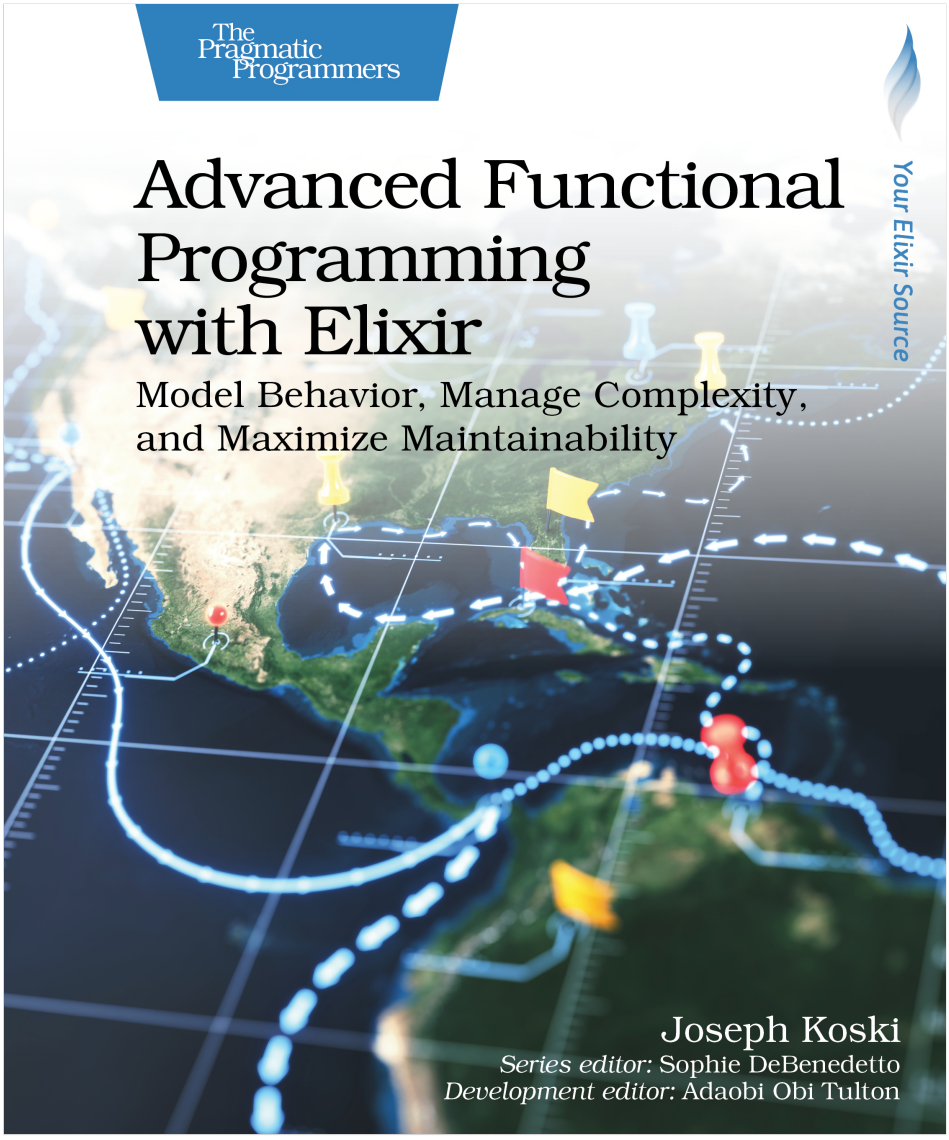

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Introduction

The question I hear most often about functional programming is, "Why bother?" It feels like overkill. Code reuse is a myth and most abstractions rot faster than they're shared. Instead of clarity, you get a maze of indirection. Onboarding suffers. And in the end, the business just wants working code—they don't care how you get there.

Fair enough. Functional programming won't fix a broken project any more than switching frameworks will fix a broken team. If your domain is unclear, your boundaries are a mess, and no one understands how the parts fit together, adding monads won't help. At best, you'll just find new ways to express confusion.

But if you care about the long game, about building systems that are easier to understand and safer to change, then functional programming helps. Not because it's clever but because it demands clarity. The goal is confidence: confidence that your code behaves, that your logic composes, that your abstractions scale.

Because once fear sets in, no one wants to touch the core, they just patch the edges and hope nothing breaks.

I, like many developers, come from a music background. There, mastery was unmistakable—you could hear it. From the outside, it looked like talent. But inside, we knew better: it came from hours of focused practice. Not playing what you already knew, but slogging through the uncomfortable bits just beyond your reach. As Anders Ericsson explains in *Peak [EP16]*, the mythic 10,000-hour rule is nonsense—it's not the time, it's the grind.

But why grind at all? As Daniel Pink shows in *Drive [Pin09]* we're motivated by mastery. We want to understand, improve, and take control. Functional programming helps by sharpening how we think—its mental models are explicit, and composable. They shift our focus from sequences to transformations, from behavior to relationships.

Most functional programming books drop you into the deep end—dense syntax, abstract theory, and jargon with little connection to real code. This one takes a different path. It doesn't pretend learning functional programming is easy. It's not. But mastery depends on knowing where you're going—step by step, with each concept building on the last. Not by simplifying the ideas, but by making them learnable: deliberate, cumulative, and within reach.

How This Book Works

You'll start with foundational patterns like equality and ordering and build toward more powerful structures: monoids, predicates, and eventually monads. Each chapter adds a concept, connects it to what came before, and pushes a little further.

The examples are written for clarity, not production. They're minimal and focused—meant to show the shape of a problem rather than cover every edge case. Alongside the book, you'll find a production-ready library that implements these abstractions—fully tested, well documented, and ready to use.[1] It's also a good place to find minimal examples and quick references.

The goal of functional programming is to build systems that stay flexible as they grow—able to absorb new requirements without buckling. After all, the only measure of quality that matters is how well your code holds up under change.

Who This Book Is For

This book is for developers who want to sharpen how they think about code. You might already be working in Elixir, or you might come from another language and are curious about functional programming—or even about Elixir itself. No prior functional programming experience is required—just the ability to write basic programs in any modern language, along with curiosity, patience, and a willingness to explore new ideas.

Online Resources

Visit the book's page on the Pragmatic Bookshelf website to download the code and access related resources.[2] If you own the ebook, each code example also includes a small gray box you can click to download the code snippet.

1. https://jkwa.github.io/funx/readme.html
2. https://pragprog.com/titles/jkelixir

Join the conversation on the book's DevTalk forum to discuss topics, share ideas, and report errata that could shape future editions.[3]

Conventions Used In This Book

This book builds functional programming mental models through the story of FunPark. We start with general FP patterns, apply them in the FunPark source code, and see these patterns in action through interactive sessions (iex). These interactive sessions appear shaded and are woven into the narrative.

The "Joe asks" asides answer the "what about…?" moments, and tip asides offer pointers to help the reader spot something important, clear up a misunderstanding, or make a connection.

With the map in hand, let's explore what's around the bend!

3. http://devtalk.com/books/advanced-functional-programming-with-elixir