Extracted from:

Seven Languages in Seven Weeks

A Pragmatic Guide to Learning Programming Languages

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Seven Languages in Seven Weeks

A Pragmatic Guide to Learning Programming Languages

Bruce A. Tate

Edited by Jacquelyn Carter

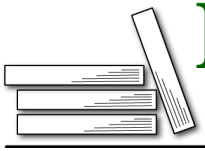
Seven Languages in Seven Weeks

A Pragmatic Guide to Learning Programming Languages

Bruce A. Tate

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Jackie Carter (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
Steve Peter (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-934356-59-3
Printed on acid-free paper.
Book version: P5.0—March 2012

CHAPTER 1

Introduction

People learn spoken languages for different reasons. You learned your first language to live. It gave you the tools to get through your everyday life. If you learned a second language, the reasons could be very different. Sometimes, you might have to learn a second language to further your career or adapt to a changing environment. But sometimes you decide to conquer a new language not because you have to but because you want to learn. A second language can help you encounter new worlds. You may even seek enlightenment, knowing every new language can shape the way you think.

So it is with programming languages. In this book, I will introduce you to seven different languages. My goal is not to make a motherly demand like your morning spoonful of cod liver oil. I want to guide you through a journey that will enlighten you and change the way you look at programming. I won't make you an expert, but I'll teach you more than "Hello, World."

1.1 Method to the Madness

Most of the time, when I'm learning a new programming language or framework, I'll look for a quick interactive tutorial. My goal is to experience the language in a controlled environment. If I want, I can go off script and explore, but I'm basically looking for a quick jolt of caffeine, a snapshot of syntactic sugar, and core concepts.

But usually, the experience is not fulfilling. If I want to get the true flavor of a language that is more than a subtle extension of one I already know, a short tutorial is *never* going to work. I need a deep, fast dive. This book will give you such an experience not once but seven times. You'll find answers to the following questions:

- *What is the typing model?* Typing is strong (Java) or weak (C), static (Java) or dynamic (Ruby). The languages in this book lean on the strong

typing end of the spectrum, but you'll encounter a broad mix of static and dynamic. You will find how the trade-offs impact a developer. The typing model will shape the way you attack a problem and control the way the language works. Every language in this book has its own typing idiosyncrasies.

- *What is the programming model?* Is it object-oriented (OO), functional, procedural, or some type of hybrid? This book has languages spanning four different programming models and, sometimes, combinations of more than one. You will find a logic-based programming language (Prolog), two languages with full support for object-oriented concepts (Ruby, Scala), four languages that are functional in nature (Scala, Erlang, Clojure, Haskell), and one prototype language (Io). Several of the languages are multiparadigm languages, like Scala. Clojure's multimethods will even let you implement your own paradigm. Learning new programming paradigms is one of the most important concepts in this book.

- *How will you interact with it?* Languages are compiled or interpreted, and some have virtual machines while others don't. In this book, I'll begin to explore with an interactive shell, if there is one. I will move on to files when it's time to attack bigger projects. We won't attack large enough projects to fully dive into packaging models.

- *What are the decision constructs and core data structures?* You'd be surprised how many languages can make decisions with things other than variations of `if`s and `while`s. You'll see pattern matching in Erlang and unification in Prolog. Collections play a vital role in just about any language. In languages such as Smalltalk and Lisp, the collections are defining characteristics of the language. In others, like C++ and Java, collections are all over the place, defining the user's experience by their absence and lack of cohesion. Either way, you'll need a sound understanding of the collections.

- *What are the core features that make the language unique?* Some of the languages will support advanced features for concurrent programming. Others provide unique high-level constructs such as Clojure's macros or Io's message interpretation. Others will give you a supercharged virtual machine, like Erlang's BEAM. Because of it, Erlang will let you build fault-tolerant distributed systems much more quickly than you can in other languages. Some languages support programming models that are laser-focused on a particular problem, such as using logic to solve constraints.

When you're through, you will not be an expert in any of these languages, but you *will* know what each uniquely has to offer. Let's get to the languages.

1.2 The Languages

Choosing the languages in this book was much easier than you might imagine. I simply asked potential readers. When we rolled up all the data, we had eight potential candidates. I struck JavaScript because it was *too* popular and replaced it with the next most popular prototype language, Io. I also struck Python because I wanted no more than one object-oriented language, and Ruby was higher on the list. That made room for a surprising candidate, Prolog, which was a top-ten language on the list. These are the languages that did make the cut and the reasons I picked them:

- *Ruby.* This object-oriented language gets high marks for ease of use and readability. I briefly considered not including any object-oriented language at all, but I found myself wanting to compare the different programming paradigms to object-oriented programming (OOP), so including at least one OOP language was important. I also wanted to push Ruby a little harder than most programmers do and give readers a flavor for the core decisions that shaped the design of Ruby. I decided to take a dive into Ruby metaprogramming, allowing me to extend the syntax of the language. I'm quite happy with the result.

- *Io.* Along with Prolog, Io is the most controversial language I included. It is not commercially successful, but the concurrency constructs with the simplicity and uniformity of syntax are important concepts. The minimal syntax is powerful, and the similarities to Lisp are sometimes striking. Io has a small footprint, is a prototype language like JavaScript, and has a unique message dispatch mechanism that I think you'll find interesting.

- *Prolog.* Yes, I know it's old, but it is also extremely powerful. Solving a Sudoku in Prolog was an eye-opening experience for me. I've worked hard to solve some difficult problems in Java or C that would have been effortless in Prolog. Joe Armstrong, creator of Erlang, helped me gain a deeper appreciation of this language that strongly influenced Erlang. If you've never had an occasion to use it, I think you will be pleasantly surprised.

- *Scala.* One of a new generation of languages on the Java virtual machine, Scala has brought strong functional concepts to the Java ecosystem. It also embraces OOP. Looking back, I see a striking similarity to C++,

which was instrumental to bridging procedural programming and OOP. As you dive into the Scala community, you'll see why Scala represents pure heresy to pure functional programmers and pure bliss to Java developers.

- *Erlang.* One of the oldest languages on this list, Erlang is gathering steam as a functional language that gets concurrency, distribution, and fault tolerance right. The creators of CouchDB, one of the emerging cloud-based databases, chose Erlang and have never looked back. After spending a little time with this distributed language, you'll see why. Erlang makes designing concurrent, distributed, fault-tolerant applications much easier than you could have ever thought possible.
- *Clojure.* Another JVM language, this Lisp-dialect makes some radical changes in the way we think about concurrency on the JVM. It is the only language in this book that uses the same strategy in versioned databases to manage concurrency. As a Lisp dialect, Clojure packs plenty of punch, supporting perhaps the most flexible programming model in the book. But unlike other Lisp dialects, the parentheses are greatly reduced, and you have a huge ecosystem to lean on, including a huge Java library and widely available deployment platforms.
- *Haskell.* This language is the only pure functional language in the book. That means you won't find mutable state anywhere. The same function with the same input parameters will give you the same output, every time. Of all the strongly typed languages, Haskell supports the most widely respected typing model. Like Prolog, it will take a little while to understand, but the results will be worth it.

I'm sorry if your favorite language didn't make the list. Believe me, I've already gotten hate mail from more than a few language enthusiasts. We included several dozen languages in the survey mentioned earlier. Those languages that I picked are not necessarily the best, but each one is unique, with something important to teach you.

1.3 Buy This Book

...if you are a competent programmer who wants to grow. That claim might seem a little nebulous, but indulge me.

Learning to Learn

Dave Thomas is one of the founders of this publishing company. He has challenged thousands of students to learn a new language every year. At

worst, by learning languages, you'll learn to fold new concepts back into the code that you write in your chosen language.

Writing this book has had a profound impact on the Ruby code that I write. It is more functional and is easier to read with less repetition. I reach for mutable variables less and do a better job with code blocks and higher-order functions. I also use some techniques that are unconventional in the Ruby community, but they make my code more concise and readable.

At best, you could launch a new career. Every ten years or so, programming paradigms change. As the Java language became more limiting for me, I experimented with Ruby to better understand its approach to web development. After a couple of successful side projects, I pushed my career hard in that direction and have never looked back. My Ruby career started with basic experimentation and grew into more.

Help for Dangerous Times

Many of the readers of this book won't be old enough to remember the last time our industry switched programming paradigms. Our shift to object-oriented programming had a couple of false starts, but the old structural programming paradigm was simply unable to handle the complexity required for modern web applications. The successful Java programming language gave us a hard shove in that direction, and the new paradigm stuck. Many developers got caught with old skills and had to completely retool the way they thought, the tools they used, and the way they designed applications.

We may be in the midst of another transformation. This time, new computer designs will be the driver. Five of the seven languages in this book have compelling concurrency models. (Ruby and Prolog are the exceptions.) Whether or not your programming language changes right away, I'm going to go out on a limb and say that the languages in this book have some compelling answers to offer. Check out Io's implementation of futures, Scala's actors, or Erlang's "Let it crash" philosophy. Understand how Haskell programmers leave mutable state behind or how Clojure uses versioning to solve some of the most difficult concurrency problems.

You can also find insight in surprising places. Erlang, the language behind the scenes for several of the cloud-style databases, is a great example. Dr. Joe Armstrong started that language from a Prolog foundation.

1.4 Don't Buy This Book

...until you've read this section and agree. I am going to make a deal with you. You agree to let me focus on the programming language rather than installation details. My part of the deal is to teach you more in a shorter time. You'll have to Google a little more, and you can't rely on me to support your installation, but when you're through the book, you'll know much more because I'll be able to dive deeper.

Please recognize that seven languages is an ambitious undertaking for both of us. As a reader, you're going to have to stretch your brain around seven different syntax styles, four programming paradigms, four decades worth of language development, and more. As an author, I have to cover an enormously broad set of topics for you. I learned several of these languages to support this book. To successfully cover the most important details of each language, I need to make some simplifying assumptions.

I Will Take You Beyond Syntax

To really get into the head of a language designer, you're going to have to be willing to go beyond the basic syntax. That means you'll have to code something more than the typical "Hello, World" or even a Fibonacci series. In Ruby, you will get to do some metaprogramming. In Prolog, you'll solve a full Sudoku. And in Erlang, you'll write a monitor that can detect the death of another process and launch another one or inform the user.

The second that I decided to go deeper than the basics, I made a commitment to you and a compromise. The commitment: I won't settle for a superficial treatment. And the compromise: I won't be able to cover some basics that you'd expect to find in dedicated language books. I will rarely go through exception processing, except where it's a fundamental feature of the language. I will not go into packaging models in detail because we'll be dealing with small projects that do not require them. I will not go over primitives that we don't need to solve the basic problems I lay out for you.

I Won't Be Your Installation Guide

One of my biggest challenges is the platform. I have had direct contact from readers of various books using three different Windows platforms, OS X, and at least five different Unix versions. I've seen comments on various message boards of many more. Seven languages on seven platforms is an insurmountable topic for a single author and probably for a multiauthor book. I can't support installation for seven languages, so I'm not going to try.

I suspect that you're not remotely interested in reading another outdated installation guide. Languages and platforms change. I'll tell you where to go to install the language, and I'll tell you what version I'm using. That way, you'll be working from up-to-date instructions from the same list as everyone else. I cannot support your installations.

I Won't Be Your Programming Reference

We've tried hard to get you strong programming reviews for this book. In some cases, we are lucky enough to get a review from the person who designed the language. I'm confident that this material will capture the spirit of each programming language pretty well by the time it has gone through the entire review process. That said, please understand that I cannot possibly fully support your endeavors in each language. I would like to make a comparison to spoken languages.

Knowing a language as a tourist passing through is far different from being a native speaker. I speak English fluently and Spanish haltingly. I know a few phrases in three other languages. I ordered fish in Japan. I asked to find a restroom in Italy. But I know my limitations. From the programming side, I speak Basic, C, C++, Java, C#, JavaScript, Ruby, and a few others fluently. I speak dozens of others haltingly, including the languages in this book. I'm not qualified to support six of the languages on this list. I write Ruby full-time and have for five years now. But I couldn't tell you how to write a web server in Io or a database in Erlang.

I would fail badly if I tried to provide an exhaustive reference for each of these languages. I could make a programming guide that's at least as long as this book on any of the separate languages in here. I will give you enough information to get started. I will walk you through examples in each languages, and you'll see examples of those programs. I will do my best to compile everything and make sure it all runs. But I couldn't support your programming efforts even if I wanted.

The languages on this list all have exceptional support communities. That's part of the reason I picked them. In each of the exercises, I try to have a section that asks you to find resources. This idea is intentional. It will make you self-reliant.

I Am Going to Push You Hard

This book is going to take you one step beyond your twenty-minute tutorial. You know Google as well as I do, and you'll be able to find one of those simple primers for every language on this list. I will give you a quick

interactive tour. You'll also get some small programming challenges and one programming project every week. It's not going to be easy, but it will be informative and fun.

If you simply read this book, you'll experience the flavor of the syntax and no more. If you look online for the answers before trying to code the exercises yourself, you'll fail. You will want to try the exercises first, fully recognizing that you'll fail at a few of them. Learning syntax is always easier than learning to reason.

If you find yourself nervous after reading this description, I suggest that you put down this book and pick up another. You won't be happy with me. You would probably be better served by seven different programming books. But if you find yourself excited about the prospect of coding better quickly, let's push on.

1.5 A Final Charge

At this point, I expected to have some sweeping, motivational words to say, but it all seemed to boil down to two words.

Have fun.