elixir

Basics Part 2
Functions

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

Agenda

- Functions in Elixir
  - Anonymous functions
  - Named functions
  - Pattern Matching
  - Guards
- Case EcCart

Eixir – Functions

- Elixir is a functional language, and functions are the engine.
- The purpose of functions is to transform data, not to change data.
- Two “types” of functions:
  - Anonymous functions
  - Named functions

Elixir – Anonymous Functions

- Anonymous functions are delimited by the keywords **fn** and **end**.
  - fn

      parameter-list -> body

      parameter-list -> body

    end
- Defined as:

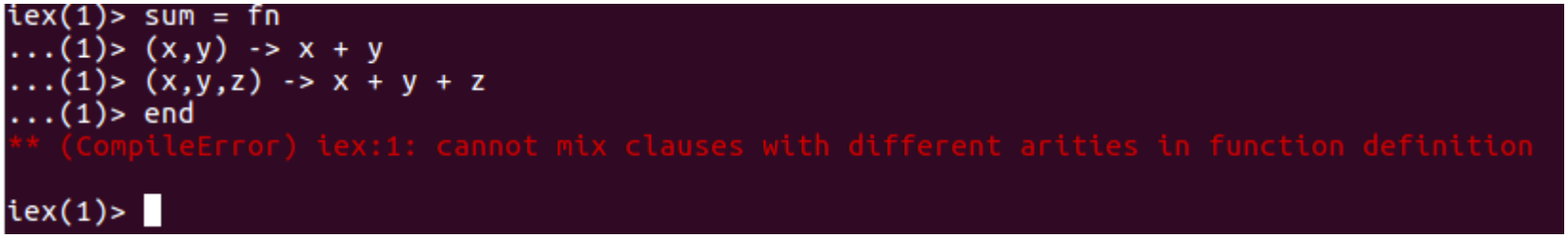
```
iex(1)> sum = fn(x,y) -> x + y end
#Function<12.52032458/2 in :erl_eval.expr/5>
iex(2)>
```

- Used as:

```
iex(2)> sum.(1,3)
4
iex(3)>
```

Elixir – Anonymous Functions

- Anonymous functions needs to have the same **arity** on the parameter list.



Elixir – Anonymous Functions

- Order in defined clauses is important

```
iex(1)> abs = fn
...(1)> (x) when x < 0 -> x * -1
...(1)> (x) -> x
...(1)> end
#Function<6.52032458/1 in :erl_eval.expr/5>
iex(2)> abs.(-10)
10
iex(3)> abs.(10)
10
iex(4)>
```

```
iex(1)> abs = fn
...(1)> (x) -> x
...(1)> (x) when x < 0 -> x * -1
...(1)> end
#Function<6.52032458/1 in :erl_eval.expr/5>
iex(2)> abs.(-10)
-10
iex(3)> abs.(10)
10
iex(4)>
```

Elixir – Anonymous Functions

- Are closures
  - The scope encloses the bindings of its variables, packaging them into something that can be saved and used later.

```
iex(1)> x = 2
2
iex(2)> sum2 = fn(num) -> num + x end
#Function<6.52032458/1 in :erl_eval.expr/5>
iex(3)> sum2.(5)
7
iex(4)> x = 3
3
iex(5)> sum2.(5)
7
iex(6)>
```

Elixir – Anonymous Functions

Sample:

.....
.....
exists.txt
.....
.....
Hello world

```
iex(1)> handle_open = fn
...(1)> {:ok , file} -> "Read: #{IO.read(file,:line)}"
...(1)> {_, error} -> "Error: #{:file.format_error(:error)}"
...(1)> end
#Function<6.52032458/1 in :erl_eval.expr/5>
iex(2)> handle_open.(File.open("exists.txt"))
"Read: Hello world\n"
iex(3)> handle_open.(File.open("no_exists.txt"))
"Error: unknown POSIX error"
iex(4)>
```

Elixir – Anonymous Functions

& notation

It’s a short helper

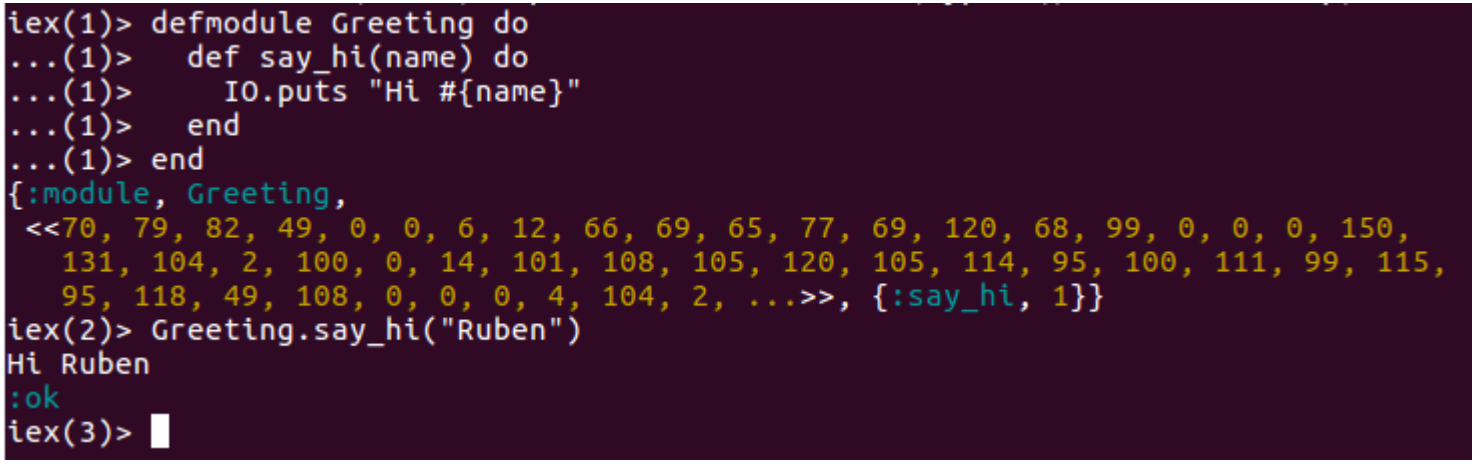
```
iex(1)> pow2 = &(&1*&1)
#Function<6.52032458/1 in :erl_eval.expr/5>
iex(2)> pow2.(2)
4
iex(3)>
```

Is the same as:

```
iex(3)> pow2 = fn(x) -> x*x end
#Function<6.52032458/1 in :erl_eval.expr/5>
iex(4)> pow2.(2)
4
iex(5)>
```

Elixir – Modules and Named Functions

- Named functions are defined inside of a module in order to organize and give structure to your code.



Elixir – Functions

- Pattern matching

```
iex(1)> defmodule Math do
...(1)>   def sum(a,b) do
...(1)>     a + b
...(1)>   end
...(1)>   def sum(a, b, c) do
...(1)>     a + b + c
...(1)>   end
...(1)> end
{:module, Math,
 <<70, 79, 82, 49, 0, 0, 5, 128, 66, 69, 65, 77, 69, 120, 68, 99, 0, 0, 0, 228,
   131, 104, 2, 100, 0, 14, 101, 108, 105, 120, 105, 114, 95, 100, 111, 99, 115,
   95, 118, 49, 108, 0, 0, 0, 4, 104, 2, ...>>, {:sum, 3}}
iex(2)>
```

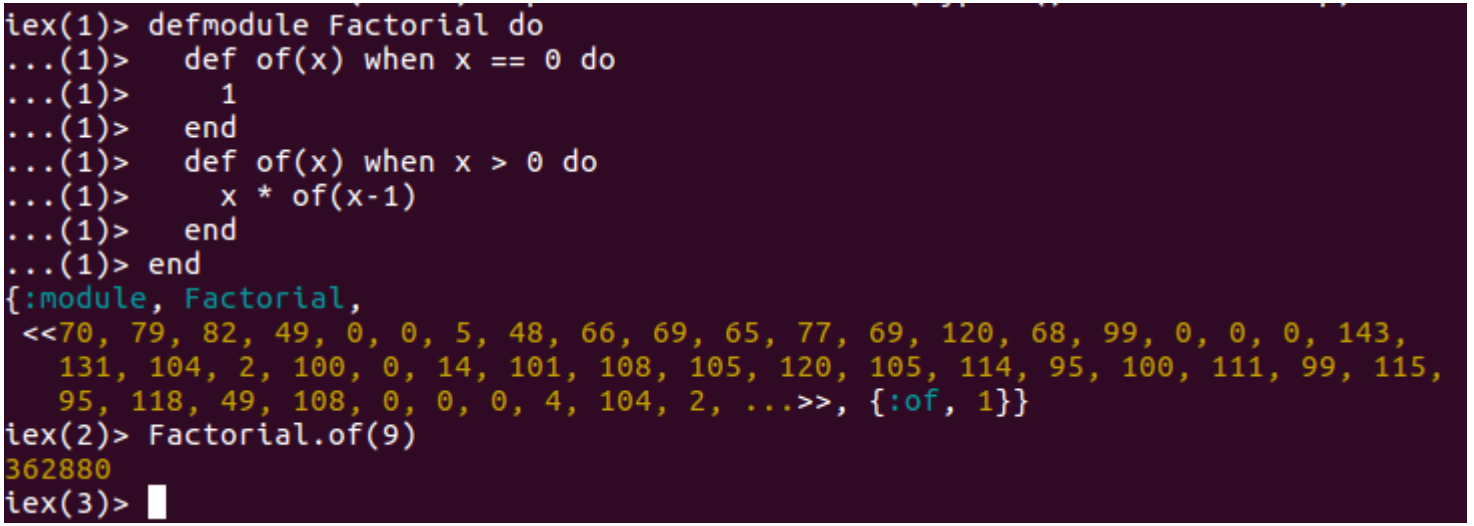
Elixir – Functions

- Pattern matching

```
iex(1)> defmodule Factorial do
...(1)>   def of(0) do 1 end
...(1)>   def of(x) do x * of(x-1) end
...(1)> end
{:module, Factorial,
 <<70, 79, 82, 49, 0, 0, 5, 20, 66, 69, 65, 77, 69, 120, 68, 99, 0, 0, 0, 143,
   131, 104, 2, 100, 0, 14, 101, 108, 105, 120, 105, 114, 95, 100, 111, 99, 115,
   95, 118, 49, 108, 0, 0, 0, 4, 104, 2, ...>>, {:of, 1}}
iex(2)> Factorial.of(9)
362880
iex(3)>
```

Elixir – Functions

- Guards
  - Are an extension of the basic pattern-matching mechanism.
  - Can be specified providing the **when** clause after the arguments list.



Elixir – Functions

- Guards

```
iex(1)> defmodule Greeting do
...(1)>   def greeting(name, age) when age < 2 do
...(1)>     IO.puts "Gaga #{name}"
...(1)>   end
...(1)>   def greeting(name, age) when age < 5 do
...(1)>     IO.puts "Hi #{name}"
...(1)>   end
...(1)>   def greeting(name, age) do
...(1)>     IO.puts "Hello #{name}"
...(1)>   end
...(1)> end
warning: variable "age" is unused
  iex:8

{:module, Greeting,
 <<70, 79, 82, 49, 0, 0, 7, 104, 66, 69, 65, 77, 69, 120, 68, 99, 0, 0, 0, 167,
   131, 104, 2, 100, 0, 14, 101, 108, 105, 120, 105, 114, 95, 100, 111, 99, 115,
   95, 118, 49, 108, 0, 0, 0, 4, 104, 2, ...>>, {:greeting, 2}}
iex(2)> Greeting.greeting("Ruben",2)
Hi Ruben
:ok
iex(3)>
```

Elixir – Functions

- Limitations of guards
  - Comparison Operators
    - ==, !=, ===, !==, >, <, <=, >=
  - Boolean and negation operators
    - Or, and, not, ! (not allowed: ||, &&)
  - Aritmetic operators
    - +, -, *, /
  - Join Operators
    - ++ <>
  - In operator
    - 5 in (0..6)
  - Type-check functions
    - Is_atom, ... is_*
  - Some other functions
    - abs(number), length(list) ...

Elixir – Functions

- Default params

```
iex(21)> defmodule Math do
...(21)>   def sum(a,b \\ 2) do
...(21)>     a + b
...(21)>   end
...(21)> end
{:module, Math,
 <<70, 79, 82, 49, 0, 0, 5, 48, 66, 69, 65, 77, 69, 120, 68, 99, 0, 0, 0, 173,
   131, 104, 2, 100, 0, 14, 101, 108, 105, 120, 105, 114, 95, 100, 111, 99, 115,
   95, 118, 49, 108, 0, 0, 0, 4, 104, 2, ...>>, {:sum, 2}}
iex(22)> Mat
MatchError    Math
iex(22)> Math.sum(6)
8
iex(23)> Math.sum(6,6)
12
iex(24)> Math.module_info(:functions)
[__info__: 1, sum: 1, sum: 2, module_info: 0, module_info: 1]
iex(25)>
```

Elixir – Modules and Named Functions

- Private functions

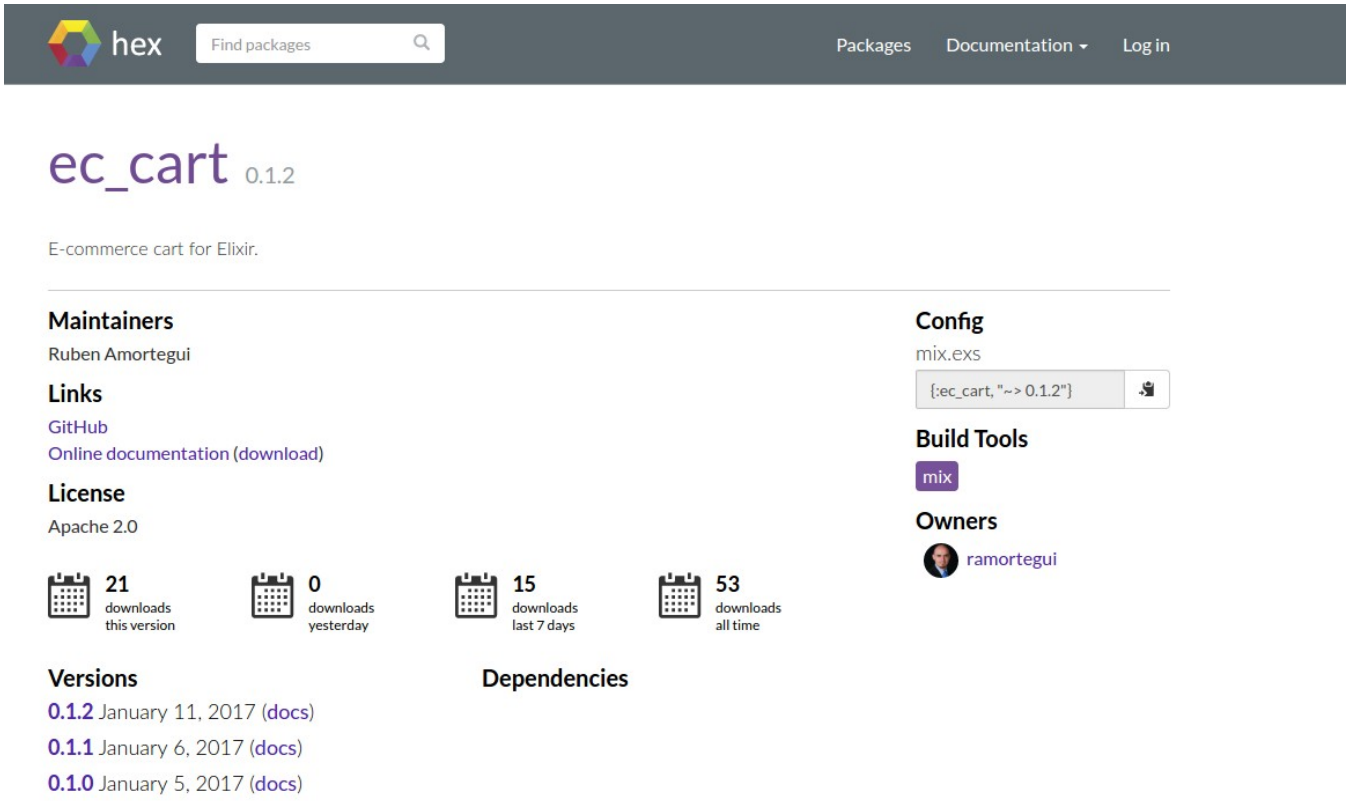
```
defmodule Factorial do
  def of(num) when num >=0 do
    _of(num,1)
  end

  defp _of(num, acc) when num == 0 do
    acc
  end
  defp _of(num, acc) when num > 1 do
    _of(num-1, acc*num)
  end
end
```

Elixir – Summary

- Functions are the engine of Elixir
  - Named functions
    - Needs to be defined on a module
    - Are identified by name and arity
    - Could be public or private
    - The arguments could have a default value
  - Anonymous functions
    - Are typed as functions
      - pass/returned by other
    - Parameter list must have the same number of arguments
  - Use pattern matching to select the right clause
  - The order of the matching clauses is important

Elixir – Ec.Cart

- https://hex.pm/packages/ec_cart



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