Riverside JS Workshop

JavaScript Skills Ladder, Lesson 1.7

JavaScript Functions & Events

Lesson Objectives:

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

- *define a function using a function keyword.*
- *use functions with multiple parameters.*
- *use different functions together.*
- *handle the events that you use in JavaScript.*
- *name your functions appropriately.*

What is a Function?

You've already seen a couple of functions in action in this course, but we haven't actually defined a function yet.

A function is a reusable chunk of code. When you put code into a function, you can *call* the function again and again to reuse its code.

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Functions </title>
  <meta charset="utf-8">
  <script>
    function countWords() {
      var sentence = "The answer to life, the universe, and everything is
42";
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
    }

    var howManyWords = countWords();
    alert(howManyWords + " words in the sentence.");
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *function.html*, and open it in a browser. You see an alert displaying the number of words in the sentence. How does it work?

OBSERVE:

```
  <script>
    function countWords() {
      var sentence = "The answer to life, the universe, and everything is
42";
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
    }

    var howManyWords = countWords();
    alert(howManyWords + " words in the sentence.");
  </script>
```

We first *define* our function, using the function keyword. We give the function a name, which must be followed by parentheses (), and then we define the body of the function. We delimit the body using the curly braces {}. All the JavaScript statements between the braces are executed every time you call the function, which you do by using its name followed by ().

In this example, the function `countWords()` returns a value. That means that the result of calling the function is the value we return at the end of the function. (When you return from a function, no statements that might follow the return statement are executed, so return is usually the last statement in the body of the function. You can put the return value of the function into a variable, and use it elsewhere in the program, like we did with the variable `howManyWords`.

The `countWords()` function figures out the number of words in a sentence by using the `split()` function to make an array of all the words, and then finding how many words there are by getting the length of the array, and returns that value so the code that calls the function gets that value as the result of the function call.

Notice that in JavaScript, we don't have to define a function before we use it:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Functions </title>
  <meta charset="utf-8">
  <script>
    var howManyWords = countWords();
    alert(howManyWords + " words in the sentence.");

    function countWords() {
```

```
      var sentence = "The answer to life, the universe, and everything is
42";
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
    }
    var howManyWords = countWords();
    alert(howManyWords + " words in the sentence.");
</script>
</head>
<body>
</body>
</html>
```

Save the changes to *function.html*, and open or refresh it in the browser. You should see the same alert again. This might seem really odd to you. How can you use a function before it's even defined? JavaScript goes through your code and looks for all function definitions before it starts executing your code, so it knows about a function you've defined below the code that uses it.

Function Parameters and Arguments

Now at this point, you might be saying "Okay, that's great, but what's the point of having a function that does *exactly the same thing* every time I call it? How often is that really going to be useful?"

When you need a function to do something *slightly different* each time you call it, you use *parameters* and *arguments*. Let's take a look at an example, and then we'll talk about the details.

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Functions </title>
  <meta charset="utf-8">
  <script>

    function countWords(sentence) {
      var sentence = "The answer to life, the universe, and everything is
42";
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
    }

    var howManyWords = countWords("The answer to life, the universe, and
everything is 42");
    alert(howManyWords + " words in the sentence.");
```

```
    var howManyWords = countWords("A short sentence");
    alert(howManyWords + " words in the sentence.");
</script>
</head>
<body>
</body>
</html>
```

Save these changes, and refresh the browser. You'll see two alerts. The first one displays the message, "The answer to life, the universe, and everything is 42" (10). The second displays the message "A short sentence" (3). Do you see how a function is useful when you can *customize* it like this?

OBSERVE:

```
  <script>
    function countWords(sentence) {
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
    }

    var howManyWords = countWords("The answer to life, the universe, and
everything is 42");
    alert(howManyWords + " words in the sentence.");

    var howManyWords = countWords("A short sentence");
    alert(howManyWords + " words in the sentence.");
</script>
```

We added a parameter named sentence to the function definition. When you put a variable name inside the parentheses in the function definition like we did here, you're saying "This function expects one value to be passed in." Then, in the body of the function, you can use that variable just like if you had declared and initialized the variable inside the function body. Notice that you do *not* use the `var` keyword for parameter names.

Now, to call a function with a parameter, you have to use an argument. An argument is a value that you put between the parentheses when you *call* the function. In this example, we type the value we want to pass to the parameter in between the parentheses in the function call.

Now, let's make another change:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Functions </title>
  <meta charset="utf-8">
```

```
  <script>
    function countWords(sentence) {
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
    }

    var testSentence = prompt("Enter a sentence, and I'll find how many words
it has:");
    var howManyWords = countWords(testSentence);
    alert(howManyWords + " words in the sentence.");
</script>
</head>
<body>
</body>
</html>
```

Save these changes, and refresh the browser. You'll be prompted to enter a sentence and then alerted with the number of words in that sentence. Now you can enter any sentence you want without even having to change the code!

OBSERVE:

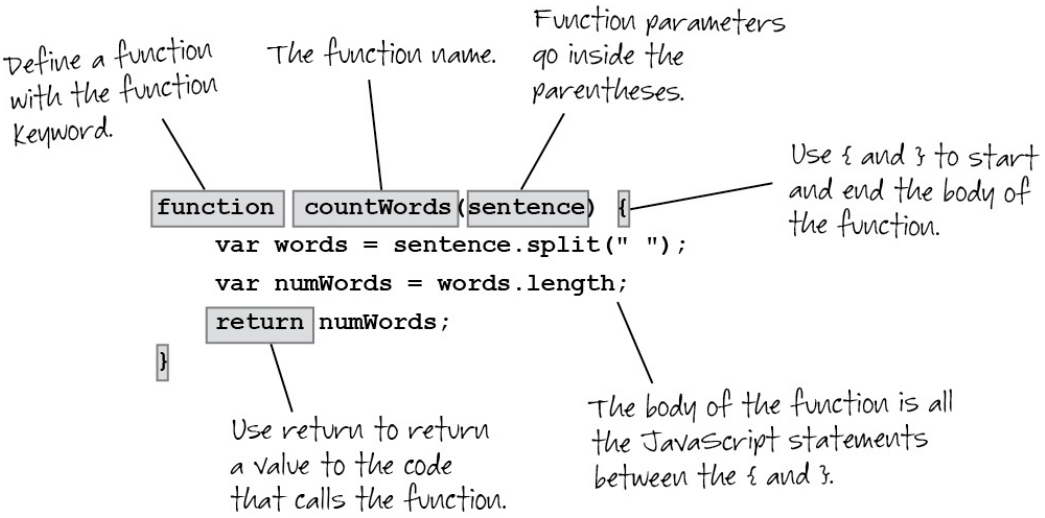
```
<script>
    function countWords(sentence) {
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
    }

    var testSentence = prompt("Enter a sentence, and I'll find how many words
it has:");
    var howManyWords = countWords(testSentence);
    alert(howManyWords + " words in the sentence.");
</script>
```

In this case, we used a variable name for the function parameter. We initialized the `testSentence` variable to a string with the value of the sentence you typed in. Then we used the `testSentence` variable as the argument.

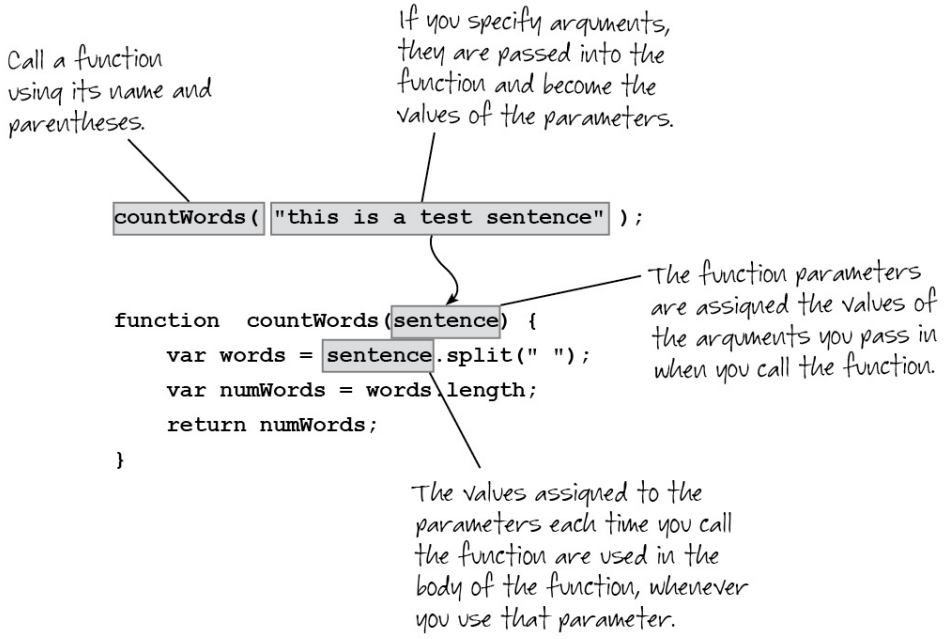
One important thing to notice here is that when you pass an argument to a parameter, you're passing the *value*. Even if you use a variable name as the argument in a function call, what gets stored in the parameter is the *value* of the argument. The parameter is a *different variable*. So in this example, the variable `testSentence` used as the argument is different from the sentence variable used as the parameter. It's the *value* that's stored in `testSentence` that gets passed. This is called *passing by value* and it's a common feature of many programming languages.

Also notice that we used different variable names for the argument and the parameter. The argument variable is named `testSentence` and the parameter variable is named sentence. That's perfectly fine because it's the *value* that matters. You can give the argument variable and the parameter variable the same name if you want—it's up to you. The code in the body of the function will always use the name of the parameter variable.

Let's review. Here's how you define a function:



And here's what happens when you call a function:



Functions with Multiple Parameters

Functions can take more than one parameter. Just remember that for each parameter, you need to pass in an argument, so the number of arguments matches the number of parameters. Create a new HTML file as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Functions with Multiple Parameters </title>
  <meta charset="utf-8">
  <script>
    function areaRect(width, height) {
      var area = width * height;
      return area;
    }

    var area = areaRect(3, 2);
    alert(area);
  </script>
</head>
<body>
</body>
</html>
```

Save this file as *functionmulti.html* in your work folder. Then open the file in your browser. In this example, the argument 3 is passed into the parameter `width`, and the argument 2 is passed into the parameter `height`. Arguments and parameters are matched in order, so the first argument will always be the value of the first parameter, and so on.

Multiple Functions

You can use functions together, and in fact, you'll find that you typically have many functions in your JavaScript. Each task that you want to do potentially more than once is a good candidate to put into a function. (You'll also find that you'll reuse functions that do common tasks in different web applications!). Reopen your *function.html* file and edit it as shown below:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Multiple Functions </title>
  <meta charset="utf-8">
  <script>
    function getSentence() {
        var sentence = prompt("Please enter a sentence");
```

```
        if (sentence == null || sentence == "") {
            alert("Please enter some words!");
        }
        else {
            var howManyWords = countWords(sentence);
            alert(howManyWords + " words in the sentence.");
        }
    }

    function countWords(sentence) {
        var words = sentence.split(" ");
        var numWords = words.length;
        return numWords;
    }

    getSentence();
    var testSentence = prompt("Enter a sentence, and I'll find how many words it has:");
    var howManyWords = countWords(testSentence);
    alert(howManyWords + " words in the sentence.");
  </script>
</head>
<body>
</body>
</html>
```

Save the changes, and reload it in your browser. You'll be prompted to enter a sentence, and you'll see an alert with the number of words in that sentence.

OBSERVE:

```
  <script>
    function getSentence() {
        var sentence = prompt("Please enter a sentence");
        if (sentence == null || sentence == "") {
            alert("Please enter some words!");
        }
        else {
            var howManyWords = countWords(sentence);
            alert(howManyWords + " words in the sentence.");
        }
    }

    function countWords(sentence) {
        var words = sentence.split(" ");
        var numWords = words.length;
        return numWords;
    }

    getSentence();

  </script>
```

Here, we have two functions: `getSentence()`, which prompts the user and makes sure they typed something, and `countWords()`, which counts the words of the sentence passed to it. This splits up the work and makes the code easier to reuse.

We call `countWords()` from the `getSentence()` function. And, of course, to start everything going, we need to call `getSentence()` as well.

Handling Events

You'll need functions whenever you use *events* in JavaScript, which will be in almost every program you write. An *event* is something that happens in your JavaScript program that you can choose to *handle*. Events can be generated by you (like if you click a mouse), or generated by the browser (like if the page has completed loading), or even generated by JavaScript itself (like if a timer goes off).

We're going to come back to events in more detail later in the course, but for now, the important thing to note is that you'll need a function whenever you want to handle an event. These functions are often called—strangely enough—*event handlers*. Let's update our *function.html* program to use a form. When you use the form to submit a sentence, your program will need to *handle* the data from the form in order to process it.

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Multiple Functions </title>
  <meta charset="utf-8">
  <script>
    window.onload = init;

    function init() {
      var button = document.getElementById("submit");
      button.onclick = getSentence;
    }

    function getSentence() {
      var sentence = prompt("Please enter a sentence");
      var sentenceInput = document.getElementById("sentence");
      var sentence = sentenceInput.value;
      if (sentence == null || sentence == "") {
        alert("Please enter some words!");
      }
      else {
        var howManyWords = countWords(sentence);
        alert(howManyWords + " words in the sentence.");
      }
    }
```

```
    function countWords(sentence) {
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
    }

    getSentence();
  </script>
</head>
<body>

  <form>
    <label for="sentence">Enter a sentence: </label>
    <input type="text" id="sentence" size="20" value=""> <br>
    <input type="button" id="submit" value="Get the number of words!">
  </form>

</body>
</html>
```

Save the changes to your file, and open or refresh it in your browser. Type in a sentence to test. When you click the "Get the number of words!" button, you should see an alert showing the number of words in the sentence.

Functions as Event Handlers

OBSERVE:

```
  <script>
    window.onload = init;

    function init() {
      var button = document.getElementById("submit");
      button.onclick = getSentence;
    }

    function getSentence() {
      var sentenceInput = document.getElementById("sentence");
      var sentence = sentenceInput.value;
      if (sentence == null || sentence == "") {
        alert("Please enter some words!");
      }
      else {
        var howManyWords = countWords(sentence);
        alert(howManyWords + " words in the sentence.");
      }
    }

    function countWords(sentence) {
      var words = sentence.split(" ");
      var numWords = words.length;
      return numWords;
```

```
  }
</script>
```

This has three functions. The first function, `init()`, runs when the page finishes loading. We say "`init()` is the load event handler." Now that you know a bit more about functions, take another look at how we assign a function to the `window.onload` property. We write:

```
window.onload = init;
```

We *don't* write:

```
window.onload = init();
```

Why? You know the answer to this now, right? Because when we write `init()`, we're *calling* the function init, which we don't want to do. We want JavaScript to call it *for* us, after the *load event* has been triggered, which the browser does for us when it finishes loading the page. We use a *load event handler* so we can access the DOM safely—that is, only *after* the page has finished loading. If we typed `window.onload = init();`, then JavaScript would try to run the `init()` function as soon as it saw that line of code, which is too soon. By assigning `window.onload` to the function *name*, we let JavaScript run the function when the page is ready.

In `init()`, we set up a button click handler, so the `getSentence()` function is called when you click the button. And here, we say "`getSentence()` is the click handler for the button." This works exactly like `window.onload` does. We want JavaScript to call the `getSentence()` function only when we click the button, which generates a *click event*, so we use the function name when we assign the *click event handler* to the button.

Setting up handlers like this is something you'll do often in your JavaScript code. Many events, like *click* and *load*, have corresponding properties, like `onclick` and `onload`, that you can use to set them up.

Walking Through the Rest of the Code

The function `getSentence()` gets the value of the "sentence" element (that is, the form input with the id "sentence"). This holds the value you typed in for the sentence. It does this first by getting the element itself, and then by getting its value. (You'll learn much more about this in the upcoming Forms lesson!).

Once we have the value of the sentence, just like before we test to see if the value is not empty by comparing to null, and comparing to the empty string, "". If the sentence is empty, then we alert the user to ask her to enter some words. If the sentence is not empty, then we call the `countWords()` function, passing in the string as the argument.

Finally, the `countWords()` function is the same as it was before. It computes the number of words in a string of words that's passed into the parameter sentence and returns that value.

One last little thing to notice: we don't have to call `getSentence()` ourselves anymore. Why? Because the function is being called when we click the form button! `getSentence()` is our click handler, so it's called when we click the button. Think through again how this works.

When you have multiple functions like this, it can get tricky to follow what's often called the *flow of execution*, because you have functions calling other functions, which are returning results, and events can happen which cause other functions to be called. Sometimes it helps to print out the code and draw on the paper, with lines showing which function is calling which other function and when. Number the lines so you know the order in which things happen. Try doing this for the example above.

Naming Functions

Before we end this lab, we should talk about naming functions. Notice that we use descriptive names, just like we do for variables. In general, the rules for function names are the same as those for variables. So stick with those and you'll be fine. Make sure you use camelCase if you're using multi-word names, like we did in this lab.

Also, notice that when we're describing a function and we say something like, "Then call function `countWords()` to figure out how many words there are in the sentence," we write `countWords()` rather than `countWords`. This is often done in books and other descriptions of functions so that you can easily distinguish that we're talking about a function rather than just a variable. But you may sometimes see them written without the `()` as well. Either way is fine when you're *describing* your code in written text, but when you're actually *writing* code, it's really important to know when to use the parentheses and when not to, right?!

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.