PRAGMATIC FUNCTIONAL JAVASCRIPT

by

Marcelo Camargo

Table of Contents

Introduction 1.1

- What does "pragmatism" mean? 1.1.1
- The second chance 1.1.2
- Less is more 1.1.3

Roots of the evil 1.2

- Null considered harmful 1.2.1
- Mutability can kill your hamster 1.2.2
- Take care with side-effects 1.2.3
- Loops are so 80's 1.2.4
- The Hadouken effect 1.2.5

Meet ESLint 1.3

- Restricting imperative syntax 1.3.1
- Plugins to the rescue 1.3.2

Modules 1.4

- The SOLID equivalence 1.4.1
- Top-level declarations 1.4.2

The power of composition 1.5

- Thinking in functions 1.5.1
- Currying and partial application 1.5.2
- Point-free programming 1.5.3
- Piping and composing 1.5.4
- Combinators 1.5.5

Types 1.6

- Why types matter 1.6.1
- Flow is your friend 1.6.2
- Don't abuse polymorphism 1.6.3
- Algebraic data types 1.6.4

Monads and functors 1.7

- What the hell is a monad? 1.7.1
- Dealing with dangerous snakes 1.7.2
- Handling state 1.7.3
- Exceptions are not the rule 1.7.4

Async programming 1.8

- So you still don't understand promises? 1.8.1
- Futures 1.8.2
- Generators and lazy evaluation 1.8.3

Awesome libraries 1.9

- Ramda 1.9.1
- Folktale 1.9.2

Welcome to Fantasy Land 1.10

- Unicorns and rainbows 1.10.1

Hacking the compiler 1.11

- Extending Babel 1.11.1
- Sweet macros 1.11.2

A bit of theory 1.12

- Lambda calculus 1.12.1

Functional languages targeting JavaScript 1.13

- LiveScript 1.13.1
- PureScript 1.13.2
- ReasonML 1.13.3
- Elm 1.13.4
- ClojureScript 1.13.5

Solving real world problems 1.14

- Integrating with external libraries 1.14.1
- Playing with files 1.14.2
- Network requests 1.14.3
- Testing 1.14.4

Final considerations 1.15

- What should I learn now? 1.15.1

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. We need to have some axioms in mind:

- **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. Most times there is a reasonable explanation about the "workaround", however, and most times there is also a good solution.
- **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.
- **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. 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", 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!