Part 1. JavaScript Loops and Variables

PLEASE READ:

In this assignment, we will be discussing two types of loops: **For loop** and **While loop**. We are going to be using loops to print a sentence a given number of times to demonstrate the differences between a JavaScript **For** loop and **While** loop.

Recall that for an **if/else statement**, there is a set of instructions that gets executed if the condition is true, and a set of instructions that gets executed if the condition is false. The same thing happens in JavaScript **For loops** and **While loops** – the set of instructions in the loop body (what’s inside the curly braces) will repeatedly execute until the loop condition is false. When that condition is false, the loop stops executing. Loops in JavaScript function in the same way that they have in Python.

In practice, a **For** loop and a **While** loop can potentially do the same things (as with this assignment), however a **For** loop is generally for cases where we want to execute something a certain number of times (i.e. we know exactly how many times we want to repeat it), whereas a **While** loop is used in all other cases (e.g. “While” it’s raining, do something; “While” you’re hungry, do something).

A **While** loop will execute the loop body as long as the While loop condition is met. Therefore, the flow chart for a **While** loop is:
A **For** loop is similar to a **While** loop, but in a more specific manner. **For** loops are generally used to repeat code a certain number of times, e.g. for counting. The three parts of a **For** loop condition statement will be discussed later in this assignment, but is illustrated in the following flow chart:

![For Loop Diagram](image)

As you can see, the main difference between a **For** loop and a **While** loop is the use of a counter. Counters count how many times a body of code is executed. The **For** loop utilizes a counter to keep track of how many times the instructions execute; the counter is declared and incremented as part of the conditional code and is updated after the body of the loop is executed. A **While** loop also uses a counter, but the counter is declared before the body of the loop is executed and updated after each loop.

**Assignment:**

**Step 1:**
In your CSE3 folder, create a new folder named HW7. Then create and save a new Notepad++ file called **Loops.html** in the HW7 folder.

**Step 2:**
Write out the basic HTML skeleton (if you don't remember what this looks like refer back to one of the earlier labs on HTML). In the head tags, set the **title** to “JavaScript Loops” using the title tags.

**Step 3:**
In the `<body>` tags, create `<center>` tags. Include a heading 1 at the top of your webpage that says, “While and For Loops”. Under `<h1>` tags, create `<div>` tags. The `<div>` tag defines a section in an HTML element and it is often used as a container. Give an id attribute to `<div>` tags with a value of “container”. We are going to use this in JavaScript later.
**Step 4:**
Let’s create `<script>` tags. Place these above the `</body>` tag:

```html
<script type = "text/javascript"> </script>
```

**Step 5:**
Now inside your script tags, let’s create two variables. The first is a variable that tells us how many times we should print the sentence we are going to create (“numLoop”), and a second variable that will tell us how many times we have already printed the sentence (“numPrinted”).

```javascript
<script type="text/javascript">
  var numLoop = 15;
  var numPrinted = 0;
</script>
```

**While Loop**

Now that we have our variables, let’s create our loop. The first kind of loop we are going to use is a **While** loop. A **While** loop can be thought of as being similar to an **if statement**, only it repeats the code inside the brackets until the conditional statement (inside the while parentheses) is false. **While** loops in JavaScript operate the same as **While** loops you have seen in Python. In our case, we want to make a loop that keeps executing the loop body until the conditional statement is false, meaning that `numLoop == numPrinted` is reached.

**Step 1**
First let’s outline the **While** loop:

```javascript
while(condition) {
  // Code to execute
}
```

**Step 2**
Let’s add the condition statement inside the parentheses. Since we want to print the sentence `numLoop` times, we need to use our `numPrinted` variable to check if we have printed the sentence `numLoop` times:

```java
while(numPrinted < numLoop) {
    // Code to execute
}
```

This loop is saying:

“While the number of times we have printed is less than the number of times we want to print it, keep printing it.”

**Step 3**

At some point we need `numPrinted` to be equal to `numLoop` so that the condition is rendered false, allowing us to break out of the loop. To achieve this, we increment `numPrinted` each time we go through the loop. Add the following line before our closing curly bracket.

```javascript
numPrinted++;  // This is our loop counter, and it will keep track of the number of times the loop has executed.
```

**NOTE:** When we use the ‘++’ operator after a variable it means that we are incrementing the variable by 1. `numPrinted++` is simply shorthand for saying `numPrinted = numPrinted + 1`.

Once we added the conditional statement and the loop counter, it should look like this:

```java
while(numPrinted < numLoop) {
    // Code to execute
    numPrinted++;  // This is our loop counter, and it will keep track of the number of times the loop has executed.
}
```

If you run this in Chrome, you won’t see anything but your heading. That is because we haven’t added anything in the loop.

**Step 4**

Let’s add code inside our loop. Remember how we added an id attribute to the `<div>` tags in the HTML code? We are going to use `document.getElementById()` function and `innerHTML` property to add a new sentence to the container. The `getElementById()` returns the element that has the ID attribute with the specified value. We are going to pass in the id value we assigned to the `<div>` tags in Step(3) under Assignment.
Let’s type the following code below the loop counter incrementation:

```javascript
document.getElementById("container");
```

And this will return an element with an id value of “container.”

**Step 5**

Now that we have our element, let’s use the `innerHTML` property to modify the HTML content of the element. We can append a new value to the HTML content by using the `+=` assignment. The right-hand side value of the “+=” assignment will be appended to the existing value of the element. Let’s modify the existing element like the following:

```javascript
while(numPrinted < numLoop) {
    // Code to execute
    numPrinted++;
    document.getElementById("container").innerHTML += "While loop #: " + numPrinted + "<br />";
}
```

If you run your code in Chrome, you should see 15 lines of “Sentence to write”

**NOTE:** The ‘+=’ assignment means that we are adding a new value to an existing value. For instance, $x += y$ is same as $x = x + y$.

**Step 6**

Now we need to put a sentence inside the parenthesis where it says, “Sentence to write.”

Use the sentence:

“While loop #: $X$”

Where $X$ is the number of times the loop has been printed (remember our variable?)

To accomplish this, we are going to use what is called string concatenation.

**Step 7**

Since we know that `numPrinted` counts how many times our `sentence` has been printed, if we substitute it for $X$ it should give us the right result. We do this concatenating using the ‘+’ operator:

```javascript
document.getElementById("container").innerHTML += "While loop #: " + numPrinted + "<br "/>";
```

**NOTE 1:** Only the words and the break tag should be inside the quotation marks (strings)

**NOTE 2:** Notice that there is a space after the colon and before the first closing quotation mark. This is so our sentence is printed: “While loop #: $X$” and not “While loop #:X”.

Once this is completed, the webpage output should look like the following:
Step 8

Right now, our print statements seem a bit boring in black font.

Let’s fix that by having them print in powderBlue. We can do this by adding an extra line of code and altering our document.getElementById("container").innerHTML function:

```javascript
while(numPrinted < numLoop)
{
    // Code to execute
    numPrinted++; 

    var text = "While loop #: " + numPrinted + "<br />
    document.getElementById("container").innerHTML += text.fontcolor("powderBlue");
}
```

What this code does is assign a piece of text we want to print to a new variable named text. Variable text will then call the method fontcolor() so that the string text now has a specified font color (in this example it is powderBlue).

The document.getElementById("container").innerHTML method simply outputs that newly colored statement onto the webpage. The output should now look like the following:
Step 9
Let’s make our print statements more dynamic by changing the font colors based on values.

Now, imagine we want our font to be a specific color based on whether the value is divisible by a certain number. This kind of check can be achieved with the use of the modulus operator. You may remember the modulus operator from Python, but in case you forgot it is represented “%”.

You can think of the modulus as essentially dividing two numbers. But instead of returning the result (known as the quotient), it will return the remainder.

Example:
7 % 4 will return 3
21%5 will return 1
16%8 will return 0

Therefore, if X%Y returns 0, then we can conclude that X is divisible by Y.

For this assignment, we want statements that contain values divisible by 4 to be printed in dodgerBlue, and statements that contain values divisible by 2 to be printed in thistle. Otherwise, if the value is neither divisible by 4 nor 2, then the statement will be printed in powderBlue.

Use the template below to add an if/else if statement inside of the loop to perform this check. You will need to determine what goes in the spots designated as “???” as well as determining what goes in the body of the if/else if statements.
NOTE: The order you check if a number is divisible by 4 or by two matters… Are numbers that are divisible by 2 also divisible by 4? What about the reverse of that statement? This is a popular problem often referred to as “fizz buzz”. You can read more about it here” https://en.wikipedia.org/wiki/Fizz_buzz

NOTE: Notice in the code above that else if is spelled out in full. In Python, else if statements can be typed as “elif”, but in JavaScript it MUST be typed out exactly as “else if” (including the space!).

Your output should now look like the following:

**While and For Loops**

<table>
<thead>
<tr>
<th>While loop #: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>While loop #: 2</td>
</tr>
<tr>
<td>While loop #: 3</td>
</tr>
<tr>
<td><strong>While loop #: 4</strong></td>
</tr>
<tr>
<td>While loop #: 5</td>
</tr>
<tr>
<td>While loop #: 6</td>
</tr>
<tr>
<td>While loop #: 7</td>
</tr>
<tr>
<td><strong>While loop #: 8</strong></td>
</tr>
<tr>
<td>While loop #: 9</td>
</tr>
<tr>
<td>While loop #: 10</td>
</tr>
<tr>
<td>While loop #: 11</td>
</tr>
<tr>
<td><strong>While loop #: 12</strong></td>
</tr>
<tr>
<td>While loop #: 13</td>
</tr>
<tr>
<td>While loop #: 14</td>
</tr>
<tr>
<td>While loop #: 15</td>
</tr>
</tbody>
</table>

**For Loop**

The next type of loop we will be using is called a **For Loop**. In Python, we used a For Loop to iterate through a list and printed out all elements contained in that list. **For Loops** in JavaScript does not have
that same functionality, but instead act more like While Loops. The difference is that the incrementing of the loop counter in a For Loop is done for you instead of having `numPrinted++` inside the loop:

```java
for ((1) declare counter; (2) conditional statement; (3) increment/decrement counter)
{
    // Loop body
}
```

Note the three main parts. Normally in a While Loop, we increment the counter in the body of the loop. In the above example, notice that the counter is declared and incremented within the set of parentheses.

**Step 1**
To emulate our While loop, let’s outline the for loop first. Add the following after the While loop:

```javascript
for (numPrinted = 15; numPrinted > 0; numPrinted-- )
{
    // Code to Execute
}
```

What this For loop is doing step by step in the order listed:
1. Set `numPrinted` to 15
2. Check to see if `numPrinted` is greater than 0
3. If `numPrinted > 0` is true, execute code inside {}
4. Decrement `numLoop` and repeat starting at step 2.

**Step 2**
Let’s add a line in the brackets to emulate our While loop.

```javascript
for (numPrinted = 15; numPrinted > 0; numPrinted-- )
{
    document.getElementById("container").innerHTML += "Sentence to write \<br \\
";
    // Code to Execute
}
```

Once you add this and run the program, you will see 15 lines of “sentence to write” followed by your While loop output.

**Step 3**
Now modify our sentence so that the correct number is printed. To do this we can simply replace the “Sentence to write \<br \\
” with the variable text:
Save your file and launch it on a webpage. What do you notice? The For Loop printed the sentence 15 times, starting the count at 15. Therefore, it prints between “15” and “1” inclusively.

Your output should now look like the following: (After “While loop #: 15”)

```
For loop #: 15
For loop #: 14
For loop #: 13
For loop #: 12
For loop #: 11
For loop #: 10
For loop #: 9
For loop #: 8
For loop #: 7
For loop #: 6
For loop #: 5
For loop #: 4
For loop #: 3
For loop #: 2
For loop #: 1
```

**Step 4**

For the next part, we want to do the same thing as in the While Loop, printing statements and their values in corresponding colors. We want to add if/else if statements to determine the font color of our text based on their values much like we did in our While Loop. The same restrictions will apply: values divisible by 4 should have dodgerBlue text, values divisible by 2 should have thistle text, and all other values should be powderBlue. You can re-use the same code structure that you had for While loop.

If implemented correctly, your FINAL output should look exactly like the following:
While and For Loops

NOTE: Common problems to pay attention to:
1. Verify that the numbers are counting from 1-15, inclusive for the While loop.
2. Verify that the numbers are counting from 15-1, inclusive for the For loop.
3. Verify that there is only one of each number, (i.e. 5 doesn’t show up twice).
4. Verify that all numbers correspond to their correct color assignment.

There must be no errors to receive credit for this assignment.

Part 2. JavaScript Function

For the second part, we are going to create a simple function that calculates a square number of given input. We are going to ask users for input and use our function to calculate the square number. If you need to review JavaScript variables and functions, please review Lab #7.

NOTE: Do not copy and paste code from the lab-write up directly. Some quotation marks can cause an error if you copy and paste them directly.

Assignment:

Step 1:
Create and save a new Notepad++ file called Function.html in the HW7 folder.

Step 2:
Write out the basic HTML skeleton (if you don't remember what this looks like refer back to one of the earlier labs on HTML). In the head tags, set the title to “JavaScript Function” using the title tags.

**Step 3:**
In the `<body>` tags, create `<center>` tags. Include a heading 1 at the top of your webpage that says, “Square Number”. Under `<h1>` tags, create `<div>` tags. The `<div>` tag defines a section in an HTML element and it is often used as a container. Give an id attribute to `<div>` tags with a value of “result”. We are going to use this in JavaScript later.

**Step 4:**
Let’s create `<script>` tags. Place these above the `</body>` tag:

```html
<script type = "text/javascript"> </script>
```

**Step 5:**
Let’s define variables and ask users for their input.

I. Create 2 variables: `userInput`, `squareResult`;
   a. The `userInput` variable will be used to store the user input, which should be integer.
   b. The `squareResult` variable will store the square number of the `userInput` variable.

II. Use the prompt function to ask for the user input and save the response to the `userInput` variable.

   ```javascript
   userInput = prompt("Enter an integer: ");
   ```

Run your program to test whether the prompt function is working properly.

**Step 6:**
Now that we have user input, let’s define our function. Name the function “square” and write one parameter named “number.”

Here is an example of function declaration: (HINT: From Lab#7)

```javascript
function displayName (name) {
    var greetingMessage = “My name is “ + name;
    return greetingMessage;
}
```

Your function declaration should look something like this:
Step 7:
Inside our function, we are going to calculate the square number of the passed-in parameter `number`. Inside our function, define a new variable named `squareNumber`. Let’s save the calculated value into the `squareNumber` variable. A square number is the product of some integer with itself, so we are going to multiply the `number` variable by itself and save it to the `squareNumber` variable.

```javascript
function square(number){
    var squareNumber = number * number;
}
```

Step 8:
Let’s write a return statement right below the multiplication so we can use the calculated variable outside of our function. Refer to the example above for the syntax. Write a return statement that returns the `squareNumber` variable.

Step 9:
Now we have our function, let’s call it! Below our function declaration, call the `square` function and pass in the `userInput` variable as a parameter. And save the returned value to the `squareResult` variable.

`squareResult = square(_FILL IN THE BLANK_);`

HINT: Review Lab#7 Part(E) if you are having trouble.

Step 10:
Now that we have the `squareResult` ready, let’s display it to our program. Like how we did Part1, we are going to use the `document.getElementById()` function and the `innerHTML` property to display the result.

I. Select the “result” HTML element by passing it to `document.getElementById()`
II. Use the `innerHTML` property and change the HTML content of the result HTML element.
III. On the right side of the equal sign, type “Square Number of `<b>`” + `userInput` + “`</b>` is `<b>`” + `squareResult` + “`</b>`.”

```javascript
document.getElementById("result").innerHTML = "Square Number of `<b>`" + userInput + "`</b>` is `<b>`" + squareResult + "`</b>`.
```
Step 11:
Test your program by launching it in Chrome. Enter various inputs to test the result.

Here are some examples:

When you enter 5:

Square Number

Square Number of 5 is 25.

When you enter 8:

Square Number

Square Number of 8 is 64.

Putting it online

Step 1:
Modify your CSE3Page.html to include a link to your Loops.html and Function.html page.

Step 2:
Put everything online and get checked off.

Checkoff: Go to your homepage via the class webpage and demonstrate to the TA/Tutor that your Loops.html and Function.html are complete. Be prepared to present and explain your code for Loops.html and run program with different inputs for Functions.html

REMEMBER to drag your HW7 folder into the WHITESPACE of your public_html/CSE3 folder!