Riverside JS Workshop

JavaScript Skills Ladder, Lesson 1.4

JavaScript Loops and Conditionals

Lesson Objectives

When you complete this skills ladder lesson, you will be able to:

- *use while loops and for loops in JavaScript.*
- *combine looping statements and conditional statements.*
- *use loops to generate HTML.*

A loop is a section of code you want to execute a specific number of times or while a specified condition is true. To help visualize this, let's consider a real life example.

Suppose you want to eat some M&Ms, and you have a bowl of them on your desk. To eat an M&M, you take one from the bowl and eat it. You want another one, so you take another M&M. You continue taking M&Ms from the bowl until they are all gone. You are *looping*: repeating the process of taking M&Ms from the bowl until there are none left.

You can write programs that loop with JavaScript by using a *while* loop or a *for* loop. Let's take a closer look at both of these kinds of loops.

The While Loop

A *while* loop executes the same section of code while a specified condition is true. So, let's write a program to eat M&Ms *while* there are M&Ms left.

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> The While Loop </title>
  <meta charset="utf-8">
  <script>
    var mms = 5;
    while (mms > 0) {
       console.log("Eat an M&M");
       mms--;
    }
    console.log("All out of M&Ms");
  </script>
```

```
</head>
<body>
</body>
</html>
```

Save it in your work folder as *mandms.html*, and load the page in your browser. If you open the JavaScript console you should see 5 messages that say "Eat an M&M" and one message that says "All out of M&Ms". (And, just as a reminder, if you want to use `alert` instead of `console.log`, you can!) Let's look at the script:

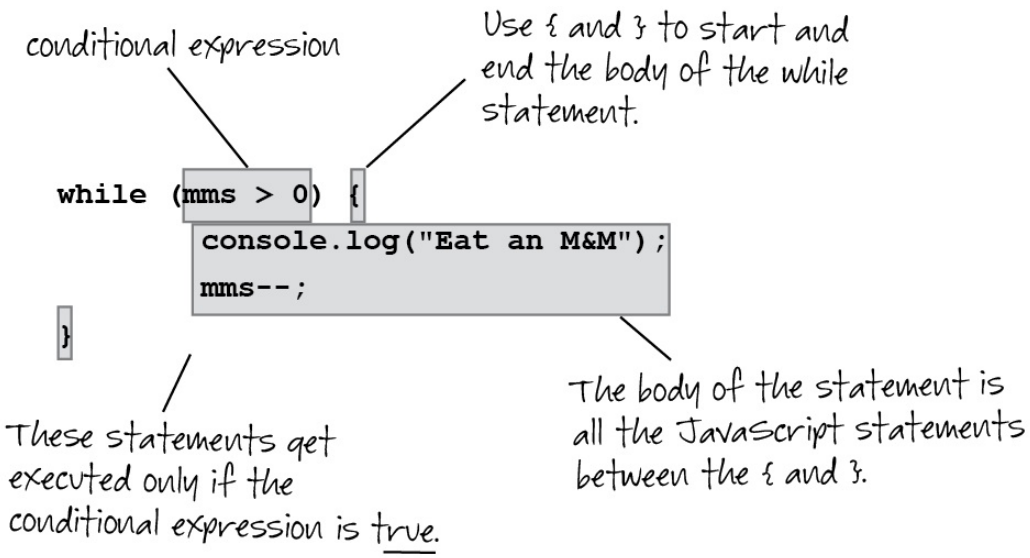
OBSERVE:

```
<script>
  var mms = 5;
  while (mms > 0) {
    console.log("Eat an M&M");
    mms--;
  }
  console.log("All out of M&Ms");
</script>
```

Notice that we *initialize* the value of the variable `mms` to 5 before the loop starts. This is an important step because `mms > 0` is used in the conditional test. If the condition is true, we keep looping and execute the code within the {braces}.

Also notice that we decrement the value of `mms` each time through the loop so the value changes. The first time through the loop, `mms` is 5, the second 4, and so on. The last time through the loop, `mms` is 1 and so this line of code decrements its value to 0. When `mms` is equal to 0, the loop terminates because the value is no longer greater than 0.

Here's a review of the different parts of the while statement:



Try experimenting with the example above by using a different initial value, and a different test. For instance, what happens if you write instead:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> The While Loop </title>
  <meta charset="utf-8">
  <script>
      var mms = 0;
      while (mms < 5) {
          console.log("Eat an M&M");
          mms++;
      }
      console.log("All out of M&Ms");
  </script>
</head>
<body>
</body>
</html>
```

Do you get the same results? Do you see why?

The For Loop

A *for* loop is similar to a while loop, but combines the initialization of the loop variable, the conditional test, and modifying the loop variable all into one statement.

CODE TO TYPE:

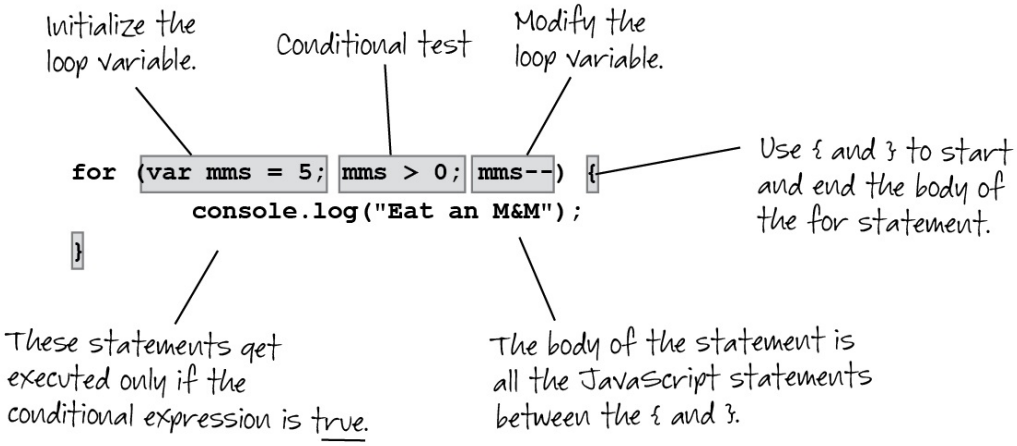
```
<!doctype html>
<html lang="en">
  <head>
  <title> The For Loop </title>
  <meta charset="utf-8">
  <script>
    var mms = 0;
    for (var mms = 5; mms > 0; mms--) {
      console.log("Eat an M&M");
       mms++;
    }
    console.log("All out of M&Ms");
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. Now check your JavaScript console. You should see the same messages you saw before (with the while loop). Let's look at the script:

OBSERVE:

```
<script>
  for (var mms = 5; mms > 0; mms--) {
    console.log("Eat an M&M");
  }
  console.log("All out of M&Ms");
</script>
```

Here's how it works: the variable `mms` is set to 5. Then the conditional test is evaluated, and if it results true, the body of the *for loop* is executed. That is, all the statements between the curly braces. After the body is executed, the variable mms is decremented, and the conditional test is evaluated again. As long as the test evaluates to true, the for loop keeps going. In this example, that happens 5 times, so you see five "Eat an M&M" strings displayed. Once `mms` gets to 0, the loop terminates, and the next statement in the code is executed, which displays "All out of M&Ms".



Can you change the for loop so `mms` starts at 0 and goes to 5 (like we did with the while loop earlier)? See if you can figure out how to do that.

While or For?

In this example, we implemented the exact same loop twice; that is, we can do the same thing with either a while loop or a for loop. So how do you know which kind of loop to use?

Sometimes either one will work, and you can pick whichever you like best. Programmers tend to favor for loops over while loops because they are a bit more concise. However, there may be times when a for loop simply doesn't fit the bill and you can use a while loop instead.

Combining Looping Statements and Conditional Statements

Suppose we want to do something different in the body of the loop depending on where we are in the loop. Maybe we want to check how many M&Ms are left before we say it's okay to eat more. Let's take a look at an example:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
  <head>
  <title> The For Loop Combined with an If Statement </title>
  <meta charset="utf-8">
  <script>
    for (var mms = 5; mms > 0; mms--) {
      if (mms >= 3) {
        console.log("Still lots of M&Ms left, so eat more!");
      } else {
```

```
        console.log("Getting low on M&Ms, take it easy!");
      }
    }
    console.log("All out of M&Ms");
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. Now check the console. In this version, you should see three of the "Still lots of M&Ms left, so eat more!" message, two of the "Getting low on M&Ms, take it easy!" message, and one of the "All out of M&Ms" message.

Here we've combined a *for loop* with an *if statement*. The for loop executes 5 times, and each time through the loop, the if statement executes and tests to see if the value of `mms` is greater than or equal to 3. If it is, we display one message in the console. If it's less than 3, we display another. So you'll see a total of 5 messages, one for each time through the loop.

You can combine JavaScript statements like this together to give you lots of options in creating behavior in your programs. Try some of your own combinations now.

BONUS: Using Loops to generate HTML

You've been very patient as we've been writing small programs to write messages to the console, but if you're anything like me, you're probably itching to write some code to actually do something with a real web page. Right?

We've still got a ways to go before we get to *really* working with web pages, but here's a taste.

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Output to a Page </title>
  <meta charset="utf-8">
  <script>
    window.onload = init;
    function init() {
      var output = "";
      for (var mms = 5; mms > 0; mms--) {
        if (mms >= 3) {
            output += "Still lots of M&amp;Ms left, so eat more!<br>";
        } else {
            output += "Getting low on M&amp;Ms, take it easy!<br>";
        }
```

```
      }
      output += "All out of M&amp;Ms";
      var p = document.getElementById("output");
      p.innerHTML = output;
    }
  </script>
</head>
<body>
  <p id="output">
  </p>
</body>
</html>
```

Don't worry if you don't understand all this code just yet. We'll get to all of it in the next few lessons. In the meantime, load the page in your browser. Instead of seeing messages about the status of your M&M stash in the console, you should now see the messages in the actual web page.

OBSERVE:

```
<script>
  window.onload = init;
  function init() {
    var output = "";
    for (var mms = 5; mms > 0; mms--) {
      if (mms >= 3) {
        output += "Still lots of M&amp;Ms left, so eat more!<br>";
      } else {
        output += "Getting low on M&amp;Ms, take it easy!<br>";
      }
    }
    output += "All out of M&amp;Ms";
    var p = document.getElementById("output");
    p.innerHTML = output;
  }
</script>
</head>
<body>
  <p id="output">
  </p>
</body>
</html>
```

Here, we build up a string, output, with the messages as we loop. Remember that the operator += is like saying "add this new value to the current value of the variable." (In this case "add" means "concatenate" because the variable is a string). So, it's like writing `output = output + "Still lots of M&Ms left, so eat more!<br>"`.

Also notice a couple of things we changed about the string. First, we are using the HTML entity & to represent an ampersand. The reason we need to do this is that HTML sees & as a special character. Now most browsers will work even if you don't use &, but it's a best practice to always use it. Also notice that we're adding a
 to the end of each line in the loop, so when we display the lines within the paragraph in the body of the page, they include those newlines at the end.

Once the loop has terminated, we add one more string on the end. Then we do a bit of magic to get the element where we want to add the string. We're using a function `document.getElementById()`, which retrieves an element using its id (notice the id attribute on the `<p>` element in the body of the HTML). Once we have the `<p>` element stashed in the variable `p`, we can use the `innerHTML` property of the element to set the content of the `<p>` element to the string that's stored in the variable output. Doing this dynamically updates the web page, and you see the string.

We wrap the whole thing in a function named `init`. Why are we doing that? Because we don't want to run the code until the page has loaded. To do that, we set the property `window.onload` to the function *name*, `init`. This tells the browser, "Wait until the page loads before you call the `init` function." You'll get a lot more detail about the reason why this is important later, but for now, just know that if you're updating a web page, it's important to *wait* until the page has been fully loaded by the browser before you update it via JavaScript code that's in the head of your page.

So now you know just a little about how to use JavaScript variables, operators, conditional statements, and loops to update a web page. There will be lots more of that type of code to come, promise!

Loops are powerful tools, and you will use them often in your scripts. Make sure you experiment with them a bit more before moving on to the next lesson. See you there!

More about the material in this lesson

The Riverside JS Workshop would like to acknowledge the generosity of O'Reilly Media, Inc. for making this material available to us through the Creative Commons License. We would also like to acknowledge the great work of Elisabeth Robson who authored this content. She is one of our favorites. Elisabeth has written a number of *Head First* programming books for web developers. We recommend them highly. You can browse her work here.