi-writes assignment for examples on writing to a page element.

Important Procedures for All Labs

Here are some general notes for perfection that you should follow for every assignment:

1. Please produce all web content to HTML5 standards.
2. Please **validate** all your files.
3. Be sure to update the header block comments for each file.
4. Be sure to check your browser's console / developer tools for error free code.
5. Test your code in Chrome and Edge at a minimum.
6. Use only your own original code for all labs.
7. Be sure to put your CSS and JavaScript in a separate files from your html.
8. Be sure to read through the lab rubric in Canvas.
9. Submit your lab in Canvas for grading.

Holler if you have any questions!

Mission Accomplished!

Now that we've tasted the bottle, its time to steal the keg. Clear the deck for action. Here comes the next assignment broadside!