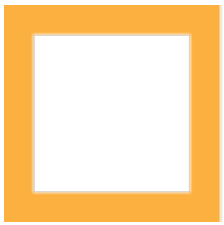
Elm Architecture
Functional Programming

Jens Egholm Pedersen and Anders Kalhauge



cphbusiness

Spring 2017

Outline

Elm Language
- Selections
- Iterations
- Sequences

Architecture
- The Monad
- The Model
- The View
- And Updates

Classic language building blocks

cphbusiness
COPENHAGEN BUSINESS ACADEMY

In LISP we still had the classic programming constructs. But Elm is pure functional, so what then?

Sequence
?

Selection
?

Iteration
?

Classic language building blocks

cphbusiness
COPENHAGEN BUSINESS ACADEMY

In LISP we still had the classic programming constructs. But Elm is pure functional, so what then?

Sequence
: ?

Selection
: **Easy**: Expressions instead of statements

Iteration
: ?

Classic language building blocks

cphbusiness
COPENHAGEN BUSINESS ACADEMY

In LISP we still had the classic programming constructs. But Elm is pure functional, so what then?

Sequence
: ?

Selection
: **Easy**: Expressions instead of statements

Iteration
: **Medium**: Recursion!

Classic language building blocks

cphbusiness
COPENHAGEN BUSINESS ACADEMY

In LISP we still had the classic programming constructs. But Elm is pure functional, so what then?

Sequence

?

Selection

Easy: Expressions instead of statements

Iteration

Medium: Recursion!

Classic language building blocks

cphbusiness
COPENHAGEN BUSINESS ACADEMY

In LISP we still had the classic programming constructs. But Elm is pure functional, so what then?

Sequence

Hard: Monads

Selection

Easy: Expressions instead of statements

Iteration

Medium: Recursion!

Destructuring - Pattern matching

cphbusiness
COPENHAGEN BUSINESS ACADEMY

- In functions calls
- In `case-of` constructs
- In `let-in` constructs

Destructuring - Pattern matching

cphbusiness
COPENHAGEN BUSINESS ACADEMY

- In functions calls
- In `case-of` constructs
- In `let-in` constructs

```
p = (3.5, 4.2)
l = [7, 9, 13],
n = { name = "Kurt", age = 34 }:
```

Destructuring - Pattern matching

- In functions calls
- In `case-of` constructs
- In `let-in` constructs

```
manhattan point =
  let
    (x, y) = point
  in
    x + y
```

```
manhattan p -- 7.7
```

```
p = (3.5, 4.2)
l = [7, 9, 13],
n = { name = "Kurt", age = 34 }:
```

Destructuring - Pattern matching

cphbusiness
COPENHAGEN BUSINESS ACADEMY

- In functions calls
- In `case-of` constructs
- In `let-in` constructs

```
p = (3.5, 4.2)
l = [7, 9, 13],
n = { name = "Kurt", age = 34 }:
```

```
manhattan point =
  let
    (x, y) = point
  in
    x + y
```

```
describe {name, age} =
  name ++ " is " ++ (toString age)
```

```
manhattan p -- 7.7
```

```
describe n -- "Kurt is 34"
```

Destructuring - Pattern matching

cphbusiness
COPENHAGEN BUSINESS ACADEMY

- In functions calls
- In `case-of` constructs
- In `let-in` constructs

```
p = (3.5, 4.2)
l = [7, 9, 13],
n = { name = "Kurt", age = 34 }:
```

```
manhattan point =
  let
    (x, y) = point
  in
    x + y
```

```
describe {name, age} =
  name ++ " is " ++ (toString age)
```

```
describe n -- "Kurt is 34"
```

```
manhattan p -- 7.7
```

```
sum list =
  case list of
    []          -> 0
    head :: tail -> head + (sum tail)
```

```
sum l -- 29
```

Selection

You can use `if-then-else` and `case-of` constructs in Elm:

```
fact n =
  if n == 0 then 1
  else n*(fact (n - 1))

case x of
  Just a  -> a
  Nothing -> 0
```

Create an Elm function that

Calculates the third product $5 * 6 = 30$ of a list of points, if the list is empty the result should be 0, if the list has less than three elements the result should be 1:

```
points = [(1, 2), (3, 4), (5, 6), (7, 8)]
```

Iteration

Where did the `while` and `for` loops go?

Iteration

Where did the `while` and `for` loops go?

There aren't any, you have to use recursion!

Iteration

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Where did the `while` and `for` loops go?

There aren't any, you have to use recursion! But you get help from the List core module:

- Create lists with:

```
List.repeat 3 (0,0) == [(0,0),(0,0),(0,0)]
List.range 3 6 == [3, 4, 5, 6]
1 :: [2,3] == [1,2,3]
1 :: [] == [1]
List.append [1,1,2] [3,5,8] == [1,1,2,3,5,8]
['a','b'] ++ ['c'] == ['a','b','c']
```

- Map and fold lists:

```
List.map sqrt [1,4,9] == [1,2,3]
List.map2 (+) [1,2,3] [1,2,3,4] == [2,4,6]
List.sum [1,2,3,4] == 10
List.product [1,2,3,4] == 24
```

Exercise 2 - List based factorial

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Create a function `factorial` that calculates $n!$ for $n > 0$ using the `List` module, especially `range` and `product` are interesting.

Exercise 3 - Iterations in Elm

Create a function, that calculates:

$$4 \cdot \sum_{n=1}^{100} (-1)^{n+1} \cdot \frac{1}{2n-1}$$

Create a function, that calculates:

$$4 \cdot \sum_{n=1}^{100} (-1)^{n+1} \cdot \frac{1}{2n-1}$$

Hint, that is the same as:

$$4 \cdot \left(\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{1}{197} - \frac{1}{199} \right)$$

In Elm, monads are hidden in the architecture, but we will return to monads in **Haskell**.

But surprisingly, we don't need much sequential processing creating web pages!

cphbusiness
COPENHAGEN BUSINESS ACADEMY

The Monad - the program

```
import Html exposing (..)
import Html.Attributes exposing (..)
import Html.Events exposing (onInput)

main =
  Html.beginnerProgram
    { model = model
    , view = view
    , update = update
    }
```

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Model

```
type alias Model =
  { name : String
  , password : String
  , passwordAgain : String
  }

model : Model
model =
  Model "" "" ""
```

cphbusiness
COPENHAGEN BUSINESS ACADEMY

View

```
view : Model -> Html Msg
view model =
  div []
    [ input
      [ type_ "text"
      , placeholder "Name"
      , onInput Name ] []
    , input
      [ type_ "password"
      , placeholder "Password"
      , onInput Password ] []
    , input
      [ type_ "password"
      , placeholder "Re-enter␣Password"
      , onInput PasswordAgain ] []
    , viewValidation model
    ]
```

cphbusiness
COPENHAGEN BUSINESS ACADEMY

View

```
viewValidation : Model -> Html msg
viewValidation model =
  let
    (color, message) =
      if model.password == model.passwordAgain then
        ("green", "OK")
      else
        ("red", "Passwords␣do␣not␣match!")
  in
    div [ style [("color", color)] ] [ text message ]
```

Update

```
type Msg
    = Name String
    | Password String
    | PasswordAgain String

update : Msg -> Model -> Model
update msg model =
  case msg of
    Name name ->
      { model | name = name }
    Password password ->
      { model | password = password }
    PasswordAgain password ->
      { model | passwordAgain = password }
```

Exercise 5 - Hello anybody

Create a hello world web site with one input field and a text field that shows `"Hello "` and the content of the input field when a button is pushed.

```
$ mkdir hello
$ cd hello
```

Copy the following into `Main.elm`

```
import Html exposing (text)

main =
  text "Hello, World!"
```

And:

```
$ elm-package install -y
$ elm-reactor
```