Table of Contents

Preface

Elixir offers developers the functional power and concurrent resilience of Erlang, with friendlier syntax, libraries, and metaprogramming. Elixir compiles to Erlang byte code, and you can mix and match it with Erlang and Erlang tools. Despite a shared foundation, however, Elixir feels very different: perhaps more similar to Ruby than to Erlang's ancestor Prolog.

Introducing Elixir will give you a gentle guide to this powerful language.

This release of *Introducing Elixir* covers version 1.3. We will update it as the language evolves. If you find mistakes or things that have broken, please let us know through the errata system.

Who This Book Is For

This book is mostly for people who've been programming in other languages but want to look around. Maybe you're being very practical, and a distributed model, with its resulting scale and resilience advantages, appeals to you. Maybe you want to see what this "functional programming" stuff is all about. Or maybe you're just going for a hike, taking your mind to a new place.

We suspect that functional programming is more approachable before you've learned to program in other paradigms. However, getting started in Elixir—sometimes even just installing it—requires a fair amount of computing skill. If you're a complete newcomer to programming, welcome, but there will be a few challenges along the way.

Who This Book Is Not For

This book is not for people in a hurry to get things done.

If you already know Elixir, you don't likely need this book unless you're looking for a slow brush-up.

If you already know Erlang, this book will give you an opportunity to see how things are different, but odds are good that you understand the key structures.

If you're already familiar with functional languages, you may find the pacing of this gentle introduction hopelessly slow. Definitely feel welcome to jump to another book or online documentation that moves faster if you get bored.

What This Book Will Do For You

You'll learn to write simple Elixir programs. You'll understand why Elixir makes it easier to build resilient programs that can scale up and down with ease. You'll be able to read other Elixir resources that assume a fair amount of experience and make sense of them.

In more theoretical terms, you'll get to know functional programming. You'll learn how to design programs around message passing and recursion, creating process-oriented programs focused more on data flow.

Most importantly, the gates to concurrent application development will be opened. Though this introduction only gets you started using the incredible powers of the Open Telecom Platform (OTP), that foundation can take you to amazing places. Once you've mastered the syntax and learned about Elixir's expectations for structuring programs, your next steps should be creating reliable and scalable applications—with much less effort than you would have needed with other approaches!

How This Book Works

This book tries to tell a story with Elixir. You'll probably get the most out of it if you read it in order at least the first time, though you're always welcome to come back to find whatever bits and pieces you need.

You'll start by getting Elixir installed and running, and looking around its shell, IEx. You'll spend a lot of time in that shell, so get cozy. Next, you'll start loading code into the shell to make it easier to write programs, and you'll learn how to call that code and mix it up.

You'll take a close look at numbers, because they're an easy place to get familiar with Elixir's basic structures. Then you'll learn about atoms, pattern matching, and guards

—the likely foundations of your program structure. After that you'll learn about strings, lists, and the recursion at the heart of much Elixir processing. Once you've gone a few thousand recursions down and back, it'll be time to look at processes, a key part of Elixir that relies on the message-passing model to support concurrency and resilience.

Once you have the foundation set, you can take a closer look at debugging and data storage, and then have a quick look at a toolset that is likely at the heart of your long-term development with Elixir: Erlang's Open Telecom Platform, which is about much much more than telephones.

Finally, you'll learn about Elixir's macro tools, features that give Elixir tremendous flexibility by letting you extend the language.

Some people want to learn programming languages through a dictionary, smashing together a list of operators, control structures, and datatypes. Those lists are here, but they're in Appendix A, not the main flow of the book.

The main point you should get from this book is that you can program in Elixir. If you don't get that, let us know!

Other Resources

This book may not be the best way for you to learn Elixir. It all depends on what you want to learn and why. If you're looking for a faster-flying introduction to the language, *Programming Elixir* (Pragmatic Publishers) jumps in more quickly and emphasizes Elixir's uniqueness more frequently. *Elixir in Action* (Manning) and *The Little Elixir & OTP Guidebook* (Manning) are similarly faster and deeper. *Metaprogramming Elixir* (Pragmatic Publishers) explores a key corner, and *Programming Phoenix* (Pragmatic Publishers) dives into a powerful Elixir-based framework.

If you like the pace of this book and want to try out your new knowledge, you might like *Études for Elixir* (O'Reilly). That book provides descriptions of short programs that you can write in Elixir, and they may ask you to stretch a bit beyond the examples you find here. It is also designed so that its chapters are in parallel with this book's chapters.

The other books in the field all cover Erlang, not Elixir. Hopefully there will be more Elixir-specific work soon. The main Elixir website includes a lot of tutorials, documentation, and links to other resources.

If your primary interest in learning Elixir is to break out of a programming rut, you should check out Bruce Tate's wild tour of *Seven Languages in Seven Weeks* (Pragmatic Publishers), which explores Ruby, Io, Prolog, Scala, Erlang, Clojure, and Haskell. Erlang gets only (an excellent) 37 pages, but that might be what you want.