

An abstract, layered geometric shape in shades of purple, resembling a stylized flower or a complex crystal, positioned on the left side of the slide.

# INTRODUCTION TO ELIXIR

By: Junior Farias

06/09/2017

**WHAT IS  
ELIXIR?**

Elixir é uma linguagem dinâmica e **funcional** concebido para construir ***aplicações escaláveis*** e ***sustentáveis***.

Elixir utiliza a maquina virtual do **Erlang**, que executa ***sistemas distribuídos***, com ***baixa latência*** e ***tolerantes a falhas***.

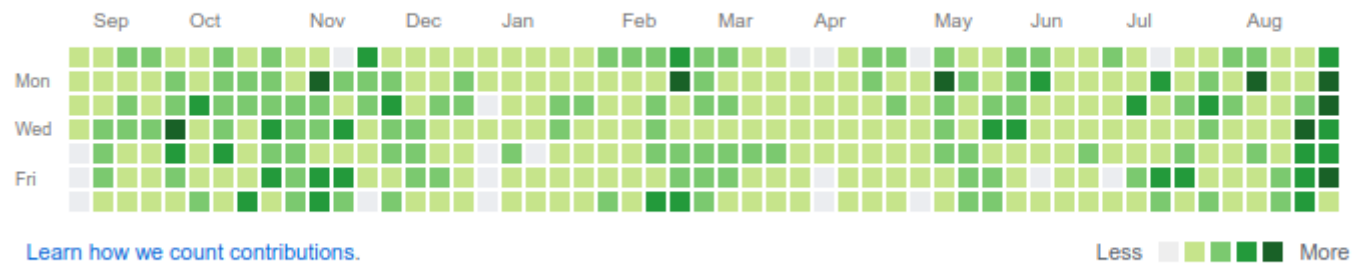


Criado por **José Valim** em torno de **2012**.  
Ex-Membro do **Rails Core Team**.  
Ele estava tentando fazer o Rails **Thread safe**,  
mas... acabou criando uma nova  
linguagem de programação. Ops!!!

#### Organizations




2,637 contributions in the last year



*Erlang* é essa linguagem feia de **1996** usada em ambiente de produção por alguns "**pequenos**" caras, tais como:

- WhatsApp
- Facebook (chat backend)
- Amazon (SimpleDB)
- AdRoll
- Heroku

2 milhões de conexões  
em um único nó



- Yahoo (Delicious)
- Ericsson (mobile networks)
- T-mobile (SMS)
- World of Warcraft
- ...

# *Elixir:*

- Compila para Erlang bytecode.
- Pode chamar qualquer biblioteca do Erlang sem perda de performance.
- Permite a produtividade dos desenvolvedores oferecendo ferramentas incríveis e bela documentação.

**WHY SHOULD I  
CARE ABOUT  
ELIXIR?**

- As CPUs hoje em dia possuem vários **transistores** e muitos **núcleos**.
- Não há nenhuma maneira de **mantê-los ocupados** para que juntos façam tudo de uma vez.
- A única maneira de mantê-los ocupados é mergulhá-los em trabalho.
- Em outras palavras:
- O futuro é **funcional** e **concorrente**.



- Elixir prova que a programação funcional não precisa ser algo matemático ou complexo.
- Com o Elixir, podemos fazer programação de ***simultaneidade***.
- Sem ter que usar ***abstrações***, como ***bloqueios*** ou ***semáforos***.
- Como podemos programar sem o estado ***Goto***? (ciência da computação, 1968).
- Como podemos programar sem o ***estado mutável***? (ciência da computação, 2016)

**SHOW ME  
THAT  
ELIXIR**

# Value Types

- *Integers*  
1, 2, 3
- *Floats*  
0.12, 1.4
- *Atoms*  
:my\_atom  
true, false, nil
- *Ranges*  
start .. end 1 .. 3
- *Regular expressions*  
~r{ regexp }

# Collection Types

- *Tuples*(object)  
{ :ok, 42, "next" }
- *Linked List*  
list = [1, 2, 3, 4]
- *Binaries*  
<<1, 2>>
- *Maps*  
• %{ key => value, key => value }

# System Types

- **PIDs** - um novo PID é criado quando você cria um novo processo.
- **Ports** - Referência a uma porta.

## Anonymous Function

- Podemos passar argumentos de qualquer tipo.
- Parênteses são opcionais.

```
iex> add = fn (a, b) → a + b end
```

```
iex> add. (1, 2)
```

3

# Named Function

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

```
iex> MyModule.say_hello(" BeeTech Elixir")
Hello BeeTech Elixir
```

# Patten Matching

Em Elixir `bee = 1` não significa que nós estamos atribuindo 1 para a variável bee. Mas sim que afirmando que o valor seja 1

```
lex> bee = 1
```

```
lex> 1 = bee
```

```
1      (você não pode fazer isso em linguagens não funcionais)
```

***Vamos fazer magica:***

```
iex> a = 1
```

```
iex> [a, 2, 3] = [1, 2, 3]
```

```
[1, 2, 3]
```

```
iex> a
```

```
1
```

# Patten Matching

Você pode ignorar valores com “\_”

```
lex> [a, _, _] = [1, 2, 3]
```

```
[1, 2, 3]
```

Você pode reutilizar o valor de uma variavel com o operador “^”

```
lex> a = 1
```

```
1
```

```
lex> [^a, 2, 3] = [1, 2, 3]
```

```
[1, 2, 3]
```

# Function Signatures

A mesma função pode ter mais de uma assinatura.

```
defmodule Fatorial do
  def beetech(0), do: 1
  def beetech(x), do: x * beetech(x-1)
end
```

## Exceptions?

```
case File.open("chain.exs") do
  { :ok, file } -> # faz algo
  { :error, reason } -> # uh oh
end
```



# Pipe Operator |>

Código em programação OOP:

```
people = DB.find_customers  
orders = Orders.for_customers(people)  
tax = sales_tax(orders, 2013)  
filing = prepare_filing(tax)
```

Nós podemos escrever isso como...

```
filing = prepare_filing(  
  sales_tax(Orders.for_customers(  
    DB.find_customers), 2013))
```

Em Elixir podemos fazer apenas:

```
filing = DB.find_customers  
        |> Orders.for_customers  
        |> sales_tax(2013)  
        |> prepare_filing
```

# MIX

- Mix é uma ferramenta de compilação que **acompanha** o Elixir que fornece tarefas para **criar**, **compilar**, **testar** sua aplicação, **gerenciando** suas **dependências** e muito mais...
- Um tipo de rake do (Ruby) com esteróides.

# MIX

→ ~ mix --help

mix	# Runs the default task (current: "mix run")
mix app.start	# Starts all registered apps
mix archive	# Lists all archives
mix archive.build	# Archives this project into a .ez file
mix archive.install	# Installs an archive locally
mix archive.uninstall	# Uninstalls archives
mix clean	# Deletes generated application files
mix cmd	# Executes the given command
mix compile	# Compiles source files
mix deps	# Lists dependencies and their status
mix deps.clean	# Deletes the given dependencies' files
mix deps.compile	# Compiles dependencies
mix deps.get	# Gets all out of date dependencies
mix deps.unlock	# Unlocks the given dependencies
mix deps.update	# Updates the given dependencies
mix do	# Executes the tasks separated by comma
mix escript.build	# Builds an escript for the project
mix help	# Prints help information for tasks
mix hex	# Prints Hex help information
mix hex.build	# Builds a new package version locally
mix hex.config	# Reads or updates Hex config
mix hex.docs	# Publishes docs for package
mix hex.info	# Prints Hex information
mix hex.key	# Hex API key tasks
mix hex.outdated	# Shows outdated Hex deps for the current project

# MIX

→ ~ mix --help

mix hex.owner	# Hex package ownership tasks
mix hex.public_keys	# Manages Hex public keys
mix hex.publish	# Publishes a new package version
mix hex.registry	# Hex registry tasks
mix hex.search	# Searches for package names
mix hex.user	# Hex user tasks
mix loadconfig	# Loads and persists the given configuration
mix local	# Lists local tasks
mix local.hex	# Installs Hex locally
mix local.phoenix	# Updates Phoenix locally
mix local.public_keys	# Manages public keys
mix local.rebar	# Installs rebar locally
mix new	# Creates a new Elixir project
mix phoenix.new	# Creates a new Phoenix v1.1.4 application
mix profile.fprof	# Profiles the given file or expression with fprof
mix run	# Runs the given file or expression
mix test	# Runs a project's tests
iex -S mix	# Starts IEx and run the default task

**ELIXIR FOR  
WEB APPS**

# Phoenix Framework

- Framework web MVC criado por **Chris McCord**.  
com participações do **José Valim**.
- Parece com Ruby on Rails, mas não é.
- Objetivo para a **alta produtividade** do desenvolvedor e **alto desempenho** da aplicação.
- Tem abstrações poderosas para a criação da web moderna com **Canais** (web sockets).

# Phoenix Framework

- Create a new project  
`$ mix phoenix.new hello_phoenix`
- Create database  
`$ mix ecto.create`
- Run server  
`$ mix phoenix.server`
- O phoenix foi projetado para ser **modular** e **flexível**.
- As outras camadas incluem **Plug** (tipo como Ruby's Rack),
- **Ecto** (tipo de ActiveRecord) e **Cowboy**, o Erlang Servidor HTTP

# Phoenix Framework

Os **aplicativos web** possuem **1 canal bidirecional** de dados em tempo real com **cada usuário** (um websocket). E escala, de verdade.

Apesar do WebSocket ser o principal canal de Transporte. O Phoenix, também suporta outros mecanismo para navegadores antigos ou dispositivos incorporados.

Ele funciona bem com o moderno mundo de **front-end (ES6)** por contar com o **Brunch**, uma ferramenta de Build rápida e simples.

É muito fácil substituir o Brunch pelo **Webpack** ou por qualquer outra ferramenta hipster de sua escolha.

\* Brunch requer Node.js (trollface).



# HOW FAST IS ELIXIR?

spoiler... a baixo!

### RAM footprint per unit of concurrency (approx)

1.3KB	Haskell ThreadId + MVar (GHC 7.6.3, 64-bit)
2.6 KB	Erlang process (64-bit)
4.0 KB	Go goroutine
9.0 KB	C pthread (minimum, 64-bit Mac OS X)
64.0 KB	Java thread stack (minimum)
<hr/>	
513 KB	C pthread (default, 64-bit Mac OS X)
1024 KB	Java thread stack (default)

Library	Throughput (req/s)	Latency (ms)
<b>Plug (elixir)</b>	54,948	3.83
<b>Gin (go)</b>	51,483	1.94
<b>Phoenix (elixir)</b>	43,063	2.82
<b>Express Cluster (node)</b>	27,669	3.73
<b>Martini (go)</b>	14,798	6.81
<b>Express (node)</b>	9,965	10.07
<b>Sinatra (ruby)</b>	9,182	6.55
<b>Rails (ruby)</b>	3,274	17.25

<https://github.com/mroth/phoenix-showdown>

**SHOWCASE**

# Showcase

- BSRBulb: biblioteca para controlar Bluetooth Smart Bulb.  
[https://github.com/diacode/bsr\\_bulb](https://github.com/diacode/bsr_bulb)
- Phoenix Trello: uma aplicação com Phoenix & React.  
<https://github.com/bigardone/phoenix-trello>  
<https://blog.diacode.com/trello-clone-with-phoenix-and-react-pt-1>
- Phoenix Toggl: um Timer feito em Phoenix & React.  
<https://github.com/bigardone/phoenix-toggl>
- Elixir Toggl API Wrapper  
<https://github.com/diacode/togglex>

**WHERE TO  
GO FROM  
HERE?**

# Next steps

- Assista todas as conversas de José Valim. Na verdade, você não vai se arrepender.

- **Books:**

Programming Elixir – Dave Thomas

Programming Phoenix – Chris McCord, Bruce Tate & José Valim.

- Elixir Getting Started Guide

<http://elixir-lang.org/getting-started/introduction.html>

- Phoenix Guide

<http://www.phoenixframework.org/docs/overview>

- Pluralsight

<https://app.pluralsight.com/library/courses/phoenix-getting-started/table-of-contents>

# THANK YOU

Questions ?