Extracted from:

Rediscovering JavaScript

Master ES6, ES7, and ES8

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Rediscovering JavaScript

Master ES6, ES7, and ES8

Venkat Subramaniam

edited by Jacquelyn Carter

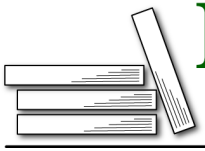
Rediscovering JavaScript

Master ES6, ES7, and ES8

Venkat Subramaniam

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Brian MacDonald
Supervising Editor: Jacquelyn Carter
Copy Editor: Liz Welch
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-546-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—June 2018

Introduction

A few days before a corporate event, the company informed me that the developers attending would be a mixture of Java, C#, and PHP programmers. I was concerned that presenting examples in Java may frustrate C# and PHP programmers. Picking any of the other two languages might have similar consequences. I made an executive decision and used JavaScript for all my examples—that way, I frustrated them all equally. Just kidding. It turned out to be a good choice—JavaScript is truly one language used by programmers who otherwise use different languages for most of their work.

JavaScript is the English of the programming world—it's native to some people, it's arguably the most widely used language, and the language itself has heavily borrowed from other languages, for greater good.

JavaScript is one of the most powerful, ubiquitous, and flexible languages. A large number of programmers fear the language for many reasons. In the past it had become infamous for being error prone and idiosyncratic. Thankfully, through the newer versions, JavaScript has evolved into a respectable language; it has come a long way since Douglas Crockford wrote *JavaScript: The Good Parts [Cro08]*.

Unlike other languages, JavaScript does not have the luxury to deprecate features. Such a measure would be considered draconian—currently working legacy code will suddenly fail on newer browsers. The language had to evolve without breaking backward compatibility.

The changes in JavaScript comes in three flavors: alternatives, additions, and advances.

Features that are error prone and outright confusing have alternative features in the newer versions. For example, `const` and `let` are the new alternatives to the messy `var` declarations. The `rest` operator, which makes it easier to create self-documenting code, is a much better alternative to `arguments`, which lacks clarity and needs explicit documentation. Also, the enhanced `for` loop removes the burden of looping that is inherent in the traditional `for` loops. While the

old way of doing things still exists, we should make a conscious effort to learn and use the newer alternatives.

When coming from other languages, programmers often say, "I wish JavaScript had…" A lot of those wishes have come true. JavaScript has adapted features found in languages like Java, C#, and Ruby, to mention a few. These additions to the language not only make the language more pleasant to use but also help solve a set of problems in a more elegant way than before.

In the vein of comparing with other languages, generators and infinite iterators in JavaScript make it possible to create lazy sequences as in languages like Haskell or Clojure. Arrow functions bring the power of lambda expressions with consistent lexical scoping while making the code concise and expressive. Template literals bring the feature of *heredocs* from languages like Ruby and Groovy to JavaScript. And the enhanced `class` syntax makes programming in JavaScript feel almost like any other object-oriented language…well, almost.

And what good is a language that does not allow you to create programs that can in turn create programs? JavaScript makes it easy to turn those meta-thoughts into useful programs using the advances in the area of metaprogramming. The `Proxy` class, along with many capabilities of the language to create dynamic, flexible, and asynchronous code, makes JavaScript a very exciting language to program in. If you have enjoyed metaprogramming in languages like Ruby, Python, and Groovy, JavaScript now has similar capabilities to create highly flexible and extensible code.

The changes in recent years bring an entirely different feeling and vibe to the language. It is a great time to be excited about programming in JavaScript. Whether you are programming the front end or writing code for the server-side back end, you can use the newer language features to make your code elegant, concise, expressive, and above all less error prone.

There is no better way to learn the language than practicing. This book has several examples for you to try out, as you learn about the new and exciting features.

Fire up your favorite IDE or text editor—let's get coding.

How to Run Modern JavaScript

JavaScript has evolved considerably but the runtime engines are still catching up. Different browsers have varied support for different features from the newer versions of JavaScript.

Sites like kangax.github.io[1] and caniuse.com[2] can help you find whether a particular browser supports a JavaScript feature you're interested in using. MDN[3] web docs is a good source for documentation of JavaScript features and support in a few different browsers. The good news is all browsers will be 100 percent features compatible within the next 20 years—or so it feels—but we can't wait that long.

If you are developing for the back end using JavaScript, you may have better control of the version of the runtime engine you use. If you are developing for the front end, you may not have much say about the browser and the version of browser your users have. The version they use may not support a particular feature, or it may be an old browser and may not support any of the features of modern JavaScript. What gives?

Here are a few options to run JavaScript in general and, in particular, to practice the examples and exercises in this book.

Run in Node.js

The easiest way to practice the code examples in this book is running them in Node.js.[4] Version 8.5 or later supports most of the latest features. I will guide you along where necessary if you need to use a command-line experimental option or an additional tool.

First, verify that Node.js is installed on your system. Point your browser to https://nodejs.org and download the latest version if you don't have it or have a fairly old version. To avoid colliding with versions of Node.js already installed on your system, use Node Version Manager[5] (NVM) if it's supported on your operating system.

Once you install the latest version of Node.js, open a command prompt and type

```
node --version
```

The version of Node.js used to run the examples in this book is

```
v9.5.0
```

The version installed on your machine may be different. If it's very old compared to the version mentioned here, consider installing a more recent version.

1. https://kangax.github.io/compat-table/es6/
2. https://caniuse.com
3. https://developer.mozilla.org
4. https://nodejs.org
5. https://github.com/creationix/nvm

If what you have is later than the version shown here, then continue using the version you have.

To run the program in Node.js, issue the `node` command followed by the filename. For example, suppose we have a file named `hello.js` with the following content:

introduction/hello.js

```
console.log('Hello Modern JavaScript');
```

Use the following command at the command prompt to run the code:

```
node hello.js
```

The command will produce the desired output:

```
Hello Modern JavaScript
```

Most IDEs that support JavaScript offer ways to more easily run the code from within the IDE. Make sure that your IDE is configured to use an appropriate version of Node.js.

Run Using the REPL

Even though I use text editors and IDEs to develop applications, I am a huge fan of *REPL*, which stands for "read-eval-print-loop." I call it the micro-prototyping environment. While in the middle of working on a function or implementing enough code to make a unit test to pass, I often reach for the REPL to quickly try out ideas. This is like how painters prime their brushes on the side of the canvas while painting.

Let's fire up the REPL and try out a snippet of code. The Node.js command `node`, when executed without any filename, runs in the REPL mode.

At the command prompt type the command `node` and press Enter. In the node prompt, which appears as >, type various JavaScript code snippets and press Enter to run immediately. The output from the execution of the snippet is shown instantly. To exit from the REPL, press Ctrl+C twice, press Ctrl+D, or type `.exit`.

Let's take the REPL for a ride. Here's an interactive session for you to try:

```
node
> languages = ['Java', 'Python', 'Ruby', 'JavaScript']
[ 'Java', 'Python', 'Ruby', 'JavaScript' ]
> word = 'Hello'
'Hello'
> word.st(hit tab)
word.startsWith  word.strike
```

```
> word.startsWith('H')
true
> languages.filter(language => language.startsWith('J'))
[ 'Java', 'JavaScript' ]
>
```

In the REPL, create a list of languages and the REPL immediately evaluates and prints the list. Now, suppose we want to pick only languages that start with J. Hmm, does string support a startsWith() function? Why guess? We can ask the REPL.

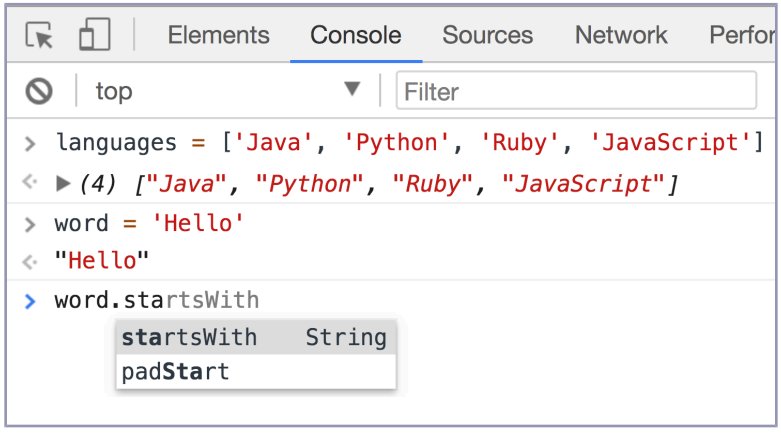
Create a variable named word and set it to the string 'Hello'. Then type word.st and press the Tab key. The REPL lists all methods of string that start with st. Then it repeats the command you had already typed. Type a after word.st and press the Tab key again. The REPL now will complete the code with word.startsWith. Proceed to complete that call and press Enter.

Finally, type the line with filter to pick words from the list that meet the expectation. The REPL immediately provides a feedback with the result of executing the call.

REPL is also a great tool to use when you are on a colleague's machine and trying to show something quickly and realize he or she is not using your favorite IDE. Instead of fiddling with his or her tool, you can open up the REPL and show some quick examples on it.

Run in the Browser Console

Much like Node.js's REPL, most browsers provide a developer console for interactive experimentation. Here's an example of using the console in Chrome, which can be invoked by choosing View > Developer > JavaScript Console or by pressing the appropriate keyboard shortcut key.



Much like an IDE, the console pops up a list of possible methods when you type a period and start typing the method. It provides an instant feedback like the REPL as well.

Run within a Browser Using Babel

In many cases, developers don't have control over the browser that their users use. Very old browsers obviously don't support any of the modern JavaScript features. The support of newer features in newer browsers also varies widely. Writing code using newer features, only to find that some browser a user is running chokes up, is no fun, especially once the application is in production. This is where transpilers come in—they translate the JavaScript you write to the good old JavaScript supported by browsers old and new.

If you are developing for the front end, you're most likely already using a transpiler like Babel.[6] Since most browsers support the older version of JavaScript, you get the best of both worlds; you can write the code using the features available in the newer versions of the language and let Babel compile it to code that will run on most browsers. With this approach, you can make use of the features without the worry of browser compatibilities, although you still need to test and verify that things actually work.

Since most examples in this book run in Node.js, we don't need to dive into Babel at this time. We'll revisit this topic toward the end of the book in *Using Decorators*, on page ?, when we need Babel.

What's in This Book?

The rest of this book is organized as follows.

Before we dig into the newer features of JavaScript, we'll quickly visit some old problem areas in Chapter 1, *JavaScript Gotchas*, on page ?. You'll learn about things to avoid and the safe alternatives to some nefarious features.

Chapter 2, *Variables and Constants*, on page ? will encourage you to replace `var` with `let` or `const` and why you should prefer `const` where possible.

JavaScript has always had support for flexible parameters, but it was not intuitive and was also error prone. Chapter 3, *Working with Function Arguments*, on page ? will show how the newer features of JavaScript make working with parameters safe, expressive, and pleasant.

6. https://babeljs.io

The enhanced for loop of modern JavaScript is the antidote for the boredom of the common loops. We discuss different ways to loop, along with the generators and how to create infinite sequences, in Chapter 4, *Iterators and Symbols*, on page ?.

The lexical scoping semantics of anonymous functions is inconsistent and confusing, to say the least. Arrow functions don't have majority of the problems that are inherent in anonymous functions. But arrow functions come with some limitations as well, as we'll see in Chapter 5, *Arrow Functions and Functional Style*, on page ?. In this chapter, we'll also see how arrow functions make it easy to create functional style code.

Hands down, one of the most exciting features of JavaScript is destructuring. In Chapter 6, *Literals and Destructuring*, on page ? we'll unleash the power of destructuring along with features like template literals and enhanced object literals.

JavaScript has supported classes for a long time, but without the class keyword. Sadly, that created problems. The newer class-related syntax in JavaScript makes writing object-oriented code much simpler, as we'll see in Chapter 7, *Working with Classes*, on page ?.

Unlike many other languages that support class-based inheritance, JavaScript has prototypal inheritance. Even though this feature is highly powerful and flexible, using it has been hard in the past—with the syntax confusing and error prone. As we'll see in Chapter 8, *Using Inheritance*, on page ?, it's now much easier, and safer, to use inheritance.

In Chapter 9, *Using Modules*, on page ?, you'll learn to work with multiple JavaScript files and the rules of module import and export.

Asynchronous programming is a way of life in JavaScript, and you need a fairly good knowledge of how promises work to master that. Chapter 10, *Keeping Your Promises*, on page ?, has you covered, I promise.

There's something magical about *metaprogramming*—the ability to create programs that can create programs. In Chapter 11, *Exploring Metaprogramming*, on page ?, we'll explore one type of metaprogramming—injection.

Then, in Chapter 12, *Deep Dive into Metaprogramming*, on page ?, we dig into another type of metaprogramming—synthesis—and how to create highly dynamic code.

Appendix 1, *Answers to Exercises*, on page ? has solutions for exercises at the end of each chapter, for you to compare notes with the solutions you create.

Finally, for your convenience, the URLs that are scattered throughout this book are gathered in one place in Appendix 2, *Web Resources*, on page ?.

Who Is This Book For?

This book is for programmers, full-stack developers, lead developers, software architects, technical managers, or just about anyone who dives into code and is interested in learning and applying modern JavaScript. If you feared JavaScript or if the language annoyed you in the past, this book will show how the language has beautifully evolved in ECMAScript 2015 (ES6), 2016 (ES7), and 2017 (ES8) and how it is now highly approachable. You can make use of these features to program the front or the back end using JavaScript.

This book assumes the reader is familiar with basics of programming—it does not teach the fundamentals of programming. Some prior knowledge of JavaScript will be helpful. Programmers who are familiar with languages like Java, C#, and Python but who are not familiar with JavaScript should be able to pick up the concepts presented fairly quickly.

If you're already familiar with the materials presented in this book, you may use this book to help train your developers.

Online Resources

You can download all the example source code for the book from the Pragmatic Bookshelf website for this book.[7] You can also provide feedback by submitting errata entries.

If you're reading the book in PDF form, you can click the link above a code listing to view or download the specific examples.

Thank you for reading this book.

7. https://www.pragprog.com/titles/ves6