OCTO Technology

ELM GETTING STARTED

Syntax & Architecture

7 APRIL 2017

FUNCTIONAL PROGRAMMING

In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. […] In functional code, the output value of a function depends only on the arguments that are passed to the function, so calling a function f twice with the same value for an argument x will produce the same result f(x) each time. ***Eliminating side effects, i.e. changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program****, which is one of the key motivations for the development of functional programming.*

From Wikipedia

WHY ELM

- Generate JavaScript with great performance and no runtime exceptions.
  From elm-lang.org
- Good Developer Experience
- Frontend development without JavaScript tooling library
- Has influenced Redux & co
- Wanna feel like a hipster ?

TRIVIA

- Create by 2012 by Evan Czaplicki as his thesis
- Open source
- Used by Prezi and NoRedInk
- 8000 persons on the slack
- There is a Elm meetup in Sydney
- Version 0.18.0

SYNTAX: FUNCTION

Functions

```
square n =
  n^2

hypotenuse a b =
  sqrt (square a + square b)

distance (a,b) (x,y) =
  hypotenuse (a-x) (b-y)
```

Anonymous functions:

```
square =
  \n -> n^2

squares =
  List.map (\n -> n^2) (List.range 1 100)
```

SYNTAX: CONDITION

Conditionals

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

If you need to branch on many different conditions, you just chain this construct together.

```
if key == 40 then
    n + 1

else if key == 38 then
    n - 1

else
    n
```

SYNTAX: UNION TYPE

Union Types

```
type List = Empty | Node Int List
```

SYNTAX: PATTERN MATCHING

```
type MyThing
  = AString String
  | AnInt Int
  | ATuple (String, Int)

unionFn : MyThing -> String
unionFn thing =
  case thing of
    AString s -> "It was a string: " ++ s
    AnInt i -> "It was an int: " ++ toString i
    ATuple (s, i) -> "It was a string and an int: " ++ s ++ " and " ++ toString i
```

Source: gist.github.com/yang-wei/4f563fbf81ff843e8b1e

SYNTAX: RECORD

```
point =                                 -- create a record
  { x = 3, y = 4 }

point.x                                 -- access field

List.map .x [point,{x=0,y=0}]           -- field access function

{ point | x = 6 }                       -- update a field

{ point |                               -- update many fields
    x = point.x + 1,
    y = point.y + 1
}

dist {x,y} =                            -- pattern matching on fields
  sqrt (x^2 + y^2)

type alias Location =                   -- type aliases for records
  { line : Int
  , column : Int
  }
```

SYNTAX: LET EXPRESSION

Let expressions are for assigning variables, kind of like a `var` in JavaScript.

```
let
  twentyFour =
    3 * 8

  sixteen =
    4 ^ 2
in
  twentyFour + sixteen
```

You can define functions and use “destructuring assignment” in let expressions too.

```
let
  ( three, four ) =
    ( 3, 4 )

  hypotenuse a b =
    sqrt (a^2 + b^2)
in
  hypotenuse three four
```

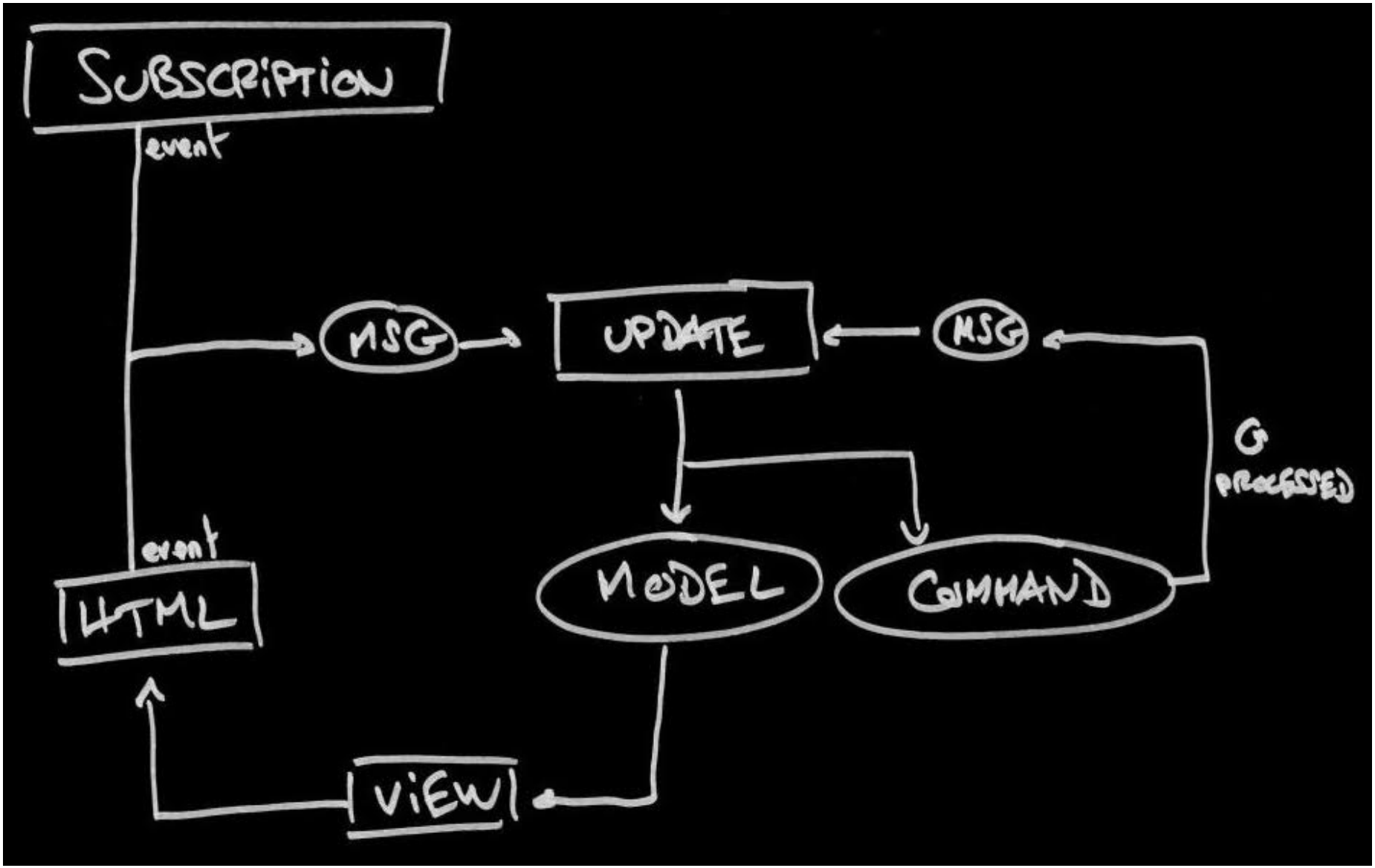
SYNTAX: UNION TYPE

Union Types

```
type List = Empty | Node Int List
```

Source: elm-lang.org/docs/syntax

ARCHITECTURE



LET'S CODE

https://github.com/ThibautGery/elm-todomvc