Extracted from:

Mastering Clojure Macros

Write Cleaner, Faster, Smarter Code

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Mastering Clojure Macros

Write Cleaner, Faster, Smarter Code

Colin Jones

Edited by Fahmida Y. Rashid

Mastering Clojure Macros

Write Cleaner, Faster, Smarter Code

Colin Jones

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The team that produced this book includes:

Fahmida Rashid (editor)
Liz Welch (copyeditor)
Dave Thomas (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-941222-22-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—August 2014

Introduction

It probably won't surprise you to learn that this is a book about macros in Clojure. The title dashes away any hope of keeping that a secret, which is fine. I'm not big on surprises anyway.

Furthermore, this book is about *mastering* macros. Now, mastery is a pretty audacious goal: some folks say it takes 10,000 hours to master any skill, which is way more time than it'd take you to memorize this book forwards and backwards. There's always more to learn. But this is a path on the journey to mastery. So pack your things—it'll be a fun trip!

Why Clojure?

There are a number of reasons to learn Clojure. The discipline of staying immutable by default, the simplicity of functions as first-class entities, the practicality of the JVM, and… well, I could go on and on, but this book's goal isn't to sell you on Clojure. There are already several fantastic books out there for that. My first Clojure book was *Programming Clojure [Hal09]*, and my most recent intro-to-Clojure recommendation has been *Clojure Programming [ECG12]* (trust me, these are actually two different books, despite their naming similarities). *The Joy of Clojure [FH11]* is excellent as well, but it can be a bit advanced for Clojure newcomers. This book probably falls into the same category: you should already know a bit of Clojure in order to get the most out of this book.

So in the interest of focus and brevity, I'm going to be assuming throughout this book that you already know some Clojure. That said, if you haven't written a single line of Clojure before, never fear! You should set this book aside briefly and go through the Clojure Koans,[1] and I'll meet you back here in a couple of hours. We'll be tackling some advanced topics in this book, so if you haven't worked through the koans, a book, or a Clojure project, you

1. http://clojurekoans.com

may need to spend some extra time with some of the chapters in this book to get more Clojure under your belt.

You'll also want to make sure you have Leiningen[2] installed and know how to do things like launch a *REPL* (Read-Eval-Print Loop, `lein repl` from a command line), read the in-REPL docstrings, and peek here and there at the source code.

From this point forward, you should know your way around a Clojure REPL. I'll do my best to keep the progression steady through the tutorial part of the book, with each step building upon the one before. But my realistic, software-estimating side says I won't think of everything, and here and there I'll get too excited and skip a step. I hope that when that happens, you'll reach for your nearest REPL, recover quickly, and forgive my exuberance.

Why Macros?

Clojure stands on the shoulders of giants, with influences from several functional and object-oriented languages, database and distributed systems technologies, and of course the tremendous force of nature that is its creator, Rich Hickey. And despite Clojure's youth, we've already started to see some cross-pollination into other language communities. But one of the real killer features of Clojure is the macro system, which is similar in many ways to Common Lisp's but brings its own modern flair to the area.

Macros in Clojure are an elegant metaprogramming system, a means to accomplish goals that might seem impossible in other languages. How hard would it be to add pattern matching or a new control flow structure to your language as a library (rather than patching the core language)? In Clojure, people like you and me have the power to do these things ourselves.

It's true in a sense that all general-purpose languages are equally powerful, but we programmers know better. Our limitations and goals are not the same as those of a Turing machine. We want lean, clean code that expresses our intent clearly while allowing us to tell the machine what to do as succinctly as possible. Macros are a way to get there.

Metaprogramming in Non-Lisps

Lisp certainly isn't the only language family with metaprogramming facilities. C has a macro system that allows textual replacement as a preprocessing step at the start of compilation. C++ has a complex and powerful template

2. http://leiningen.org/

metaprogramming feature that's itself Turing-complete. Plenty of dynamic languages, like JavaScript, Python, and Ruby, have `eval` functions that acts on strings to produce a result. Ruby, in particular, has quite a few nice features, and Scala's macro system even allows you to manipulate the parse tree! But here's fair warning: once you've experienced Clojure's macros, you may not want to go back.

You may notice that in this book, sometimes I'll refer to Clojure, and sometimes to Lisp in general (since Clojure is part of the Lisp family of languages). This shouldn't be too confusing, but let's take a moment to clarify, just to make sure. When I talk about Lisp, the advice will apply across all the major Lisps (including Clojure, Scheme, and Common Lisp). Elixir is an interesting edge case: it's not quite a Lisp, but its macro system sure walks and talks like one. But just because I mention Clojure specifically doesn't mean that the advice *only* applies to Clojure. It may well apply in other Lisps too, but not necessarily.

Who Is This Book For?

This book assumes that you've programmed in Clojure a bit but that you're not yet a master of Clojure macros. Experienced Clojure programmers will no doubt recognize situations they've encountered before, whereas newer Clojure folks will learn to avoid some of the stumbles the rest of us have made when learning macros. Macros are typically one of the most difficult features, and this book will help anyone who wants to understand how they work and when to use them.

What's in This Book?

Chapter 1, *Build a Solid Foundation*, on page ? introduces the basic building blocks that will allow you to understand how macros work and what kinds of things it's possible to do with them.

Chapter 2, *Advance Your Macro Techniques*, on page ? shows how and why to use syntax-quoting, unquoting, and gensyms. These are some the trickiest concepts for new macro writers, but also some of the most useful.

Chapter 3, *Use Your Powers Wisely*, on page ? takes a step back to dig into some of the problems macros can create, and how you can avoid those problems.

Chapter 4, *Evaluate Code in Context*, on page ? starts the field guide portion of the book, and covers the first main use case you see in the wild: wrapping a block of code in a context for its execution.

Chapter 5, *Speed Up Your Systems*, on page ? digs into how macros can help you write very fast Clojure code without sacrificing concision and cleanliness.

Chapter 6, *Build APIs That Say Just What They Mean*, on page ? outlines some ways that macros allow you to provide easy-to-use APIs that let users write only the minimum amounts of code.

Chapter 7, *Bend Control Flow to Your Will*, on page ? shows you how you can invent your own loops and other control flow mechanisms without relying on the language to provide you with new ones.

Chapter 8, *Implement New Language Features*, on page ? goes a step further and shows you how you can steal some of the best features from other languages and introduce them to your Clojure code with macros.

How to Read This Book

There are lots of code examples in this book, and I suggest trying them out in your own REPL, creating a toy project for the longer examples. Leiningen[3] is the Clojure build tool of choice, and `lein new macrobook` will give you a new project to play around in. The examples in this book have been tested using Clojure 1.6.0, so you're safest using that version of Clojure when you try things out. But since the macro system changes so rarely, I anticipate that many earlier and future versions will work just fine as well.

This book is intended to be read front to back, but if you're the adventurous or time-constrained sort, here are a few ideas to get you the information you're looking for:

The first two chapters, Chapter 1, *Build a Solid Foundation*, on page ? and Chapter 2, *Advance Your Macro Techniques*, on page ?, teach you all the building blocks you'll need to write your own macros. If you already have a pretty good working knowledge of macros and have written and debugged several of them yourself, you may wish to skip straight to Chapter 2, *Advance Your Macro Techniques*, on page ?.

Everyone should read Chapter 3, *Use Your Powers Wisely*, on page ?, where you'll see some of the problems that macros can cause and why you don't want to use them all the time.

In the remaining chapters, starting with Chapter 4, *Evaluate Code in Context*, on page ?, we'll look in depth at some use cases for macros, with examples

3. http://leiningen.org

from both the Clojure language itself and community libraries. In these chapters, you'll see reasons you'd use macros in practice, despite their imperfections, and sometimes you'll see ways to accomplish the same goals with normal functions.

It's fine to skip around once you've finished Chapter 3, *Use Your Powers Wisely*, on page ?. By that point you'll have all the basic tools you need to understand the rest of the book. The macros you see will generally get more complex as the chapters progress, but there are minimal dependencies among the later chapters.

Online Resources

Take a look at this book's official website,[4] where you can order copies of this book as gifts and download the book's source code. You can also submit error reports.[5]

Please send specific book questions or commentary directly to me via the forums on the official website, but there are also some wonderful community resources for more general questions, or ones outside the scope of this book. I highly recommend visiting the #clojure IRC channel on Freenode and the Clojure mailing list[6] for those questions that aren't specifically about this book.

This book is a brief one, but don't be fooled–there's a lot of material to cover. Don't hesitate to try things out at the REPL and reread sections that move a bit too quickly. A small time investment now could pay off in deeper understanding later on.

Now let's get started by digging into the details of how Clojure macros work.

4. http://pragprog.com/book/cjclojure/clojure-macros
5. http://pragprog.com/book/cjclojure/errata
6. http://groups.google.com/group/clojure