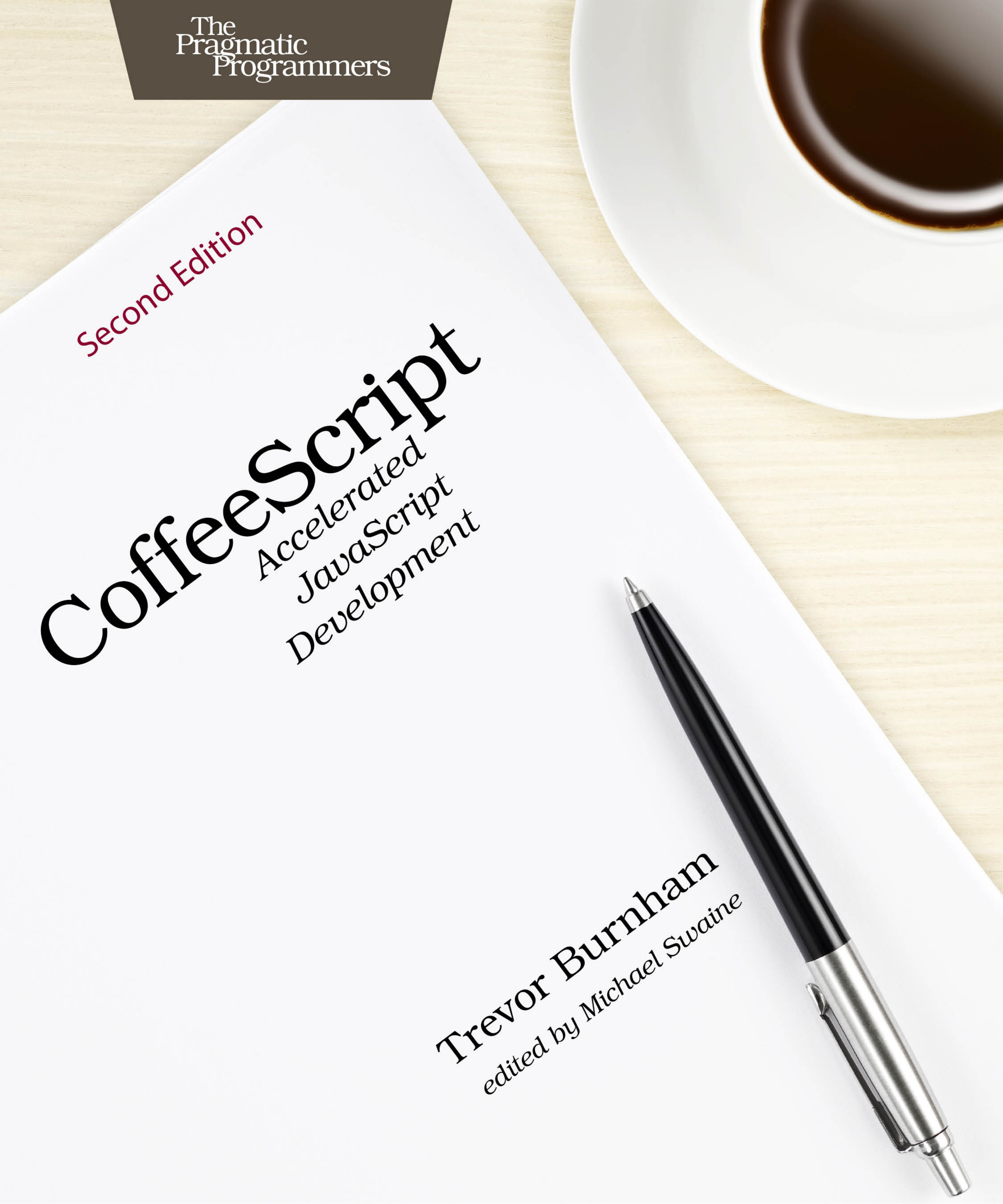
Extracted from:

CoffeeScript

Accelerated JavaScript Development, Second Edition

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



The Pragmatic Programmers

Second Edition

CoffeeScript

Accelerated JavaScript Development

Trevor Burnham

edited by Michael Swaine

CoffeeScript

Accelerated JavaScript Development, Second Edition

Trevor Burnham

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The team that produced this book includes:

Michael Swaine (editor)
Potomac Indexing (indexer)
Cathleen Small (copyeditor)
Dave Thomas (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

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

Printed in the United States of America.
ISBN-13: 978-1-94122-226-3
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—February 2015

Introduction

JavaScript was never meant to be the most important programming language in the world. It was hacked together in ten days, with ideas from Scheme and Self packed into a C-like syntax. Even its name was an awkward fit, referring to a language with little in common besides a few keywords. (For the story behind that, see Peter Seibel's interview with Brendan Eich, the creator of JavaScript, in *Coders at Work [Sei09]*.) But once JavaScript was released, there was no controlling it. As the only language understood by all major browsers, JavaScript quickly became the lingua franca of the Web. And with the introduction of Ajax in the early 2000s, what began as a humble scripting language for enhancing web pages suddenly became a full-fledged rich application development language.

As JavaScript's star rose, discontent came from all corners. Some pointed to its numerous little quirks and inconsistencies.[1] Others complained about its lack of classes and inheritance. And a new generation of coders, who had cut their teeth on Ruby and Python, were stymied by its thickets of curly braces, parentheses, and semicolons.

A brave few created frameworks for web application development that generated JavaScript code from other languages, notably Google's GWT and 280 North's Objective-J. But few programmers wanted to add a thick layer of abstraction between themselves and the browser. No, they would press on, dealing with JavaScript's flaws by limiting themselves to "the good parts" (as in the title of Douglas Crockford's now-classic book).

And then CoffeeScript came along.

The New Kid in Town

On Christmas Day 2009, Jeremy Ashkenas first released CoffeeScript, a little language he touted as "JavaScript's less ostentatious kid brother." The project

1. http://wtfjs.com/

quickly attracted hundreds of followers on GitHub as Ashkenas and other contributors added a bevy of improvements each month. The language's compiler, originally written in Ruby, was replaced in March 2010 by one written in CoffeeScript.

After its 1.0 release on Christmas 2010, CoffeeScript became one of GitHub's "most watched" projects. And the language attracted another flurry of attention in April 2011, when David Heinemeier Hansson confirmed rumors that CoffeeScript support would be included in Ruby on Rails 3.1.

Why did this little language catch on so quickly? Three reasons come to mind: familiarity, safety, and readability.

The Good Parts Are Still There

JavaScript is a vast language. It contains multitudes of features, which obscure its essence. JavaScript offers many of the benefits of a functional language while retaining the familiar feel of an imperative one. This subtle power is one of the reasons that JavaScript tends to confound newcomers. Functions can be passed around as arguments and returned from other functions, and objects can have new methods added at any time. In short, *functions are first-class objects*.

All that power is still there in CoffeeScript, but with a syntax that encourages you to use it wisely. Gone is the `function` keyword, along with its accompanying curly braces. Instead, blocks of function code are demarcated with `->` or `=>` and indentation. Likewise, the crucial yet often baffling `this` keyword has a distinctive symbol to serve in its stead, `@`. These are small changes, but they go a long way toward making the way a program works more obvious.

The Compiler Is Here to Help

Imagine a language with no syntax errors, a language where the computer forgives you your typos and tries as best it can to comprehend the code you give it. What a wonderful world that would be! Sure, the program wouldn't always run the way you expected, but hey, that's what testing is for.

Now imagine that you write that code once and send it out to the world, typos and all, and millions of computers work around your small mistakes in subtly different ways. Suddenly statements that your computer silently skipped over are crashing your entire app for thousands of users.

Sadly, that's the world we live in. JavaScript has no standard interpreter. Instead, hundreds of browsers and other environments run JavaScript in their own way. Debugging cross-platform inconsistencies is a huge pain.

CoffeeScript can't cure all of these ills, but the compiler tries its best to generate JavaScript Lint-compliant output,[2] which is a great filter for common human errors and nonstandard idioms. And if you type something that just doesn't make any sense, such as 2 = 3, the CoffeeScript compiler will tell you. Better to find out sooner than later.

It's All So Clear Now

Writing CoffeeScript can be highly addictive. Why? Take this piece of JavaScript:

```
function cube(num) {
  return Math.pow(num, 3);
}
var list = [1, 2, 3, 4, 5];
var cubedList = [];
for (var i = 0; i < list.length; i++) {
  cubedList.push(cube(list[i]));
}
```

Now here's an equivalent snippet of CoffeeScript:

```
cube = (num) -> Math.pow num, 3
list = [1, 2, 3, 4, 5]
cubedList = (cube num for num in list)
```

For those of you keeping score, that's half the character count and less than half the line count! Those kinds of gains are common in CoffeeScript. And as Paul Graham once put it, "Succinctness is power."[3]

Shorter code is easier to read, easier to write, and, perhaps most critically, easier to change. Gigantic heaps of code tend to lumber along, as any significant modifications require a Herculean effort. But bite-sized pieces of code can be revamped in a few swift keystrokes, encouraging a more agile, iterative development style.

It's worth adding that switching to CoffeeScript isn't an all-or-nothing proposition—CoffeeScript code and JavaScript code can interact freely. CoffeeScript's strings are just JavaScript strings, and its numbers are just JavaScript numbers. Even its classes work in JavaScript frameworks like Backbone.js.[4] So don't be afraid of calling JavaScript code from CoffeeScript code or vice versa. We'll talk about using CoffeeScript with two of JavaScript's

2. http://www.javascriptlint.com/
3. http://www.paulgraham.com/power.html
4. http://documentcloud.github.com/backbone/

most popular libraries in Chapter 5, *Web Applications with jQuery and Backbone.js*, on page ?.

But enough ancient history. Coding is believing, everything else is just meta, and as Jeff Atwood once said, "Meta is murder."[5] So let's talk a little bit about the book you're reading now, and then—in just a few pages, I promise!—we'll start banging out some code.

Who This Book Is For

If you're interested in learning CoffeeScript, you've come to the right place! However, because CoffeeScript is so closely linked to JavaScript, there are really two languages running through this book—and not enough pages to teach you both. Therefore, I'm going to assume that you know *some* JavaScript.

You don't have to be John "JavaScript Ninja" Resig. In fact, if you're only an amateur JavaScripter, great! You'll learn a lot about JavaScript as you go through this book. Check the footnotes for links to additional resources that I recommend. If you're new to programming entirely, you should definitely check out *Eloquent JavaScript [Hav11]*, which is also available in an interactive online format.[6] If you've dabbled a bit but want to become an expert, pick up *Effective JavaScript [Her12]*.[7] And if you want a comprehensive reference, no one does it better than the Mozilla Developer Network.[8]

You may notice that I talk about Ruby a lot in this book. Ruby inspired many of CoffeeScript's great features, such as implicit returns, splats, and postfix conditionals. And thanks to the Rails Asset Pipeline, which makes CoffeeScript compilation fully automatic, CoffeeScript has a huge following in the Ruby world. So if you're a Rubyist, *great!* You've got a head start. If not, don't sweat it; everything will fall into place once you have a few examples under your belt.

If anything in the book doesn't make sense to you, I encourage you to post a question about it on the book's forum.[9] While I try to be clear, the only entities to whom programming languages are completely straightforward are computers —and they buy very few books.

5. http://www.codinghorror.com/blog/2009/07/meta-is-murder.html
6. http://eloquentjavascript.net/
7. http://effectivejs.com/
8. https://developer.mozilla.org/en/JavaScript/Guide
9. https://forums.pragprog.com/forums/347

Embedding JavaScript in CoffeeScript

This is as good a place as any to mention that you can stick JavaScript inside of CoffeeScript code by surrounding it with backticks, like so:

```
console.log `impatient ? useBackticks() : learnCoffeeScript()`
```

The CoffeeScript compiler simply ignores everything between the backticks. That means that if, for instance, you declare a variable between the backticks, that variable won't obey conventional CoffeeScript scope rules.

In all my time writing CoffeeScript, I've never once needed to use backtick escapes. They're an eyesore at best and dangerous at worst. So in the immortal words of Troy McClure: "Now that you know how it's done—don't do it."

How This Book Is Organized

We'll start our journey by introducing the tools you'll need to compile, run, and debug CoffeeScript code. After that, the next three chapters will take you through the nuts and bolts of the language. Each of those chapters includes a small command-line project at the end. Finally, the last three chapters are dedicated to building and testing a Trello-like web application called Coffee-Tasks.

To master CoffeeScript, you'll need to know how it works with the rest of the JavaScript universe. So after learning the basics of the language, we'll take brief tours of jQuery, the world's most popular JavaScript framework, and Node.js, an exciting new project that lets you run JavaScript outside of the browser. While we won't go into great depth with either tool, we'll see that they go with CoffeeScript like chocolate and peanut butter. And by combining their powers, we'll be able to write an entire task management app in a matter of hours.

The code presented in this book, as well as errata and discussion forums, can be found on its PragProg page: http://pragprog.com/titles/tbcoffee2/coffeescript

What's Changed Since CoffeeScript 1.0.0?

Despite being a very young language, CoffeeScript has been remarkably stable in the years since its 1.0.0 release. As of this writing, the latest release is 1.8.0. For the most part, code written for CoffeeScript 1.0.0 will be accepted by the CoffeeScript 1.8.0 compiler, and vice versa. The few exceptions mainly have to do with syntactic edge cases involving implicit parentheses and indentation. Those who like to keep their code as paren-free as possible will be happy to know that ever since CoffeeScript 1.7.0, chaining code like this has worked as expected:

```
$('h1')
  .slideDown 100, => $('h2').show()
  .fadeIn 300
```

Personally, I prefer to use explicit parentheses in most cases, and you'll see that preference expressed in the code shown throughout this book. But this change has definitely made CoffeeScript's behavior in the absence of parentheses more aligned with human expectations.

But perhaps the biggest leap CoffeeScript has taken in the last few years is not in the language itself, but in the tooling. Since CoffeeScript 1.6.1, the compiler has included the ability to generate source maps, a debugger's dream come true. Before that, having to look at compiled JavaScript when an error was thrown was one of the most frequent complaints developers had about CoffeeScript. Happily, source maps solve that problem. We'll talk more about source maps in *Using Source Maps*, on page ?.

Another tooling addition is Literate CoffeeScript, which embodies an approach to coding advocated by the great Donald Knuth. In a .coffee file, you embed comments in code. But in a .litcoffee file, you do the opposite, writing a document in Markdown syntax with embedded snippets of code. The compiler simply extracts those snippets and ignores the rest of the code. The result is a human-readable narrative that doubles as a machine-readable program. Although I don't use Literate CoffeeScript in this book, I definitely think it's a cool concept.

For a (nearly) comprehensive list of changes the CoffeeScript project has gone through over time, see the changelog.[10]

The CoffeeScript Community

A great language is of little use without a strong community. If you run into problems, who you gonna call?

Posting a question to StackOverflow[11] (being sure to give your question the coffeescript tag) is a terrific way to get help, especially if you include a snippet of the code that's hassling you. If you need a more immediate answer, you can usually find friendly folks in the #coffeescript channel on Freenode IRC. For more problems, such as possible bugs, you should create an issue on

10. http://coffeescript.org/#changelog
11. http://stackoverflow.com

GitHub.[12] You can also request new language features there. CoffeeScript is still evolving, and the whole team welcomes feedback.

What about documentation? You've probably already seen the snazzy official docs.[13] There's also an official wiki.[14] And now there's this book.

Which brings us to me. I run @CoffeeScript on Twitter; you can reach me there or by good old-fashioned email at trevorburnham@gmail.com.

These are exciting times for web development. Welcome aboard!

12. http://github.com/jashkenas/coffee-script/issues
13. http://coffeescript.org
14. http://github.com/jashkenas/coffee-script/wiki