Creating a JavaScript

Welcome to of the Core JavaScript Skills Ladder, Phase 1! In this phase, you will learn the basics of JavaScript programming and how to use it to suit your professional and creative goals.

Phase 1 Objectives:

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

- *develop the syntax and structure of JavaScript programs, including statements, expressions, variables, and operators.*
- *collect values using loops, arrays and objects.*
- *add and remove web page elements using the Document Object Model (DOM).*
- *validate and respond to user input using functions and events.*
- *create website menus with CSS and JavaScript.*
- *build a dynamic, interactive, front-end web application.*

To get the most from this material, you'll want to work through the exercises, keying in all the code examples. You will need to set up your computer to create and serve web pages.

First, you'll need a code editor of some kind. There are several very good editors and Integrated Design Environments (IDE) available online. You don't need to spend a lot of money. Many are free. In fact, if you choose, you can use the simple text editor that's already installed on your computer.

Second, you'll need a web server to serve the web pages you'll be creating. Again there are a couple of options, but basically it's a choice between Apache and Microsoft's IIS. XAMPP, for example, is a free package includes Apache, and it runs on most operating systems.

Alright, I'll assume your setup and ready to go. Let's dive right into JavaScript and create your first script. Open a new file in your code editor and follow along.

CODE TO TYPE:

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

  <script>
    var age = 10;
    var ageInDogYears = age * 7;
    alert("Age in dog years: " + ageInDogYears);
  </script>

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

Save the file in your work folder as *dogyears.html*, and open it in a browser. You'll see an alert dialog popup; it looks like this:

Where to Put Your Script

While the HTML probably looks familiar to you, we also introduced a new element—the `<script>` element. This element tells the browser to expect JavaScript. In this case, we placed the `<script>` element in the head of our HTML document. That's generally a good place for a short script that will be used only by the page on which it's located.

You can also place the script in the body of your HTML if you like. Modify your code as shown:

CODE TO TYPE:

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

  <script>
    var age = 10;
    var ageInDogYears = age * 7;
    alert("Age in dog years: " + ageInDogYears);
  </script>

</head>
<body>

  <script>
    var age = 10;
    var ageInDogYears = age * 7;
    alert("Age in dog years: " + ageInDogYears);
  </script>

</body>
</html>
```

Save your file again, and refresh or open it in your browser.

The browser runs your JavaScript code as soon as it sees it, while it's parsing your HTML document. So, if you place your JavaScript in the `<head>` of your document, it will run earlier than if you place it in the `<body>`. In JavaScript, like in HTML, the browser evaluates and runs your code top-down, so JavaScript statements at the top run before JavaScript statements at the bottom.

As you'll see a bit later, ordering is important!

Linking to an External Script

If you're going to use your JavaScript in multiple HTML files, or you just want to keep your JavaScript and HTML separate, you can link to a JavaScript file. It's similar to how you link to CSS files, except that instead of using the `<link>` tag, you use the `<script>` tag. First, copy and paste all the JavaScript (between the `<script>` tags) from dogyears.html into a new file in your work folder; name that file *dogyears.js*. Then, remove that JavaScript from dogyears.html and change the `<script>` element so that it links to the new file. Modify your code as shown:

CODE TO TYPE:

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

  <script src="dogyears.js">
  </script>

</head>
<body>
  <script>
    var age = 10;
    var ageInDogYears = age * 7;
    alert("Age in dog years: " + ageInDogYears);
  </script>

</body>
</html>
```

We moved the `<script>` element back to the `<head>` of the document. Even though we don't have any JavaScript between the opening and closing script tags, we still include both. This is important! If you don't include both the opening and closing script tags, you may get weird behavior in your browser.

Save your file again and open it a browser or refresh the browser if it's already opened. You should see the same dialog as before.

Linking to an external JavaScript file from the `<head>` of the document is the most common way to include JavaScript in a web page. If you "view source" code of web pages, you'll usually see the JavaScript included like that. We'll be using the same method frequently, as well as placing our JavaScript in the `<head>` element, throughout the rest of the Skills Ladder.

The <script> Element

The `<script>` element is pretty straightforward and you now know everything you need to know about `<script>` for this Skills Ladder. But, when you check out how other pages link to JavaScript files, you might notice that some links include a `type` attribute.

The `type` attribute used to be required because browsers used to support different scripting languages; for instance, for a long time, IE supported JScript, while Netcape supported JavaScript. Now that those browsers have standardized a bit more, JavaScript is the default scripting language for all the major browsers, so the type attribute is optional. Still, if you'd like to use it (maybe you know that some of your users still use old browsers), you can—it's still supported! Edit *dogyears.html* as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> My First JavaScript </title>
  <meta charset="utf-8">
  <script src="dogyears.js" type="text/javascript">
  </script>
</head>
<body>
</body>
</html>
```

Create and Calling a Function

When you're writing JavaScript, most of the time you'll put your code in *functions*. You'll also use many of the built-in functions available in JavaScript. We'll cover functions in greater detail later in the Skills Ladder, for now, think of a function as a way to package a chunk of code so that it's more convenient to access and reuse later.

Just to get warmed up using functions, let's change the script you've already written in *dogyears.js*, and put that code inside a function. Modify the code as shown:

CODE TO TYPE:

```
function dogYears() {
    var age = 10;
    var ageInDogYears = age * 7;
    alert("Age in dog years: " + ageInDogYears);
}
```

We've put all the code that was in your file into a function named `dogYears`. The function keyword indicates that we're defining a function named dogYears. All the code in the function goes between an opening and closing brackets {}. By convention, we indent the code inside the function, so it's easier to see that the code is part of the function and not the code around it, and also to keep the brackets matching.

Save your *dogyears.js* again and open it a browser or refresh the browser if it's already opened. Did you see the alert?

No! You did not see the alert because now all the JavaScript code is in a function, and the only way to get the function's code to run is to call it. To do that, add one more line of code to *dogyears.js*. Modify your code as shown.

CODE TO TYPE:

```
function dogYears() {
    var age = 10;
    var ageInDogYears = age * 7;
    alert("Age in dog years: " + ageInDogYears);
}
dogYears();
```

Save your JavaScript file, and open it a browser or refresh the browser if it's already opened. Do you see the alert now?

Now that you've packaged the code into a function, it'll be convenient to use that code whenever you want—behold the power of functions! You could type `dogYears();` as many times as you want in your JavaScript file and you'd get the equivalent number of alerts. You can reuse the code in the `dogYears()` function without having to type it repeatedly.

Try adding a few more calls to the `dogYears()`function as shown:

CODE TO TYPE:

```
function dogYears() {
    var age = 10;
    var ageInDogYears = age * 7;
    alert("Age in dog years: " + ageInDogYears);
}
dogYears();
dogYears();
dogYears();
dogYears();
```

Open *dogyears.html* again. How many alerts do you see now? It might seem like overkill to get that many alerts, but you can see how nice it is to have that code packaged up into a function.

Built-In Functions vs. Creating Your Own Functions

Most of what you do in JavaScript is writing and using functions. For example, `alert` is a built-in function that takes a string (a series of characters between quotation marks) and displays that string in a dialog box. Let's try another built-in function, `prompt`. Type this code into your editor as shown:

CODE TO TYPE:

```
function dogYears() {
    var age = 10;
    var age = prompt("Enter your dog's age: " );
```

```
    var ageInDogYears = age * 7;
    alert("Age in dog years: " + ageInDogYears);
}
dogYears();
dogYears();
dogYears();
dogYears();
```

Note If you are using IE, you may need to change a browser setting for this script to work properly. Select **Tools | Internet Options | Security** and check that "Allow websites to prompt for information using scripted windows" is enabled.

Reload *dogyears.html* to see how `prompt` works. It prompts you to enter your dog's age. So instead of just setting the dog's age to 10 every time we call the function dogYears, we ask the user to enter their dog's age, storing that value in a variable named age, and then displaying that age multiplied by 7.

Don't worry about all the details of variables and functions yet. We're just getting your feet wet with JavaScript and getting a basic script working for now. We'll come back to all these details shortly, I promise!

Using Other People's Scripts

Just about every web page on the internet uses JavaScript. If you find a page that contains JavaScript you like, it may be convenient to copy and paste the JavaScript from the web page into yours and use it. That's absolutely fine when you're learning JavaScript and you want to try things out. But it's definitely *not* fine for you to use other people's scripts in a live web page without getting their permission first!

If you find a script you like, and you want to use it, make sure you ask permission from the person upon whose web page you found it. Typically, people are open to sharing their scripts, as long as you give them attribution for them. Using scripts without permission is copyright infringement, so stay on the safe (and honorable) side, and ask permission before you use other people's scripts.

Script Libraries

There are lots of places you can get *script libraries* now as well. These are scripts that are designed to be downloaded, or linked to, and used by other people. One of the most popular script libraries on the internet is *jQuery*, a library designed to make many common tasks you do with JavaScript (to interact with a web page, or create UI effects, for instance) easier. You can

use this library without asking anyone's permission, as long as you leave all the information in the JavaScript files you download intact so others know you're using the jQuery library.

Perform a *view source* on a few of your favorite web pages and see if you can find the JavaScript. You might find that some of them link to well-known libraries, like jQuery, or have their own code in the page or linked to from the page, or both!

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.