

Extracted from:

Programming Clojure

This PDF file contains pages extracted from Programming Clojure, 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 printer versions; the content is otherwise identical.

Copyright © 2009 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
Programmers

Programming Clojure



Stuart Halloway

Edited by Susannah Davidson Pfalzer



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 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

<http://www.pragprog.com>

Copyright © 2009 Stuart Halloway.

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-10: 1-934356-33-6

ISBN-13: 978-1-934356-33-3

Printed on acid-free paper.

P2.0 printing, September 2009

Version: 2009-10-27

Preface

Clojure is a dynamic programming language for the Java Virtual Machine (JVM), with a compelling combination of features:

- *Clojure is elegant.* Clojure's clean, careful design lets you write programs that get right to the essence of a problem, without a lot of clutter and ceremony.
- *Clojure is Lisp reloaded.* Clojure has the power inherent in Lisp but is not constrained by the history of Lisp.
- *Clojure is a functional language.* Data structures are immutable, and most functions are free from side effects. This makes it easier to write correct programs and to compose large programs from smaller ones.
- *Clojure simplifies concurrent programming.* Many languages build a concurrency model around locking, which is difficult to use correctly. Clojure provides several alternatives to locking: software transactional memory, agents, atoms, and dynamic variables.
- *Clojure embraces Java.* Calling from Clojure to Java is direct and fast, with no translation layer.
- *Unlike many popular dynamic languages, Clojure is fast.* Clojure is written to take advantage of the optimizations possible on modern JVMs.

Many other languages cover *some* of the features described in the previous list. My personal quest for a better JVM language included significant time spent with Ruby, Python, and JavaScript, plus less intensive exploration of Scala, Groovy, and Fan. These are all good languages, and they all simplify writing code on the Java platform.

But for me, Clojure stands out. The individual features listed earlier are powerful and interesting. Their clean synergy in Clojure is *compelling*.

We will cover all these features and more in Chapter 1, *Getting Started*, on page 23.

Who This Book Is For

Clojure is a powerful, general-purpose programming language. As such, this book is for experienced programmers looking for power and elegance. This book will be useful for anyone with experience in a modern programming language such as C#, Java, Python, or Ruby.

Clojure is built on top of the Java Virtual Machine, and it is *fast*. This book will be of particular interest to Java programmers who want the expressiveness of a dynamic language without compromising on performance.

Clojure is helping to redefine what features belong in a general-purpose language. If you program in Lisp, use a functional language such as Haskell, or write explicitly concurrent programs, you will enjoy Clojure. Clojure combines ideas from Lisp, functional programming, and concurrent programming and makes them more approachable to programmers seeing these ideas for the first time.

Clojure is part of a larger phenomenon. Languages such as Erlang, F#, Haskell, and Scala have garnered attention recently for their support of functional programming and/or their concurrency model. Enthusiasts of these languages will find much common ground with Clojure.

What Is in This Book

Chapter 1, *Getting Started*, on page 23, demonstrates Clojure's elegance as a general-purpose language, plus the functional style and concurrency model that make Clojure unique. It also walks you through installing Clojure and developing code interactively at the REPL.

Chapter 2, *Exploring Clojure*, on page 47, is a breadth-first overview of all of Clojure's core constructs. After this chapter, you will be able to read most day-to-day Clojure code.

Chapter 3, *Working with Java*, on page 81, shows you how to call Java from Clojure and call Clojure from Java. You will see how to take Clojure straight to the metal and get Java-level performance.

The next two chapters cover functional programming. Chapter 4, *Unifying Data with Sequences*, on page 113, shows how all data can be

unified under the powerful sequence metaphor. Chapter 5, *Functional Programming*, on page 149, shows you how to write functional code in the same style used by the sequence library.

Chapter 6, *Concurrency*, on page 179, delves into Clojure's concurrency model. Clojure provides four powerful models for dealing with concurrency, plus all of the goodness of Java's concurrency libraries.

Chapter 7, *Macros*, on page 213, shows off Lisp's signature feature. Macros take advantage of the fact that Clojure code is data to provide metaprogramming abilities that are difficult or impossible in anything but a Lisp.

Chapter 8, *Multimethods*, on page 246, covers Clojure's answer to polymorphism. Polymorphism usually means "take the *class* of the *first* argument and dispatch a method based on that." Clojure's multimethods let you choose *any function of all* the arguments and dispatch based on that.

There is already a thriving Clojure community. Chapter 9, *Clojure in the Wild*, on page 267, introduces third-party libraries for automated testing, data access, and web development. You will see how to use these libraries to build Snippet, a database-backed web application for posting and reading code snippets.

At the end of most chapters there is an extended example demonstrating the ideas from that chapter in the context of a larger application: Lancet. Lancet³ is a Clojure-based build system that works with Apache Ant. Starting from scratch, you will build a usable subset of Lancet by the end of the book.

Appendix A, on page 286, lists editor support options for Clojure, with links to setup instructions for each.

How to Read This Book

All readers should begin by reading the first two chapters in order. Pay particular attention to Section 1.1, *Why Clojure?*, on page 23, which provides an overview of Clojure's advantages.

3. <http://github.com/stuarthalloway/lancet>

Experiment continuously. Clojure provides an interactive environment where you can get immediate feedback; see Section 1.2, *Using the REPL*, on page 34 for more information.

After you read the first two chapters, skip around as you like. But read Chapter 4, *Unifying Data with Sequences*, on page 113 before you read Chapter 6, *Concurrency*, on page 179. These chapters lead you from Clojure's immutable data structures to a powerful model for writing correct concurrency programs.

As you make the move to longer code examples in the later chapters, make sure that you use an editor that does Clojure indentation for you. Appendix A, on page 286, will point you to common editor options.

For Functional Programmers

- Clojure's approach to FP strikes a balance between academic purity and the realities of execution on the current generation of JVMs. Read Chapter 5, *Functional Programming*, on page 149 carefully to understand how Clojure idioms differ from languages such as Haskell.
- The concurrency model of Clojure (Chapter 6, *Concurrency*, on page 179) provides several explicit ways to deal with side effects and state and will make FP appealing to a broader audience.

For Java/C# Programmers

- Read Chapter 2, *Exploring Clojure*, on page 47 carefully. Clojure has very little syntax (compared to Java), and we cover the ground rules fairly quickly.
- Pay close attention to macros in Chapter 7, *Macros*, on page 213. These are the most alien part of Clojure, when viewed from a Java or C# perspective.

For Lisp Programmers

- Some of Chapter 2, *Exploring Clojure*, on page 47 will be review, but read it anyway. Clojure preserves the key features of Lisp, but it breaks with Lisp tradition in several places, and they are covered here.
- Pay close attention to the lazy sequences in Chapter 5, *Functional Programming*, on page 149.

- Get an Emacs mode for Clojure that makes you happy before working through the code examples in later chapters.

For Perl/Python/Ruby Programmers

- Read Chapter 6, *Concurrency*, on page 179 carefully. Intraprocess concurrency is very important in Clojure.
- Embrace macros (Chapter 7, *Macros*, on page 213). But do not expect to easily translate metaprogramming idioms from your language into macros. Remember always that macros execute at read time, not runtime.

Notation Conventions

The following notation conventions are used throughout the book.

Literal code examples use the following font:

```
(+ 2 2)
```

The result of executing a code example is preceded by a ->:

```
⇒ (+ 2 2)
   4
```

Where console output cannot easily be distinguished from code and results, it is preceded by a pipe character (|):

```
⇒ (println "hello")
   | hello
   nil
```

When introducing a Clojure form for the first time, I will show the grammar for the form like this:

```
(example-fn required-arg)
(example-fn optional-arg?)
(example-fn zero-or-more-arg*)
(example-fn one-or-more-arg+)
(example-fn & collection-of-variable-args)
```

The grammar is informal, using ?, *, +, and & to document different argument-passing styles, as shown previously.

Clojure code is organized into *libs* (libraries). Where examples in the book depend on a library that is not part of the Clojure core, I document that dependency with a use form:

```
(use '[lib-name :only (var-names+)])
```


This form of use brings in only the names in var-names, making each function's origin clear. For example, a commonly used function is `str-join`, from the `clojure.contrib.str-utils` library:

```
(use '[clojure.contrib.str-utils :only (str-join)])
⇒ (str-join "-" ["hello", "clojure"])
   "hello-clojure"
```

Clojure returns `nil` from a successful call to `use`. For brevity, this is omitted from the example listings.

While reading the book, you will enter code in an interactive environment called the REPL. The REPL prompt looks like this:

user=>

The user before the prompt tells the namespace you are currently working in. For most of the book's examples, the current namespace is irrelevant. Where the namespace is irrelevant, I will use the following syntax for interaction with the REPL:

```
⇒ (+ 2 2)      ; input line without namespace prompt
   4           ; return value
```

In those few instances where the current namespace is important, I will use this:

```
⇒ user=> (+ 2 2) ; input line with namespace prompt
   4           ; return value
```

Web Resources and Feedback

Programming Clojure's official home on the Web is the *Programming Clojure* home page⁴ at the Pragmatic Bookshelf website. From there you can order electronic or paper copies of the book and download sample code. You can also offer feedback by submitting errata entries⁵ or posting in the forum⁶ for the book.

In addition to the book, I have written a number of articles about Clojure. These are all available under the “clojure” tag at the Relevance blog.⁷

-
4. <http://www.pragprog.com/titles/shcloj/programming-clojure>
 5. <http://www.pragprog.com/titles/shcloj/errata>
 6. <http://forums.pragprog.com/forums/91>
 7. <http://blog.thinkrelevance.com/tags/clojure>

Downloading Sample Code

The sample code for the book is available from one of two locations:

- The *Programming Clojure* home page⁸ links to the official copy of the source code and is updated to match each release of the book.
- The *Programming Clojure* git repository⁹ is updated in real time. This is the latest, greatest code and may sometimes be *ahead* of the prose in the book.

Individual examples are in the `examples` directory, unless otherwise noted. The `Lancet` examples have their own separate `lancet` directory.

Throughout the book, listings begin with their filename, set apart from the actual code by a gray background. For example, the following listing comes from `examples/preface.clj`:

```
Download examples/preface.clj  
(println "hello")
```

If you are reading the book in PDF form, you can click the little gray box preceding a code listing and download that listing directly.

With the sample code in hand, you are ready to get started. We will begin by meeting the combination of features that make Clojure unique.

8. <http://www.pragprog.com/titles/shcloj>

9. <http://github.com/stuarthalloway/programming-clojure>

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Programming Clojure's Home Page

<http://pragprog.com/titles/shcloj>

Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates

<http://pragprog.com/updates>

Be notified when updates and new books become available.

Join the Community

<http://pragprog.com/community>

Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy

<http://pragprog.com/news>

Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/shcloj.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)