Intro to Elm

Josiah Mory / @kickinbahk

What is Elm?

Elm is a programming language

-

It compiles to HTML5: HTML, CSS and JavaScript and used to build web apps.

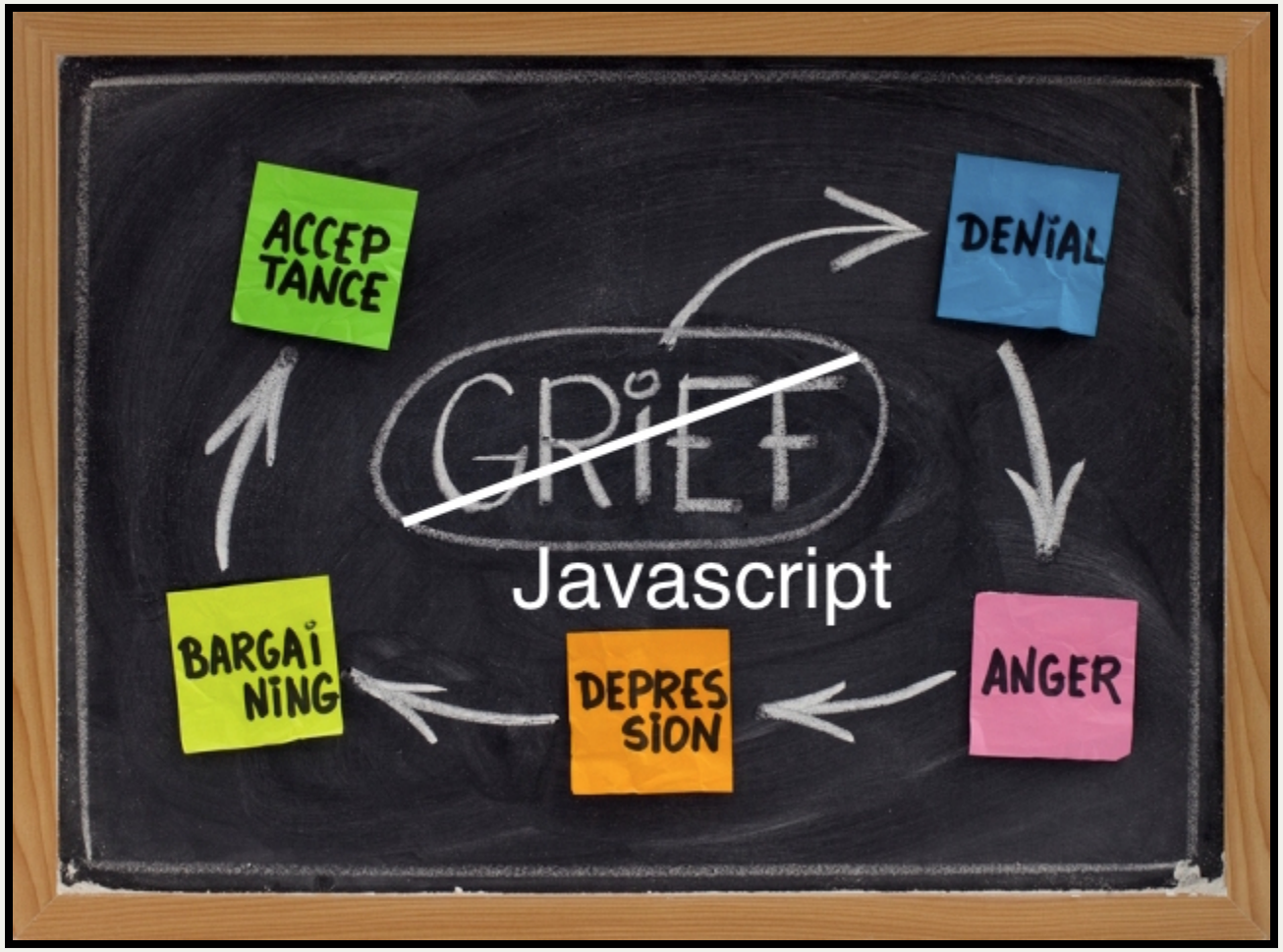
Similar to CoffeeScript or TypeScript,
Elm Compiles to Javascript

Elm doesn’t replicate or try to mend the intricacies of JavaScript.

Elm is a language and ecosystem of its own, and it just happens to compile to JavaScript.

Evan Czaplicki - Creator of Elm

Inspired by the Kubler-Ross model



Syntax

Comments

```
-- a single line comment

{- a multiline comment
 {- can be nested -}
-}
```

Literals

```
-- Boolean
True  : Bool
False : Bool

-- Int or Float depending on usage
42    : number
3.14  : Float

'a'   : Char
"abc" : String

-- multi-line String
"""
This is useful for holding JSON or other
content that has "quotation marks".
"""
```

Lists

```
[1..4]
[1,2,3,4]
1 :: [2,3,4]
1 :: 2 :: 3 :: 4 :: []
```

Conditionals

```
if powerLevel > 9000 then "OVER 9000!!!" else "meh"

-- OR

if key == 40 then
    n + 1

else if key == 38 then
    n - 1

else
    n
```

Functions

```
sayHello name = String.append "Hello " name
-- function : String -> String

sayHello name = \
  String.append "Hello " name
-- function : String -> String
```

Elm Architecture

Model

```
type alias Model = { ... }
```

Update

```
type Action = Reset | ...

update : Action -> Model -> Model
update action model =
  case action of
    Reset -> ...
    ...
```

View

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

Why Elm?

1. Functional Reactive Programming

Functional

- Functions are First Class
- Pure Functions - always return the same value given the same arguments

Pure functions

-

always return the same value given the same arguments, with no side-effects.

In essence, the function must not depend on anything else besides the arguments, and it must not mutate anything.

Reactive

-

something that a component can start listening for, and react to the events as it pleases

In Elm, these listenable things are signals. The component using a signal knows how to utilize it, but the signal has no knowledge of the component(s) that it is affecting.

2. General refactoring experience is:

1. Make changes, any changes.
2. The compiler tells what you missed.
3. Go to 1.

3. Strong static types

Find errors fast with readable compiler messages.

4. No `null` or `undefined`

Never miss a potential problem.

5. Total immutability

Focus on what you are writing right now without worrying about outer state.

6. Purely functional

Leads to decoupled and easily refactorable code.

7. No runtime exceptions

Gives incomparable reliability.

8. Reactive by design

FRP isn’t opt-in, it is baked right in the language.

What I think of Elm...