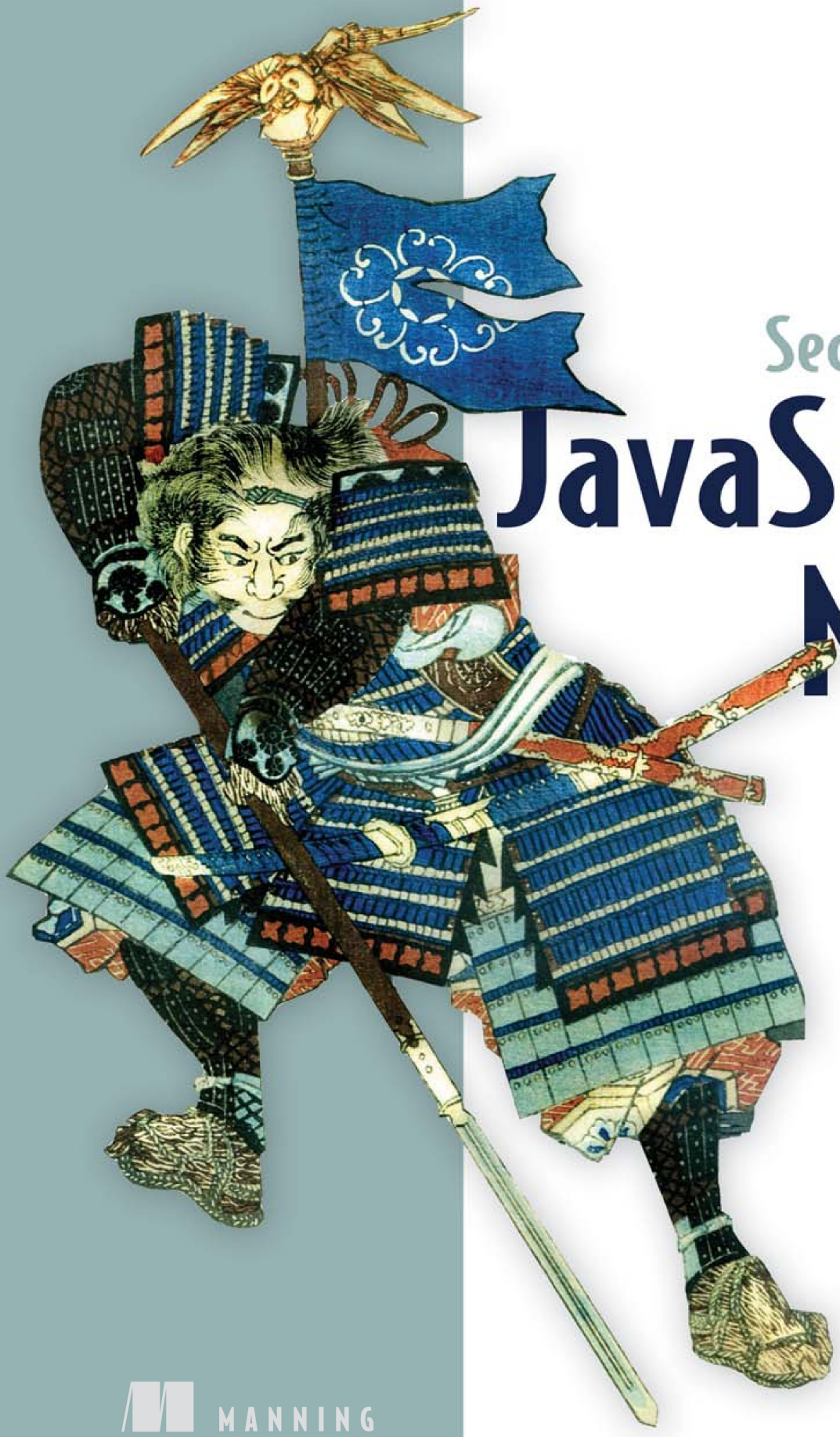

Secrets of the JavaScript Ninja

SECOND EDITION

John Resig
Bear Bibeault
Josip Maras

MANNING

ES6 cheat sheet

Template literals embed expressions into strings: `` `${ninja}` ``.

Block-scoped variables:

- Use the new `let` keyword to create block-level variables: `let ninja = "Yoshi"`.
- Use the new `const` keyword to create block-level constant variables whose value can't be reassigned to a completely new value: `const ninja = "Yoshi"`.

Function parameters:

- *Rest parameters* create an array from arguments that weren't matched to parameters:

```
function multiMax(first, ...remaining){ /*...*/}multiMax(2, 3, 4, 5); //first: 2;
     remaining: [3, 4, 5]
```

- *Default parameters* specify default parameter values that are used if no value is supplied during invocation:

```
function do(ninja, action = "skulk"){ return ninja + " " + action;}
do("Fuma"); //"Fuma skulk"
```

Spread operators expand an expression where multiple items are needed: `[...items, 3, 4, 5]`.

Arrow functions let us create functions with less syntactic clutter. They don't have their own `this` parameter. Instead, they inherit it from the context in which they were created:

```
const values = [0, 3, 2, 5, 7, 4, 8, 1];
values.sort((v1,v2) => v1  - v2); /*OR*/ values.sort((v1,v2) => { return v1  - v2;});
value.forEach(value => console.log(value));
```

Generators generate sequences of values on a per-request basis. Once a value is generated, the generator suspends its execution without blocking. Use `yield` to generate values:

```
function *IdGenerator(){
  let id = 0;
  while(true){ yield ++id; }
}
```

Promises are placeholders for the result of a computation. A promise is a guarantee that eventually we'll know the result of some computation. The promise can either succeed or fail, and once it has done so, there will be no more changes:

- Create a new promise with `new Promise((resolve, reject) => {});`.
- Call the `resolve` function to explicitly resolve a promise. Call the `reject` function to explicitly reject a promise. A promise is implicitly rejected if an error occurs.
- The promise object has a `then` method that returns a promise and takes in two callbacks, a success callback and a failure callback:

```
myPromise.then(val => console.log("Success"), err => console.log("Error"));
```

- Chain in a `catch` method to catch promise failures: `myPromise.catch(e => alert(e));`.

(continued on inside back cover)

Praise for the First Edition

Finally, from a master, a book that delivers what an aspiring JavaScript developer requires to learn the art of crafting effective cross-browser JavaScript.

—Glenn Stokol, Senior Principal Curriculum Developer, Oracle Corporation

Consistent with the jQuery motto, "Write less, do more."

—André Roberge, Université Saint-Anne

Interesting and original techniques.

—Scott Sauyet, Four Winds Software

Read this book, and you'll no longer blindly plug in a snippet of code and marvel at how it works—you'll understand "why" it works.

—Joe Litton, Collaborative Software Developer, JoeLitton.net

Will help you raise your JavaScript to the realm of the masters.

—Christopher Haupt, greenstack.com

The stuff ninjas need to know.

—Chad Davis, author of *Struts 2 in Action*

Required reading for any JavaScript Master.

—John J. Ryan III, Princigration LLC

This book is a must-have for any serious JS coder. Your knowledge of the language will dramatically expand.

—S., Amazon reader

Secrets of the JavaScript Ninja, Second Edition

JOHN RESIG
BEAR BIBEAULT
and JOSIP MARAS

MANNING
Shelter Island

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964
Email: orders@manning.com

Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964

Development editor: Dan Maharry
Technical development editor: Gregor Zurowski
Review editor: Ozren Harlovic
Project editor: Tiffany Taylor
Copy editor: Sharon Wilkey
Proofreader: Alyson Brener
Technical proofreaders: Mathias Bynens, Jon Borgman
Typesetter: Gordan Salinovic
Cover designer: Marija Tudor

ISBN 9781617292859
Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 – EBM – 21 20 19 18 17 16

contents

author's introduction

It's incredible to think of how much the world of JavaScript has changed since I first started writing *Secrets of the JavaScript Ninja* back in 2008. The world in which we write JavaScript now, while still being largely centered around the browser, is nearly unrecognizable.

The popularity of JavaScript as a full-featured, cross-platform language has exploded. Node.js is a formidable platform against which countless production applications are developed. Developers are actually writing applications in one language—JavaScript—that are capable of running in a browser, on a server, and even in a native app on a mobile device.

It's more important now, than ever before, that a developer's knowledge of the JavaScript language be at its absolute peak. Having a fundamental understanding of the language and the ways in which it can be best written will allow you to create applications that can work on virtually any platform, which is a claim that few other languages can legitimately boast.

Unlike previous eras in the growth of JavaScript, there hasn't been equal growth in platform incompatibilities. It used to be that you would salivate over the thought of using the most basic new browser features and yet be stymied by outdated browsers that had far too much market share. We've entered a harmonious time in which most users are on rapidly updated browsers that compete to be the most standards-compliant platform around. Browser vendors even go out of their way to develop features specifically targeted at developers, hoping to make their lives easier.

The tools that we have now, provided by browsers and the open source community, are light years ahead of old practices. We have a plethora of testing frameworks to choose from, the ability to do continuous integration testing, generate code-coverage reports, performance-test on real mobile devices around the globe, and even automatically load up virtual browsers on any platform to test from.

The first edition of the book benefited tremendously from the development insight that Bear Bibeault provided. This edition has received substantial help from Josip Maras to explore the concepts behind ECMAScript 6 and 7, dive into testing best practices, and understand the techniques employed by popular JavaScript frameworks.

All of this is a long way of saying: how we write JavaScript has changed substantially. Fortunately, this book is here to help you keep on top of the current best practices. Not only that, but it'll help you improve how you think about your development practices as a whole to ensure that you'll be ready for writing JavaScript well into the future.

John Resig

acknowledgments

The number of people involved in writing a book would surprise most people. It took a collaborative effort on the part of many contributors with a variety of talents to bring the volume you are holding (or ebook that you are reading onscreen) to fruition.

The staff at Manning worked tirelessly with us to make sure this book attained the level of quality we hoped for, and we thank them for their efforts. Without them, this book would not have been possible. The "end credits" for this book include not only our publisher, Marjan Bace, and editor Dan Maharry, but also the following contributors: Ozren Harlovic, Gregor Zurowski, Kevin Sullivan, Janet Vail, Tiffany Taylor, Sharon Wilkey, Alyson Brener, and Gordan Salinovic.

Enough cannot be said to thank our peer reviewers who helped mold the final form of the book, from catching simple typos to correcting errors in terminology and code, and helping to organize the chapters in the book. Each pass through a review cycle ended up vastly improving the final product. For taking the time to review the book, we'd like to thank Jacob Andresen, Tidjani Belmansour, Francesco Bianchi, Matthew Halverson, Becky Huett, Daniel Lamb, Michael Lund, Kariem Ali Elkoush, Elyse Kolker Gordon, Christopher Haupt, Mike Hatfield, Gerd Klevesaat, Alex Lucas, Arun Noronha, Adam Scheller, David Starkey, and Gregor Zurowski.

Special thanks go to Mathias Bynens and Jon Borgman, who served as the book's technical proofreaders. In addition to checking each and every sample of example code in multiple environments, they also offered invaluable contributions to the technical accuracy of the text, located information that was originally missing, and kept abreast of the rapid changes to JavaScript and HTML5 support in the browsers.

John Resig

I would like to thank my parents for their constant support and encouragement over the years. They provided me with the resources and tools that I needed to spark my initial interest in programming—and they have been encouraging me ever since.

Bear Bibeault

The cast of characters I'd like to thank for this seventh go-around has a long list of "usual suspects," including, once again, the membership and staff at coderanch.com (formerly JavaRanch). Without my involvement in CodeRanch, I'd never have gotten the opportunity to begin writing in the first place, and so I sincerely thank Paul Wheaton and Kathy Sierra for starting the whole thing, as well as fellow staffers who gave me encouragement and support, including (but certainly not limited to) Eric Pascarello, Ernest Friedman-Hill, Andrew Monkhouse, Jeanne Boyarsky, Bert Bates, and Max Habibi.

My husband, Jay, and my dogs, Little Bear and Cozmo, get the usual warm thanks for putting up with the shadowy presence who shared their home and rarely looked up from his keyboard except to curse Word, the browsers, my fat-fingered lack of typing skills, or anything else that attracted my ire while I was working on this project.

And finally, I'd like to thank my coauthors, John Resig and Josip Maras, without whom this project would not exist.

Josip Maras

The biggest thanks go to my wife, Josipa, for putting up with all the hours that went into writing this book.

I would also like to thank Maja Stula, Darko Stipanicev, Ivica Crnkovic, Jan Carlson, and Bert Bates: all of them for guidance and useful advice, and some of them for being lenient on my "day job" assignments as book deadlines were approaching.

Finally, I would like to thank the rest of my family—Jere, two Marijas, Vitomir, and Zdenka—for always being there for me.

about this book

JavaScript is important. That wasn't always so, but it's true now. JavaScript has become one of the most important and most widely used programming languages today.

Web applications are expected to give users a rich user interface experience, and without JavaScript, you might as well just be showing pictures of kittens. More than ever, web developers need to have a sound grasp of the language that brings life to web applications.

And like orange juice and breakfast, JavaScript isn't just for browsers anymore. The language has long ago knocked down the walls of the browser and is being used on the server thanks to Node.js, on desktop devices and mobiles through platforms such as Apache Cordova, and even on embedded devices with Espruino and Tessel.

Although this book is primarily focused on JavaScript executed in the browser, the fundamentals of the language presented in this book are applicable across the board. Truly understanding the concepts and learning various tips and tricks will make you a better all-around JavaScript developer.

With more and more developers using JavaScript in an increasingly JavaScript world, it's more important than ever to grasp its fundamentals so you can become an expert ninja of the language.

Audience

If you aren't at all familiar with JavaScript, this probably shouldn't be your first book. Even if it is, don't worry too much; we try to present fundamental JavaScript concepts in a way that should be understandable even for relative beginners. But, to be honest,

this book will probably best fit people who already know some JavaScript and who wish to deepen their understanding of JavaScript as a language, as well as the browser as the environment in which JavaScript code is executed.

Roadmap

This book is organized to take you from an apprentice to a ninja in four parts.

Part 1 introduces the topic and sets the stage so that you can easily progress through the rest of the book:

- Chapter 1 introduces JavaScript the language and its most important features, while suggesting current best practices we should follow when developing applications, including testing and performance analysis.
- Because our exploration of JavaScript is made in the context of browsers, in chapter 2 we'll set the stage by introducing the lifecycle of client-side web applications. That will help us understand JavaScript's role in the process of developing web applications.

Part 2 focuses on one of the pillars of JavaScript: functions. We'll study why functions are so important in JavaScript, the different kinds of functions, as well as the nitty-gritty details of defining and invoking functions. We'll put a special focus on a new type of function—generator functions—which are especially helpful when dealing with asynchronous code:

- Chapter 3 begins our foray into the fundamentals of the language, starting, perhaps to your surprise, with a thorough examination of the *function* as defined by JavaScript. Although you may have expected the *object* to be the target of our first focus, a solid understanding of the function, and JavaScript as a functional language, begins our transformation from run-of-the-mill JavaScript coders to JavaScript ninjas!
- We continue this functional thread in chapter 4, by exploring the exact mechanism of invoking functions, as well as the ins and outs of implicit function parameters.
- Not being done with functions quite yet, in chapter 5 we take our discussion to the next level by studying two closely related concepts: *scopes* and *closures*. A key concept in functional programming, closures allow us to exert fine-grained control over the scope of objects that we declare and create in our programs. The control of these scopes is the key factor in writing code worthy of a ninja. Even if you stop reading after this chapter (but we hope you don't), you'll be a far better JavaScript developer than when you started.
- We conclude our exploration of functions in chapter 6, by taking a look at a completely new type of function (*generator functions*) and a new type of object (*promises*) that help us deal with asynchronous values. We'll also show you how to combine generators and promises to achieve elegance when dealing with asynchronous code.

Part 3 deals with the second pillar of JavaScript: objects. We'll thoroughly explore object orientation in JavaScript, and we'll study how to guard access to objects and how to deal with collections and regular expressions:

- Objects are finally addressed in chapter 7, where we learn exactly how JavaScript's slightly strange flavor of object orientation works. We'll also introduce a new addition to JavaScript: classes, which, deep under the hood, may not be exactly what you expect.
- We'll continue our exploration of objects in chapter 8, where we'll study different techniques for guarding access to our objects.
- In chapter 9, we'll put a special focus on different types of collections that exist in JavaScript; on arrays, which have been a part of JavaScript since its beginnings; and on maps and sets, which are recent addition to JavaScript.
- Chapter 10 focuses on regular expressions, an often-overlooked feature of the language that can do the work of scores of lines of code when used correctly. We'll learn how to construct and use regular expressions and how to solve some recurring problems elegantly, using regular expressions and the methods that work with them.
- In chapter 11, we'll learn different techniques for organizing our code into modules: smaller, relatively loosely coupled segments that improve the structure and organization of our code.

Finally, part 4 wraps up the book by studying how JavaScript interacts with our web pages and how events are processed by the browser. We'll finish the book by looking at an important topic, cross-browser development:

- Chapter 12 explores how we can dynamically modify our pages through DOM-manipulation APIs, and how we can handle element attributes, properties, and styles, as well as some important performance considerations.
- Chapter 13 discusses the importance of JavaScript's single-threaded execution model and the consequences this model has on the event loop. We'll also learn how timers and intervals work and how we can use them to improve the perceived performance of our web applications.
- Chapter 14 concludes the book by examining the five key development concerns with regard to these cross-browser issues: browser differences, bugs and bug fixes, external code and markup, missing features, and regressions. Strategies such as feature simulation and object detection are discussed at length to help us deal with these cross-browser challenges.

Code conventions

All source code in listings or in the text is in a `fixed-width font like this` to separate it from ordinary text. Method and function names, properties, XML elements, and attributes in the text are also presented in this same font.

In some cases, the original source code has been reformatted to fit on the pages. In general, the original code was written with page-width limitations in mind, but sometimes you may find a slight formatting difference between the code in the book and that provided in the source download. In a few rare cases, where long lines could not be reformatted without changing their meaning, the book listings contain line-continuation markers.

Code annotations accompany many of the listings; these highlight important concepts.

Code downloads

Source code for all the working examples in this book (along with some extras that never made it into the text) is available for download from the book's web page at https://manning.com/books/secrets-of-the-javascript-ninja-second-edition.

The code examples for this book are organized by chapter, with separate folders for each chapter. The layout is ready to be served by a local web server, such as the Apache HTTP Server. Unzip the downloaded code into a folder of your choice, and make that folder the document root of the application.

With a few exceptions, most of the examples don't require the presence of a web server and can be loaded directly into a browser for execution, if you so desire.

Author Online

The authors and Manning Publications invite you to the book's forum, run by Manning Publications, where you can make comments about the book, ask technical questions, and receive help from the authors and other users. To access and subscribe to the forum, point your browser to https://manning.com/books/secrets-of-the-javascript-ninja-second-edition and click the Author Online link. This page provides information on how to get on the forum once you are registered, what kind of help is available, and the rules of conduct in the forum.

Manning's commitment to our readers is to provide a venue where a meaningful dialogue between individual readers and between readers and the authors can take place. It's not a commitment to any specific amount of participation on the part of the authors, whose contribution to the book's forum remains voluntary (and unpaid). We suggest you try asking the authors some challenging questions, lest their interest stray!

The Author Online forum and the archives of previous discussions will be accessible from the publisher's website as long as the book is in print.

About the authors

John Resig is a staff engineer at Khan Academy and the creator of the jQuery JavaScript library. In addition to the first edition of *Secrets of the JavaScript Ninja*, he's also the author of the book *Pro JavaScript Techniques*.

John has developed a comprehensive Japanese woodblock print database and image search engine: Ukiyo-e.org. He's a board member of the Japanese Art Society of America and is a Visiting Researcher at Ritsumeikan University working on the study of Ukiyo-e.

John is located in Brooklyn, NY.

Bear Bibeault has been writing software for well over three decades, starting with a Tic-Tac-Toe program written on a Control Data Cyber supercomputer via a 100-baud teletype. Having two degrees in electrical engineering, Bear should be designing antennas or something like that, but since his first job with Digital Equipment Corporation, he has always been much more fascinated with programming.

Bear has also served stints with companies such as Dragon Systems, Works.com, Spredfast, Logitech, Caringo, and more than a handful of others. Bear even served in the U.S. military, leading and training a platoon of anti-tank infantry soldiers—skills that come in handy during scrum meetings. "That's *Sergeant* Bear to you, trainee!"

Bear is currently a senior front-end developer for a leading provider of object storage software that provides massive storage scalability and content protection.

In addition to the first edition of this book, Bear is also the author of a number of other Manning books, including *jQuery in Action* (first, second, and third editions), *Ajax in Practice*, and *Prototype and Scriptaculous in Action*; and he has been a technical reviewer for many of the web-focused "Head First" books by O'Reilly Publishing, such as *Head First Ajax*, *Head Rush Ajax*, and *Head First Servlets and JSP*.

In addition to his day job, Bear also writes books (duh!), runs a small business that creates web applications and offers other media services (but not wedding videography—never, ever wedding videography), and helps out at CodeRanch.com as a "marshal" (uber moderator).

When not planted in front of a computer, Bear likes to cook *big* food (which accounts for his jeans size), dabble in photography and video, ride his Yamaha V-Star, and wear tropical print shirts.

He works and resides in Austin, Texas, a city he loves, except for the completely insane traffic and drivers.

Josip Maras is a post-doctoral researcher in the faculty of electrical engineering, mechanical engineering, and naval architecture, University of Split, Croatia. He has a PhD in software engineering, with the thesis "Automating Reuse in Web Application Development," which among other things included implementing a JavaScript interpreter in JavaScript. During his research, he has published more than a dozen scientific conference and journal papers, mostly dealing with program analysis of client-side web applications.

When not doing research, Josip spends his time teaching web development, systems analysis and design, and Windows development (a couple hundred students over the last six years). He also owns a small software development business.

In his spare time, Josip enjoys reading, long runs, and, if the weather allows, swimming in the Adriatic.

about the cover illustration

The figure on the cover of *Secrets of the JavaScript Ninja, Second Edition* is captioned "Noh Actor, Samurai," from a woodblock print by an unknown Japanese artist of the mid-19th century. Derived from the Japanese word for *talent* or *skill*, Noh is a form of classical Japanese musical drama that has been performed since the 14th century. Many characters are masked, with men playing male and female roles. The samurai, a hero figure in Japan for hundreds of years, was often featured in the performances, and in this print the artist renders with great skill the beauty of the costume and the ferocity of the samurai.

Samurai and ninjas were both warriors excelling in the Japanese art of war, known for their bravery and cunning. Samurai were elite soldiers, well-educated men who knew how to read and write as well as fight, and they were bound by a strict code of honor called Bushido (The Way of the Warrior), which was passed down orally from generation to generation, starting in the 10th century. Recruited from the aristocracy and upper classes, analogous to European knights, samurai went into battle in large formations, wearing elaborate armor and colorful dress meant to impress and intimidate. Ninjas were chosen for their martial arts skills rather than their social standing or education. Dressed in black and with their faces covered, they were sent on missions alone or in small groups to attack the enemy with subterfuge and stealth, using any tactics to assure success; their only code was one of secrecy.

The cover illustration is from a set of three Japanese prints owned for many years by a Manning editor, and when we were looking for a ninja for the cover of this book, the

striking samurai print came to our attention and was selected for its intricate details, vibrant colors, and vivid depiction of a ferocious warrior ready to strike—and win.

At a time when it is hard to tell one computer book from another, Manning celebrates the inventiveness and initiative of the computer business with book covers based on two-hundred-year-old illustrations that depict the rich diversity of traditional costumes from around the world, brought back to life by prints such as this one.

Part 1

Warming up

This part of the book will set the stage for your JavaScript ninja training. In chapter 1, we'll look at the current state of JavaScript and explore some of the environments in which JavaScript code can be executed. We'll put a special focus on the environment where it all began—*the browser*—and we'll discuss some of the best practices when developing JavaScript applications.

Because our exploration of JavaScript will be done in the context of browsers, in chapter 2 we'll teach you about the lifecycle of client-side web applications and how executing JavaScript code fits into this lifecycle.

When you're finished with this part of the book, you'll be ready to embark on your training as a JavaScript ninja!

1 JavaScript is everywhere

This chapter covers

- The core language features of JavaScript
- The core items of a JavaScript engine
- Three best practices in JavaScript development

Let's talk about Bob. After spending a few years learning how to create desktop applications in C++, he graduated as a software developer in the early 2000s and then went out into the wide world. At that point, the web had just hit its stride, and everybody wanted to be the next Amazon. So the first thing he did was learn web development.

He learned some PHP so that he could dynamically generate web pages, which he usually sprinkled with JavaScript in order to achieve complex functionality such as form validation and even dynamic on-page clocks! Fast-forward a couple of years, and smartphones had become a thing, so anticipating a large new market opening up, Bob went ahead and learned Objective-C and Java to develop mobile apps that run on iOS and Android.

Over the years, Bob has created many successful applications that all have to be maintained and extended. Unfortunately, jumping daily between all these different programming languages and application frameworks has really started to wear down poor Bob.

Now let's talk about Ann.Two years ago, Ann graduated with a degree in software development, specializing in web- and cloud-based applications. She has created a few medium-sized web applications based on modern Model–view–controller (MVC) frameworks, along with accompanying mobile applications that run on iOS and Android. She has created a desktop application that runs on Linux, Windows, and OS X, and has even started building a serverless version of that application entirely based in the cloud. And *everything she has done has been written in JavaScript.*

That's extraordinary! What took Bob 10 years and 5 languages to do, Ann has achieved in 2 years and in *just one language.* Throughout the history of computing, it has been rare for a particular knowledge set to be so easily transferable and useful across so many different domains.

What started as a humble 10-day project back in 1995 is now one of the most widely used programming languages in the world. JavaScript is quite literally *everywhere,* thanks to more-powerful JavaScript engines and the introduction of frameworks such as Node, Apache Cordova, Ionic, and Electron, which have taken the language beyond the humble web page. And, like HTML, the language itself is now getting long overdue upgrades intended to make JavaScript even more suitable for modern application development.

In this book, we're going to make sure you know all you need to know about JavaScript so that, whether you're like Ann or like Bob, you can develop all sorts of applications on a green field or a brown field.

Do you know?

- What are Babel and Traceur, and why are they important to today's JavaScript developers?
- What are the core parts of any web browser's JavaScript API used by web applications?

1.1 Understanding the JavaScript language

As they advance through their careers, many JavaScript coders like Bob and Ann reach the point where they're actively using the vast number of elements that form the language. In many cases, however, those skills may not be taken beyond fundamental levels. Our guess is that this is often because JavaScript, using a C-like syntax, bears a surface resemblance to other widespread C-like languages (such as C# and Java), and thus leaves the impression of familiarity.

People often feel that if they know C# or Java, they already have a pretty solid understanding of how JavaScript works. But it's a trap! When compared to other mainstream languages, JavaScript is much more *functionally* oriented. Some JavaScript concepts differ fundamentally from those of most other languages.

These differences include the following:

- *Functions are first-class objects*—In JavaScript, functions coexist with, and can be treated like, any other JavaScript object. They can be created through literals, referenced by variables, passed around as function arguments, and even returned as function return values. We devote much of chapter 3 to exploring some of the wonderful benefits that functions as first-class objects bring to our JavaScript code.
- *Function closures*—The concept of function closures is generally poorly understood, but at the same time it fundamentally and irrevocably exemplifies the importance of functions to JavaScript. For now, it's enough to know that a function is *a closure when it actively maintains ("closes over") the external variables used in its body.* Don't worry for now if you don't see the many benefits of closures; we'll make sure all is crystal clear in chapter 5. In addition to closures, we'll thoroughly explore the many aspects of functions themselves in chapters 3 and 4, as well as identifier scopes in chapter 5.
- *Scopes*—Until recently, JavaScript didn't have block-level variables (as in other C-like languages); instead, we had to rely only on global variables and function-level variables.
- *Prototype-based object orientation*—Unlike other mainstream programming languages (such as C#, Java, and Ruby), which use class-based object orientation, JavaScript uses prototypes. Often, when developers come to JavaScript from class-based languages (such as Java), they try to use JavaScript as if it were Java, essentially writing Java's class-based code using the syntax of JavaScript. Then, for some reason, they're surprised when the results differ from what they expect. This is why we'll go deep into prototypes, how prototype-based object-orientation works, and how it's implemented in JavaScript.

JavaScript consists of a close relationship between objects and prototypes, and functions and closures. Understanding the strong relationships between these concepts can vastly improve your JavaScript programming ability, giving you a strong foundation for any type of application development, regardless of whether your JavaScript code will be executed in a web page, in a desktop app, in a mobile app, or on the server.

In addition to these fundamental concepts, other JavaScript features can help you write more elegant and more efficient code. Some of these are features that seasoned developers like Bob will recognize from other languages, such as Java and C++. In particular, we focus on the following:

- *Generators*, which are functions that can generate multiple values on a per-request basis and can suspend their execution between requests
- *Promises*, which give us better control over asynchronous code
- *Proxies*, which allow us to control access to certain objects
- *Advanced array methods*, which make array-handling code much more elegant

- *Maps*, which we can use to create dictionary collections; and *sets*, which allow us to deal with collections of unique items
- *Regular expressions*, which let us simplify what would otherwise be complicated pieces of code
- *Modules*, which we can use to break code into smaller, relatively self-contained pieces that make projects more manageable

Having a deep understanding of the fundamentals and learning how to use advanced language features to their best advantage can elevate your code to higher levels. Honing your skills to tie these concepts and features together will give you a level of understanding that puts the creation of any type of JavaScript application within your reach.

1.1.1 How will JavaScript evolve?

The ECMAScript committee, in charge of standardizing the language, has just finished the ES7/ES2016 version of JavaScript. The ES7 version is a relatively small upgrade to JavaScript (at least, when compared to ES6), because the committee's goal going forward is to focus on smaller, yearly incremental changes to the language.

In this book we thoroughly explore ES6 but also focus on ES7 features, such as the new async function, which will help you deal with asynchronous code (discussed in chapter 6).

NOTE When we cover features of JavaScript defined in ES6/ES2015 or ES7/ES2016, you'll see these icons alongside a link to information about whether they're supported by your browser.

Yearly incremental updates to the language specification are excellent news, but this doesn't mean that web developers will instantly have access to the new features after the specification has been released. JavaScript code has to be executed by a JavaScript engine, so we're often left waiting impatiently for updates to our favorite JavaScript engines that incorporate these new and exciting features.

Unfortunately, although the JavaScript engine developers are trying to keep up and are doing better all the time, there's always a chance that you'll run into features that you are dying to use but that are yet to be supported.

Luckily, you can keep up with the state of feature support in the various browsers via the lists at https://kangax.github.io/compat-table/es6/, http://kangax.github.io/compat-table/es2016plus/, and https://kangax.github.io/compat-table/esnext/.

1.1.2 Transpilers give us access to tomorrow's JavaScript today

Because of the rapid release cycles of browsers, we usually don't have to wait long for a JavaScript feature to be supported. But what happens if we want to take advantage of