Extracted from:

CoffeeScript

Accelerated JavaScript Development

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

CoffeeScript

Accelerated JavaScript Development

Trevor Burnham

Foreword by Jeremy Ashkenas

edited by Michael Swaine

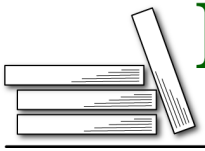
CoffeeScript

Accelerated JavaScript Development

Trevor Burnham

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Michael Swaine (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
David Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-934356-78-4
Printed on acid-free paper.
Book version: P1.0—July 2011

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.[1] 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.[2] 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 Douglas Crockford's 2008 similarly titled book).

That is, until now.

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 quickly attracted hundreds of followers on GitHub as Ashkenas and other contributors added a bevy of new features 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.

1. See Peter Seibel's interview with Brendan Eich, the creator of JavaScript, in *Coders at Work* [Sei09].
2. http://wtfjs.com/

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 vast. It contains multitudes. JavaScript offers many of the best features of functional languages while retaining the feel of an imperative language. 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; objects can have new methods added at any time; in short, *functions are first-class objects*.

All that power is still there in CoffeeScript, along with a syntax that encourages you to use it wisely.

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 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 doesn't have a standard interpreter. Instead, hundreds of browsers and server-side frameworks 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[3], 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) {
```

3. http://www.javascriptlint.com/

```
  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."[4]

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.[5] So don't be afraid of calling JavaScript code from CoffeeScript code or vice versa. As an example, we'll talk about using CoffeeScript with one of JavaScript's most popular libraries in Chapter 5, *Web Interactivity with jQuery*, on page ?.

But enough ancient history. Coding is believing, everything else is just meta, and as Jeff Atwood once said,"Meta is murder."[6] 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 hot 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

4. http://www.paulgraham.com/power.html
5. http://documentcloud.github.com/backbone/
6. http://www.codinghorror.com/blog/2009/07/meta-is-murder.html

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."

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.[7] If you've dabbled a bit but want to become an expert, head to the JavaScript Garden.[8] And if you want a comprehensive reference, no one does it better than the Mozilla Developer Network.[9]

You may notice that I talk about Ruby a lot in this book. Ruby inspired many of CoffeeScript's great features, like implicit returns, splats, and postfix `if`/`unless`. And thanks to Rails 3.1, 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.[10] While I try to be clear, the only entities to whom programming languages are completely straightforward are computers—and they buy very few books.

7. http://eloquentjavascript.net/
8. http://javascriptgarden.info/
9. https://developer.mozilla.org/en/JavaScript/Guide
10. http://forums.pragprog.com/forums/169

How This Book Is Organized

We'll start our journey by discovering the various ways that we can compile and run CoffeeScript code. Then we'll delve into the nuts and bolts of the language. Each chapter will introduce concepts and conventions that tie into our ongoing project (see the next section).

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 multiplayer game in just a few hours.

No matter what level you're at, be sure to do the exercises at the end of each chapter. They're designed to be quick yet challenging, illustrating some of the most common pitfalls CoffeeScripters fall into. Try to solve them on your own before you check the answers in Appendix 1, *Answers to Exercises*, on page ?.

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

About the Example Project: 5x5

The last section of each chapter applies the new concepts to an original word game called 5x5. As its name suggests, 5x5 is played on a grid five tiles wide and five tiles high. Each tile has a random letter placed on it at the start. Then the players take turns swapping letters on the grid, scoring points for all words formed as a result of the swap (potentially, this can be four words at each of the two swapped tiles: one running horizontally, one vertically, and two diagonally—only left-to-right diagonals count).

Scoring is based on the Scrabble point value of the letters in the formed words, with a multiplier for the number of distinct words formed. So, at the upper limit, if eight words are formed in one move, then the point value of each is multiplied by eight. Words that have already been used in the game don't count.

We'll build a command-line version of the game in Chapters 2–4, then move it to the browser in Chapter 5, *Web Interactivity with jQuery*, on page ?,

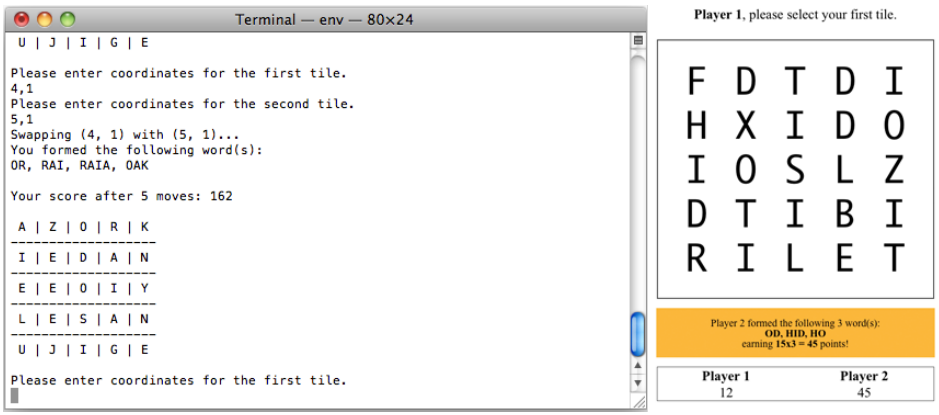


Figure 1—In the console and web versions of our project, the game logic code will be the same.

and finally add multiplayer capability in Chapter 6, *Server-Side Apps with Node.js*, on page ?. Moving the code from the command line to the browser to the server will be super-easy—they all speak the same language!

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 (being sure to tag your question coffeescript) is a terrific way to get help, especially if you post a snippet of the code that's hassling you.[11] If you need a more immediate answer, you can usually find friendly folks in the #coffeescript channel on Freenode IRC. For relaxed discussion of CoffeeScript miscellany, try the Google Group.[12] For more serious problems, such as possible bugs, you should create an issue on GitHub.[13] 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 at http://coffeescript.org. There's also an official wiki at http://github.com/jashkenas/coffee-script/wiki. And now there's this book.

11. http://stackoverflow.com
12. http://groups.google.com/forum/#!forum/coffeescript
13. http://github.com/jashkenas/coffee-script/issues

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!