The Pragmatic Programmers

grox.io Learning

Programmer Passport

Crystal: Fast as C & Slick as Ruby

Bruce A. Tate

Edited by Jacquelyn Carter

Crystal

Bruce A. 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

The book you're reading is in beta. This means that we update it frequently. 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: Sept 16, 2019

- Initial beta release. Welcome to the Programmer Passport! We'll continue to add on chapters on a regular schedule. We'd love to hear what you have to say.

B2.0: Oct 1, 2019

- Beta release of Crystal Chapter 2. We focus on working with data in collections and using modules and classes. Enjoy!

B3.0: Oct 15, 2019

- In this beta release, we'll move away from introducing the language and focus on using it to solve a problem. We'll pay special attention to both types and macros, using both of them to build a rover. We move away from exercises from Exercism to include our first project, the Mastermind game.

B2.0: Nov 1, 2019

- Send your thoughts to our editor, Jackie Carter, if you're so inclined! She's been evacuated from her home. I think she may be moving back in now. This version is lightly edited. We'll post a fully edited version of this chapter if there are substantive changes. Even so, the chapter is content complete.

Beta release of the final Crystal beta, Chapter 4. We work with shards for dependencies and embrace concurrency. We build a basic web server project.

If you like what you see, let your friends know! It's an extraordinary value. The next language we'll do together will be Pony. We hope you're enjoying Crystal!

P.S. We're considering doing a rotation or two based on technologies that Joe Armstrong helped with or loved himself to commemorate his passing. Let us know at bruce@grox.io if you have a strong opinion!

Preface

I opened up my passport yesterday and looked at all of the stamps sprinkled throughout, triggering fond memories of the places and people I've encountered in my travels. Learning new languages is very much that way for me. Welcome to Programmer Passport.

We're glad you're here. Twenty years ago in The Pragmatic Programmer, my friend and business partner Andy Hunt and my longtime mentor Dave Thomas charged us with learning a new language every year. I took that advice and never gave up the habit. In this series we'll learn languages together because doing so will make us better programmers. I strongly believe learning even obscure languages teaches common patterns that can be applied in your day to day job.

Ten years later, I wrote a book called Seven Languages in Seven Weeks, and that experience opened a whole new world to me. It's my fondest hope that Programmer Passport will do the same for you, offering a new language with new content every two weeks. Here's what you can expect.

- Every two months, we'll vote on the next language to cover. I'll suggest a few languages that build on the one we're doing currently and we'll choose one of those.
- Twice a month we'll release a new piece of the book and a new video.
- We'll mix in a few links for you to explore with projects or exercises for you to try.

It's the first week, so please be patient while we work the bugs out and give you features to make the experience more pleasant. We hope you enjoy the journey together!

CHAPTER 1

Crystal Clear Philosophy

For as long as we've been building things and taking midday meals, engineers have been searching franticly for a free lunch. You've been part of enough quests to know no such thing exists. Crystal's central philosophy, from the beginning, is enticing: fast like C, slick like Ruby. Adding C-like performance to a Ruby-like language, for free, is an ambitious goal. Almost every great language has clear goals like this one.

This book will tease out what makes Crystal special. Fair warning, though, we're going to talk a lot about Ruby. If you're already working up a righteous anger, give us a few minutes to explain why. Historically, Crystal was initially intended to be a fully compatible replacement to Ruby, but one that is more type safe and performant. If you don't have a Ruby background, don't worry. Your dutiful guide will bring you along gently.

The first chapter will focus on Crystal's promises of performance and ease of use, making sure to linger on the direct benefits of Crystal's approach to deliver both of these goals. Every lunch has a price so we'll foreshadow those hidden costs that might sneak in. At this early stage, we'll spend most of our time together getting the tools for Crystal set up and working some basic exercises to limber up our muscles. Our goals will be pretty modest:

- Get the language installed
- Understand the basic tools at our disposal
- Learn to use the playground and notebook to explore Crystal
- Understand the simple primitive types and operators in the language

Along the way, I'll point out some of the places you can go to find interesting exercises, and this book will offer a few puzzles and exercises as well.

Meet Crystal

Without further ado, it's time to introduce Crystal, the language that's built to do many of the things C can do but with the friendliness of Ruby. Based on rapid adoption within the Ruby community, it's a quickly growing language and it has an impressive list of strengths, including some nice feature groupings you don't usually find together in the same language.

Crystal was initially designed to be a drop in replacement for Ruby, but one with a type system friendlier to compile-time checking, tooling and compilers. As the language has grown it has dropped complete compatibility with Ruby to better support the goals of the type system. Still, Crystal and Ruby syntax are remarkably similar. Let's look at what makes Crystal special.

The Essence

With heavy Ruby inspiration for library structure, syntax, and code organization, Crystal is plenty smooth. As a language with some type optimizations and a compiler, its performance in production is extremely fast compared to Ruby.

Ruby is an interesting, sometimes controversial language that is extremely concise and productive in the short term, at the cost of less protection from runtime bugs and slower runtime performance. Crystal provides a bit more rigor with compilation and a type system. The compiler can catch many problems before a program is ever run.

Sometimes, with languages such as Java, a type system can levy pretty heavy taxes of extra keystrokes. Rather than requiring the developer to explicitly express types, the Crystal compiler can infer most of them based on clues in the rest of a program. Crystal certainly maintains many of Ruby's syntactic benefits, as you can see in this idiomatic example from the first few pages of Crystal's online tutorial:[1]

```
require "http/server"

server = HTTP::Server.new do |context|
  context.response.content_type = "text/plain"
  context.response.print "Hello world! The time is #{Time.now}"
end

address = server.bind_tcp 8080
puts "Listening on http://#{address}"
server.listen
```

1. https://crystal-lang.org/reference/overview/http_server.html

I'm a particular fan of Ruby's syntax, and Crystal's is nearly identical. If you've done very much with Ruby, the syntax of the previous example will, um, crystallize for you right away, from the code blocks to string interpolation. That code is a simple web server that we'll cover in more detail later in this chapter. We'll get into language features as the book goes on, but for now, let's whet your appetite.

Fact 1. Crystal's type system addresses problems such as nil pointers.

What you might not get from the previous example is that Crystal has a much different type system than Ruby, one that makes nil pointer exceptions much more rare. The type system is more static, but not oppressively so from a programming requirements standpoint. Consider this snippet that declares two different arrays:

```
jagged_glass = [] of String?
fluffy_pillows = [] of String

jagged_glass[0] = nil  # OK
fluffy_pillows[0] = nil # ERROR!
```

The first array can have nil values, the second can't. You'll see that the program will force you to think about where nil values can occur and even raise exceptions when you make a mistake. We'll dive more into the type system as we go.

Type systems can't keep you from making many logical errors, but they can save you from a whole class of problems including softening the ways nil values can occur in production.

Fact 2. Crystal's speed sometimes makes it a reasonable C replacement.

Crystal is indeed a compiled language, so the results you get are in the same ballpark as what you'd find in C. Since Crystal has a much better story for higher level programming constructs and automatic garbage collection, it's often a better choice for attacking those low-level components you can sometimes find yourself building as a performance optimization. You'll build C interfaces with *bindings*. They are easy to declare, like this:

```
lib C
fun cos(value : Float64) : Float64
end
```

That binding makes the C function cos, which takes a C double and returns a double, available to Crystal programs. Slick. You don't have to fully understand what's happening here. For now, understand that the Crystal language makes language interfaces pretty simple.

Fact 3. Crystal has lightweight threads called fibers.

Parallelism and concurrency aren't the same. Crystal's support for parallelism is still young, several days old as I write this. Still, the story for concurrency is very good. If you want to build solutions that take advantage of the latency of things like database access, you can use fibers through Crystal channels. They look like this:

```
spawn do
puts "I'm feeling sleepy."
sleep 5.seconds
puts "I'm wide awake."
end

Fiber.yield
puts "Just a small town girl, living in a lonely world..."

sleep 5.seconds
```

That program produces this output:

```
I'm feeling sleepy.
Just a small town girl, living in a lonely world...
I'm wide awake.
```

That strategy is called cooperative multitasking. Programs have to voluntarily release control. This program is just a small taste of Crystal's concurrency, an interesting take not all of us have seen before. The `spawn` function starts up a thread and does the work inside the `do`/`end` block, printing messages to the console and sleeping.

After the spawn, the thread explicitly yields control to other threads and then does some more prints and sleeps. Cooperative multitasking has the benefit of reducing overhead, but also ups the ante, making bugs potentially more serious. It's surprisingly efficient so it allows a nice level of concurrency, without having to manage as many details as C developers.

Now you've had the tiniest glimpse of how this Crystal is cut. With luck, your curiosity should be kicking in. With those high level details in mind, we can shift our attention to the language's creation and why the creators made the decisions they did.

The Creation of Crystal

The article The Story Behind #CrystalLang[2] tells the story of the creation of Crystal. In Argentina in 2011, Ary Borenszweig wondered if he would be able

2. https://manas.tech/blog/2016/04/01/the-story-behind-crystal.html