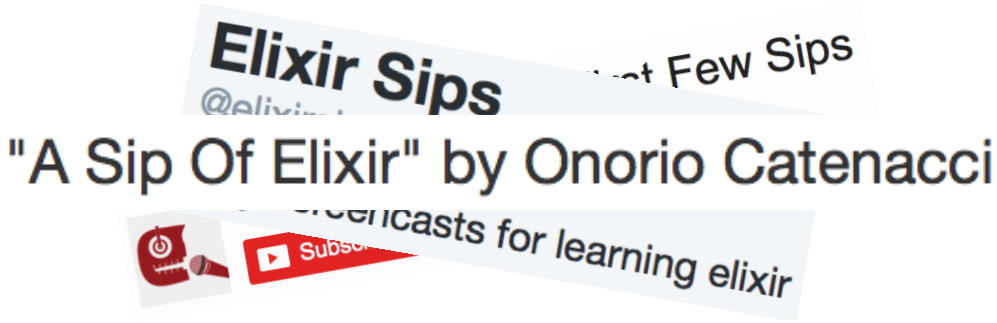
Elixir: love at first sip


Elixir Sips
Few Sips
"A Sip Of Elixir" by Onorio Catenacci
casts for learning elixir

Andrea Leopardi

@whatyouhide

andrea@leopardi.me

Questions

Interrupt me!

Elixir

> Elixir is a dynamic, functional language designed for building scalable and maintainable applications.

Nice!

Erlang

- ERicsson LANGuage
- Built to handle telecom applications
- In 1986, ~30 years ago!
- Features:
  - Highly available
  - Extremely concurrent
  - Fault tolerant

What does it look like

```elixir
defmodule Slugger do
  @doc "Slugs a string."
  @spec slufigy(String.t) :: String.t
  def slugify(str) do
    str
    |> String.downcase
    |> String.replace("'", "")
    |> String.replace(~/[?!,\-:;.]/, "-")
  end
end
```

Dynamic

- Dynamically typed (strongly typed)
- Runtime code evaluation

Functional

Immutable data structures

```
list = [:foo, :bar, :baz]

List.delete_at(list, 1)
#=> [:foo, :baz]

list
#=> [:foo, :bar, :baz]
```

Functional

High-order functions

```
sum = fn(n) ->
  fn(x) -> x + n end
end

add42 = sum.(42)
add42.(1)
#=> 43
```

```
Enum.map [1, 2, 3, 4], fn(x) ->
  x * 3
end
#=> [3, 6, 9, 12]
```

Not *purely* functional

Side effects:

```
iex> IO.puts "Hello world!"
Hello world!
:ok
```

Call-dependent results:

```
iex> :erlang.now()
{1426, 179468, 348062}
iex> :erlang.now()
{1426, 179478, 386986}
```

Scalable

```
for _ <- 1..1_000_000 do
  spawn fn ->
    "hello"
  end
end
```

Thanks Erlang!

Maintainable

- Nice syntax
- *Extremely* extensible
- Well documented

Elixir <3 Erlang

Pattern matching

= is not what it looks like.

```
list = [1, 2, 3, 4]

[first, second|rest] = list

first  #=> 1
second #=> 2
rest   #=> [3, 4]
```

```
tuple = {:my, 3, "elements", :tuple}

{:my, how_many, "ele" <> rest_of_the_string, :tuple} = tuple

how_many           #=> 3
rest_of_the_string #=> "ments"
```

Pattern matching

Can be used in function heads:

```
def do_action(:something) do
  something()
end

def do_action(:something_else) do
  something_else()
end
```

Pattern matching + guards = polymorphism

(on steroids!)

```
defmodule Math do
  def factorial(n) when n < 0 do
    raise "negative"
  end

  def factorial(0) do
    1
  end

  def factorial(n) do
    n * factorial(n - 1)
  end
end
```

Processes

```
pid = spawn fn ->
  IO.puts "Hello world!"
end
#=> #PID<0.61.0>
```

Lightweight processes

```
:timer.tc fn ->
  Enum.each 1..100_000, fn(_) ->
    spawn(fn -> :foo end)
  end
end
#=> {1735741, :ok}
```

That's about **1.7s** for 100k spawned processes!

Sending messages

```
pid = spawn fn ->
  :timer.sleep 20_000
end

send pid, "Hello!"
#=> "Hello!"
```

Receiving messages

```
pid = spawn fn ->
  receive do
    {from, :hello} -> send from, {self(), "Hello to you!"}
    {from, :bye}   -> send from, {self(), "Bye bye!"}
  end
end
```

```
send pid, {self(), :hello}
#=> {#PID<0.79.0>, :hello}
```

```
receive do
  {_from, message} -> "Response: #{message}"
end
#=> "Response: Hello to you!"
```

```elixir
defmodule Actor do
  # "Client"
  def start(initial_state) do
    spawn fn -> loop(initial_state) end
  end

  def get(pid) do
    send pid, {:get, self()}
    receive do
      state -> state
    end
  end

  def put(pid, state) do
    send pid, {:put, self(), state}
  end

  # "Server"
  def loop(state) do
    receive do
      {:get, from} ->
        send(from, state)
        loop(state)
      {:put, from, new_state} ->
        loop(new_state)
    end
  end
end
```

```elixir
actor = Actor.start(1)
Actor.set(actor, 2)

Actor.get(actor)
#=> 2
```

OTP

Open Telecom Platform

Ah, like CoffeeScript for JavaScript

NO

More like Clojure for Java

Elixir is what would happen if **Erlang**, **Clojure** and **Ruby** somehow had a baby and it wasn't an accident. -- *Devin Torres*

Interop

Erlang

```
random:uniform().
%=> 0.4435846174457203
```

Elixir

```
:random.uniform
#=> 0.4435846174457203
```

Interop

Erlang

```
lists:map(fun(El) -> El + 2 end, [1, 2, 3]).
%=> [3, 4, 5]
```

Elixir

```
:lists.map(fn(el) -> el + 2 end, [1, 2, 3])
#=> [3, 4, 5]
```

Protocols

```
defprotocol Blank do
  def blank?(data)
end
```

```
defimpl Blank, for: List do
  def blank?(list) do
    Enum.empty? list
  end
end
```

```
Blank.blank? []        #=> true
Blank.blank? [1, 2, 3] #=> false
```

Pipe operator

This...

```
Enum.map(List.flatten([1, [2], 3]), fn(x) -> x * 2 end)
```

...becomes this:

```
[1, [2], 3]
|> List.flatten
|> Enum.map(fn(x) -> x * 2 end)
```

Pipe operator

Non-ridicolous (still simplified) example:

```
def eval_file(path) do
  path
  |> File.read!
  |> Tokenizer.tokenize
  |> Parser.parse
  |> Interpreter.eval
end
```

Nice, uh?

Tooling

Great REPL (IEx):

```
iex(1)> 3 + 4
7
iex(3)> v(1) + 4
11
```

Tooling

Built-in templating language (EEx):

```
EEx.eval_string "Hello, <%= name %>", [name: "José"]
#=> "Hello, José"
```

Tooling

Build/test/project management tool (Mix):

```
mix new my_new_project
cd my_new_project
mix compile
mix test
```

Tooling

Package manager (Hex):

```
def dependencies do
  [{:cowboy, "~> 1.0"},
   {:plug, github: "elixir-lang/plug"}]
end
```

(not in the core)

Ok, so that's pretty much it...

WAIT

METAPROGRAMMING

MACROS!

HOMOICONICITY!

Homoiconicity

> In a homoiconic language the primary representation of programs is also a data structure in a primitive type of the language itself.

The representation of Elixir code is valid Elixir code!

Quoting

```elixir
quote do
  1 + 2
end
#=> {:+, _metadata, [1, 2]}
```

Looks not-so-different from Lisp:

```lisp
(+ 1 2)
```

```elixir
{op, meta, args} = quote(do: 1 + 2)

new_args = Enum.map(args, fn(x) -> x * 2 end)
#=> [2, 4]

Code.eval_quoted {op, meta, new_args}
#=> 6
```

Unquoting

```elixir
a = 1

quote do
  a + 3
end
#=> {:+, _metadata, [{:a, [], Elixir}, 3]}
```

```elixir
a = 1

quote do
  unquote(a) + 3
end
#=> {:+, _metadata, [1, 3]}
```

Macros

```elixir
defmodule MyMacros do
  defmacro unless(condition, do: something) do
    quote do
      if !unquote(condition) do
        unquote(something)
      end
    end
  end
end
```

```elixir
MyMacros.unless 2 + 2 == 5 do
  "Math still works"
end
#=> "Math still works"
```

Demo time!

May the demo gods be with me

Questions?

Andrea Leopardi

@whatyouhide (twitter/github)

andrea@leopardi.me