

Head First JavaScript Programming

Wouldn't it be dreamy if there was
a JavaScript book that was more
fun than going to the dentist and
more revealing than an IRS form?
It's probably just a fantasy...



Eric T. Freeman
Elisabeth Robson

O'REILLY®

Beijing • Cambridge • Köln • Sebastopol • Tokyo

Table of Contents (summary)

	Intro	xxv
1	A quick dip into JavaScript: <i>Getting your feet wet</i>	1
2	Writing real code: <i>Going further</i>	43
3	Introducing functions: <i>Getting functional</i>	79
4	Putting some order in your data: <i>Arrays</i>	125
5	Understanding objects: <i>A trip to Objectville</i>	173
6	Interacting with your web page: <i>Getting to know the DOM</i>	229
7	Types, equality, conversion, and all that jazz: <i>Serious types</i>	265
8	Bringing it all together: <i>Building an app</i>	317
9	Asynchronous coding: <i>Handling events</i>	381
10	First-class functions: <i>Liberated functions</i>	429
11	Anonymous functions, scope, and closures: <i>Serious functions</i>	475
12	Advanced object construction: <i>Creating objects</i>	521
13	Using prototypes: <i>Extra-strength objects</i>	563
	Appendix: The Top Ten Topics (we didn't cover): <i>Leftovers</i>	623

Table of Contents (the real thing)

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 programming?



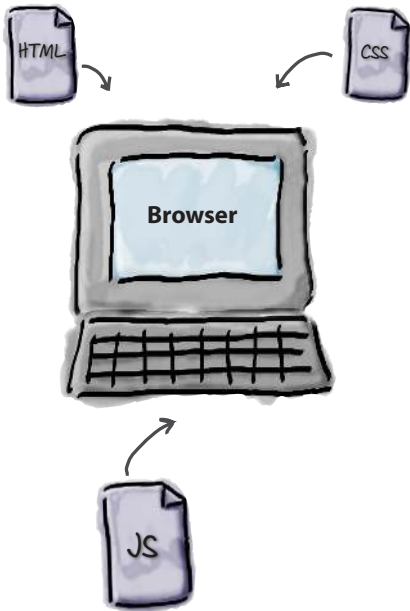
Who is this book for?	xxvi
We know what you're thinking	xxvii
We think of a "Head First" reader as a learner.	xxviii
Metacognition: thinking about thinking	xxix
Here's what WE did:	xxx
Here's what YOU can do to bend your brain into submission	xxxi
Read Me	xxxii
Tech Reviewers	xxxv
Acknowledgments*	xxxvi

a quick dip into javascript

1

Getting your feet wet

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're going to 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 in your web 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.



The way JavaScript works	2
How you're going to write JavaScript	3
How to get JavaScript into your page	4
JavaScript, you've come a long way baby...	6
How to make a statement	10
Variables and values	11
Back away from that keyboard!	12
Express yourself	15
Doing things more than once	17
How the while loop works	18
Making decisions with JavaScript	22
And, when you need to make LOTS of decisions	23
Reach out and communicate with your user	25
A closer look at console.log	27
Opening the console	28
Coding a Serious JavaScript Application	29
How do I add code to my page? (let me count the ways)	32
We're going to have to separate you two	33



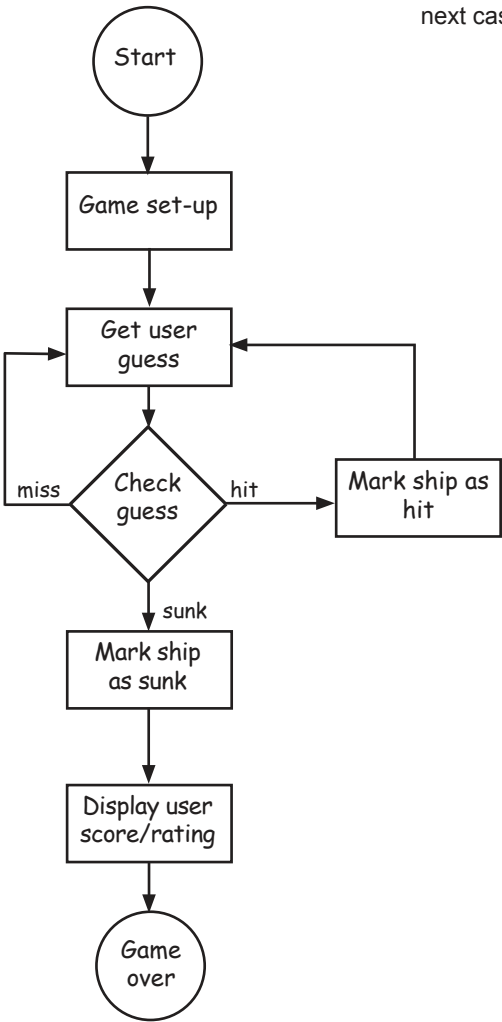
writing real code



2

Going further

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 casual startup, we won't stand in your way; the code is yours.

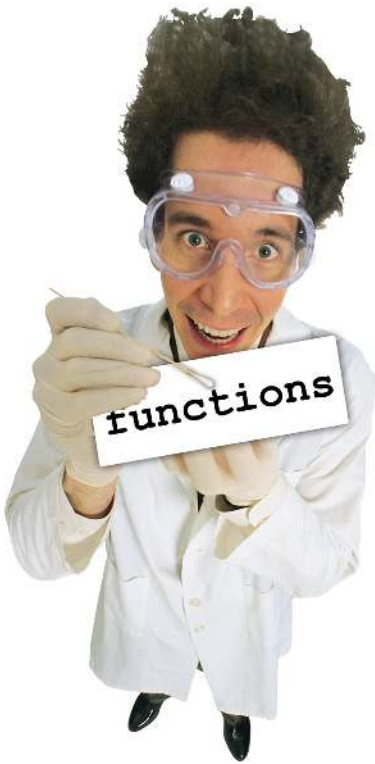


Let's build a Battleship game	44
Our first attempt...	44
First, a high-level design	45
Working through the Pseudocode	47
Oh, before we go any further, don't forget the HTML!	49
Writing the Simple Battleship code	50
Now let's write the game logic	51
Step One: setting up the loop, getting some input	52
How prompt works	53
Checking the user's guess	54
So, do we have a hit?	56
Adding the hit detection code	57
Provide some post-game analysis	58
And that completes the logic!	60
Doing a little Quality Assurance	61
Can we talk about your verbosity...	65
Finishing the Simple Battleship game	66
How to assign random locations	67
The world-famous recipe for generating a random number	67
Back to do a little more QA	69
Congrats on your first true JavaScript program, and a short word about reusing code	71

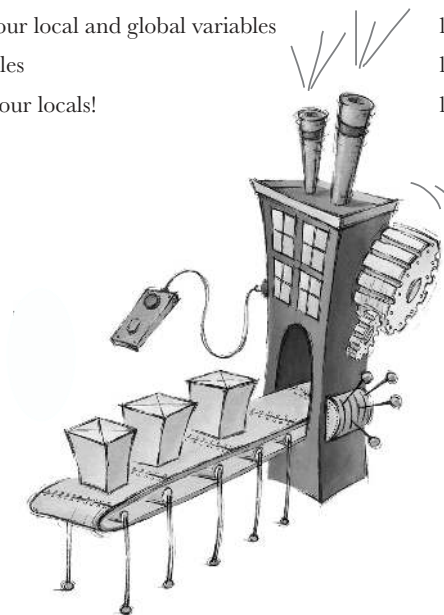
introducing functions

3 Getting functional

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, 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?	81
By the way, did we happen to mention FUNCTIONS?	83
Okay, but how does it actually work?	84
What can you pass to a function?	89
JavaScript is pass-by-value.	92
Weird Functions	94
Functions can return things too	95
Tracing through a function with a return statement	96
Global and local variables	99
Knowing the scope of your local and global variables	101
The short lives of variables	102
Don't forget to declare your locals!	103





putting some order in your data

4 Arrays There’s more to JavaScript than numbers, strings and booleans. So far you’ve been writing JavaScript code with **primitives**—simple strings, numbers and booleans, like “Fido”, 23, and true. And 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 we need a little more *ummph*. 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?	126
How to represent multiple values in JavaScript	127
How arrays work	128
How big is that array anyway?	130
The Phrase-O-Matic	132
Meanwhile, back at Bubbles-R-Us...	135
How to iterate over an array	138
But wait, there’s a better way to iterate over an array	140
Can we talk about your verbosity?	146
Redoing the for loop with the post-increment operator	147
Quick test drive	147
Creating an array from scratch (and adding to it)	151
And the winners are...	155
A quick survey of the code...	157
Writing the printAndGetHighScore function	158
Refactoring the code using printAndGetHighScore	159
Putting it all together...	161

understanding objects

5

A trip to Objectville

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*! We 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 to 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”?!	174
Thinking about properties...	175
How to create an object	177
What is Object-Oriented Anyway?	180
How properties work	181
How does a variable hold an object? Inquiring minds want to know...	186
Comparing primitives and objects	187
Doing even more with objects...	188
Stepping through pre-qualification	190
Let's talk a little more about passing objects to functions	192
Oh Behave! Or, how to add behavior to your objects	198
Improving the drive method	199
Why doesn't the drive method know about the started property?	202
How this works	204
How behavior affects state... Adding some Gas-o-line	210
Now let's affect the behavior with the state	211
Congrats on your first objects!	213
Guess what? There are objects all around you! (and they'll make your life easier)	214



interacting with your web page

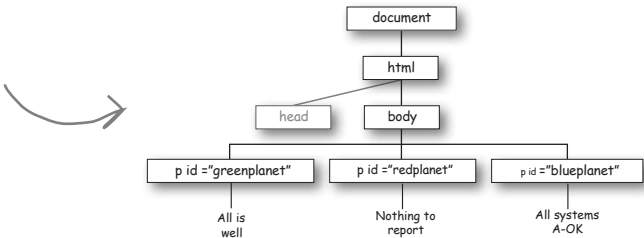
6

Getting to know the DOM

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 see just how we can use it, along with JavaScript, to teach your page a few new tricks.



The “crack the code challenge.”	230
So what does the code do?	231
How JavaScript really interacts with your page	233
How to bake your very own DOM	234
A first taste of the DOM	235
Getting an element with getElementById	240
What, exactly, am I getting from the DOM?	241
Finding your inner HTML	242
What happens when you change the DOM	244
A test drive around the planets	247
Don’t even think about running my code until the page is fully loaded!	249
You say “event handler,” I say “callback”	250
How to set an attribute with setAttribute	255
More fun with attributes! (you can GET attributes too)	256
Don’t forget getElementById can return null too!	256
Any time you ask for something, you need to make sure you got back what you expected...	256
So what else is a DOM good for anyway?	258

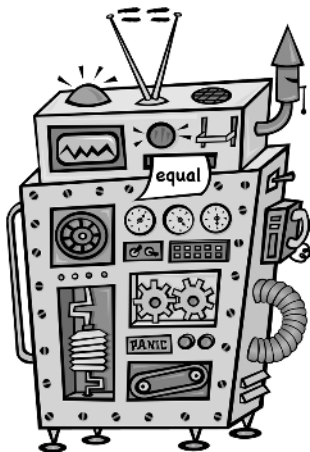


types, equality, conversion, and all that jazz

7

Serious types

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 way back about JavaScript? That it didn't have the luxury of a silver-spoon, academic, peer-reviewed language definition? Well that's true, but the 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.

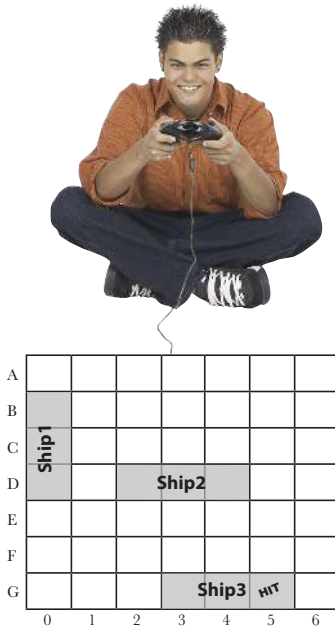


The truth is out there...	266
Watch out, you might bump into undefined when you aren't expecting it...	268
How to use null	271
Dealing with NaN	273
It gets even weirder	273
We have a confession to make	275
Understanding the equality operator (otherwise known as ==)	276
How equality converts its operands (sounds more dangerous than it actually is)	277
How to get strict with equality	280
Even more type conversions...	286
How to determine if two objects are equal	289
The truthy is out there...	291
What JavaScript considers falsey	292
The Secret Life of Strings	294
How a string can look like a primitive and an object	295
A five-minute tour of string methods (and properties)	297
Chair Wars	301

bringing it all together

8 Building an app

Put on your toolbelt. That is, the toolbelt with all your new coding skills, your knowledge of the DOM, and even some HTML & 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 structure 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.



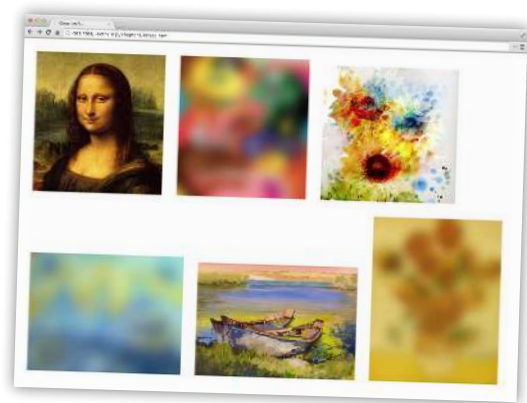
This time, let's build a REAL Battleship game	318
Stepping back... to HTML and CSS	319
Creating the HTML page: the Big Picture	320
Adding some more style	324
Using the hit and miss classes	327
How to design the game	329
Implementing the View	331
How displayMessage works	331
How displayHit and displayMiss work	333
The Model	336
How we're going to represent the ships	338
Implementing the model object	341
Setting up the fire method	342
Implementing the Controller	349
Processing the player's guess	350
Planning the code...	351
Implementing parseGuess	352
Counting guesses and firing the shot	355
How to add an event handler to the Fire! button	359
Passing the input to the controller	360
How to place ships	364
Writing the generateShip method	365
Generate the starting location for the new ship	366
Completing the generateShip method	367

asynchronous coding

9

Handling events

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 learn how and why we should write code that reacts to events.



What are events?	383
What's an event handler?	384
How to create your first event handler	385
Test drive your event	386
Getting your head around events... by creating a game	388
Implementing the game	389
Test drive	390
Let's add some more images	394
Now we need to assign the same event handler to each image's onclick property	395
How to reuse the same handler for all the images	396
How the event object works	399
Putting the event object to work	401
Test drive the event object and target	402
Events and queues	404
Even more events	407
How setTimeout works	408
Finishing the image game	412
Test driving the timer	413



first class functions

10

Liberated functions

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... This chapter is going to be a bit like Willy Wonka giving a 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	430
Function declarations versus function expressions	431
Parsing the function declaration	432
What's next? The browser executes the code	433
Moving on... The conditional	434
How functions are values too	439
Did we mention functions have First Class status in JavaScript?	442
Flying First Class	443
Writing code to process and check passengers	444
Iterating through the passengers	446
Passing a function to a function	447
Returning functions from functions	450
Writing the flight attendant drink order code	451
The flight attendant drink order code: a different approach	452
Taking orders with first class functions	454
Webville Cola	457
How the array sort method works	459
Putting it all together	460
Take sorting for a test drive	462



anonymous functions, scopes, and closures

11

Serious 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’re going to show you how to **really handle** functions. This won’t be a super long chapter, but it will be intense, and at the end you’re going to be more expressive with your JavaScript than you thought possible. You’re also going to be ready to take on a coworker’s code, or jump into an open source JavaScript library, because we’re going to cover some common coding idioms and conventions around functions. And if you’ve never heard of an **anonymous function** or a **closure**, boy are you in the right place.



Taking a look at the other side of functions...	476
How to use an anonymous function	477
We need to talk about your verbosity, again	479
When is a function defined? It depends...	483
What just happened? Why wasn't fly defined?	484
How to nest functions	485
How nesting affects scope	486
A little review of lexical scope	488
Where things get interesting with lexical scope	489
Functions Revisited	491
Calling a function (revisited)	492
What the heck is a closure?	495
Closing a function	496
Using closures to implement a magic counter	498
Looking behind the curtain...	499
Creating a closure by passing a function expression as an argument	501
The closure contains the actual environment, not a copy	502
Creating a closure with an event handler	503
How the Click me! closure works	506

advanced object construction

12

Creating objects

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 **object constructors** come in. With constructors we can create objects much more easily, and we can create objects that all adhere to the same **design blueprint**—meaning we can use constructors to ensure each object has the same properties and includes the same methods. And with constructors 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 and after this chapter you’ll be talking constructors just like you grew up in Objectville.



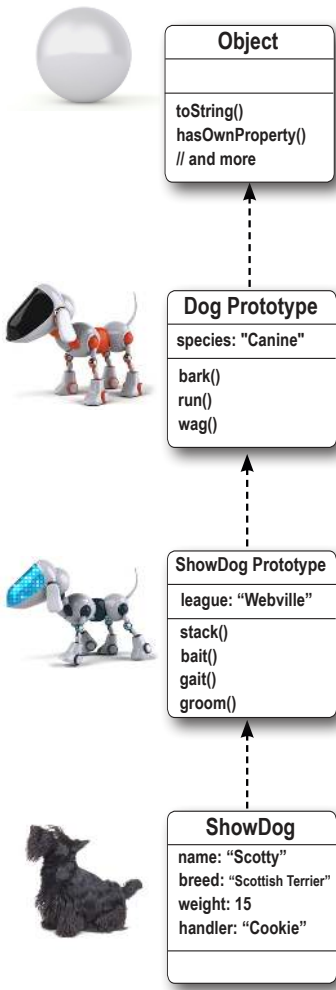
Creating objects with object literals	522
Using conventions for objects	523
Introducing Object Constructors	525
How to create a Constructor	526
How to use a Constructor	527
How constructors work	528
You can put methods into constructors as well	530
It’s Production Time!	536
Let’s test drive some new cars	538
Don’t count out object literals just yet	539
Rewiring the arguments as an object literal	540
Reworking the Car constructor	541
Understanding Object Instances	543
Even constructed objects can have their own independent properties	546
Real World Constructors	548
The Array object	549
Even more fun with built-in objects	551

using prototypes

13

Extra strength objects

Learning how to create objects was just the beginning. It's time to put some muscle on our objects. We need more ways to create **relationships** between objects and to **share code** among them. And, we need ways to extend and enhance existing objects. In other words, we need more tools. In this chapter, you're going to see that JavaScript has a very powerful **object model**, but one that is a bit different than the status quo object-oriented language. Rather than the typical class-based object-oriented system, JavaScript instead opts for a more powerful **prototype** model, where objects can inherit and extend the behavior of other objects. What is that good for? You'll see soon enough. Let's get started...



Hey, before we get started, we've got a better way to diagram our objects	565
Revisiting object constructors: we're reusing code, but are we being efficient?	566
Is duplicating methods really a problem?	568
What are prototypes?	569
Inheriting from a prototype	570
How inheritance works	571
Overriding the prototype	573
How to set up the prototype	576
Prototypes are dynamic	582
A more interesting implementation of the sit method	584
One more time: how the sitting property works	585
How to approach the design of the show dogs	589
Setting up a chain of prototypes	591
How inheritance works in a prototype chain	592
Creating the show dog prototype	594
Creating a show dog Instance	598
A final cleanup of show dogs	602
Stepping through Dog.call	604
The chain doesn't end at dog	607
Using inheritance to your advantage...by overriding built-in behavior	608
Using inheritance to your advantage...by extending a built-in object	610
Grand Unified Theory of Everything	612
Better living through objects	612
Putting it all together	613
What's next?	613

14

Appendix: Leftovers

The top ten topics (we didn't cover)

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. This really *is* the end of the book. Except for the index, of course (a must-read!).



#1 jQuery	624
#2 Doing more with the DOM	626
#3 The Window Object	627
#4 Arguments	628
#5 Handling exceptions	629
#6 Adding event handlers with addEventListener	630
#7 Regular Expressions	632
#8 Recursion	634
#9 JSON	636
#10 Server-side JavaScript	637