Elixir Study Group

Kick-off meet-up

Hosted by

DeuSpace.be

Why are we here tonight?

Things I know

- We are interested in **Elixir**.
- We need to **learn.**
- Learning together is **fun.**
- Elixir is **fun.**
- Our meet-ups well be practical: we'll write **code**.

Things I don’t know

- **How often** we get together.
- Your **background** and your **level**.
- Let’s **get acquainted**!

Thus

- Let's **experiment**!
- **Everyone** is welcome to take initiative and organize the next session, possibly in a different way!

Erlang the platform and Elixir

Erlang the language

Elixir

Erlang the VM

- In use for ~20 years.
- Designed at Ericsson
- Fault tolerant, scalable, distributed, responsive systems.
- Erlang the language is a functional and concurrent programming language with a dynamic type system
- Erlang the language is rather conservative

What makes Erlang unique

- **Lightweight isolated process** as the building block
- Processes communicate via **asynchronous messages**
- Error handling approach: **LET IT CRASH** approach
- OTP

Hello, Mike!

Hello, Joe!

Google: “Erlang the movie”
and then: “Erlang the movie the sequel”

Elixir

- **Modern**
- **Functional**
- **Concurrent**

- **Transparent** integration with the **Erlang** world
- Protocol-based **polymorphism**
- **Macros**
- Focus on **tooling**

Most importantly…

Elixir

possesses the gene of programmer happiness!

Functional Programming for the uninitiated

Two pillars

- **Higher-order functions**
- **Immutability**

Higher-Order Functions

```
words = ["takes", "one", "or", "more",
  "functions",  "as", "an", "input", "or",
    "outputs", "a", "function"]

Enum.max_by words, fn(word) ->
    String.length(word) end  #=> "functions"
```

Consequences of Immutability

- You **can't** modify an *object* **in-place**. Any modification produces a **new** *object*.

Consequences of Immutability

- There is no question of equality and identity, like in Java .

```
Date a = new Date(123);
Date b = new Date(123);
Date c = a;
System.out.println(a == b);              //=> false
System.out.println(a.equals(b));         //=> true
System.out.println(a == c);              //=> true
```

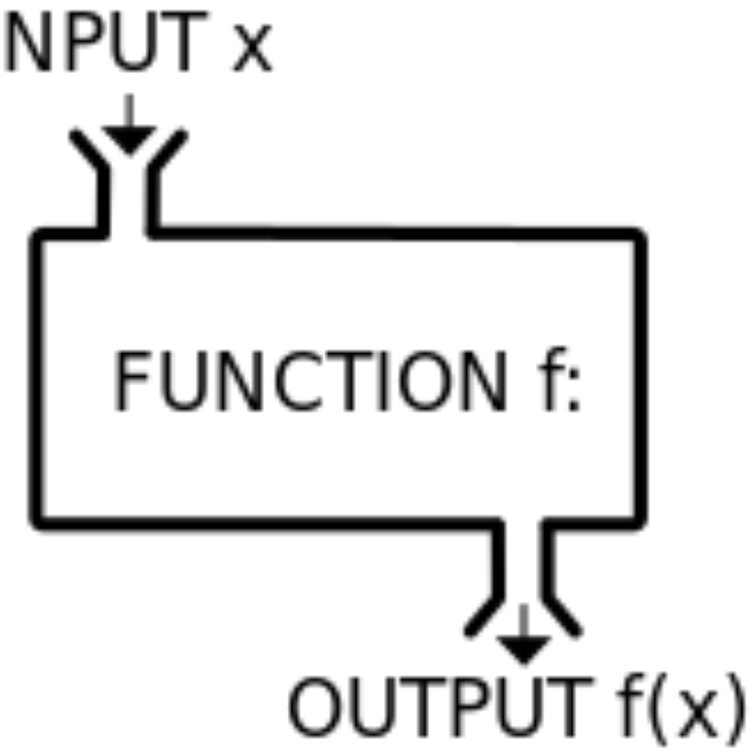
Equality and Identity the Functional Way

```
a = {2014, 9, 11}
b = {2014, 9, 11}
c = a

IO.inspect a == b  #=> true
IO.inspect a == c  #=> true
IO.inspect b == c  #=> true
```

Functions transform data

Pure functions:



Functions transform data

```
defmodule Example do
  def word_signature word do
    without_spaces = String.strip(word)
    downcased = String.downcase(without_spaces)
    letters = String.split(downcased, "", trim: true)
    sorted = Enum.sort(letters)
    Enum.join(sorted)
  end
end

IO.inspect  Example. word_signature(” Higher ”)
```

Functions transform data

```
defmodule Example do
  def word_signature word do
    Enum.join(Enum.sort(String.split(
          String.downcase(
                String.strip(word)), "", trim: true)))
  end
end

IO.inspect  Example. word_signature(” Higher ”)
```

OR:

```
defmodule Example do
  def word_signature word do
    word |> String.strip
         |> String.downcase
         |> String.split("", trim: true)
         |> Enum.sort
         |> Enum.join
  end
end

IO.inspect  Example. word_signature(” Higher ”)
```

FP & OO

- Functional programming does not contradict **Object Orientedness** if objects are **immutable**. Example: Scala.

- But neither **Erlang** nor **Elixir** has classes

- You **separate** code and data

Abstracting with modules

```
peter = Person.new("Peter","Peterson", 20)

IO.inspect  Person.can_drink_alcohol?(peter)
                                        #=> false

peter = Person.birthday(peter)

IO.inspect  Person.can_drink_alcohol?(peter)
                                        #=> true
```

Abstracting with modules

```
defmodule Person do

  defstruct firstname: nil, lastname: nil, age: 0

  @legal_drinking_age 21

  def new(fname, lname, age) do
    %Person{firstname: "Peter",
                      lastname: "Peterson", age: 20}
  end
```

Abstracting with modules

```
def can_drink_alcohol?(person) do
  person.age >= @legal_drinking_age
  end

  def birthday(person) do
    %{person | age: person.age + 1}
  end

end
```

WiFi:
DevSpace-5GHz
DevSpace-2.4GHz
passwd: devspace2012ftw!

twitter:
@elixir_be
@xavierdefrang
@less_software

Elixir docs:

http://elixir-lang.org/docs/stable/elixir/

Exercises:

https://github.com/belgian-elixir-study-group/meetup-materials

```
git clone
https://github.com/belgian-elixir-study-group/meetup-
materials.git
```