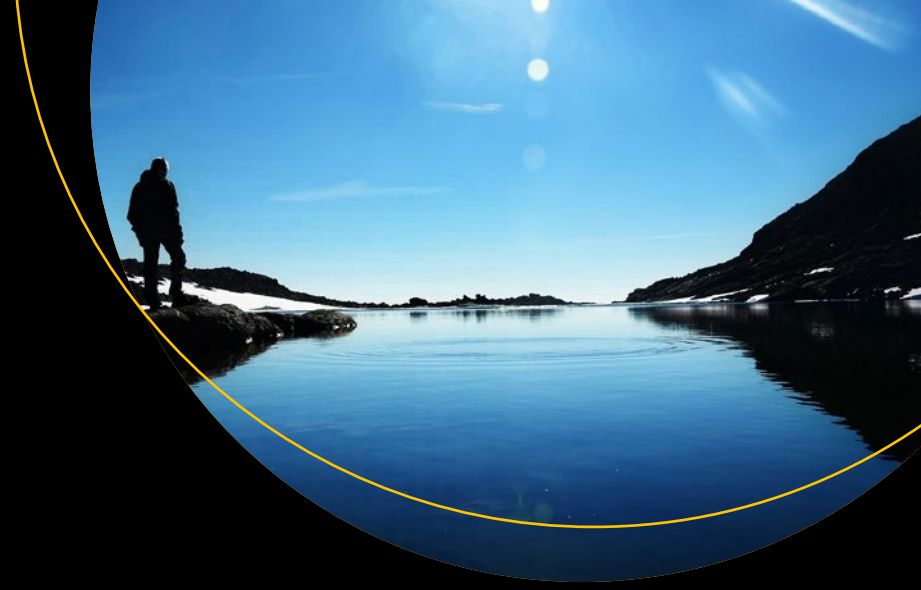

Beginning JavaScript

The Ultimate Guide to Modern JavaScript Development

—

Third Edition

—

Russ Ferguson

Apress®

Beginning JavaScript

The Ultimate Guide to Modern JavaScript Development

Third Edition

Russ Ferguson

Apress®

Beginning JavaScript

Russ Ferguson
Ocean, NJ, USA

ISBN-13 (pbk): 978-1-4842-4394-7 ISBN-13 (electronic): 978-1-4842-4395-4
https://doi.org/10.1007/978-1-4842-4395-4

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Coordinating Editor: Nancy Chen

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484243947. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

This space is dedicated to my brother, Rodd, and my Dad.

If not for my Dad, none of this would be possible.
Thanks, Dad.

—Russ

Table of Contents

About the Author

Russ Ferguson is a freelance developer and instructor in the New York City area. He has worked with companies of all sizes, from startups to some of the largest organizations in the world. These companies have spanned industries including cable television, book publishing, finance, and advertising. He has worked on projects for companies like Bank of America, General Mills, LG, Viacom, and DC Comics.

About the Technical Reviewer

Toby Jee is software programmer currently located in Sydney, Australia. He loves Linux and open source projects. He programs mainly in Java, JavaScript, TypeScript, and Python. In his spare time, Toby enjoys walkabouts, reading, and playing guitar.

Acknowledgments

I need to thank everyone at Apress for working with me and keeping me on course to get this book finished. Nancy, Toby, Louise, James, and Jade, thank you.

CHAPTER 1

Introduction to JavaScript

JavaScript has changed a lot over the years. We are currently in a time where there is a JavaScript library for just about anything you would like to build. JavaScript lives both on the client and the server, on the desktop and on mobile devices.

The goal of this book is to help you get an understanding of how the language works, what can be done with it, the resources available, and the some of the ecosystem around the language and tools. At times I will point out things that may be asked on a technical interview, all in an effort to help you get your arms around this growing community. Some of the topics I will cover are

- Understanding JavaScript syntax and structures
- Creating scripts that are easy to understand and maintain
- Using tools to debug JavaScript
- Handling events
- How JavaScript works on the server
- The frameworks that exist to make JavaScript a strongly typed language
- JavaScript application frameworks and how they work
- Retrieving data from the server

JavaScript is essential in modern web development; single page applications (SPAs) make up the majority of sites being created. Understanding JavaScript lets you add interactivity to your website and lowers the learning curve for things like frameworks. This is not to say that you need frameworks for everything, but to add any level of interactivity to your site, you need JavaScript.

R. Ferguson, *Beginning JavaScript*, https://doi.org/10.1007/978-1-4842-4395-4_1

Enough introduction—you got this book to learn about JavaScript, so let's start by talking quickly about JavaScript on a high level before diving right into it.

In this chapter, you'll learn

- Why JavaScript is important to you as a developer
- How to add JavaScript to a web document
- Object-oriented programming (OOP) in relation to JavaScript

Chances are that you have already come across JavaScript and already have an idea of what it is and what it can do, so I'll move quite swiftly through some basics of the language and its capabilities first. If you know JavaScript well already, and you simply want to know more about the newer and more accessible features and concepts, you may skip a head. However, there may be some information you've forgotten, and a bit of a review doesn't hurt, either.

The Why of JavaScript

As discussed, JavaScript is everywhere. It plus HTML and CSS are all the tools you need to develop a website.

You can work on both the client side and the server using JavaScript. This makes the demand for JavaScript developers very high. And high demand means various job opportunities and competitive rates for developers. As of this writing, the average salary of a JavaScript developer is $110,841 per year in the United States according to Indeed, a job website (`www.indeed.com/salaries/Javascript-Developer-Salaries`). So not only is JavaScript a language worth taking a look at, it can be a good addition to your toolset a developer. Let's have a quick discussion of what JavaScript is and then move onto writing some code.

What Is JavaScript?

JavaScript is an interpreted scripting language. The host environment will provide access to all the objects needed to execute the code.

The primary example for JavaScript is the ability to add interactivity to a website. This works because the interpreter is embedded into a web browser so you do not need to add any software.

This has made JavaScript a language that is easily accessible because all you need is a text editor and a browser. On the client side, you can add levels of interactivity like responding to button clicks and validating the contents of a form. It will also allow you to take advantage of the APIs (application programming interfaces) that are built into a browser. Adding something like geolocation capabilities to your site is an example of this.

Another use case is to execute JavaScript on the server, using an environment like Node.js. An example of server-side JavaScript is the ability to do things like make requests from a database and respond to HTTP requests and create files.

The majority of this book will focus on the client side; however, there is very little difference from a code perspective.

JavaScript in a Web Page and Essential Syntax

Applying JavaScript to a web document is very easy; all you need to do is use the `script` tag:

```
<script>
  // Your code here
</script>
```

While this is the simplest way of adding JavaScript to the page, it is not recommended. This is an example of inline JavaScript. One of the reasons not to build your site this way is that it becomes hard to maintain over time. Imagine, as your website grows in size, how difficult it would be to keep everything organized.

The preferred way to add JavaScript to an HTML page is to refer to an external `.js` file:

```
<script src="js/myfile.js"></script>
```

Note HTML 5 requires that the `script` tag have its own closing tag. This will ensure backwards combability with older browsers. In addition, adding the `script` tag right before the ending body tag is considered a best practice.

JavaScript Syntax

Learning any programming language is very similar to learning a foreign language. There are rules to follow and that may require a different way of thinking.

There is a lot of problem solving in programming. You spend time looking at a situation and trying to figure out how to use code to solve the problem. With that in mind, let's take that angle in how we discuss JavaScript.

If you want to hold onto a piece of information to be used later, this is called a *variable*. If you remember any high school algebra, there are always examples where a word or letter represents a value:

```
5 + x = 10
```

In this example, x is a variable that represents a number. Using JavaScript, you can declare a variable and give it a value and then use it later:

```
var x = 5;
5 + x = 10;
```

The above code is not perfect JavaScript, but it illustrates the idea of how variables work. The first line uses the keyword `var`; this is built into the language and can only be used when declaring a variable. At the end of each line is a semicolon, which can be thought of as the period at the end of a sentence. The JavaScript interpreter does not require you to have it. If you do not add semicolons, the interpreter will add them for you. In order to have more control and to make it easier to read, it is recommended that you add them on your own.

The keyword `var` is not the only way to declare a variable. Other keywords like `let` and `const` can also be used. I will cover what makes them different and when one should be used over another in Chapter 3.

Another scenario is when you have some code and you only want to run it when you need it. JavaScript calls this a *function*. You can write a function, add into it all the commands that you want it to execute, and then have it wait until you need it, like so:

```
function doMath(num1, num2){
      let sum = num1 + num2;
      return sum;
}
```

This sample function has the name `doMath`. This allows the function to be referenced or called from other parts of your code. This function also accepts two arguments or parameters, `sum1` and `sum2`. Think of them as variables for your function; they just represent data that the function will need in order to execute properly. In this case, it's a set of numbers.

Next are the curly braces ({ }). They contain your code block. Everything that you need to make this function work goes here. You can add as many lines of code as you need. As a general rule, a function should perform only one thing. Using `doMath` as an example, it only adds numbers together. If you want something else to happen, another function should be written. This will help the debugging process. It is also important to note that the JavaScript language itself contains functions to perform things like string manipulation and math.

Let's say you want to leave notes to yourself. In this case, you need to have something in your code that is not going to execute as code. Adding comments to your code can be declared in two variations:

```
 // single line comment

*/
multi line
comment
/*
```

Adding multiline comments is also useful when you are debugging your code. For example, if you do not want to delete what you have, but you do not want the code to execute, you can make it a comment.

So far, you now know how to solve a few problems: how to hold onto data, execute functions, and leave reminders in the code.

Now let's take some time to talk about how the code executes in your browser.

Code Execution

The browser reads the page from top to bottom, so the order in which code executes depends on the order of the script blocks. A *script block* is the code between the `<script>` and `</script>` tags; if you have an external `.js` file, it will also read from top to bottom. (Also note that it's not just the browser that can read your code; the user of

a website can view your code, too, so you shouldn't put anything secret or sensitive in there.) There are three script blocks in this next example:

```
<!DOCTYPE html>
<html>
    <head>
         <script type="text/javascript">
                alert("First script Block");
                alert("First script Block - Second Line");
          </script>
     </head>
     <body>
           <h1>Test Page</h1>
           <script type="text/javascript">
                 alert("Second script Block");
           </script>
    <p>Some more HTML</p>
          <script type="text/JavaScript">
                alert("Third script Block");
                function doSomething() {
                      alert("Function in Third script Block");
                }
          </script>
        </body>
</html>
```

If you try it out, you'll see that the `alert()` dialog in the first script block appears and displays the message

```
First script Block
```

That's followed by the next `alert()` dialog in the second line displaying the message

```
First script Block - Second Line.
```

The interpreter continues down the page and comes to the second script block, where the `alert()` function displays

```
Second script Block
```

And the third script block follows it with an `alert()` statement that displays

```
Third script Block
```

Although there's another `alert` statement inside the function a few lines down, it doesn't execute and display the message. This is because it's inside a function definition (`function doSomething()`), and code inside a function executes only when the function is called.

So far in this chapter, you've looked at the JavaScript language, seen some of the syntax rules, learned about some of the main components of the language (albeit briefly), and run a few JavaScript scripts. You've covered quite a lot of distance. Before you move onto a more detailed examination of the JavaScript language in the next two sections, let's explore the important parts of the JavaScript language: *functions* and *objects*.

Functions

In the last section, you saw some code that was not executed until you explicitly asked for it. Functions are extremely flexible in JavaScript. They can be assigned to variables and passed as a property to other functions as an argument. Functions can also return another function. Let's take a look at a few examples:

```
var doMath = function(num1, num2) {
        var result = num1 + num2;
         return result;
};

var myResult = doMath(2,3);
```

This example illustrates how you can assign a function directly to a variable and then use that variable to call the function. This function takes two numbers as arguments. When the numbers are passed to the function, they are added together, and the result is returned back to the code that calls the function:

```
function message() {
      return 'It's. the information age ';
}

function displayMessage(msgFunction, person){
   consoe.log(msgFunction() + person) //It's the information age brother!
}

displayMessage(message, "brother!");
```

This example passes a function over to another function. The second function receives a function as an argument and then executes the function inside a `log` command. This function originally called `message` returns a string that will be displayed alongside the string that is also passed inside your `console.log` method.

Functions are a very important part of the JavaScript language, and I will cover them in detail in Chapter 5.

One other concept I will introduce here is the concept of an *object*. I will also go into more detail about objects in Chapter 4.

Objects

Objects are central to the way you use JavaScript. Objects in JavaScript, in many ways, are like objects in the world outside of programming. (It does exist; I just had a look.) In the real world, an object is just a "thing" (many books about object-oriented programming compare objects to nouns): a car, a table, a chair, and the keyboard I'm typing on. Objects have

- **Properties** (analogous to adjectives): The car is *red*.
- **Methods** (like verbs in a sentence): The method for starting the car might be to *turn ignition key*.
- **Events:** Turning the ignition key results in the *car starting* event.

Object-oriented programming tries to make programming easier by modeling real-world objects. Let's say you are creating a car simulator. First, you create a car object, giving it properties like *color* and *current speed*. Then you need to create methods: perhaps a *start* method to start the car, and a *brake* method to slow the car, into which you need to pass information about how hard the brakes should be pressed so that you can determine the slowing effect. Finally, you need to know when something happens with your car. In OOP, this is called an *event*. For example, when the gas tank is low, the car sends a notification (a light on the dashboard) letting you know it's time to fill up. In this code, you will listen for such an event so that you can do something about it.

Object-oriented programming works with these concepts. This way of designing software is now very commonplace and influences many areas of programming—but most importantly to you, it's central to JavaScript programming.

Some of the objects you'll be using are part of the language specification: the `String` object, the `Date` object, and the `Math` object, for example. These objects provide lots of useful functionality that could save you tons of programming time. You can use the `Date` object, for example, to obtain the current date and time from the client (such as a user's device). It stores the date and provides lots of useful date-related functions such as converting the date/time from one time zone to another. These objects are usually referred to as *core objects* because they are independent of the implementation. The browser also makes itself available for programming through objects you can use to obtain information about the browser and to change the look and feel of the application. For example, the browser makes available the `Document` object, which represents a web page available to JavaScript. You can use this in JavaScript to add new HTML to the web page being viewed by the user of the web browser. If you used JavaScript with a different host, a Node.js server, for example, you'd find that the server hosting JavaScript exposes a very different set of host objects because their functionality is related to things you want to do on a web server.

Note Even though this section covers JavaScript from an object-oriented perspective, the language itself is a multi-paradigm language where you use it as a functional, imperative, or event-driven language.

As you progress through the book, you'll get a more in-depth look at objects: the objects central to the JavaScript language, the objects that the browser makes available for access and manipulation using JavaScript, and your own custom objects. For now,

though, all you need to know is that objects in JavaScript are *entities* you can use to add functionality to web pages, and that they can have properties and methods. The `Math` object, for example, has among its properties one that represents the value of pi and among its methods one that generates a random number.

Summary

In this chapter, you learned about JavaScript at a high level. You learned why JavaScript can be a good tool in your developer toolbox. You also learned some basic syntax and how to add comments to your code. You saw how to use variables to hold onto data for later use and how functions can be used to have code ready on demand. Another subject was code execution (how the browser reads code from top to bottom). The largest section was the discussion of objects. If you are going to work with JavaScript either on the client or on the server, it is important to understand that working with objects is at the core of JavaScript.

Next, you will take a look at some of the tools you, as a JavaScript developer, have available to you.

CHAPTER 2

JavaScript and Development Tools

There is a lot to get your head around if you are new to developing JavaScript applications. A question often asked is "Where do I start?" This is the goal of this chapter, to help you locate resources and tools that can help you on the path to developing applications with JavaScript. I will also go over some basic usage of these tools. This chapter is not designed to be a definitive guide or as product placement, but it should point you in the right direction.

Here are a few subjects I will go over in this chapter:

- Tutorials and resources
- Integrated development environments (IDEs)
- A quick introduction to Node.js and npm
- Git and GitHub

Tutorials and Resources

There are a lot of websites you can use to gather information about JavaScript, HTML, and CSS. Here are some of the more useful sites:

- MDN web docs at `https://developer.mozilla.org/en-US/docs/Web/JavaScript`
- Free Code Camp at `www.freecodecamp.org/`
- Khan Academy at `www.khanacademy.org/computing/computer-programming/html-css-js`

R. Ferguson, *Beginning JavaScript*, https://doi.org/10.1007/978-1-4842-4395-4_2