THE EXPERT'S VOICE® IN WEB DEVELOPMENT

Beginning jQuery

WRITE BETTER AND MORE EFFICIENT JAVASCRIPT WITH JQUERY

Jack Franklin

Apress®

Beginning jQuery

Jack Franklin

Apress®

Beginning jQuery

ISBN-13 (pbk): 978-1-4302-4932-0

ISBN-13 (electronic): 978-1-4302-4933-7

President and Publisher: Paul Manning
Lead Editor: Louise Corrigan
Technical Reviewer: Ian Devlin
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Mark Powers
Copy Editor: Kimberly Burton-Weisman
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

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.

Apress and friends of ED books 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 Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com/9781430249320. For detailed information about how to

locate your book's source code, go to www.apress.com/source-code/.

Dedicated to Mum, Dad and Sam

Contents at a Glance

Contents

Foreword

I'm a Christian, I love my family, I work for appendTo, and I love to learn! I think Jack and I share the common desire to learn.

I first noticed Jack Franklin when he launched the JavaScript Playground web site. I watched as he regularly posted many relevant topics about the front-end development world. I then saw him branch out and experiment with server-side JavaScript—recording and sharing screencasts, speaking at conferences, and now writing this book.

I was honored when he contacted me to write the foreword for his book. jQuery has come a long way since 2006. There have been many books written about it and I'm certain there will be many more to come. The thing I like about Jack is that he is first and foremost a JavaScript scholar. As your read through his book, he takes special care to introduce his readers to proper JavaScript concepts in order to shield them from confusion down the road.

jQuery tends to be an easy library for many developers and designers to learn, but the danger comes when they start to feel friction with the actual JavaScript language, not the jQuery library. Jack appreciates this friction and tries to alleviate that roadblock for his readers.

Jack gives a good overview of the main topics that jQuery covers and provides numerous code examples and snippets for his readers to grasp. I personally find that the technical books I most enjoy reading are ones that have code sprinkled here and there so that I can fully grasp the concepts explained in the prose.

jQuery is a fast-evolving library and new versions come out frequently. As a result, new features are added and others are deprecated from version to version. If you are new to jQuery or need a quick refresher, this book will navigate you toward the appropriate API methods and techniques you'll need to become proficient with the jQuery library.

Elijah Manor
Senior Architect & Trainer for appendTo
Microsoft Regional Director & Microsoft ASP.NET MVP

About the Author

Jack Franklin is a web developer and computer science student from the world heritage city of Bath, in the UK. He started creating web sites in 2005 and has experience in a number of web languages, including HTML, CSS, PHP, Ruby, Python, and others, although his focus is JavaScript. He runs the popular online resource JavaScript Playground (`http://javascriptplayground.com`) and has released a number of open-source jQuery plug-ins online.

About the Technical Reviewer

Ian Devlin is an Irish web and app developer who resides in Germany, where he works for pixolith, a web agency in Düsseldorf. He started his working life as a software developer mainly using C, and eventually turned his attention toward web technologies. In addition to writing on his own web site (`www.iandevlin.com`), Ian writes for HTML5 Doctor and .net magazine, curates at HTML5 Gallery, and has written for Dev.Opera and *PC Pro*. He has also written a book called *HTML5 Multimedia Develop and Design* (Peachpit Press, 2011). Outside of all that, he loves European history and taking walks in the countryside.

Acknowledgments

I’ve been fortunate to have so many people help me along my journey to get me to this stage.

The first is Richard Quick, who first got me hooked on the web when I attended my first conference. He went out of his way to make me able to attend and I got home late that evening inspired, knowing it was a path I wanted to venture down.

Then there’s all the folks who I got to know when I moved to Bath, who gave me advice, put up with my questions and were always willing to help. People like Dan Dineen, Justin Owen, Jamie Rumbelow, Phil Sturgeon, Julian Cheal everyone at Storm and the guys at Riot. Thanks to Alex Older for giving me my first speaking opportunity at his conference in Bristol too.

Next on the list are my peers at University who put up with me being incredibly unsociable whilst I stayed in coding: Aaron, Grant, Dave, James, Ollie, Cat, Sophie, Helen and loads more.

The opportunity to write a book came about because I started blogging, and a crucial part of that was Toby Howarth, who kindly donated his time to make the site look nice. Thanks to the people who helped spread my articles across the internet and gave me advice: Stuart Robson, Elijah Manor, Anthony Killeen, Dan Sheerman, Addy Osmani, Sindre Sorhus, Rachel Shillcock, Adam Onishi, Michael Heap and Peter Cooper to name but a few.

For the duration of writing this book, I had just started working for Kainos, who were incredibly welcoming and supportive of me. Thanks to everyone there, but in particular to Stuart McKee, Luke McNeice, Will Hamill, Michael Allen, Steven Alexander, James Hughes and Tom Gray.

I was also lucky to meet and work with some other really smart people through work; people like Tim Paul, Roo Reynolds, Tom Loosemore, Ben Howdle and Alice Newton.

Finally, everyone at Apress deserves a medal for putting up with me, and the barrage of questions I sent their way, in particular Mark, Louise and Ian. Along with the rest of the Apress team they’ve turned my mess of words into this book you’re reading.

CHAPTER 1

JavaScript You Need to Know

---

jQuery is a framework that's built on top of JavaScript, not a language in its own right. It is possible to write jQuery with barely any knowledge of JavaScript, but it's not something I would recommend. If you want to be able to confidently write jQuery plug-ins for your site, or alter plug-ins others have written, you need to be familiar with basic JavaScript. This is why I'm starting with JavaScript that you need to know. This chapter will cover:

- JavaScript scripts on a web page
- Variables and objects in JavaScript
- JavaScript functions
- Conditionals
- Looping over arrays and objects
- Debugging JavaScript

If you are familiar with JavaScript, you might feel like skipping this chapter. That's fine, but please consider skimming it first to ensure that you are comfortable with everything covered. Resist the temptation to skip to the jQuery parts—because you will struggle with it. Trust me, in a couple of chapters' time, this will all seem worth it. Many developers I've helped online have dived into jQuery eagerly before becoming stuck due to a lack of understanding the language jQuery is built on. When you're writing jQuery, you're writing JavaScript, but using the jQuery library. I cannot stress how important it is that you make sure the content covered in this chapter is content that you are comfortable with before moving on. I suggest that you try out the code as you go through. Don't fool yourself into thinking you understand it because you've read it; there is no substitute for typing out the code yourself.

To run the code, I recommend JS Console (`www.jsconsole.com`), a tool by Remy Sharp that allows you to execute JavaScript and see the results. You can enter the code in the top bar and hit Enter to see the results. This is really useful for short lines of code. Figure 1-1 shows an example of JS Console.

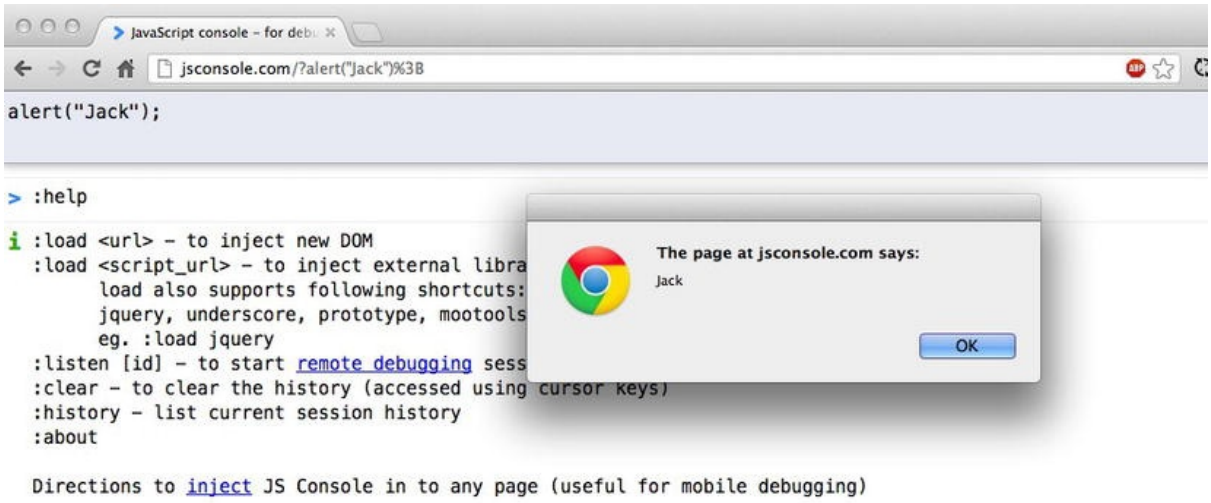

Figure 1-1. Running the code alert ("Jack") and viewing the results on JS Console

For larger pieces of code, it's best to set up an `index.html` page and include your JavaScript file in there. I'll explain how to do that in the next section of this chapter. Throughout this chapter, I will often use the alert function to demonstrate the value of a certain variable. This is purely used for demonstration of concepts. In real life when I need to check the variable, I don't ever use alerts—I use a browser's JavaScript console. The reason for using alerts for basic examples is that it's much easier to get started with. There's no need to load up the developer tools, which take time to get accustomed to. Once you progress into more complex code, you will spend time exploring the developer tools. At the end of this chapter, I'll show you exactly how I do that, before moving on to jQuery.

Using JavaScript on a Web Page

When you have a basic web page and wish to add some JavaScript to run, you have two options. First, you can add your code inline, within a `script` tag, like so:

```
<script type="text/javascript">
  //write code here
</script>
```

Or, you can create an external JavaScript file with the `.js` file extension and then load it in, again through the `script` tag:

```
<script type="text/javascript" src="path/to/your/file.js">
```

Note that you have to close the `script` tag. Even though there's nothing between it, it's not a self-closing tag. Within your JS file, you are free to write JavaScript.

Within a typical HTML file, there are typically two places people use to load their

external JS files. The first is within the `head`, and the second is just before the closing `</body>` tag. In the past, scripts were always loaded into the `head` element, but with performance and page loading speeds more critical than ever, it's often recommended to place your scripts at the bottom of your page. This is an approach I side with, too.

The browser renders the page from top to bottom, and when it comes across your scripts, it pauses rendering the page to load in your JS. Thus, the page loads slower (or, more importantly, feels that way to the user) because the rendering is blocked by your loading JavaScript files. Hence, putting the scripts just before the closing `</body>` tag means that when the time comes to load your scripts, the rest of the page has been loaded.

Before moving on to looking at the language itself, there's one more thing I'd like to note. If you're using the new HTML5 doctype (`<!DOCTYPE html>`) rather than one of its more long-winded predecessors, you don't actually need to define the type attribute on your script tags. Simply,

```
<script src="path/to/your/file.js"></script>
```

is enough. This does not cause issues in older browsers—neither does the HTML5 doctype—and I highly recommend using it.

Syntax Conventions

JavaScript's syntax is pretty basic and clear, but there are certain subtleties that you will discover on the way. There's often more than one way to do things, but the community has certain conventions that have stuck over time. One convention that I want to mention straightaway is semicolons. Often in JavaScript, adding a semicolon at the end of a line is optional, and you will see tutorials that don't do it. However, the convention is to always use a semicolon at the end of a line, and that's what I'll be following here. There are obviously certain circumstances when you can't use one, and you will see those, but in any situation where a semicolon is optional, I'll use one. I recommend you do, too.

Another consideration to make is for white space. It is insignificant in JavaScript, so you can layout code the way you like in terms of white space. Whenever you are inside a set of braces, you should indent by one tab, but other than that, you will find yourself adapting your own standard.

Comments

Before continuing, at this stage it's worth discussing comments. JavaScript allows us to insert comments. This is content that will be ignored and not treated as code, so you can put anything you want in them. It's useful for documenting your code. There are two syntaxes for comments—one for a single line comment and one for a multiline comment:

```
//this is a single line comment, denoted by two forward sl
/* this is a multi-line comment, started with a slash and
and ended with an asterisk and a slash */
```

Use these when you like to remind yourself about a piece of code and what it does, or to provide references for the future you. After not working on code for a long period of time, comments can really help you remember why you wrote what you wrote.

Variables

Often when coding, we want to save the state of something. Perhaps we want to remember that the current color of our background is red, or the calculation we just performed totaled 33. JavaScript, like most languages, has *variables*: a place to store information. To create one, you simply declare it with the `var` keyword, name it, and then set it to equal to something. You can also declare a variable without explicitly setting its value. If you do this, the variable will be set to `undefined`, a special value in JavaScript that simply means that this variable has not been set to anything.

```
var twoPlusThree = 5;
var twoPlusTwo = 2 + 2;
var notYetDefined;
```

Here I declared three variables. The first, `twoPlusThree`, is set to the value `5`. The second, `twoPlusTwo`, is set to be the result of `2+2`. Here you meet one of JavaScript's many operators, `+`. These operators perform operations on values. Most of them are obvious. Along with `+` (addition), there's `-` (subtraction), `/` (division), `*` (multiplication), and many more. You'll meet more throughout the book, so don't worry too much about them now. The third variable, `notYetDefined`, does not have a value and is set to `undefined`, because I declared a variable (that is, I created a new variable) but did not set a value.

Variables can contain letters, digits, and underscores. They cannot start with a number. So the variable name `0abc` is *not* valid, whereas `abc0` is. Typically, most developers do not use digits in variable names, and either stick to camelCase or the underscore notation.

Note Notice my naming convention for variables. I'm using what's known as camelCase. The first word in the variable name should start with a lowercase letter but then every other word in the name should start with a capital letter. I'll be using this throughout the book. There are other popular naming conventions, most notably the_underscore_method. This keeps all words in lowercase and separates them with underscores. This is more popular in other languages. The majority of the JavaScript community uses camelCase.

Of course, once you set a variable to a value, it doesn't mean you can't change the value. All variables can have their values changed. It's done very similarly to the way

you declare a variable, with the only difference being the missing `var` keyword at the beginning. That's only needed when you declare a variable.

```
var totalCost = 5;
totalCost = 5 + 3;
```

Here you see I've set the `totalCost` to 5, and then updated it again to be 5 + 3 (which I could just write as 8, obviously).

Types

Before continuing, you will notice that so far I've set all the variables as nondecimal numbers. In JavaScript (and all programming languages), there is the notion of types. There are a number of types that a variable can be. The most common are the number type and the string type. There's also the Boolean type, which can only be set to `true` or `false`. When working with JavaScript, you usually won't have to worry too much about types. Even if a variable is declared with an integer value (e.g., 5), it can then be updated to be a string value, as follows:

```
var testVariable = 5;
testVariable = "Jack";
```

You can see here I've changed the type of `testVariable` from an integer to string, and JavaScript doesn't complain at all. Along with strings, numbers, and Booleans, the two other types you need to concern yourself with (for now) are arrays and objects. I will cover both in more detail very shortly, but for now, just know that an *array* is essentially a list of values. These values can be of any type, and not all values within an array have to be the same type. You can create an array by listing values between square braces, like so:

```
var squares = [1, 4, 9, 16, 25];

var mixed = [1, "Jack", 5, true, 6.5, "Franklin"];
```

For now, that's all you need to know about arrays. I will cover them in more detail before this chapter is over.

The other type, object, is more easily explained with an example. Let's say you have the concept of a car in your application. This car has a certain number of wheels and seats, is a certain color, and has a maximum speed. You could model this car with four separate variables:

```
var carWheelCount = 4;
var carColour = "red";
var carSeatCount = 5;
var carMaximumSpeed = 99;
```

It would be easier if you could have just one variable—`car`—that contained all this