JavaScript for App Development

By Mark Lassoff, Founder Framework Television

Section Five

Loops.

Loops allow us to repeat a section of code again and again until some predetermined condition is met. Many of the procedures we execute when coding include loops. Imagine simulating dealing a deck of cards, or bowling or even driving a car.

To some degree, all of these processes have loops.

In this section you'll learn the basics of creating loops with the JavaScript language. We'll cover the three fundamental loop types in JavaScript: While Loops, Do…While Loops and For Loops.

Section 5 Goals

In this section of the course your goals are:

- ☐ Understand How to Construct a While Loop
- ☐ Create a Do…While Loop
- ☐ Understand the important fundamental difference between While loops and Do…While Loops
- ☐ Use the compact For Loop

Watch This: Section 5 Video

As always your course videos are available on YouTube, Roku and other locations. However, only those officially enrolled have access to this course

guide, are able to submit assignments, work with the instructor, and get this guide.

Watch this section video at: https://www.youtube.com/watch?v=8KoitM375vY

Creating the Fundamental While Loop

The `while` loop continues to execute a block of code while some condition is true. In the most generic form the loop looks like this:

```
while(true)
{
        ...Do this stuff

}
```

In the parenthesis after the while statement goes a continuation condition that the statement evaluates, such as `x <100`. While x is less than 100 the loop will continue to iterate.

Let's take a look at a fully coded `while` loop.

```
<div id="output"></div>
<script>
  var x = 0;
  var out = "";
  while(x < 11)
    {
      out += x;
      out += "<br/>";
      x++;
    }
  document.getElementById('output').innerHTML = out;
</script>
```

Before we enter the loop, we create two variables. Variable `x` will be used as the loop counter. Variable out will hold the output. Notice that `x` is initialized before the while statement.

The loop continues to iterate while `x < 11` is true. First time through the loop x (which contains a value of 0) is added to the output. Next, a break is added to the output. Then x is incremented by one before the loop is run again. On the second trip through the loop, the value of x is 1. 1 is also less than eleven so the loop continues.

Do This

The best way to learn loops is to create them. Create loops that do the following:

1. Create a loop that starts counting at 100 and counts down to 0, displaying each number.
2. Create a loop that counts to 0 to 1000, counting by 10's and display each number. Your output should look something like this:
   10
   20
   30
   40
   …

The Do…While Loop

The `do...while` loop is the while loop turned upside down. It is functionally the same as the `while` loop. It's structure does lend it self to one major difference in application (more on this in a bit.)

The do…while loop is structured to have the continuation condition at the end of the loop. Let's look at the generic structure of a `do...while` loop.

```
do
{

        … all this stuff

} while (true)
```

Again in the `do...while` loop the loop will continue iterating while the continuation condition is true.

You might wonder why we have these two relatively similar structures: the `while` loop and the `do...while` loop. **This is what you need to remember:** Use the `do...while` loop when you need to guarantee that the loop will iterate at least once *even if the the continuation condition is not initially true*.

If you use the `while` loop, the loop will not iterate if the initial condition is found to be **false**.

With that understood, let's take a look at a full coded `do...while` loop.

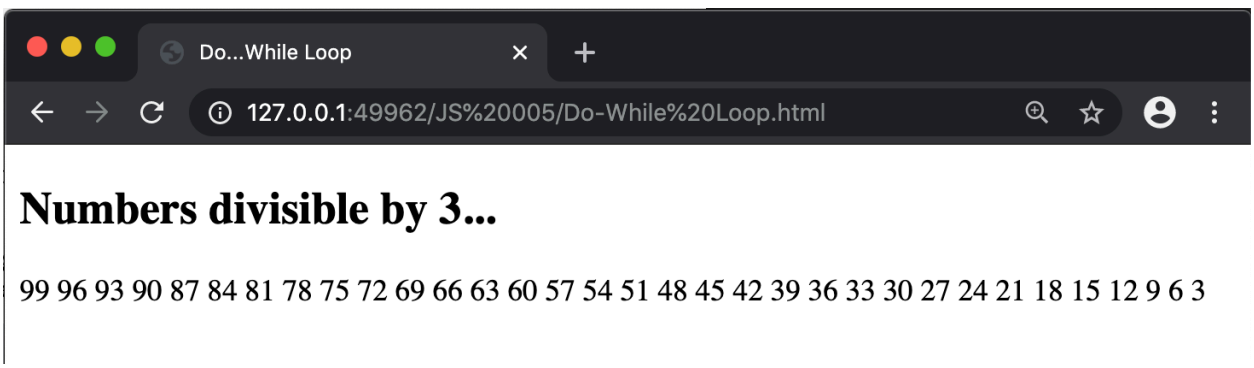
```
<h2>Numbers divisible by 3...</h2>
<div id="output"></div>

<script>
var x = 100;
do
{
   if(x % 3 ==0)
      {
         document.getElementById('output').innerHTML += " " + x;
      }
   x--;
}while(x>0);

</script>
```

Trace the execution of this script carefully. When we start the script the value of `x` is 100. Inside the loop we determine if the value of `x` is evenly divisible by three. If it is we append that value to the output. Then we reduce `x` by one using the decrement operator.

After the decrement, we run our test and determine if x is greater than zero. Once `x` reaches zero, we'll exit the `do...while` loop. If you have keyed in the loop above correctly, the output should look something like this:



Do This: Modify the Loops

Change the `do...while` loop above and make the continuation condition initially false. (For example if initially `x = -1` then the test `x > 0` will fail.) Run the loop. What happens? Do you understand why? Change the initial value of `x` to 99. What happens now? Would this be different with a `while` loop? How and why?

The For Loops (My Favorite!)

The `for` loops is the one I use most frequently. The `for` loops is essentially a loop shorthand that combines all of the components of a loop into a single line of code. The generic format of a for loop is:

```
for(initialization, continuation condition, counter)
{
   ...do this stuff
}
```

The key to understanding how the `for` loop works is understanding the three components. *Initialization* sets up the loop counter. For example if our counter stated at *zero initialization* might look like: `i=0`. The loop will continue iterating while the continuation condition is true. The continuation condition might be `i<100`. The counter indicates how will change the value of the counter variable as the loop executes. For example, `x--;` might be our counter.

Let's take a look at a fully coded `for` loop:

```
<div id="output"></div>
<script>
   for(var i=0; i < 25; i=i+5)
      {
         document.getElementById("output").innerHTML += i + " ";
      }
</script>
```

The `for` loop above will output as follows:

```
0,5,10,20
```

Nice and compact, right?

We start by initializing the counter `i` at 0. The loop continues iterating while `i < 25`, and each time we iterate through the loop we increase the value of `i` by 5.

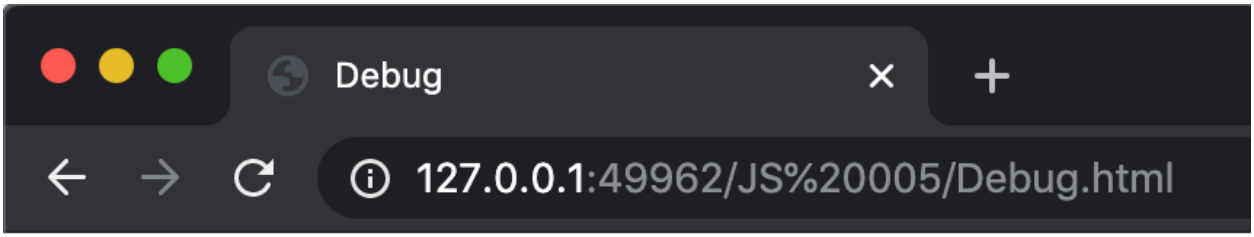
Do This: Debugging

```
<div id="output"></div>
<script>
   var from = prompt("Where do you want to start counting?");
   var stop = parseInt(
                                          prompt("Where do you want to
stop counting?"));

   for(var i = from; i < stop + 1; i++)
   {
      document.getElementById('output').innerHTML += i +
"<br/>";
   }

</script>
```

If you run this code and enter the numbers 10 and 20 it appears to work fine.



10
11
12
13
14
15
16
17
18
19
20

But try enter the values 20 an 1. You will find that the code is only designed to count forwards. Debug the code so it can count forwards or backwards. Good luck!

Submit This: Lab Exercise

Section A: Using a loop or loops have the following patterns output on the web browser. (You MUST use a loop or loops to produce these. You can't simply output the hard coed pattern.)

Pattern #1

```
*
**
***
****
*****
******
*******
********
*********
**********
*********
********
*******
******
*****
****
***
**
*
```

Pattern #2

```
*-*-*-*-*-*-
*-*-*-*-*-*-
*-*-*-*-*-*-
*-*-*-*-*-*-
```

Section B: Prompt the user for a number of rows and a number of columns. Using the * output the * character in the row and column pattern requested by the user. If the user requests 2 rows and 2 columns you should output:

```
**
**
```

Please save your file in the following format to insure proper credit:

`LastName_Exercise5A.html and LastName_Exercise5B.html.`

Remember every exercise must be submitted in order for you to earn certification. Once you've completed this exercise, you're ready to move on to Section 6.