FAQ

Why are snippets written sometimes with semicolons and sometimes without?

> see #6

There are two schools in JavaScript, people who use them, and people who don't. We've made the
choice here to use them, and now, we strive to be consistent with that choice. If some are missing, please let us know and we will take care of the oversight.

Aren't external libraries like _ (ramda) or $ (jquery) making calls impure?

> see #50

Those dependencies are available as if they were in the global context, part of the language. So, no, calls can still be considered as pure.
For further reading, have a look at this article about CoEffects

What is the meaning of `f a` in signature?

> see #62

In a signature, like:

```
map :: Functor f => (a -> b) -> f a -> f b
```

`f` refers to a `functor` that could be, for instance, Maybe or IO. Thus, the signature abstracts

the choice of that functor by using a type variable which basically means that any functor might be used where `f` stands as long as all `f` are of the same type (if the first `f a` in the signature represents a `Maybe a`, then the second one **cannot refer to** an `IO b` but should be a `Maybe b`. For instance:

```
let maybeString = Maybe.of("Patate")
let f = function (x) { return x.length }
let maybeNumber = map(f, maybeString) // Maybe(6)

// With the following 'refined' signature:
// map :: (string -> number) -> Maybe string -> Maybe number
```

Is there any "real world" examples available?

> see #77, #192

Should you haven't reached it yet, you may have a look at the Chapter 6 which presents a simple flickr application.
Other examples are likely to come later on. By the way, feel free to share with us your experience!

Why does the book uses ES5? Is any ES6 version available?

> see #83, #235

The book aims at being widely accessible. It started before ES6 went out, and now, as the new
standard is being more and more accepted, we are considering making two separated editions with
ES5 and ES6. Members of the community are already working on the ES6 version (have a look to
#235 for more information).

What the heck is that reduce function about?

> see #109

Reduce, accumulate, fold, inject are all usual functions in functional programming used to combine the elements of a data structure successively. You might have a look at this talk to get some

more insights about the reduce function.

Wouldn't you use a simplified English rather than the current style?

> see #176

The book is written in its own style which contributes to make it consistent as a whole. If you're not familiar with English, see it as good training. Nevertheless, should you need help to understand the meaning sometimes, there are now several translations
available that could probably help you.

What is Either? What is Future? What is Task?

> see #194

We introduce all of those structures throughout the book. Therefore, you won't find any use of a
structure that hasn't previously be defined. Do not hesitate to read again some old parts if you ever feel uncomfortable with those types.
A glossary/vade mecum will come at the end to synthesize all these notions.

Where do map, filter, compose ... methods come from?

> see #198

Most of the time, those methods are defined in specific vendor libraries such as `ramda` or `underscore`. You should also have a look at
support.js
in which we define several implementations used for the exercises. Those functions are really common in functional programming and even though their implementations may vary a bit, their
meanings remain fairly consistent between libraries.