

Elm Workshop



Day 1

Ce facem azi?

1. Setup
2. Intro
3. Prânz
4. Probleme
5. Arhitectura

1. Setup

General Info

Wi-Fi

SSID: DevNest-Guest

Pass:

Elm Compiler

<https://guide.elm-lang.org/install/elm.html>

Editor / IDE

Whatever you like, we will use Visual Studio Code

New Elm Project

Terminal Commands


```
> mkdir elm-workshop
```

```
> cd elm-workshop
```

```
> elm init
```

Folder Structure

 elm-workshop

 elm.json

 src

Elm Scratch Module

```
module Scratch exposing (..)

greeting : String
greeting = "Hello world"
```

Folder Structure

- elm-workshop
 - elm.json
 - src
 - Scratch.elm

Elm Scratch Module

```
module Scratch exposing (..)

greeting : String
greeting = "Hello world"
```

Terminal

```
> elm repl
> import Scratch exposing (..)
> greeting
"Hello world" : String
```

2. Intro

History



Evan Czaplicki

“Elm: Concurrent FRP for functional GUIs”

2012

2012

- elm 0.1

2019

- elm 0.19.1

Roadmap

<https://github.com/elm/compiler/blob/master/roadmap.md>

How it works

Elm code is compiled to JavaScript code

Elm is really different from JS



Constants / Definitions

```
name : Type  
name = value
```

Examples

```
greeting : String  
greeting = "servus"
```

```
theAnswer : Int  
theAnswer = 42
```

```
someBoolean : Bool  
someBoolean = True
```

Operations

Addition (+)
Difference (-)
Multiplication (*)
Exponentiation (^)
Float Division (/)
Integer Division (//)

Concatenation (++)

Logical And (&&)
Logical Or (||)
Logical Negation (not)

Equality (==)
Inequality (/=)

Examples

```
greeting : String  
greeting = "hello" ++ " world"
```

```
ans : Int  
ans = (2 * 2) ^ (2 + 2)
```

```
toBe : Bool  
toBe = True
```

```
theQuestion : Bool  
theQuestion = toBe || not toBe
```

Single Input Functions

function : Input -> Output

function input = value

input : Input

input = value

output : Output

output = function input

Examples

add1 : Int -> Int

add1 n = n + 1

square : Int -> Int

square x = x ^ 2

ex1 : Int

ex1 = add1 5 -- equals to 6

ex2 : Int

ex2 = square 8 -- equals to 64

Multi Input Functions

```
function : Input1 -> Input2 -> Output
```

```
function in1 in2 = value
```

```
output : Output
```

```
output = function input1 input2
```

```
partial: Input2 -> Output
```

```
partial = function input1
```

```
output2 : Output
```

```
output2 = partial input2
```

Examples

```
add : Int -> Int -> Int
```

```
add x y = x + y
```

```
number : Int
```

```
number = add 5 10 -- equals 15
```

```
bump : Int -> Int
```

```
bump = add 1
```

```
bumpSeven : Int
```

```
bumpSeven = bump 7 -- equals 8
```

Branching

```
if condition
then result-1
else result-2
```

Examples

```
title : String
title =
    if isMarried then "mrs" else "miss"
```

```
isEven : Int -> Bool
isEven n =
    if modBy 2 n == 0
    then True
    else False
```

```
-- In reality this would be written:
-- isEven n = modBy 2 n
```

List

```
list : List Type  
list = [value1, value2, value3]
```

```
emptyList : List Type  
emptyList = []
```

```
numbers : List Int  
numbers = [1, 2, 3, 4, 5]
```

```
truths : List Bool  
truths = [True, True, True, False]
```

```
lists : List (List String)  
lists =  
  [ ["O", "A", "B", "AB"]  
    , ["True", "False"]  
    , ["🐵", "🐵", "🐵"]  
    , []  
  ]
```


List Functions

```
value : Value
```

```
value = List.function list
```

```
list : List Int
```

```
list = [3,2,1]
```

```
sorted : List Int
```

```
sorted = List.sort list
```

```
add1s : List Int
```

```
add1s = List.map add1 list
```

```
odds : List Int
```

```
odds = List.filter isOdd list
```

```
mySum : Int
```

```
mySum = List.sum list
```

Maybe

```
something : Maybe Int  
something = Just 42
```

```
missing : Maybe String  
missing = Nothing
```

```
firstElem : Maybe Int  
firstElem = List.head []
```

```
numberM : Maybe Int  
numberM = String.toInt "13"
```

```
bumpNumber : Maybe Int  
bumpNumber = Maybe.map add1 numberM
```

```
Number : Int  
Number = Maybe.withDefault 0 numberM
```

Tuple

```
group: (Type1, Type2)  
group = (value1, value2)
```

```
example : (Int, String)  
example = (1, "one")
```

```
index : Int  
index = Tuple.first example
```

```
text : String  
text = Tuple.second example
```

3. Prânz

4. Probleme

Problem 1

```
isLeapYear : Int -> Bool  
isLeapYear year = ???
```

A leap year is divisible by 4, but not divisible by 100, unless divisible by 400

Problem 1

```
isLeapYear : Int -> Bool
isLeapYear year =
  divisible year 4
  && (
    not (divisible year 400)
    || divisible year 100
  )

divisible : Int -> Int -> Bool
divisible a b = modBy b a == 0
```

A leap year is divisible by 4, but not divisible by 100, unless divisible by 400

Problem 2

daysBetween : Int -> Int -> Int

daysBetween startYear endYear = ???

Calculate number of days between two years

Problem 2

```
daysBetween : Int -> Int -> Int
daysBetween startYear endYear =
  List.range startYear endYear
    |> List.map yearToDays
    |> List.sum
```

```
yearToDays : Int -> Int
yearToDays y =
  if isLeapYear y
  then 366
  else 365
```

Calculate number of days between two years

Problem 3

```
cnpStuff : String -> Bool  
cnpStuff cnp = ???
```

Calcularea ultimei cifre se face folosind constanta "279146358279", după cum urmează:

- fiecare cifră din primele 12 cifre ale C.N.P. este înmulțită cu corespondentul său din constantă
- rezultate sunt însumate și totalul se împarte la 11
- dacă restul împărțirii este mai mic de 10, acela reprezintă valoarea componentei **C**
- dacă restul împărțirii este 10, valoarea componentei **C** este 1

[https://ro.wikipedia.org/wiki/Cod_numeric_personal_\(Rom%C3%A2nia\)](https://ro.wikipedia.org/wiki/Cod_numeric_personal_(Rom%C3%A2nia))

Problem 3

```
cnpStuff : String -> Bool
cnpStuff cnp =
    String.length cnp == 13
    && lastDigit cnp == Just (control cnp)

lastDigit : String -> Maybe Int
lastDigit cnp =
    cnp
    |> String.toList
    |> List.reverse
    |> List.head
    |> Maybe.andThen (String.toInt << String.fromChar)

control : String -> Int
control cnp =
    cnp
    |> String.toList
    |> List.map (Maybe.withDefault 0 << String.toInt << String.fromChar)
    |> List.map2 (*) [2,7,9,1,4,6,3,5,8,2,7,9]
    |> List.sum
    |> modBy 11
    |> modify

modify : Int -> Int
modify x = modBy 10 x + x // 10
```

Calcularea ultimei cifre se face folosind constanta "279146358279", după cum urmează:

- fiecare cifră din primele 12 cifre ale C.N.P. este înmulțită cu corespondentul său din constantă
- rezultate sunt însumate și totalul se împarte la 11
- dacă restul împărțirii este mai mic de 10, acela reprezintă valoarea componentei **C**
- dacă restul împărțirii este 10, valoarea componentei **C** este 1

[https://ro.wikipedia.org/wiki/Cod_numeric_personal_\(Rom%C3%A2nia\)](https://ro.wikipedia.org/wiki/Cod_numeric_personal_(Rom%C3%A2nia))

5. Arhitectura

Thank you for attending!

Useful links

Elm packages

<https://package.elm-lang.org/>

Elm Search by Type

<https://klaftertief.github.io/elm-search/>

The Official Elm Guide

<https://guide.elm-lang.org/>