Extracted from:

Learn Functional Programming with Elixir

New Foundations for a New World

The Pragmatic Bookshelf

Raleigh, North Carolina

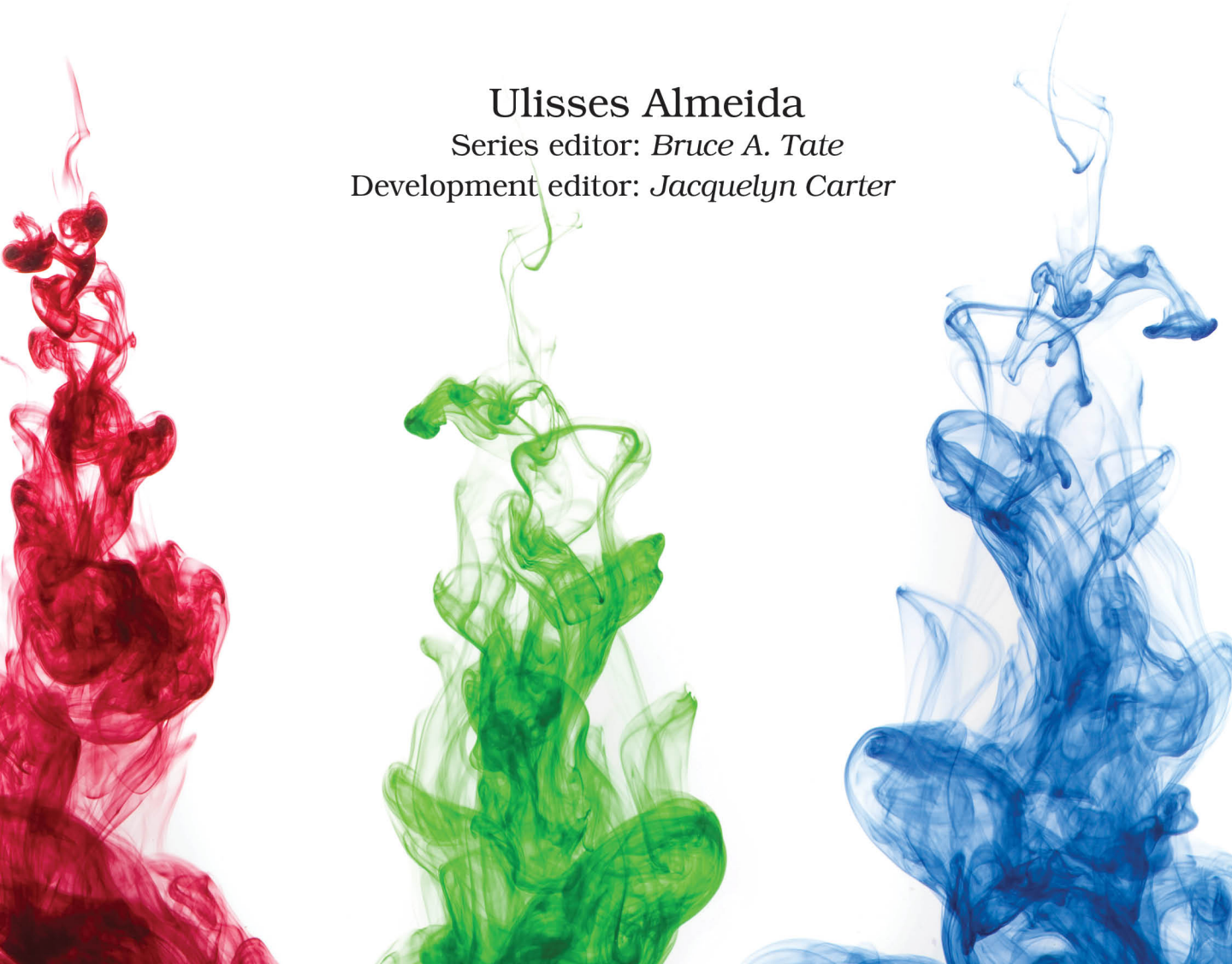
The Pragmatic Programmers

Your Elixir Source

Learn Functional Programming with Elixir

New Foundations for a New World

Ulisses Almeida
Series editor: *Bruce A. Tate*
Development editor: *Jacquelyn Carter*



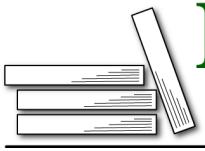
Learn Functional Programming with Elixir

New Foundations for a New World

Ulisses Almeida

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
Series editor: Bruce A. Tate
Copy Editor: Candace Cunningham, Nicole Abramowitz
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: 978-1-68050-245-9
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—February 2018

Acknowledgments

When it is your first time writing a book, it's a great challenge. But when English isn't your native language, it's a challenge on a whole new level. I did it, but I wasn't alone. This book has reached this level of quality with the help of several amazing and kind people. I would like to highlight my editor, Jackie Carter. Her contribution is what makes the release of this book possible. Her experience and knowledge guided me in the right direction with patience and humor. We faced tough decisions, rewrites, and corrections, and she was always there to help and keep me motivated. I'm really grateful to have worked with her.

Bruce Tate, the series editor, took the first technical look at the book early in the writing process. His experience was invaluable to me. He helped me transform introductions from boring to engaging, and helped me prioritize the essential and useful functional programming techniques.

The Elixir core members Andrea Leopardi and James Fish provided great technical advice throughout the writing of this book. Our technical reviewers did superb work in catching broken code and pointing out concepts that needed more clarity: thank you to Bernardo Araujo, Stéfanni Brasil, João Britto, Thiago Colucci, Mark Goody, Gábor László Hajba, Maurice Kelly, Nigel Lowry, Max Pleaner, Juan Ignacio Rizza, Kim Shrier, Carlos Souza, Elomar Souza, and Richard Thai. Thank you also to our beta reviewers who did an excellent job in reporting issues, especially Luciano Ramalho, who shared his experience by providing excellent insights for the first examples of this book.

Thank you to Susannah Davidson Pfalzer for the excellent onboarding to The Pragmatic Bookshelf, and for the early tips on how to write a great book; Candace Cunningham, our copyeditor, who helped make the text fluid and enjoyable to read; Janet Furlow, who helped with production details and extractions; and Katharine Dvorak, who did an amazing job in guiding the book through the final steps. She was always ready to answer any questions and help me with book promotion.

Thank you to Hugo Baraúna from Plataformatec and Adriano Almeida from Casa do Código for introducing me to The Pragmatic Bookshelf; and to my coworkers from Plataformatec, who helped keep me motivated, especially João Britto and José Valim, who always helped me answer hard questions about Elixir.

Finally, I want to thank my family—Ana Guerra, Sandra Regina, and Thamiris Herrera—and friends for helping me focus on this project and filling me with good energy. Thanks to them, I was able to keep my motivation to work hard and finish the book.

Introduction

As a child, I played many games that were very similar—games like Super Mario Bros., Donkey Kong, The Lion King, and Aladdin. I could switch between them without much work; the learning ramp-up was quick. They all shared the same core mechanics: you move straight to the right, jump on platforms, and avoid being hit by enemies. They were all 2D platform games.

Switching between programming languages is similar. In my work, I have needed to switch between Ruby, JavaScript, and CoffeeScript, and between Java, Python, and Objective-C. It wasn't too painful to do. All these languages are very different, but in some ways they are similar. I could use object-oriented programming with all of them. When I learned how to create objects and methods, all the dots started to connect and the languages became familiar quickly.

After playing 2D platform games, I switched to fighting games. They were still games. They were still 2D. However, the challenges and mechanics were completely different. Instead of going straight to the right and jumping the obstacles, I needed to punch and kick the enemies in a limited space. I needed to think differently to master this type of game.

That's how I felt when I switched to functional programming. Where were my objects and methods? I made the mistake of applying the concepts that I was used to in a paradigm where they aren't necessary. I was messing up the codebase. I needed to change my thinking. I couldn't program like I had before.

Switching to a new paradigm is very different from simply switching between languages. You need to think differently, or you'll get in trouble.

I invite you to reset your mind before learning functional programming. After reading this book you'll see your old code from a very different perspective. The best part is that most of today's main languages support some functional concepts. Even if you can't switch to Elixir today, you'll be able to apply useful functional concepts in your daily language.

Is This Book for You?

This book is tailored for beginners in functional programming and Elixir. I expect you have some experience in building simple algorithms, debugging errors, and running commands in a terminal, and that you have at least an entry-level knowledge of software development. Any experience in other languages will help you out. You don't need to be an expert because we'll start from scratch.

If you're an object-oriented programmer ready to take the next step, or a college student graduating and looking for a language to work with, this book is for you. If you've tried to program in Elixir before and had a hard time because of the functional programming concepts, this book will give you the knowledge that you need to become a future expert. If you're already an Elixir or functional programming expert, you may find some valuable tips here, but this book probably isn't for you.

What's in This Book?

You'll find a beginner's guide to functional programming concepts and an introduction to Elixir. The book is divided into seven chapters:

Chapter 1, *Thinking Functionally*, on page ?, introduces the main concepts of functional programming that will persist throughout the book. You'll learn why functional concepts matter and help you create better software.

In Chapter 2, *Working with Variables and Functions*, on page ?, you'll start learning Elixir from scratch, from simple expressions to modules. We'll explore the base building blocks of a functional program: *functions*. Anonymous and named functions are introduced here.

Then, in Chapter 3, *Using Pattern Matching to Control the Program Flow*, on page ?, you'll learn how to create conditional code with functions. Pattern matching plays the central role.

Repetition is a fundamental task in any programming language. In Chapter 4, *Diving into Recursion*, on page ?, you'll learn the functional way: recursive functions.

In Chapter 5, *Using Higher-Order Functions*, on page ?, we'll explore how to create better functions that hide complex code. We'll cover how to create functions that can receive or return functions; you'll learn higher-order functions.

Chapter 6, *Designing Your Elixir Applications*, on page ?, is about creating a larger application and organizing it. We'll explore how to model data, create contracts, and achieve polymorphism using Elixir.

Finally, in Chapter 7, *Handling Impure Functions*, on page ?, we'll look at the concept that finishes this journey: how to work with impure functions. We'll explore the pros and cons of four strategies: conditional code, exception handling, monads, and Elixir's with.

At the end of the book you'll find two appendixes. In Appendix 1, *Adding Rooms to the Game*, on page ?, you'll find extra challenges for the game you developed in Chapter 6, *Designing Your Elixir Applications*, on page ?. In Appendix 2, *Answers to Exercises*, on page ?, you'll find the answers for the exercises.

Using Elixir

Elixir is a functional programming language that runs in the Erlang VM, a powerful environment to run distributed systems. I've chosen Elixir for this book because of its fun syntax, the vibrant community, and the production-ready tooling. Elixir syntax lets you focus on what's important while learning functional programming.

Installing Elixir

Elixir needs Erlang to run; the Elixir installer installs Erlang for you. There's not a lot to say about the Elixir install steps if you follow the official Elixir installation guide.[1] It covers everything you need to know to install Elixir in each of the main operating systems. Read the guide, and be sure to install the latest Elixir version (1.6.0 or newer) so you can follow along with the examples in the book.

Running the Code

For some examples, you'll need to write commands in your terminal. They will look like this:

```
$ elixir -v
Erlang/OTP 20 [erts-9.2] [source] [64-bit] [smp:4:4] [ds:4:4:10]
[async-threads:10] [hipe] [kernel-poll:false]

Elixir 1.6.0 (compiled with OTP 19)
```

The command elixir -v goes after the $ sign. Press Enter after typing the command to see the result. If you try that command, the result will show you that you have Elixir 1.6.0 installed (or a newer version).

1. https://elixir-lang.org/install.html

We'll also work with some Elixir tools that use the terminal, especially in Chapter 6, *Designing Your Elixir Applications*, on page ?. The main tool we'll use in many examples is Elixir's interactive shell, IEx. Try it:

```
$ iex
Erlang/OTP 20 [erts-9.2] [source] [64-bit] [smp:4:4] [ds:4:4:10]
[async-threads:10] [hipe] [kernel-poll:false]

Interactive Elixir (1.6.0) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)>
```

You'll find this interactive shell very useful for quickly trying Elixir code and concepts, and gathering information to debug local and remote systems. Type the code that runs inside the IEx shell after the iex> prompt and press Enter to see the result. For example,

```
iex> IO.puts "Hello, World"
Hello, World
:ok
```

Inside IEx, you can press the Tab key to use autocomplete. You can exit by pressing Ctrl+C two times.

Moreover, some code will look like this:

introduction/hello_world.exs
```
IO.puts "Hello, World!"
```

The top line has the name of the file, with an exs (for script files) or ex (for compiled files) extension. You can execute the code inside of the files using the terminal, like this:

```
$ elixir hello_world.exs
Hello, World!
```

That's everything you need to know to use Elixir and run most of the examples in the book.

Online Resources

You can find all the examples, a form to submit errata, and a community forum for this book on the Pragmatic Bookshelf website.[2] Additionally, you can get in touch with me and your fellow readers in the Elixir community forum for this book.[3]

2. https://pragprog.com/book/cdc-elixir/learn-functional-programming-with-elixir
3. https://elixirforum.com/t/learn-functional-programming-with-elixir-pragprog/5114