elixir

Elixir - Basics

Ruben Amortegui
@ramortegui
http://rubenamortegui.com
https://github.com/ramortegui

Agenda

- About Elixir
- Tools (iex and mix)
- Concepts (Pattern Matching, Immutability)
- Basics (types, operators, functions, lists and recursion, maps, collections, strings, control flow)

About Elixir

- Functional, concurrent, general purpose programming language.
- Build on top of Erlang
- Runs over Erlangs virtual Machine (BEAM)

About Elixir - Features

- Scalabiliy
  - Creates light weight processes (code runs inside processes)
- Fault tolerance
  - Supervisors (detects when a process dies and creates a new one)
- Build on top of Erlang
  - Supports erlang calls
- Runs over Erlang’s virtual Machine (BEAM)
  - Can use erlang libraries: e.g. :math

Basic Tools

- iex (Interactive elixir shell)
  - Commands: pwd() ls(), c, h, i
- mix (helper to build, run, compile, manage dependencies).
  - mix new project: To create a the default structure
  - iex -s mix: To compile and load your project
  - mix deps.get: To get dependencies
  - mix hex: .... To publish your code
  - mix test: To run the test suite

Elixir – Pattern Matching

- It’s a core part of elixir. The operator is **=**

```
iex> a = 3
3
iex> 3 = a
3
iex> 4 = a
** (MatchError) no match of right hand side value: 3
```

  - Tuples and lists

```
{a,b,c} = {"1", "5", "6"}
[head|tail] = [1,2,3,4]
```

  - Pin operator ^
    - Match the value of the variable.
- It’s used in functions too.

Elixir – Immutability

- Immutable data is known data
- Sample

  a = 10

  something(a)

  print a
- Benefits:
  - You don’t need to worry that any code might mutate your data.

Elixir – Basics 1

- Value Types
  - Integer, Float, Atoms, Ranges, Regular Expresions
- System Types
  - PIDs
- Collection Types
  - Tuples {:name, "Ruben", :age, 36}
  - Lists [1,2,3,4]
  - Maps %{ "data" => 123 }
  - Binaries <<91,93>>

Elixir – Basics 2

- Operators
  - Comparison
    - ===, !==, ==, !=, >, >=, <, <=
  - Booelan
    - or , and, not (Receive only booleans) true || false
    - ||, &&, ! (Any value != false or nill is true)
  - Arithmetic
    - +, -, *, /, div, rem
  - Join
    - binary <> binary
    - list1 ++ list2, list1 -- list2
  - In
    - a in enum (check if a exists on enum array)

Elixir – Basics 3

- Pipe operator |>
  - It’s used to pass the result of an expression as the first parameter of another expression

    e.g:

    Count the number of words in a String

    iex> Enum.count(String.split("This is a String"))

    iex> "This is a String" |> String.split |> Enum.count
  - On a file

    "This is a String"

    |> String.split

    |>Enum.count

Elixir – Anonymous Functions

- What is anonymous funciton

  Are basic types and are created using the fn keyword:

  - fn

    parameter-list -> body

    parameter-list -> body

    end

- How to define

  sum = fn(x,y) -> x + y end

- How to use

  sum.(1,3) # => 4

Elixir – Anonymous Functions

File: exist.txt

Hello world

```
handle_open = fn
  {:ok, file} -> "Read data: #{IO.read(file, :line)}"
  {:error, error} -> "Error: #{:file.format_error(error)}"
end

handle_open.(File.open("exists.ex"))
"Read data: \"hello world\"\n"


iex> handle_open.(File.open("not_exists.ex"))
"Error: no such file or directory"
```

Elixir – Modules and Named Functions

- Organize code

```
defmodule Name do
    def print_name(name) do
        IO.puts "Hello #{name}"
    end
end
```

- Iex(1)> Name.print_name("Ruben")

```
hello Ruben
:ok
```

Elixir – Modules and Named Functions

- guards

```
defmodule Name do
    def greeting(name,age) when age < 2 do
        IO.puts "Gaga #{name}"
    end
    def greeting(name,age) when age <5  do
        IO.puts "Hi #{name}"
    end
    def greeting(name,age)   do
        IO.puts "Hello #{name}"
    end
end
```

- Iex(1)> Name.print_name("Ruben",1)

```
Gaga Ruben
:ok
```

Elixir – List and Recursion

- How to define a List

  [], [1,2,3]
- Patter matching

  [a,b,c] = [1,2,3]
- Sample
  - Sum numbers of a list
- Tail Recursion
  - Sum numbers of a list

Elixir – Maps, Keyword Lists, and Structs

- What is:
  - Keyword list (list of tuples – key needs to be an atom)

    [name: "Ruben", age: 36]

    kw = [{:name, "Ruben"},{ :age, 36}]
    - Kw[:name]
  - Map
    - map = %{ "name" => "Ruben" }
    - map["name"]
  - Structs (Define map structure)

    defmodule User do

        defstruct [name: "unknown"]

    end
    - %User{ username => "test" }
    - %User{}

Elixir – Processing Collections

- Enum
  - Module to process collections
  - Iex>h Enum
- Comprehensions (a way to iterate over an Enumerable)
  - for var <- Enum, do: code
  - for x <- [1,2,3], do: IO.puts(x)

Elixir – Strings

- "String"
  - A String in Elixir is a UTF-8 encoded binary.
  - Use String module to manipulate strings.

    iex> String.capitalize("ruben")
  - It’s represented as binary list

    iex> i "Ruben"
  - Contatenation with <>

    iex> i ("Ruben" <> " Dario")

Elixir – Control Flow

- If (cond) do body end
- If (cond) do body1 else body2 end
- Case

```
x = 10
case x do
  0 ->   "This clause won't match"
  _ ->   "This clause would match any value (x = #{x})"
end
#=> "This clause would match any value (x = 10)"
```

- Cond

```
cond do
  1 + 1 == 1 ->        "This will never match"
  2 * 2 != 4 ->        "Nor this"
  true ->        "This will"
end
#=> "This will"
```

References

- Jurić, S. (2015). Elixir in action. Shelter Island, NY: Manning Publications.
- Thomas, D. (2016). Programming Elixir 1.3: functional, concurrent, pragmatic, fun. Releigh, NC: Pragmatic Bookshelf.

Thanks!

Q & A?

@ramortegui

http://rubenamortegui.com

https://github.com/ramortegui