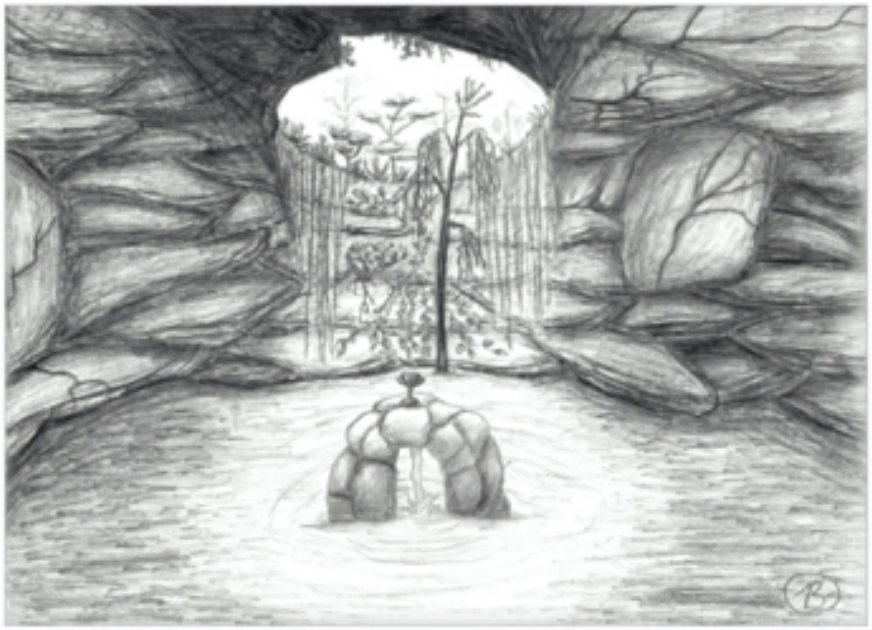
The Pragmatic Programmers

grox.io Learning

Programmer Passport

Elixir



Bruce A. Tate

Edited by Jacquelyn Carter

Programmer Passport: Elixir

Bruce Tate

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*.

For sales, volume licensing, and support, please contact *support@pragprog.com*.

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

ISBN-13: pending
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—Monthname, yyyy

Contents

Change History

As with the other books in the Programmer Passport series, the book you're reading is in beta. This means that we update it frequently, and it has not received the full editorial pass it will receive before it's final form. Here is the list of the major changes that have been made at each beta release of the book, with the most recent change first.

B1.0: March 16, 2000

- In this Programmer Passport language, you'll see more content than you've seen so far. That's true because these chapters will form the foundation of Groxio's Elixir and OTP coursework. In particular, you'll see two chapters instead of one for the first two units. AFter that, we'll see!
- Elixir tooling. The second language of our Joe Armstrong celebration, Elixir would not exist without Joe's work on the BEAM. In this release, the first chapter focuses on Elixir's tooling, including Mix, IEx, and Hex.
- Elixir data. We'll focus on primitive types, and the techniques you'll use to design Elixir modules.

CHAPTER 1

Sweet Tooling

Elixir is one of the most important languages created in the past decade. It's a functional language, meaning the underlying concepts deal with mathematical functions. It's an immutable language, meaning Elixir programs won't *mutate* or *change* values, opting instead to have functions that *transform* values. Its smooth, friendly, Ruby-inspired syntax makes it understandable by a generation of object oriented programmers. Its Erlang foundations make Elixir massively scalable with excellent features for reliability.

Most importantly, Elixir has libraries and tooling for solving some of the most important problems of our day. It has several growing communities within the overall Elixir umbrella, and each community has impressive libraries and infrastructure.

Phoenix is a community that's growing rapidly. It has a web server that's stunningly reliable and concurrent. Though Elixir isn't fast when measured on a single core, it's incredibly concurrent, leading Phoenix to accumulate staggering statistics at scale. A single Phoenix box can serve hundreds of thousands of concurrent users for some use cases. The OTP foundation leads to great uptime numbers for Elixir applications. Phoenix offers these advantages alongside a development model that's rich enough for experts but simple enough for intermediate developers. A new library called Phoenix LiveView leverages these strengths to build highly interactive web applications without involving custom JavaScript, leading to excellent productivity.

The Nerves project is another Elixir community that is growing hand over fist. Most developers of embedded devices use C, and a few are starting to use Python. Nerves offers better tooling alongside the reliability features of Elixir. As embedded chip designers begin to embrace multiple cores, the concurrency advantages of Elixir will begin to tell. Like Phoenix, Nerves is experiencing explosive growth.

The Elixir tooling stack is quite rich for such a young language. It has a package manager called Hex, one that is open to both Elixir and Erlang projects. Hex provides a place to share common infrastructure and also a way to resolve dependencies. Another tool, called mix, provides a way to wrap up development tasks such as running tests, compiling projects, and the like. Elixir also provides a command line shell, debugging services that work both locally and remotely, and an inspector that makes it easy to see all of an application's processes.

Elixir is the second language for our Joe Armstrong tribute and celebration. Elixir is a language based on Joe's creation, Erlang. Before we dive into Elixir, we're going to spend a few paragraphs on why Erlang remains so important, even thirty years after its creation.

Based on Erlang

A team at Erickson, including Joe Armstrong, was building applications to work with phone switches. Erlang, the movie,[1] tells this story. These telecom programs had to be extremely reliable, with real time performance, and excellent concurrency. Because of Joe's experience with Prolog, they wanted a language that would work in a declarative style. They quickly ruled out languages like C and Smalltalk because they were not declarative or expressive enough for the problems Erickson was solving. They also ruled out existing functional languages like Prolog and ML because those languages did not have the support for concurrency or low-level constructs for dealing with the bits that often showed up in hardware problems. Reluctantly, they decided to build a new language from scratch.

Erlang Escapes the Lab

Over time, the team built Erlang, and the new language quickly accumulated an impressive list of successes in Erickson. They established a way of building generic services called GenServers, and wrapped that up into a library called OTP (the Open Telephony Protocol). This library established a strategy for dealing with concurrency and failure in a uniform way. Eventually, Erlang was released beyond Erickson and established a growing following as a language for building reliable infrastructure.

1. https://www.youtube.com/watch?v=xrIjfIjssLE

OTP and the BEAM: Erlang's Crown Jewels

The centerpieces of the Erlang language are the virtual machine it runs on, called the BEAM, and the OTP framework for running scalable, reliable services. As it grows, the BEAM is becoming known for very high concurrency, responsive performance, high reliability, and excellent support for processes. OTP runs on the BEAM, and it describes an application's processes and lifecycles so that individual pieces can be easily shut down and restarted upon failure.

These changes make Erlang one of the most reliable programming languages in existence. When any part of a distributed Erlang system crashes, the infrastructure can simply shut it down and restart it.

These advantages have gathered a fierce following among a small set of infrastructure and application developers. So far, Erlang has yet to break through into the main stream. Most people believe that when a BEAM-based language does break through, it will be Elixir. Let's find out why José Valim developed Elixir.

Beyond Erlang

José was a leader in the community supporting Ruby on Rails, one of the most successful web development frameworks ever built. José was an author, core team member, and framework developer in that Ruby space with a large following, but he was growing increasingly frustrated with some of the problems he encountered related to scalability and programming abstractions. Ruby was an excellent language, but not appropriate for the types of problems José was solving most frequently:

- The Rails framework was not explicit, so it was difficult to debug and extend.
- The Ruby language did not support concurrency well, and José felt concurrency would become increasingly important.
- Ruby applications did not scale particularly well for many users.

In 2011, José started working on a programming language to solve some of these problems. He studied programming languages and settled on Erlang as a foundation.

Let's clarify one misconception right now. Elixir is not just Erlang with a different syntax. It is a modern language with important new features, and a wide suite of tools that Erlang has traditionally lacked. In this section, we'll look at some of those highlights.

Mass Appeal

As a longtime member of the Ruby on Rails community, José has a keen understanding of language adoption. To grow rapidly, a language must be easy to learn, and that means it needs convenient and popular syntax, strong documentation, and approachable tooling. Elixir has all of these things in spades. Based on Ruby, Elixir's syntax is far more approachable to a large audience than Erlang. It's not that Elixir's syntax is inherently better. The language Elixir syntax is based on, Ruby, is just far more popular than the language Erlang is based on, Prolog.

Having effective and consistent documentation is also critical. From the very beginning, José took many of the successful documentation strategies from Ruby and moved them to Elixir. He also enlisted help from those who are good at documentation. The result is not just a tool set, but a culture around good documentation.

Elixir also provides tools that are critical to those who would build early libraries: a good build tool called `mix` and a package manager to store and share dependencies called Hex. These tools ensured that when others were ready to contribute to Elixir, they could immediately build and share tools.

New Abstractions

In the thirty years since Erlang's creation, the state of the art for what makes an effective language has advanced. Elixir built in several critical advancements that Erlang supports only poorly, or not at all. Protocols provide a way to safely extend types to support new functions. Streams are an abstraction for very long or infinite data sets. Structs allow the rapid creation of structured data types. Macros allow the rapid creation of advanced functions in Elixir itself. Elixir pipes make Elixir easier to learn by letting beginners write programs as a series of transformations. We'll look at each of these features throughout the next four chapters.

Taken separately, these features are interesting. Taken together, they elevate Elixir programming as a whole. More higher level abstractions in the hands of a good programmer improve productivity, allow better reliability, and make language learning easier.

These features help Elixir extend the reach of the BEAM to more developers, and potentially make each developer more effective. Effectively, Elixir is drawing a whole new user base into the Erlang ecosystem.