Extracted from:

Getting Clojure

Build Your Functional Skills One Idea at a Time

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Getting Clojure

Build Your Functional Skills One Idea at a Time

Russ Olsen

edited by Michael Swaine

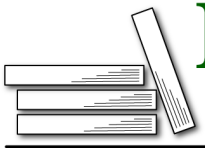
Getting Clojure

Build Your Functional Skills One Idea at a Time

Russ Olsen

The Pragmatic Bookshelf

Raleigh, North Carolina



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
Development Editor: Michael Swaine
Copy Editor: Candace Cunningham
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: 978-1-68050-300-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—May 2018

To

Mikey

Felicia

Jackson

Charlie & Jennifer

Tim & Emily & Evan

Nicholas & Jonathan

Meg & Alex & Zachary

and

Scott Downie

The future is in your hands.

Preface

I have been writing this book—at least in my head—for decades. As a professional programmer who has used everything from assembly language and FORTRAN to C and Python and Java and Ruby, I've spent much of my career longing to get my hands on a better programming language. In my mind's eye I could see exactly what I wanted: I wanted a language that had the stripped-down syntax of Lisp. I wanted a language that embraced the power of functional programming. I wanted a language that was fast enough to run the real-world applications that I was writing. Most of all I wanted a language that had all the above *and* ran on the computing platforms I worked with. For years my imaginary programming language remained just that—imaginary. I spent a lot of that time mentally composing a book that would lay out the case for my dream programming language.

Then in 2009 I found the programming language of my dreams. It was a Lisp with macros and S-expressions and it was adamantly functional. It was as fast as I needed it to be and, because it was compiled, it held out the prospect of getting still faster. Best of all, it lived where I lived, in the practical if messy world of Java and JAR files and JVMs. Ironically, the language I'd been waiting for all this time was called *Clojure*.

So I began to learn Clojure and the book in my head got longer and longer. And then I started typing.

Who Is This Book For?

This book is for people with some programming experience—perhaps in Java or JavaScript or Ruby or Python—who want to learn Clojure. Notice I didn't say *learn to program in Clojure*. Certainly by the time you get to Chapter 21 you will be able to write functions and map vectors and quote lists and do all the other things that come with being able to sling Clojure code. But behind every programming language lies a vision: a vision of how programs should

be built. The vision behind Clojure is of a radically simple language framework holding together a sophisticated collection of programming features. Thus learning Clojure—really understanding the language—involves much more than just working out where the parentheses and quotes go and knowing which library to use when parsing a URL. To really get Clojure you need to understand the framework and how it pulls together the major chunks of the language. So in the pages that follow we'll focus on the ideas behind Clojure. Each chapter will take a feature (or two or three) from the language, and explain the syntax and the mechanics behind that feature so that you know how to use it. Then we turn to the important questions:

- *What is the thinking behind this feature?*
- *How do Clojure programmers use this feature in real life?*
- *What are the dangers, the things to watch for, as you use this feature?*

As I say, I'll assume that you have some programming background. And, since most programmers find that learning Clojure is an exhilarating intellectual experience, you're definitely going to need an open mind.

How Is This Book Organized?

Generally we're going to start small and work toward the bigger ideas:

- The first chapter is a quick *this is what Clojure code looks like and how you make it go*.
- Next we look at Clojure's pair of sequential data structures: the list and the vector. Then we move on to maps, a data structure that lets you associate a key with a value. Along the way we'll run into yet another data type, the keyword, which is like a string with an attitude; and we'll explore sets, which are like maps without the values.
- Rounding out the fundamentals of Clojure we take a long look at Clojure's riff on `if` statements and Boolean logic.
- With the very basics out of the way we move on to a couple of chapters about functions. We discover just how to build a more fully featured (dare I say functional?) function and then discover that in Clojure functions are values akin to numbers and strings.
- From there we move on to names and namespaces, Clojure's answer to the twin questions of *How do I associate a name with a value?* and *How do I avoid name collision–induced insanity?*

- Next we turn to Clojure sequences, both the garden variety and the lazy kind. We'll see how Clojure programmers frequently use sequences as an elegant replacement for the loops that you find in more traditional programming languages.
- Then we look at destructuring, a feature you can use to dig your data back out of all those data structures.
- Next we look at building more sophisticated data containers and adding to them the behavior we need.
- At this point you'll know more than enough Clojure to be dangerous, so we'll move on to talking about how to ensure your code actually works.
- From there we turn to some of the knottier questions of writing real-world programs, including *How do I interact with Java?* as well as *How do I manage mutable state?*—not to mention *How do I write programs that can live with more than one thread?*
- Finally we take a long look at Clojure's syntax, which will lead us to think about how Clojure expressions get evaluated. This will get us to macros and how they enable you to extend the syntax of the language without taking your hands off the keyboard.

As I say, I'm not going to tell you everything about Clojure. Instead, the goal is to get you to the point where you can figure the rest out on your own.

About the Code Examples

As I write this, there are two industrial-strength Clojure implementations in wide use: First there's the original Java VM–based Clojure. Then there's ClojureScript, which compiles to JavaScript and runs anywhere JavaScript runs, including on your favorite browser. To keep things simple, I'll stick to discussing the original Java-based Clojure implementation. Mostly it doesn't matter: the differences between Clojure and ClojureScript are almost all related to the environments in which they run. In every way that matters, Clojure and ClojureScript are the same language.

The pages that follow include excerpts from various open source projects. Obviously the code for these projects is covered by the open source licenses under which they were released. In particular, the Clojure source code is copyrighted by Rich Hickey, who generously released it under the terms of the Eclipse Public License.

Online Resources

This book is overflowing with examples of Clojure in action—examples that you can download from the Pragmatic Bookshelf website.[1] You will also find a community forum there that you can use to ask questions, report any mistakes you find in the book, and generally connect with other readers.

Russ Olsen

russ@russolsen.com

May 2018

1. https://pragprog.com/book/roclojure/getting-clojure