

# Table of Contents (the real thing)

## How to use this book *Intro*

**Your brain on JavaScript.** Here *you* are trying to *learn* something, while here your *brain* is doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing JavaScript?

"Who is this book for?"

"We know what you're thinking"

"Metacognition: thinking about thinking"

"Here's what YOU can do to bend your brain into submission"

"Read Me"

"Tech reviewers"

"Acknowledgments for the first edition"

## Chapter 1

**JavaScript gives you superpowers.** The **true programming language** of the web, JavaScript lets you **add behavior** to your web pages. No more dry, boring, static pages that just sit there looking at you—with JavaScript, you'll be able to reach out and touch your users, react to interesting events, grab data from the web to use in your pages, draw graphics right into those pages, and a lot more. And once you know JavaScript, you'll also be in a position to create **totally new** behaviors for your users.

You'll be in good company, too. JavaScript's not only one of the **most popular** programming languages, it's also **supported** in all modern browsers and is used in many environments outside of the browser. More on that later; for now, let's get started!

“The way JavaScript works”

“How you’re going to write JavaScript”

“How to get JavaScript into your page”

“JavaScript, you’ve come a long way...”

“How to make a statement”

“Variables and values”

“Constants, another kind of variable”

“Back away from that keyboard!”

“Express yourself”

“Doing things more than once”

“How the while loop works”

“Making decisions with JavaScript”

“And, when you need to make LOTS of decisions...”

“Reach out and communicate with your user”

“A closer look at console.log”

“Opening the console”

“Coding a Serious JavaScript Application”

“How do I add code to my page? (let me count the ways!)”

“We’re going to have to separate you two”

## Chapter 2

**You already know about variables, types, expressions... we could go on.** The point is, you already know a few things about JavaScript. In fact, you know enough to write some **real code**. Some code that does something interesting, some code that someone would want to use. What you’re lacking is the **real experience** of writing code, and we’re going to remedy that right here and now. How? By jumping in head first and coding up a casual game, all written in JavaScript. Our goal is ambitious, but we’re going to take it one step at a time. Come on, let’s get this started, and if you want to launch the next startup, we won’t stand in your way; the code is yours.

“Let’s build a Battleship game”

“First, a high-level design”

“Working through the pseudocode”

“Oh, before we go any further, don’t forget the HTML!”

“Writing the Simple Battleship code”

“Now let’s write the game logic”

“Step 1: Setting up the loop, getting some input”

“How prompt works”

“Step 2: Checking the user’s guess”

“Adding the hit detection code”

“Step 3: Hey, you sank my battleship!”

“Step 4: Provide some post-game analysis”

“Doing a little quality assurance”

“Can we talk about your verbosity...”

“Finishing the Simple Battleship game”

“The recipe for generating a random number”

“Congrats on your first true JavaScript program, and a short word about reusing code”

## Chapter 3

**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*.

[“What’s wrong with the code, anyway?”](#)

[“By the way, did we happen to mention FUNCTIONS?”](#)

[“Okay, but how does it actually work?”](#)

[“What can you pass to a function?”](#)

[“JavaScript is pass-by-value”](#)

[“Weird Functions”](#)

[“Functions can return things too”](#)

[“Tracing through a function with a return statement”](#)

[“Global and local variables”](#)

[“Knowing the scope of your local and global variables”](#)

[“There’s more to the story”](#)

[“Don’t forget to declare your locals!”](#)

[“The short lives of variables”](#)

## [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.

[“Can you help Bubbles-R-Us?”](#)

[“How to represent multiple values in JavaScript”](#)

[“How arrays work”](#)

“How big is that array anyway?”

“The Phrase-O-Matic”

“Meanwhile, back at Bubbles-R-Us...”

“How to iterate over an array”

“But wait, there’s a better way to iterate over an array”

“It’s that time again... can we talk about your verbosity?”

“Redoing the for loop with the post-increment operator”

“Creating an array from scratch (and adding to it)”

“And the winners are...”

“A quick survey of the code”

“Writing the printAndGetHighScore function”

“Refactoring the code using printAndGetHighScore”

“Putting it all together”

## Chapter 5

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.

“Did someone say “objects”?!”

“Thinking about properties...”

“How to create an object”

“What is “object-oriented” anyway?”

“How properties work”

“How does a variable hold an object? Inquiring minds want to know...”

“Comparing primitives and objects”

“Doing even more with objects”

“Does the taxi cut it?”

“Let’s talk a little more about passing objects to functions”

“The Auto-O-Matic”

“Oh, behave! Or, how to add behavior to your objects”

“Improving the drive method”

“Uh-oh, not so fast...”

“Why doesn’t the drive method know about the started property?”

“How “this” works”

“Method shorthand”

“How behavior affects state”

“Now let’s affect the behavior with the state”

“Congrats on your first objects!”

“Guess what? There are objects all around you!”

## Chapter 6

**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.

“In the last chapter, we left you with a little challenge...the “crack the code challenge””

“So what does the code do?”

“How JavaScript really interacts with your page”

“How to bake your very own DOM”

“A first taste of the DOM”

“Getting an element with getElementById”

“What, exactly, am I getting from the DOM?”

“Finding your inner HTML”

“What happens when you change the DOM”

“Don’t even think about running my code until the page is fully loaded!”

“You say “event handler,” I say “callback””

“Why stop now? Let’s take it further...”

“How to set an attribute with setAttribute”

“More fun with attributes!”



“Meanwhile, back at the solar system...”

“So what else is a DOM good for, anyway?”

## Chapter 7

**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.

“This time, let’s build a REAL Battleship game”

“Stepping back...to HTML and CSS”

“Creating the HTML page: the Big Picture”

“Adding some more style”

“Using the hit and miss classes”

“Designing the game”

“Implementing the view”

“The model”

“You’re gonna need a bigger boat...and game board”

“How we’re going to represent the ships”

“Putting it all together”

“Wait, can we talk about your verbosity again?”

“A view to a kill...”

“Implementing the controller”

“Processing the player’s guess”

“Getting a player’s guess”

“How to place ships”

“Avoiding a collision!”

“Congrats, it’s startup time!”

## Chapter 9

**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 to rethink our approach to JavaScript coding, and look at how and why we should write code that reacts to events.

[“What are events?”](#)

[“What's an event handler?”](#)

[“Creating an event handler”](#)

[“Getting your head around events...by creating a game”](#)

[“Implementing the game”](#)

[“Let's add some more images”](#)

[“How to reuse the same handler for all the images”](#)

[“How the event object works”](#)

[“Putting the event object to work”](#)

[“Events and queues”](#)

[“How setTimeout works”](#)

[“Finishing the image game”](#)

## [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.

[“The mysterious double life of the function keyword”](#)

[“How functions are values too”](#)

[“If functions are values, we can assign them to variables”](#)

[“Did we mention functions have first class status in JavaScript?”](#)

[“Taking a look at the other side of functions...”](#)

[“How to use an anonymous function”](#)

[“We need to talk about your verbosity, again”](#)

[“We can make the code even shorter with arrow functions”](#)

[“Creating arrow functions”](#)

[“Webville Cola”](#)

[“Understanding the array sort method”](#)

[“Putting it all together”](#)

[“Meanwhile, back at Webville Cola”](#)

[“Introducing higher-order functions”](#)

[“Filtering with higher-order functions”](#)

[“Don’t forget your anonymous and arrow functions”](#)

[“Using reduce to get the total cases sold”](#)

[“Chaining map, filter, and reduce”](#)

[“Iterating with forEach”](#)

## [Chapter 11](#)

**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.

“Getting serious about functional syntax”

“Spreading out your arguments”

“There’s something we haven’t told you about functions...”

“Function declarations are “hoisted””

“We’ve done the function declarations; now we do everything else”

“We need to talk about scope”

“Taking functions beyond global scope”

“A lexical scope refresher”

“Another look at our outer/inner functions”

“Using scope for encapsulation”

“Two important JavaScript scope rules”

“Solving the mystery”

“How to make a closure”

“Using closures to implement a magic counter”

“Looking behind the curtain...”

“Implementing a counter with a closure”

“How makeTimer works”

“Implementing onlyOnceMaker”

## Chapter 12

So, far we’ve been crafting objects by hand. For each object, we’ve used an **object literal** to specify each and every property. That’s okay on a small scale, but for serious code we need something better. That’s where **classes** come in. With classes we can create objects much more easily, and we can create objects that all adhere to the same **design blueprint**—meaning we can use classes to ensure each object has the same properties and includes the same methods. And with classes we can write object code that is much more **concise** and a lot less error-prone when we’re creating lots of objects. So, let’s get started...

“Creating objects with object literals”

[“Using conventions for objects”](#)

[“Introducing classes”](#)

[“How to define a class”](#)

[“How to create an object from a class”](#)

[“How classes work”](#)

[“Let’s add some methods”](#)

[“It’s production time!”](#)

[“The basic Car class”](#)

[“Implementing the Taxi class with extends”](#)

[“Adding new methods to the Taxi class”](#)

[“Implementing the RocketCar class”](#)

[“Using an object literal to clean up our constructor”](#)

[“Reworking the Car constructor”](#)

[“Accessor properties”](#)

[“Using getters”](#)

[“What’s a getter without a setter?”](#)

[“Static properties and methods”](#)

[“Counting our car production”](#)

## [Appendix A](#)

**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](#).

[“#1 Modules”](#)

“#2 JSON”

“#3 Promises”

“#4 Destructuring assignment”

“#5 Symbols and BigInt”

“#6 Map and Set”

“#7 Doing more with the DOM”

“#8 The window object”

“#9 Server-side JavaScript”

“#10 Recursion”