Elm taught me to write code 10 years after my teachers thought me

Alex Nicol - Brighton Web Development Meetup - June 2025

Hi, I’m Alex Nicol

I write code for a living
for fun
because I like it
to make positive changes in people’s lives

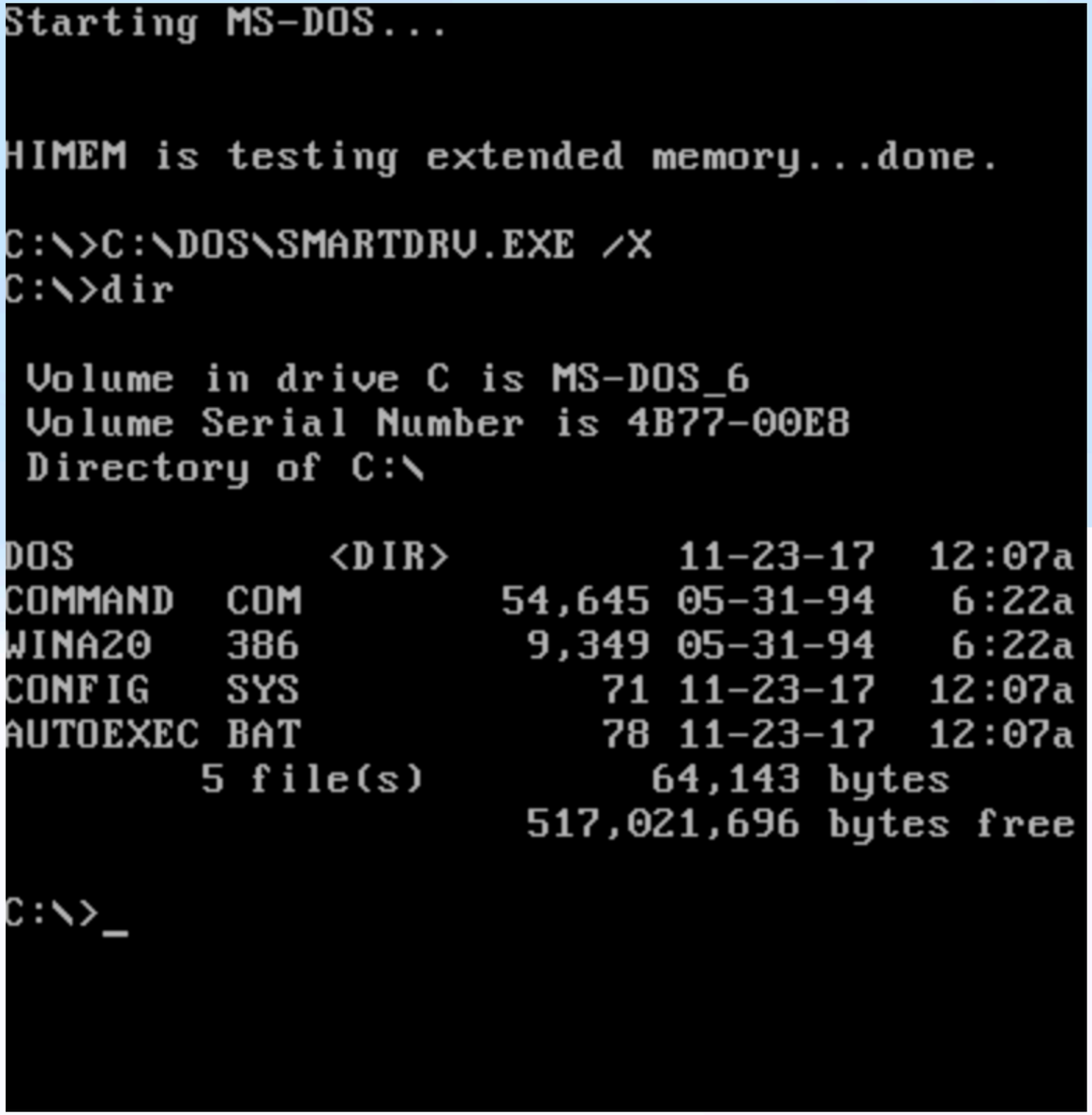
Thank you Gavin!

Let’s start with bit of context

I don’t remember life without computers

But I do remember MS-DOS

And having to type command to do “things”

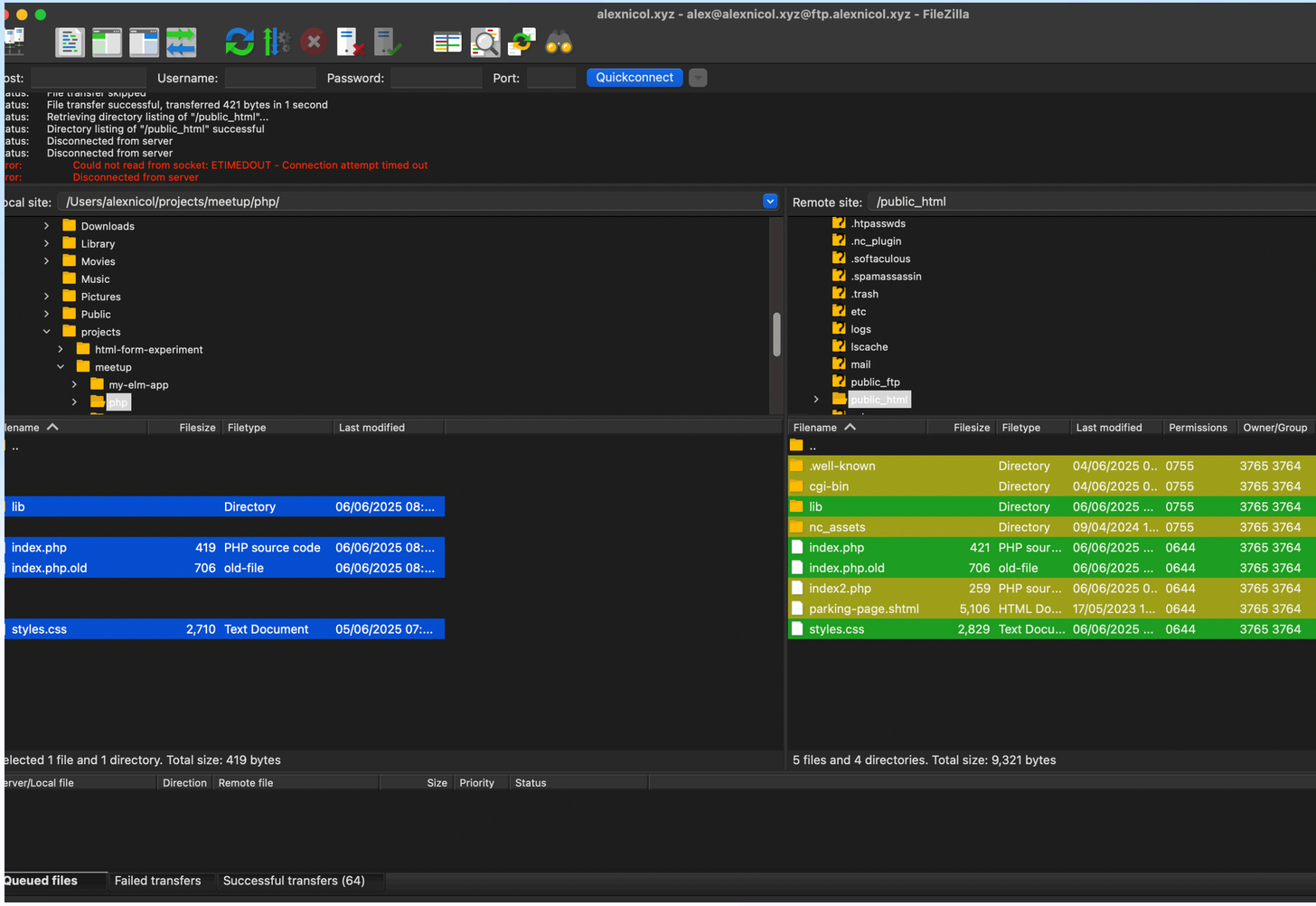


I was first exposed to programming when I was about 10 years old

In my teenager years, I slowly grew my web development skills

My tools of choice were

- HTML, CSS, PHP, SQL
- Notepad++
- Dreamweaver
- FileZilla
- phpMyAdmin



```
    <title>alexnicol.xyz</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    <link rel='stylesheet' type='text/css' media='screen' href='styles.css'>
</head>

<body>
    <div class="u-container">

        <?php include 'lib/header.php'; ?>

        <?php

        $user = 'alexwmfr_user';
        $pass = 'password';

        function connectToDatabase()
        {
            global $user, $pass;
            return new PDO('mysql:host=127.0.0.1;dbname=alexwmfr_db', $user, $pass);
        }


        $dbh = connectToDatabase();

        $sql = "SELECT * FROM `titles`;";
        $stmt = $dbh->prepare($sql);
        $stmt->execute();

        $results = $stmt->fetchAll(PDO::FETCH_ASSOC);
        foreach ($results as $row) {
            echo $row['title'] . '<br>';
        }
        $dbh = null;
```

Yet, I hadn’t been told formally how to code

I knew that’s what I wanted to do

Fast forward to 2009

I’m 18, baccalaureat in my pocket; and I’m about to start a master at an engineering school

What’s programming in 2009?

- Windows 7 was just released
- Angular / React / Vue did not exist yet
- HTML 5 was just released, CSS Flexbox too
- PHP 5.3 - no return or variable types
- Laravel did not exist yet
- JavaScript did not have Class or arrow functions
- Node.js was just released
- iPhone 3GS (0.6GHz, 1 core, 256MB RAM)

2 first years of the master: not much, a little bit of Python

Year 3: Object Oriented Programming

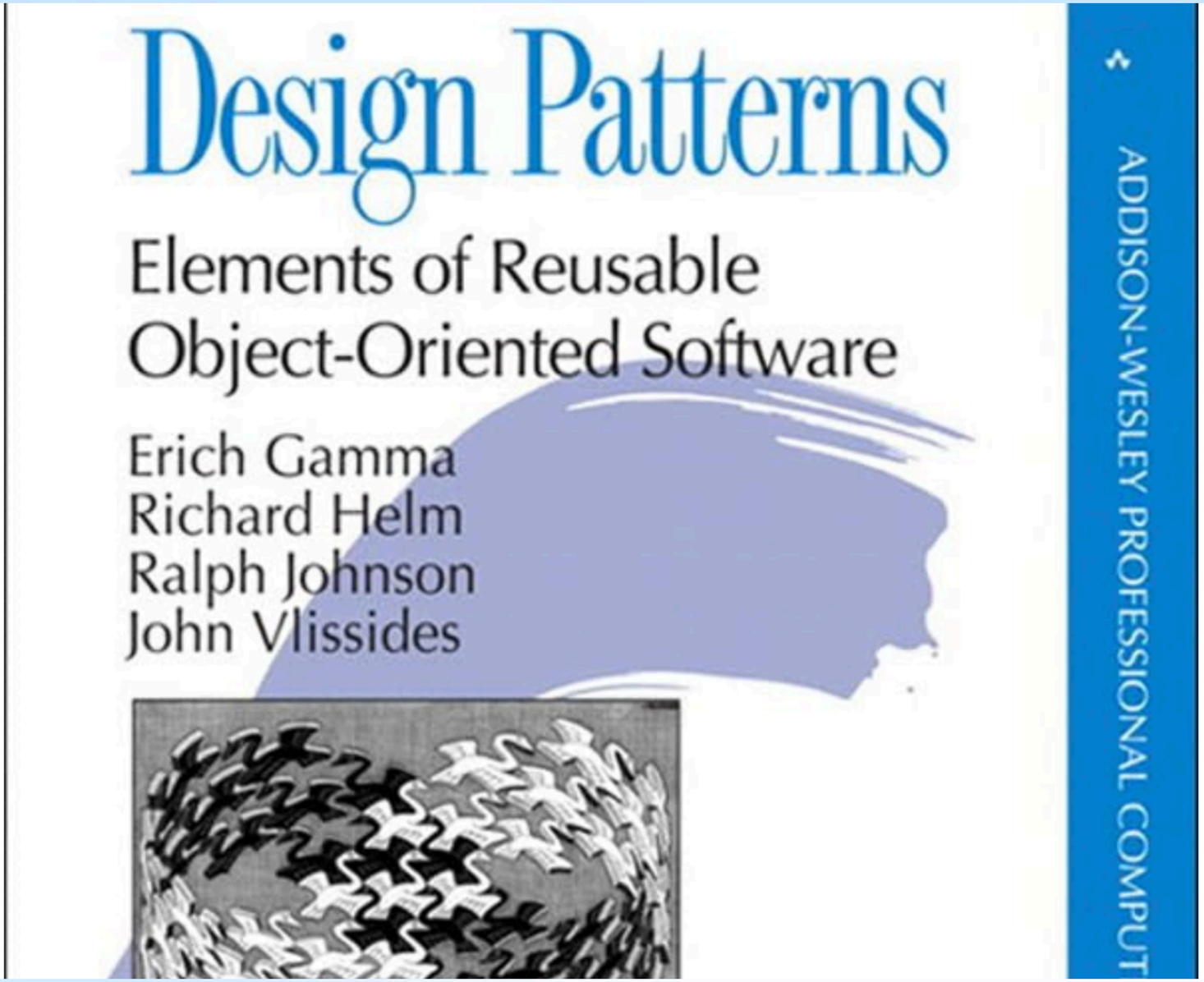

OBJECT ORIENTED PROGRAMMING
imgflip.com

Year 3: Object Oriented Programming, UML, Meta-programming, Design Patterns, RDBS, ORM

I could see how to apply OOP

- Web application & databases
- Video games / 3D / Virtual Reality
- Artificial Intelligence & Multi agent simulation

OOP opens a world of endless possibilities to me; this is the start of nearly 10 years of making classes for everything in all the languages I know.




```python
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []    # creates

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
```

```javascript
class SkinnedMesh extends THREE.Mesh {
  constructor(geometry, materials) {
    super(geometry, materials);

    this.idMatrix = SkinnedMesh.defaultMatrix();
    this.bones = [];
    this.boneMatrices = [];
    //...
  }
  update(camera) {
    //...
    super.update();
  }
  static defaultMatrix() {
    return new THREE.Matrix4();
```

But, we’re not here to talk about OOP.

Fast forward to 2021: BridgeU & Elm

Disclaimer: I won’t talk about Functors, ADTs, Monads

because I cannot explain them

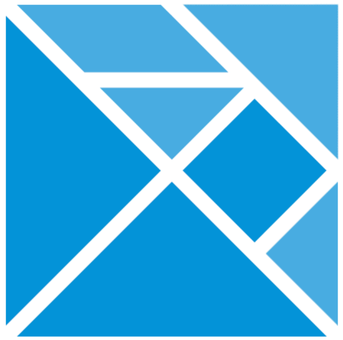
because you don’t need to know anything about them to enjoy FP

What is Elm?

elm-lang.org

elm

Examples Docs Community News



A delightful language for reliable web applications.

Playground Guide

or download the installer.

“Everything in core fits together like Lego.”
Atle Wee Førre, Software Engineer, Equinor

What is Elm?

- Purely functional language
- Strictly typed (inference possible)
- Immutability
- Interoperability with JavaScript
- Compiles to JavaScript
- Compiler written in Haskell

Why Elm?

- Pure function + Immutability =
  no side effects
- Strictly typed + compiled language =
  no exceptions + no unhandled runtime
  errors + reliable refactoring
- **Built to make safe programs**
- **Built to make programmers feel safe**
- **Cannot read property X of undefined?**
  -> GONE!

Hand up if you've written code in:
Haskell, Erlang, OCaml, Clojure, Lisp, F#

```
div [ class "jumbotron" ]
    [ h1 [] [ text "Welcome to Dunder Mifflin!" ]
    , p []
        [ text "Dunder Mifflin Inc. (stock symbol "
        , strong [] [ text "DMI" ]
        , text <|
            """
            ) is a micro-cap regional paper and office
            supply distributor with an emphasis on servicing
            small-business clients.
            """
        ]
    ]
```

Let’s play a game, function or not function

Hint: an Elm function can be defined as

```
functionName arguments =
    somecode
```

And can be implemented as

```
functionName someArgument
```

```
div [ class "jumbotron" ]
    [ h1 [] [ text "Welcome to Dunder Mifflin!" ]
    , p []
        [ text "Dunder Mifflin Inc. (stock symbol "
        , strong [] [ text "DMI" ]
        , text <|
            """
            ) is a micro-cap regional paper and office
            supply distributor with an emphasis on servicing
            small-business clients.
            """
        ]
    ]
```

div

```
div : List (Attribute msg) -> List (Html msg) -> Html msg
```

Represents a generic container with no special meaning.

class

```
class : String -> Attribute msg
```

Often used with CSS to style elements with common properties.

Note: You can have as many `class` and `classList` attributes as you want. They all get applied, so if you say `[ class "notice", class "notice-seen" ]` you will get both classes!

text

```
text : String -> Html msg
```

Just put plain text in the DOM. It will escape the string so that it appears exactly as you specify.

```
text "Hello World!"
```

Assume everything is a function

Assume everything is a function

Think about what HTML is:

- a root node, the start of tree
- that can attributes
- that can have children nodes
- nodes can have attributes, children nodes, or contain some text
- It’s essentially a function, that takes a list of attributes and a list of children nodes, just like its definition in Elm

```
functionName: arg1 type -> arg2 type -> return type
functionName arg1 arg2 =
    arg1 + arg2
```

```
add: Float -> Float -> Float
add arg1 arg2 =
    arg1 + arg2
```

Passing a function as a parameter

```
function highOrderFunction(
  fn: (x: number) => number,
  value: number
): number {
  return fn(value);
}
```

```
local | 2 references
highOrderFunction : (number -> number) -> number -> number
highOrderFunction fn a  =
    fn a
```

Partial function / function currying

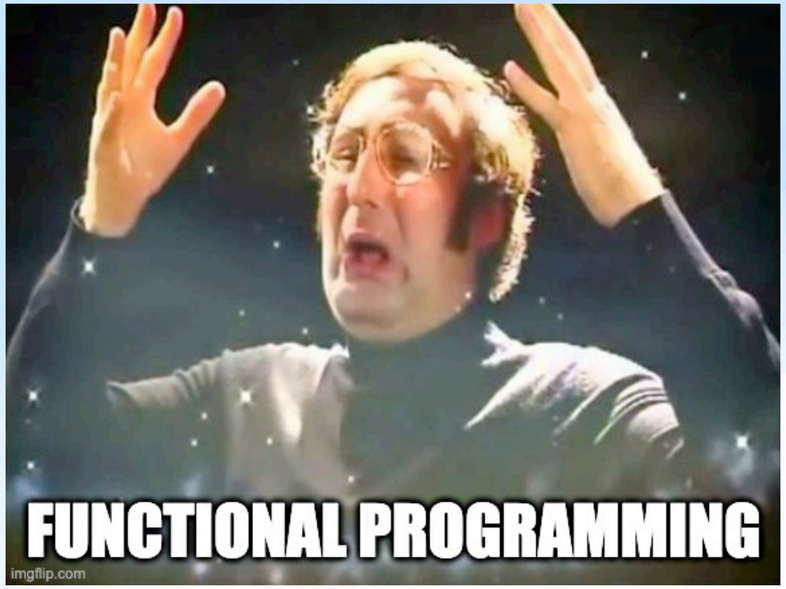
```
local | 3 references
multiply : Int -> Int -> Int
multiply a b =
    a * b


local | 3 references
multiplyByTwo : Int -> Int
multiplyByTwo x =
    multiply 2 x


local | 2 references
multipleTwoByThree : Int
multipleTwoByThree =
    multiplyByTwo 3
```

```
function multiply(x: number, y: number): number {
  return x * y;
}
function multiplyPartial(x: number): (y: number) => number {
  return (y: number) => x * y;
}
function multiplyByTwo(x: number): number {
  return multiplyPartial(2)(x);
}
function multiplyByThree(): number {
  return multiplyByTwo(3);
}
function multiplyByTwo_(): (x: number) => number {
  return multiplyPartial(2);
}
function multiplyByThree_(x: number): number {
  return multiplyByTwo_()(3);
}
```

Elm syntax

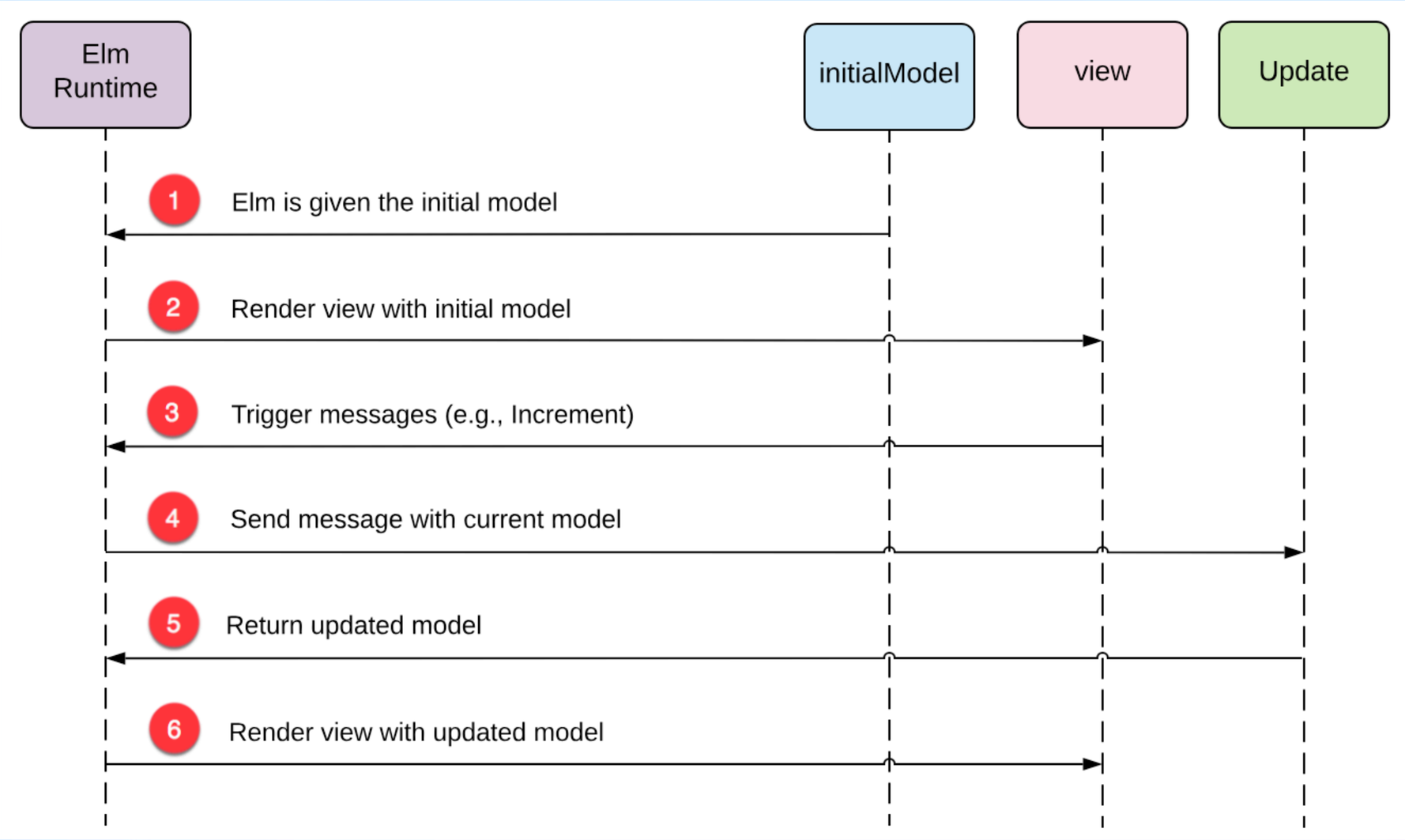
Things that I hadn’t seen before Elm

- Pattern Matching
- Function currying
- Pipe operator
- Maybe/Nothing
- Let/In (a function body can only call one other function unless let/in)


FUNCTIONAL PROGRAMMING
imgflip.com

Elm architecture

Model View Update



Prior Art

Redux has a mixed heritage. It is similar to some patterns and technologies, but is also different from them in important ways. We'll explore some of the similarities and the differences below.

Developer Experience

Dan Abramov (author of Redux) wrote Redux while working on his React Europe talk called "Hot Reloading with Time Travel". His goal was to create a state management library with a minimal API but completely predictable behavior. Redux makes it possible to implement logging, hot reloading, time travel, universal apps, record and replay, without any buy-in from the developer.

Dan talked about some of his intent and approach in Changelog episode 187.

Influences

Redux evolves the ideas of Flux, but avoids its complexity by taking cues from Elm. Even if you haven't used Flux or Elm, Redux only takes a few minutes to get started with.

Live coding: Let’s do what Elm is good at, refactoring!

And hopefully I won’t set my computer on fire

I can’t advise you to use Elm in your day job

State of Elm

And why I decided to ditch Elm at BridgeU (for Svelte)

- No new versions since 2019 (0.19, 0.20 has been in dev for a while)
- Hard to recruit experienced people
- Easy to learn for people with little frontend experience
- Very hard to learn for framework specialists
- Scalability is hard
- Limited support for modern browser APIs

So, 4 years of Elm, what now?

So, 4 years of Elm, now what?

- I’ve left BridgeU, BridgeU has left the Elm chat
- Elm has completely changed my approach to programming
- Write simple code
- Code is meant to be refactored
- I have high exigence from a programming language ecosystem; types, error messages, introspection, IDE integration, compiler
- Elm remains my favourite programming experience

Things are about to get better for native JS

- Pattern Matching
- Pipe Operator
- Immutable notation (maybe?)
- in the meantime there are also a ton of FP libraries for Javascript (fp-ts, Ramda, Immer,…), some are harder to use than others
- Immutable.js + strict TypeScript will do a lot for you

What have I kept from Elm: Simple is better

- Avoid complex types in TypeScript, but do use type definitions
- Avoid side effects, unless you need them by design
- Prefer pure functions and immutability


IMPERATIVE
OOP
FP
SIMPLE CODE
imgflip.com

How ever long you’ve been coding:
- broaden your horizon,
- assume others don’t know,
- assume you don’t know,
- assume you’ll need to change your code,
- get the tools to work for you,
- keep things simple.

Thank you!

hey@alexnicol.dev