elm

Introduction to elm

Inspired by presentations from [1] E. Czaplicki, [2] J. Fairbank, [3] K. Yank - see references at the end

Why elm?

*****   Syntax   *****

TEA – The Elm Architecture

Functions

```
greet name = "Hello " ++ name

greet "Thomas"
-- Hello Thomas



add x y = x + y

add 2 3
-- 5
```

Functions are “pure” (stateless, no side effect)

```
add x y = x + y

add 2 3 -- 5
add 2 3 -- 5
add 2 3 -- 5
add 2 3 -- 5
add 2 3 -- 5
...
add 2 3 -- 5
```

JavaScript is dynamically typed – “typed” XD ...

```
var x = 3;
'5' + x - x // 50
'5' - x + x // 5


   16 == [16]  // true
   16 == [1,6] // false
"1,6" == [1,6] // true


[] - [] // 0
[] + [] // ''
```

Elm is statically typed – types are checked at compilation

```elm
life : Int
life = 42

isTrue : Bool
isTrue = True

numbers : List Int
numbers = [ 1, 2, 3 ]
```

Elm is statically typed – types are checked at compilation

```
-- function with 1 parameter
greet : String -> String
greet name =
    "Hello " ++ name


-- function with 2 parameters
add : Int -> Int -> Int
add x y =
    x + y
```

JavaScript is imperative – describe how to get the result

```
function doubleNumbers( numbers ) {
    const doubled = []
    const l = numbers.length

    for ( let i = 0; i < l; i++ ) {
        doubled.push(2 * numbers[i])
    }

    return doubled
}

doubleNumbers( [1, 2, 3] )
// [2, 4, 6]
```

Elm is declarative – describe what the result is

```elm
numbers = [ 1, 2, 3 ]

-- function doubling one number
double n = 2 * n

-- function doubling all numbers of a list
doubleNumbers list =
    List.map double list

doubleNumbers numbers
-- [ 2, 4, 6 ]
```

Computation flow in JavaScript

```
// double -> keep only values < 5 -> square

function process( numbers ) {
    const processed = []
    const l = numbers.length

    for ( let i = 0; i < l; i++ ) {
        const doubled = 2 * numbers[i]
        if (doubled < 5) {
            processed.push( doubled * doubled )
        }
    }

    return processed
}
```

Computation flow in elm – the pipe operator

```
[ 1, 2, 3 ]
    |> List.map double
    |> List.filter lowerThan5
    |> List.map square


double x = 2 * x
lowerThan5 x = x < 5
square x = x * x
```

Computation flow in elm – the pipe operator

```
[ 2, 4, 6 ]
    |> List.filter lowerThan5
    |> List.map square
```

```
lowerThan5 x = x < 5
square x = x * x
```

Computation flow in elm – the pipe operator

```
[ 2, 4 ]
    |> List.map square
```

```
square x = x * x
```

Computation flow in elm – the pipe operator

```
[ 4, 16 ]
```

Tuples

```
dog : ( String, Int )
dog = ( "Tucker", 11 )


name = Tuple.first dog -- "Tucker"
age = Tuple.second dog -- 11
```

Records

```
dog : { name : String, age : Int, breed : String }
dog =
    { name = "Tucker"
    , age = 11
    , breed = "Sheltie"
    }




dog.name  -- "Tucker"
dog.age   -- 11
dog.breed -- "Sheltie"
```

Records – type aliases

```elm
dog : Dog
dog =
    { name = "Tucker"
    , age = 11
    , breed = "Sheltie"
    }


type alias Dog =
    { name : String
    , age : Int
    , breed : String
    }
```

Data is immutable – create state, don’t mutate it

```elm
dog : Dog
dog =
    { name = "Tucker"
    , age = 11
    , breed = "Sheltie"
    }

olderDog = { dog | age = dog.age + 1 }

dog.age      -- 11
olderDog.age -- 12
```

Custom types

```
type alias Dog =
    { name : String
    , age : Int
    , breed : Breed
    }


type Breed
    = Sheltie
    | GoldenRetriever
    | Mix Breed Breed
```

Custom types

```elm
type Breed
    = Sheltie
    | StBernard
    | Mix Breed Breed


dog : Dog
dog =
    { name = "Sally"
    , age = 2
    , breed = Mix Sheltie StBernard
    }
```

Type parameters – polymorphism

```
[ 1, 2, 3 ] : List Int


-- List is defined similarly to that
type List a
    = Empty
    | Cons a (List a)


Cons 1 (Cons 2 (Cons 3 Empty))
```

Maybes – or how to deal with nonexistent values

```
type List a
    = Empty
    | Cons a (List a)

type Maybe a
    = Nothing
    | Just a
```

Maybes – or how to deal with nonexistent values

```
type List a
    = Empty
    | Cons a (List a)

type Maybe a
    = Nothing
    | Just a

head : List a -> Maybe a
head list =
    ...
```

Maybes – or how to deal with nonexistent values

```
type List a
    = Empty
    | Cons a (List a)

type Maybe a
    = Nothing
    | Just a

head : List a -> Maybe a
head list =
    case list of
        Empty -> ...
        Cons someA _ -> ...
```

Pattern matching

Maybes – or how to deal with nonexistent values

```elm
type List a
    = Empty
    | Cons a (List a)

type Maybe a
    = Nothing
    | Just a

head : List a -> Maybe a
head list =
    case list of
        Empty -> Nothing
        Cons someA _ -> Just someA
```

Pattern matching

Functions are total – no forgotten case!

```
printHead : List a -> String
printHead list =
    case head list of
        Just h -> "The head is " ++ (toString h)
```

Functions are total – no forgotten case!

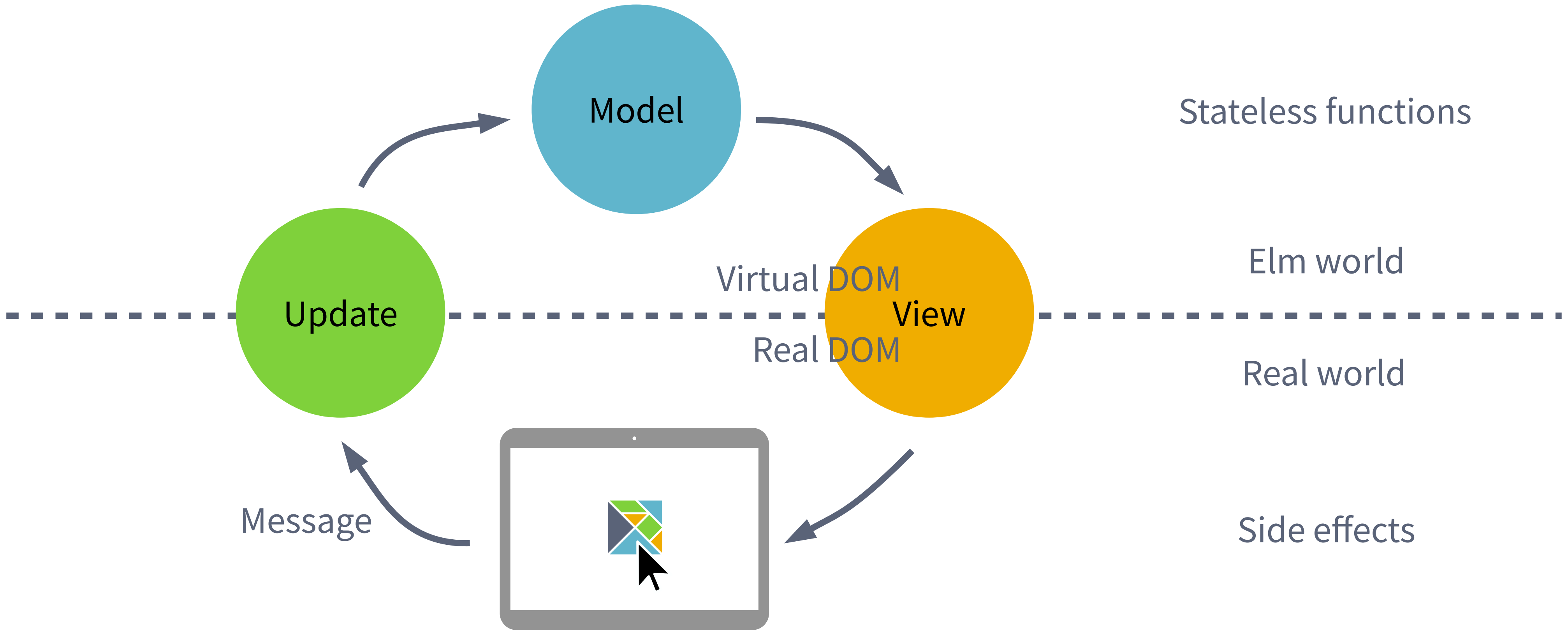
```
-- MISSING PATTERNS ------------------------------------------ elm

This `case` does not have branches for all possibilities:

4|>  case head list of
5|>      Just h -> "The head is " ++ (toString h)

Missing possibilities include:

    Nothing

I would have to crash if I saw one of those. Add branches for them!

Hint: If you want to write the code for each branch later,
use `Debug.todo` as a placeholder.
Read <https://elm-lang.org/0.19.0/missing-patterns>
for more guidance on this workflow.
```

Why elm?

Syntax

*****   TEA – The Elm Architecture   *****

Model – View – Update – unidirectional architecture



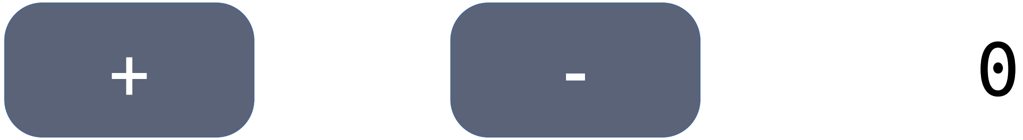
Stateless functions

Elm world

Real world

Side effects

Model – View – Update – let’s make a simple counter



Increment Decrement Model = Int

Questions ?

References

1. Evan Czaplicki, Curry On 2015, Prague, Let’s be Mainstream!,
   http://www.elmbark.com/2016/03/16/mainstream-elm-user-focused-design

2. Jeremy Fairbank, Codemash 2017, Toward a Better Front-end Architecture: Elm,
   https://speakerdeck.com/jfairbank/codemash-2017-toward-a-better-front-end-architecture-elm

3. Kevin Yank, 2017, Elm in Production: Surprises & Pain Points,
   https://www.youtube.com/watch?v=LZj_1qVURL0