# PRAGMATIC FUNCTIONAL JAVASCRIPT

by

Marcelo Camargo

# Table of Contents

PRAGMATIC
FUNCTIONAL
JAVASCRIPT
by
Marcelo Camargo

# What does "pragmatism" mean?

Pragmatism, noun, a **practical** aproach to problems and affairs. There is no word that can describe this book better. We are going to focus on practical applications of the beautiful and enchanting theory built around functional programming. This book is made for people with basic knowledges on JavaScript programming; you definitely shouldn't have exceptional abilities to understand the things here written: it is made to be practical, simple, concise and after that we except you to be able to write real world applications using the functional paradigm and finally say "I **do** work with functional programming!".

When possible, we'll provide the problem that we are trying to solve with alternative implementations and techniques in other paradigms. Programming is made to solve problems, but sometimes solving a problem may generate several other small problems. Before continuing, we need to set some premises:

- **Programming languages are not perfect**: seriously. They are built and designed by humans, and humans are not perfect (far from that!). They may contain failures and arbitrarily defined features, however, most times there is a reasonable explanation about the "workaround".
- **Problem solving may generate other problems**: if you are worried only about "getting shit done", this might be a bit controversial for you. When you solve a problem, have in mind that your solution is not free from introducing problems that other people will have to solve later. Being open to criticism is a good thing here; this is how technology evolves.
- **The right tool for the right job**: this is pragmatism. We pick the tool that solves the problem and introduces the lowest number of side-effects. Good programmers analyse trade-offs. There are a lot of programming languages published and dozens of paradigms, and they are not made/discovered only because "somebody likes writing that way" or "somebody wants to have their name in a programming language" (at least most times), but because new problems arise. It is sensible, for example, to use Erlang on telephone systems, Agda on mathematical proofs and Go on concurrent systems, but it is definitely insane to use Brainfuck on web development and PHP on compiler development!

# The second chance

> And God said, "let there be functions", and there were functions.

In the beginning, computers were very slow, way slower than running Android Studio on your 2GB RAM machine! When the first physical computers appeared and programming languages started to be designed and implemented, there were mainly 2 mindsets:

1. Start from the Von Neumann architecture and **add abstraction**;
2. Start from mathematics and **remove abstraction**.

When it all started, dealing with high levels of abstraction was very hard and costly, memory and storage were really small in power, so imperative programming languages got a lot more visibility than functional ones because imperative languages had a lower level of abstraction that was way easier to implement and map directly to the hardware. It could not be the best way to design programs and express ideas, but at least it was the faster one to run.

But computers improved a lot, and functional programming finally got its deserved chance! Functional programming is not a new concept. I'm serious, it is really old and came directly from mathematics! The core concepts and the functional computational model came even before physical computers existed. It had its origins on lambda calculus, and it was initally only a formal system developed in the 1930s to investigate computability.

A lot of modern programming languages support well functional programming. Even Java surrended to this and implemented lambda expressions and streams. Most of you program in languages that were created; I want to show you the one which was discovered.