Extracted from:

Clojure Applied

From Practice to Practitioner

This PDF file contains pages extracted from *Clojure Applied*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2015 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Clojure Applied

From Practice to Practitioner

Ben Vandgrift
Alex Miller

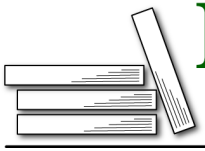
edited by Jacquelyn Carter

Clojure Applied

From Practice to Practitioner

Ben Vandgrift
Alex Miller

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

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 *https://pragprog.com*.

The team that produced this book includes:

Jacquelyn Carter (editor)
Potomac Indexing, LLC (index)
Eileen Cohen (copyedit)
Dave Thomas (layout)
Janet Furlow (producer)
Ellie Callahan (support)

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

Copyright © 2015 The Pragmatic Programmers, LLC.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-68050-074-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—September 2015

Foreword

For most people, the process of learning Clojure proceeds in three stages. First are the fundamentals: when do I use parentheses and when do I use brackets? How is a vector different from a list? And what's the syntax for a function literal again?

The middle stage of learning Clojure is when you work out how it all fits together. How do I assemble all these first-class functions into working code? How do I turn laziness into an ally instead of a confusing adversary? And just how do I get anything done with data structures that I can't modify?

Once you have enough understanding of the language to be able to bootstrap from what you know to what you want to know, you enter the third and final stage of learning Clojure. This is when you explore the Clojure ecosystem of libraries and applications, using your new knowledge to work out what other people have built and how they went about building it. This is also when most people start making their own contributions. It's all fun, but the third stage is when the *serious* fun kicks in.

Given how minimal Clojure's syntax is, the first stage usually goes quickly and painlessly. The third stage generally takes quite a bit longer, but most people don't seem to notice since they're having such a good time. It's that middle phase that tends to be the challenge: you understand the basics but aren't quite sure what you do with them. For example, functional programming is a straightforward idea: you write your code as a series of functions that do nothing but take some arguments and produce a result. Getting the hang of actually writing functional programs—of assembling these little bits of algorithm into a working whole—is a much bigger challenge. The same is true of many of the other concepts embraced by Clojure: simple and easy to grasp, harder to assemble into a working whole. The middle stage of learning Clojure is when you really learn Clojure.

Clojure Applied is aimed squarely at the middle stage of learning Clojure. If you know how to write Clojure functions but still aren't 100 percent sure how

• Click HERE to purchase this book now. discuss

you do this functional programming *thing*, this book can help you. If you know how to fill up all those persistent data structures with data, but you aren't sure where to go from there, this is the book for you. If you find yourself struggling with the issues of state and when and how it should change, keep reading. As the name suggests, *Clojure Applied* is about using Clojure to solve real problems and build real code—and get to the serious fun.

Most people find that once they learn Clojure it's hard to go back. Get used to Clojure's finely honed minimalism, its elegant and powerful approach to programming, and you wonder how you ever lived without it. As I say, there's no going back. But with this book, Alex and Ben have come back. They've come back for you. They've come back to be your guide, to help you through the middle stage of learning Clojure, the *after the basics but before mastery* part of the trip. Enjoy the ride.

Russ Olsen

Herndon, VA, April 1, 2015

• Click HERE to purchase this book now. discuss

Acknowledgments

This book wouldn't have been possible without the help of many people. They did their level best to make this book spectacular, and where it falls short, the fault is entirely ours.

We'd like to thank the staff at Pragmatic Bookshelf, our editor Jacquelyn Carter, managing editor Susannah Pfalzer, and the publishers Andy Hunt and Dave Thomas. Jackie provided us with excellent advice and dealt patiently with our often hectic schedules.

Thanks to Rich Hickey for creating our favorite programming language and for helping us think about good problems. Thanks to everyone at Cognitect for being great colleagues and supporting both Clojure and the book. Additionally, thanks to the Clojure community—you continually amaze us with your ideas and contributions.

Thanks to those who reviewed the book both on and off the record: Mario Aquino, Kevin Archie, Aaron Arnett, Stu Halloway, Bridget Hillyer, Adam Hunter, Ben Kamphaus, David McNeil, Andrew Mertz, Ryan Neufeld, Russ Olsen, Alex Stangl, and Larry Staton, Jr.

An especially heartfelt thanks to Russ Olsen for his excellent foreword, and for the many times he gave sound advice on how to sound as patient, encouraging, and friendly as he does by reflex alone.

Thank you to those who posted feedback through the discussion forums, errata page, and other online communities. A special thanks to Stig Brautaset, who contributed generously to the asynchronous shopping example in *Shopping with a Pack*, on page ?.

From Alex:

Thanks first to Ben, both for asking me to participate in this effort, and then for not kicking me out when I suggested substantial changes in direction.

• Click HERE to purchase this book now. discuss

Thanks to my daughter Ella for an unending stream of humorous faux book reviews and requests to read the book. Thanks to my son Truman for (mostly) not playing drums outside my office during writing time. Thanks to my son Beck for the reminder that there's always time to get shot by a NERF gun. Finally, the biggest thanks go to my wife Mary for her love and support despite far too many late nights of writing. It's all done now, I promise!

From Ben:

Thank you Alex, for sharing this journey with me, pushing the book in the right directions, and being a rock I could count on. You've helped make this work *much* better than I could've alone.

Thanks to many friends for constant encouragement, particularly Carl and Elaine. You two won't read this book, but you're the reason I thought I could write it.

Caffeinated thanks to Central Coffee[1] and Not Just Coffee,[2] where most of my writing got done.

Most of all, thank you to my wife Christina for her support, patience, occasional kicks in the shin, and for allowing me to continue working on the text while we planned our wedding. So oh oh.

1. http://tinyurl.com/centralcoffee
2. http://www.notjust.coffee/

Introduction

Taking a programming language out of the toy box and into the workplace can be a daunting prospect. If you haven't designed or developed a life-size application in Clojure before, you may not know where to begin, how to proceed, or what your day-to-day development process will look like. You're not alone, and we're here to help. We're going to show you how to take your Clojure practice to the professional level.

Clojure's focus on data, Lisp syntax, and functional leanings can empower you to write elegant applications. Learning to take full advantage of these facilities, though, is more than just syntax. Think about the game of chess.

Understanding how to play chess is more than understanding which pieces can move where. Broader concerns are involved: choosing an opening, pressuring and holding the center, the transition to the midgame, trapping your opponent's king. You can play the game the minute you understand the mechanics, but achieving any level of satisfying victory requires an understanding of the larger concepts.

Learning Clojure is no different. The syntax and behavior are only the first step toward proficiency. Understanding the language's principles and putting them into practice is the next step.

When you're new to any topic it helps to have strong guidelines with which to operate. Rules become practice, practice becomes habit, and habit becomes instinct. Before long, you'll have a nose for doing the right thing.

As you strengthen your practice, you'll know which rules can be bent and evolve your own personal style. You'll eventually outgrow the techniques we present, but by then—we hope!—you'll look fondly back at this text as one of your stepping stones to mastery.

• Click HERE to purchase this book now. discuss

Putting Clojure to Work

All Clojure applications build upon a foundation of immutable values. Immutability is present not just for simple scalar values but also for composite values like lists, vectors, maps, and sets. The axiom of immutability underlies our approach to data transformation, state, concurrency, and even API design.

In this book, you'll learn to build Clojure applications from bottom to top, then how to take those systems to production. This process starts from simple concepts and builds larger units of code until the application has full functionality.

We'll start by looking at how to model a problem domain with domain entities and relationships. You'll learn how to choose the best data structure to use when collecting values and entities in your domain, including some lesser-known and more-specialized options. If none of the available collections is sufficient, you'll even learn how to build your own custom collection that can work with the existing Clojure core library.

Once you have your data representation, you need to consider how to transform both your entities and collections. For this, you can rely primarily on the tools of functional programming. Most of the functions you'll create in a Clojure program are pure: they transform one immutable value (whether it's an entity, collection, sequence, or tree) to another immutable value without side effects. This pairing of immutable values and pure functions makes your code easy to understand, easy to test, and immune to many of the problems caused by unmanaged mutability.

Building Applications

Once we've developed a representation of our data and the basic operations upon it, we need to consider how to build up from there into larger structures that compose an application. This will require things like state, concurrency, and components.

The combination of immutable values and pure functions provides exactly the foundation we need to create and maintain state. In Clojure, state is the current value referenced by an identity. State changes happen when an update function transforms the current value to a new value. Clojure has several stateful reference types that can establish a shared identity. You'll learn how to select the best reference type for your needs.

Although this state model is simple, it's the secret to Clojure's suitability for writing concurrent programs. When you can rely on immutable values and

• Click HERE to purchase this book now. discuss

a simple model for state changes, it becomes much easier to use concurrency to scale up your processing. You'll learn how to leverage Clojure concurrency techniques for both doing work in the background and processing data in parallel.

We then need to move to bigger units of code to accomplish bigger goals. You'll learn how to leverage namespaces to organize code and how to design components. Components expose functionality through an API and can contain state and manage concurrency. Components can even aggregate other components and act as application subsystems.

Finally, it's time to assemble the whole application by gluing together components. You'll learn how to load your system configuration, instantiate components, connect those components, and provide entry points for the application as a whole.

This process of building systems from bottom to top is logical, but it's unlikely that you'll follow it in linear order in every real application. You may start at the bottom developing a data model, but you may also start at the top, determining how a system will be broken up into subsystems or components, and how those components are connected. Most likely you'll bounce back and forth and do both!

Both directions of work allow us to gain a greater understanding of the problem. Only an iterative process combining information from both will allow the shape of the final solution to emerge. Nevertheless, you should expect that the end application will contain the pieces we discussed in the preceding text when you get to the end.

From Build to Deploy

After you've seen an overview of how to build a Clojure application, you'll need to consider other concerns, such as testing, integration, and deployment.

When you look into testing Clojure code, you'll find that Clojure developers lean away from example-based unit testing and more toward other approaches such as interactive development in the REPL and model- or property-oriented approaches that can survey a wider range of inputs for correctness. This approach provides more coverage in less time and creates tests that are easier to maintain over time. However, it's a shift in thinking, and some practice is required to yield the greatest benefits.

You may also need to connect your Clojure-based application to other systems, either by integrating a web or other user interface, exposing an API service,

• Click HERE to purchase this book now. discuss

or consuming external APIs. The Clojure approach to these problems, unsurprisingly, treats data (and the transmission of data over wires) with importance. You'll learn about some of the available options and which ones to use in different situations to maximize performance or extensibility.

Finally, you need to deploy your application to cloud-based containers. We'll look at some of the most popular choices and how to choose among them.

About This Book

This book is a bridge from introductory material to solving real problems with Clojure, providing a guide to thinking about problems in ways that are harmonious with the tools provided.

Who This Book Is For

You should be familiar with Clojure's basic concepts and syntax to read this book. You'll learn to connect the pieces you already know to support the larger goal of building great applications.

How to Read This Book

Parts 1 and 2 should be read in order, because each chapter builds on previous topics. It's a narrative, not a reference. Part 3 can be read in any order. Each chapter in Part 3 is self-contained but may depend at times on content discussed in Parts 1 and 2.

Code and Notation

Whether in a code block or embedded in text, code uses the following font:

```
(println "Hello!")
```

For commands to be typed in a REPL, the namespace is shown (in this case, user) and a slightly different highlighting scheme used. Any output relevant to the evaluated statement is in code font immediately below the executed code:

```
user=> (println "Hello!")
Hello!
```

The location of source code that can be found in the accompanying source bundle precedes the code:

cljapplied/src/preface.clj

```
(+ 5 4 3 2 1)
```

In several instances we add indentation (as you might see in Clojure source) to output, for clarity:

```
user=> (current-customer)
#Customer{:cname "Danny Tanner",
          :email "danny@fullhouse.example.com",
          :membership-number 28374}
```

In other instances, we use a nonstandard (but useful) convention to denote code that has been elided for brevity. Three commas ,,, mark a section of code that's been removed for brevity. You may see this type of ellipsis referred to as the *Fogus comma*.[1]

Online Resources

The code examples we use are available online at the Pragmatic Bookshelf website.[2] A community forum is also available there for discussion, questions, and submitting any errata you find.

To get the most out of the examples, you'll want to install Clojure. A number of good online resources[3] are available, but the easiest way to get Clojure on your system is to install Leiningen.[4]

Alex Miller and Ben Vandgrift

August 2015

1. http://blog.fogus.me/2013/09/04/a-ha-ha-ha-aah/
2. https://pragprog.com/book/vmclojeco/clojure-applied
3. http://ben.vandgrift.com/2014/03/14/clojure-getting-started.html
4. http://leiningen.org/