Purely Functional Programming - More than just a flavor of FP

Adrian Sieber | CTO at qthority.com | 10 years of JavaScript experience

2019-11-27

What Is Functional Programming?

Functional Programming treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

From Wikipedia

Advantages?

Eliminating side effects (changes in state that do not depend on the function inputs) can make understanding a program easier.

From Wikipedia

What About JavaScript?

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles.

From Wikipedia

How Do We Use It?

JavaScript Functional Programming — map, filter and reduce

[redacted] Follow

Oct 24, 2017 · 6 min read

Well, that should be easy, right?

Example

```
const people = [{name: 'John'}, {name: 'Anna'}]
const newPeople = people.map(x => {
  x.color = 'green'
  return x
})
```

What's wrong with this code?

Unexpectedly, the Arrays Look the Same ¯\(o,o)/¯

```
>> people
0: Object { name: "John", color: "green" }
1: Object { name: "Anna", color: "green" }

>> newPeople
0: Object { name: "John", color: "green" }
1: Object { name: "Anna", color: "green" }
```

Somehow we mutated the `people` array ...

What We Actually Wanted

```
const people = [{name: 'John'}, {name: 'Anna'}]
const newPeople = people.map(x => {
  const temp = Object.assign({}, x)
  temp.color = 'green'
  return temp
})
```

So What's the Root Cause?

. . . JavaScript *supports** functional . . .

* But you can still easily shoot yourself in the foot . . .

Another Example

```
const title = 'functional programming is awesome'
const emphasizedTitle = exclaim(title)

console.log(emphasizedTitle)

// FUNCTIONAL PROGRAMMING IS AWESOME!!!
```

Yay, Functional! Right?

- Immutable => Returns a new string and old one isn't changed
- Pure function => Output only depends on input
- No side effects

. . . *or are there?*

WELP!!!

```
function exclaim (string) {
  Rocket.launch({target: gpsModule.getCurrentLocation()})
  globalState.nukeWasLaunched = true
  console.log('We just nuked you. Bye, bye sauerkrautfresser!')
  return string.toUpperCase() + '!!!'
}
```

Oh well, look at those unfortunate side effects:

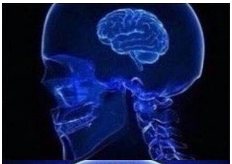
- Depends on location
- Modifies global state
- Logs to standard out
- Does something completely unexpected in the background

Then Let's Just Use One of the Functional Libraries?

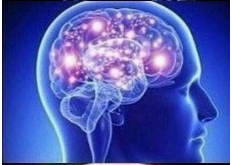
> *Immutability works best when it is pervasive. But the JavaScript language and ecosystem is designed around mutable data, you can't enforce immutability from a library, . . .*

reaktor.com/blog/fear-trust-and-javascript

Functional Concepts



Typed + Functional Concepts



Typed + Functional Architecture

Purely Functional

Purely Functional Programming (e.g. Elm)

- Everything – **no exception** – is written in a functional style
- Everything – **no exception** – is immutable
- Everything – **no exception** – is strongly typed *
- Compiles to JavaScript

* While not strictly part of the definiton, there are no commonly used untyped pure FP languages

Safety++++++++

You can't shoot yourself in the foot.
Even if you try really hard.
The compiler just won't let you!

Example

```
import String exposing (toUpper)

exclaim : String -> String
exclaim string =
  (toUpper string) ++ "!!!"
```

You can run it on ellie-app.com

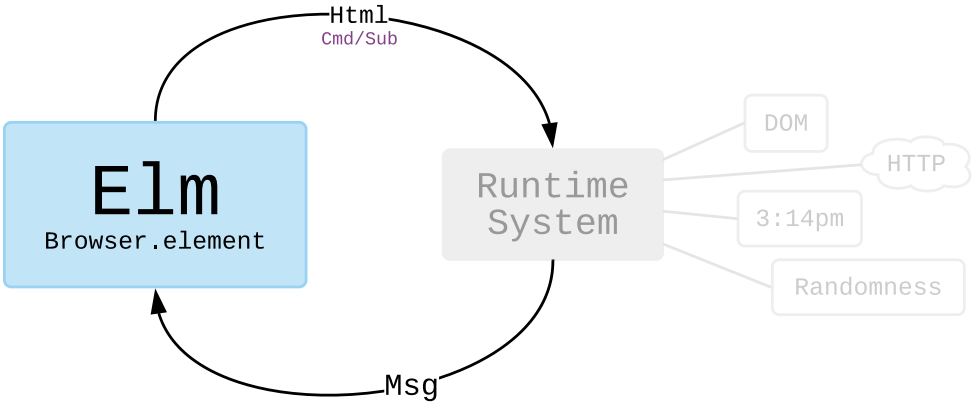
Launch a Rocket, Maybe?

```
exclaim : String -> String
exclaim string =
  (toUpper string) ++ "!!!"
```

- What would `launch rocket` return?
- Only `main` function is allowed to have side effects
- How would you execute several statements if everything is an expression?

There is no way to write that code!

So How Do You Launch a Rocket?



```
type Msg = LaunchRocket
         | LaunchInitiated (Result Http.Error String)

type Model = Waiting
           | Launching
           | LaunchSuccess String
           | LaunchFailure String

update : Msg -> Model -> (Model, Cmd Msg)
...

view : Model -> Html Msg
...
```

Side effects are already visible in the type signature!

All Purely Functional Languages

- **Elm** - No bullshit HTML UIs and apps
- **PureScript** - Full power of JavaScript
  - Compiles to *readable* JavaScript
  - Advanced FP features
- **Haskell** - Mother of all Purely FP languages
  - Compiles to binaries
  - C like performance

Not Completely Functional

- **Rust**
  - Unpure functions
  - No tail-call optimization for recursion
  - https://www.fpcomplete.com/blog/2018/10/is-rust-functional
- **Swift** - Also Object oriented, unpure, mutating, ...
- **Scala** - Also Object oriented
- **Clojure** - Also Object oriented

You can use them in a more or less functional way, but it depends on your discipline and skills.

Summary of Advantages

- You can finally sleep in peace
- When it compiles it runs
- Parallelization at the flip of a switch
- Testing is straight forward
- Onboarding is a breeze
- Maintainability is through the roof

Any questions?

PS: We are looking for working students!
PS2: More rants on Twitter @AdrianSieber