Table of Contents (the real thing)

Get ready for your first superpower. You’ve got some programming under your belt; now it’s time to really move things along with **functions**. Functions give you the power to write code that can be applied to all sorts of different circumstances, code that can be **reused** over and over, code that is much more **manageable,** code that can be **abstracted** away and given a simple name so you can forget all the complexity and get on with the important stuff. You’re going to find not only that functions are your gateway from scripter to programmer, but that they’re the key to the JavaScript programming style. In this chapter, we’re going to start with the basics—the mechanics, the ins and outs of how functions really work—and then you’ll keep honing

your function skills throughout the rest of the book. So, let's get a good foundation started, *now*.

Chapter 4

There's more to JavaScript than numbers, strings, and booleans. So far you've been writing JavaScript code with **primitives**—strings, numbers, and booleans, like "Fido", 23, and true. You can do a lot with primitive types, but at some point you've got to deal with **more data**. Say, all the items in a shopping cart, or all the songs in a playlist, or a set of stars and their apparent magnitude, or an entire product catalog. For that you need a little more *oomph*. The type of choice for this kind of ordered data is a JavaScript **array**, and in this chapter we're going to walk through how to put your data into an array, how to pass it around, and how to operate on it. We'll be looking at a few other ways to **structure your data** in later chapters, but let's get started with arrays.

So far, you've been using primitives and arrays in your code. And you've approached coding in quite a **procedural manner**, using simple statements, conditionals, and for/while loops with functions—that's not exactly **object-oriented**. In fact, it's not object-oriented *at all!* You did use a few objects here and there without really knowing it, but you haven't written any of your own objects yet. Well, the time has come to leave this boring procedural town behind and create some **objects** of your own. In this chapter, you're going to find out why using objects is going to make your life so much better—well, better in a **programming sense** (we can't really help you with your fashion sense *and* your JavaScript skills all in one book). Just a warning: once you've discovered objects, you'll never want to come back. Send us a postcard when you get there.

You've come a long way with JavaScript. In fact, you've evolved from a newbie to a scripter to, well, a **programmer**. But, there's something missing. To really begin leveraging your JavaScript skills, you need to know how to interact with the web page your code lives in. Only by doing that are you going to be able to write pages that are **dynamic**, pages that react, that respond, that update themselves after they've been loaded. So how do you interact with the page? By using the **DOM**, otherwise known as the **document object model**. In this chapter, we're going to break down the DOM and show you how you can use it, along with JavaScript, to teach your pages a few new tricks.

It's time to get serious about our types. One of the great things about JavaScript is you can get a long way without knowing a lot of details of the language. But to truly **master the language**, get that promotion, and get on to the things you really want to do in life, you have to rock at **types**. Remember what we said about JavaScript back in Chapter 1? That it didn't have the luxury of a silver-spoon, academic, peer-reviewed language definition? Well, that's true, but the lack of an academic life didn't stop Steve Jobs and Bill Gates, and it didn't stop JavaScript either. It does mean that JavaScript doesn't have the...well, the most thought-out type system, and we'll find a few **idiosyncrasies**

along the way. But don't worry, in this chapter we're going to nail all that down, and soon you'll be able to avoid all those embarrassing moments with types.

"The truth is out there..."

"Watch out, you might bump into undefined when you aren't expecting it..."

"How to use null"

"Dealing with NaN"

"It gets even weirder..."

"We have a confession to make"

"Understanding the equality operator (otherwise known as ==)"

"How equality converts its operands"

"How to get strict with equality"

"Even more type conversions"

"How to determine if two objects are equal"

"The truthy is out there..."

"What JavaScript considers falsey"

"The Secret Life of Strings"

"How a string can look like a primitive and an object"

"How template literals work"

"A five-minute tour of string properties and methods"

"Chair Wars"

Chapter 8

Put on your toolbelt. That is, the toolbelt with all your new coding skills, your knowledge of the DOM, and even some HTML and CSS. We're going to bring everything together in this chapter to create our first true **web application**. No more **silly toy games** with one battleship and a single row of hiding places. In this chapter, we're building the **entire experience**: a nice big game board, multiple ships, and user input right in the web page. We're going to create the page struc-

ture for the game with HTML, visually style the game with CSS, and write JavaScript to code the game's behavior. Get ready: this is an all-out, pedal-to-the-metal development chapter where we're going to lay down some serious code.

After this chapter, you're going to realize you aren't in Kansas anymore. Up until now, you've been writing code that typically executes from top to bottom—sure, your code might be a little more complex than that, and make use of a few functions, objects, and methods, but at some point the code just runs its course. Now, we're awfully sorry to break this to you this late in the book, but that's **not how you**

typically write JavaScript code. Rather, most JavaScript is written to **react to events**. What kind of events? Well, how about a user clicking on your page, data arriving from the network, timers expiring in the browser, changes happening in the DOM...and that's just a few examples. In fact, all kinds of events are happening **all the time**, behind the scenes, in your browser. In this chapter we're going rethink our approach to JavaScript coding, and look at how and why we should write code that reacts to events.

Chapter 10

Know functions, then rock. Every art, craft, and discipline has a key principle that separates the intermediate players from the rock-star virtuosos—when it comes to JavaScript, it's truly understanding **functions** that makes the difference. Functions are fundamental to JavaScript, and many of the techniques we use to **design and organize** code depend on advanced knowledge and use of functions. The path to learning functions at this level is an interesting and often mind-bending one, so get ready...The next two chapters are going to be a bit like taking Willy Wonka's tour of the chocolate factory—you're going to encounter some wild, wacky, and wonderful things as you learn more about JavaScript functions.

You've put functions through their paces, but there's more to learn. In this chapter, we take it further; we get hard-core. We'll show you how to up your syntax game with advanced techniques for handling arguments, parameters, and assignments. We'll then take another look at scope and some of the finer points of how JavaScript manages it. This journey through the subtleties of scope brings us to the heart of closures—a concept often shrouded in mystery but pivotal in mastering JavaScript. In the end, you'll be more expressive with your JavaScript than you thought possible.

We've covered a lot of ground, and you're almost finished with this book. We'll miss you, but before we let you go, we wouldn't feel right about sending you out into the world without a little more preparation. We can't possibly fit everything you'll need to know into this relatively small chapter. Actually, we *did* originally include everything you need to know about JavaScript programming (not already covered by the other chapters), by reducing the type point size to .00004. It all fit, but nobody could read it. So we threw most of it away, and kept the best bits for this "top ten" Appendix A.