Functions

Mitsiu Alejandro Carreño Sarabia

Agenda

- Recap
- Functions
- First project
- If
- Case
- Homework

Recap

Course changes

- 1st Partial = 30%
  - Class exercise / Homework = 10%
  - Theorical evaluation - Functional paradigm = 20%
  - Theorical evaluation - HTML = 20%
  - Practical evaluation - Functional paradigm (paper based, NO computer!) = 25%
  - Practical evaluation - HTML (paper based, NO computer!) = 25%
- 2nd Partial = 30%
  - Class exercise / Homework = 10%
  - Theorical evaluation = 40%
  - Practical evaluation (maybe in computer) = 50%
- 3rd Partial = 40%
  - Theorical evaluation - Client-Server = 30%
  - Theorical evaluation - side effects = 30%
  - Practical evaluation (in computer) = 40%

Recap

- What is an expression?
- What is a value?
- What does this simbol means?

$$\Rightarrow$$

- Values are expressions?
- Expressions are values?
- What is imperative programming?
- What is state in computer science?
- Which primitive data types exist in elm?
- Exemplify 3 non-Int values
- Which conditions does the (+) operator has?

Functions

Variables

In elm we can create variables just by giving a name and binding a value:

```
myName = "Mitsiu"
```

Elm response is: Mitsiu : String

But let's look what happens if we try with a numerical value

Elm is unable to determine "age" type with certainty, it can be:

- A float that happens to have no decimal
- An integer

▦ Data types & operators

Let's improve our communication hability remember that to say that an expression e has type t we write:

$$e : t$$

```
1 -- This is an inline comment, below is a type annotation
2 age : Int
3 age = 32
```
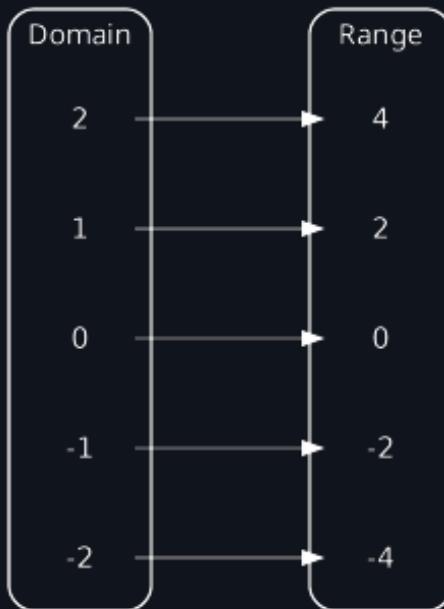
Line 1 is a type annotation it help's auto-document our code and clears all ambiguity about our true intent.

Functions

In math we find:

$$f(x) = 2 * x$$

- f is the name of the function
- f(x) the function f has an input x
- = 2 * x describe what the function does

Domain		Range
2	→	4
1	→	2
0	→	0
-1	→	-2
-2	→	-4

Let's dig deeper into our function

$$f(x) = 2 * x$$

- f is the name of the function
- f(x) the function f has an input x
- = 2 * x describe what the function does

The typing rule for * is:

$$e1 * e2 : number$$

$$if$$

$$e1 : number$$

$$and$$

$$e2 : number$$

What can be inferred about the input and output of our function f?

$$f(x) = 2 * x$$

Let's start to code our first function in elm First our type annotation

```
1 -- Functions also have type annotations
2 f : number -> number
```

The function parameter has type number and the function returns a number

```
1 f : number -> number
2 f x =
3     -- This is the function body
4     2 * x
```

Function inputs and outputs

In the functional paradigm all functions must receive an input and return an expression

```
1 -- The function f receives a number and produces a number
2 f : number -> number
3 f x =
4     2 * x
```

Our type annotation describe:

- The function is called "f"
- Functions are values and they have a type ":"
- List of input types "number" separated by "->"
- The final type is the function output type "number"

🟧 Function application

To use the functions we just defined, we have to apply it.

1. We specify the function to apply by it's name
2. We have to provide a value for the input "x"

```
1  -- Function definition
2  f : number -> number
3  f x =
4      2 * x
5
6  -- Function application
7  f 4
```

So we would have that

$$f4 \Rightarrow 8$$

First project

First project

Let's create our first elm project.

1. Create a folder to store all our exercies and homeworks (for example C:\Documents\web)
2. In the previous folder create a new folder for todays exercise (Ex1-functions)
3. Open a terminal in that folder

```
pwd
cd C:\Documents\web\Ex1-functions
```

4. Once you are in the correct folder run the command:

```
elm init
# Accept the following questions
```

Elm init

`elm init` is the command to bootstrap an elm project, it creates:

1. The file "elm.json":

    - elm.json is the file describing all libraries required for our project to run.

2. The folder "src"

    - The folder to write our code

Coding

Let's create a file "Helper.elm" in the "src" folder.

```
-- File: Ex1-functions/src/Helper
module Helper exposing (..)

double : number -> number
double x =
    x * 2
```

Let's make sure our terminal is at "Ex1-functions" (pwd)

```
pwd
elm repl
```

REPL stands for Read Eval Print Loop

```
import Helper

Helper.double

Helper.double 4

:exit
```

More functions

1. A new function "square" that takes a number and square's it (x^2)

2. A new function called "Greet" so that it receives a String name and return "Hello " + name

3. A new function "above5 : number -> Bool" that evaluates if a given number is greater than (>) 5

If

The if expression has one of the following structures:

$$\text{if } e1 : \text{Bool then}$$
$$e2 : \alpha$$
$$\text{else}$$
$$e3 : \alpha$$

$$\text{if } e1 : \text{Bool then}$$
$$e2 : \alpha$$
$$\text{else if } e3 : \text{Bool then}$$
$$e4 : \alpha$$
$$\text{else}$$
$$e5 : \alpha$$

Like all other expressions, an if expression returns a value of type alpha

In Elm we must provide the else branch.

More functions

1. A new function "ifBoolTranslate" that get's a Bool and return a String either "Positivo" or "Negativo"

2. A new function "ifNumberSign" that get's a number and return a string either "Positive", "Negative", "Neutral"

Case

Case expression

Another usefull structure are case expressions:

$$case \ e1 : \alpha \ of$$
$$pat1 : \alpha \ ->$$
$$e2 : \beta$$
$$pat2 : \alpha \ ->$$
$$e3 : \beta$$

Notice that all possible values of type alpha must be evaluated consider:

```
caseExample : Int -> Bool
caseExample num =
    case num of
        3 ->
            True


        _ ->
            False
```

1. A new function "getNameById" that get's an Int and return a String

2. A new function "getGradeByName" that get's a String and return a Float

Formatter and clean code

Formatter and clean code

Let's standarize how we write code:

```
# Check if our code complies the standarized elm rules:
elm-format src/ --validate
# Apply the format rules and modify our files:
elm-format src/
```

Check we provide all annotations:

```
elm-review \
--template jfmengels/elm-review-common/example \
--rules NoMissingTypeAnnotation,NoMissingTypeAnnotationInLetIn
```

Verify our code compiles and is valid:

```
elm make src/*
```

Check unit testing if present:

```
elm-test
```

Homework

Homework

1. Create a github account with your "alumnos.upa.edu.mx" email