@trenpixster

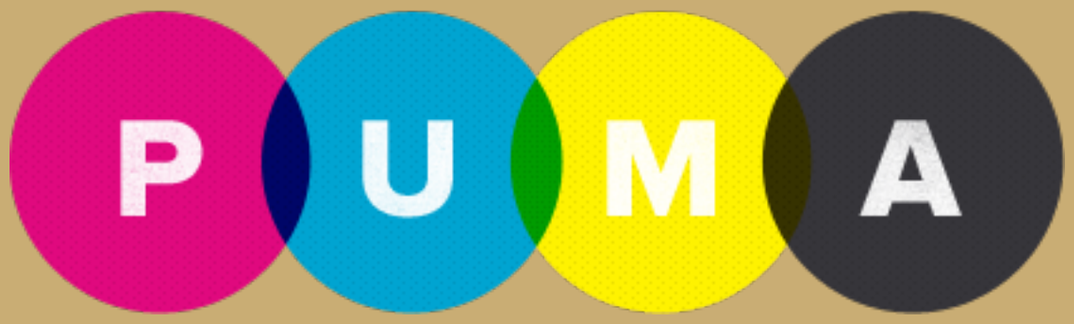
@tjsousa

talkdesk

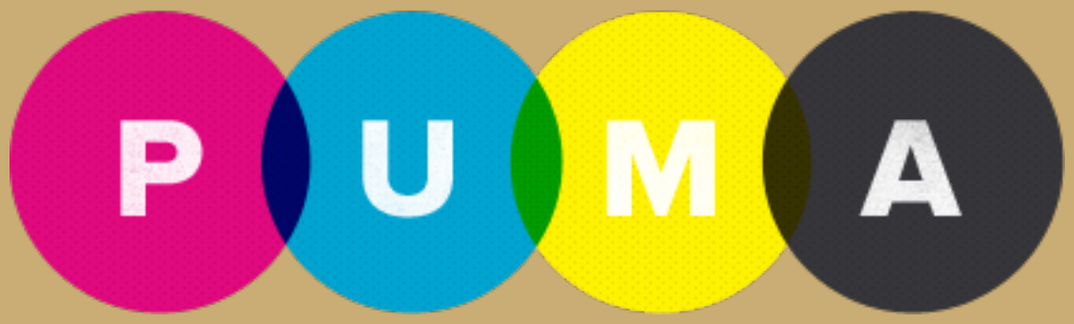
talkdesk

talkdesk

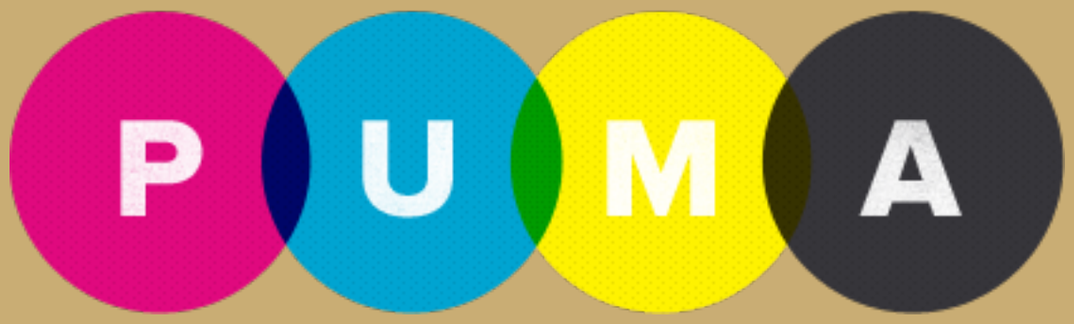
talkdesk



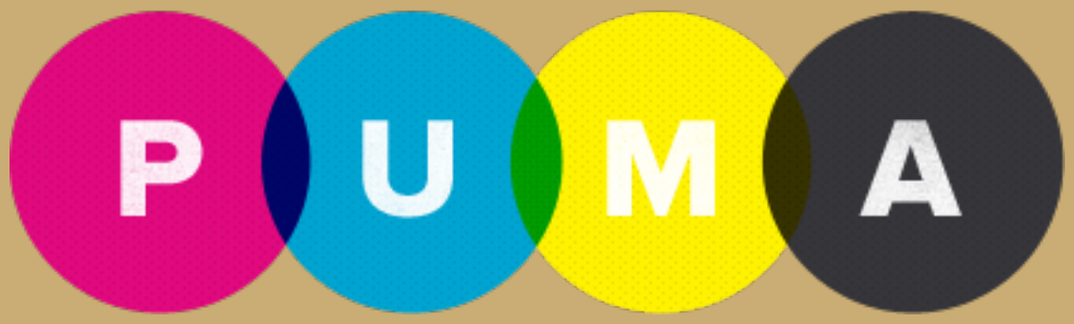
talkdesk



talkdesk



talkdesk



talkdesk

PUMA

talkdesk

PUMA

JOSÉ VALUM ES

josé valim

José Valim

josevalim

Plataformatec
Kraków, Poland
jose.valim@plataformatec.com…
http://blog.plataformatec.com.br/
Joined on May 7, 2008

3.2k Followers **223** Starred **23** Following

Organizations

Contributions | Repositories | Public activity

Follow

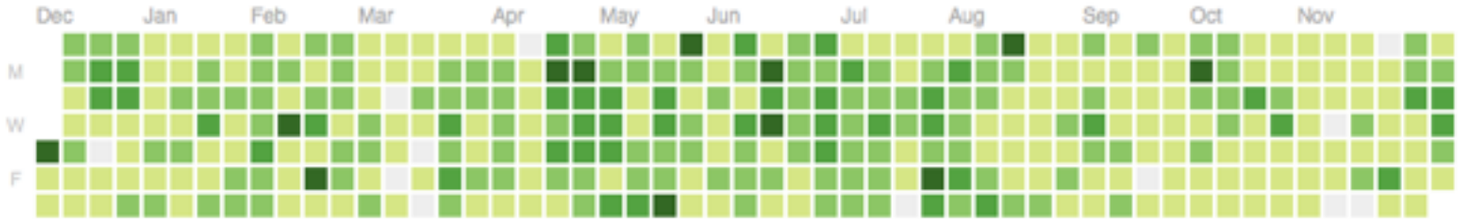
Popular repositories

- **inherited_resources** — 2,424 ★
- **rails-footnotes** — Every Rails page has footnotes that gives info… — 1,255 ★
- **enginex** — An executable which creates a bare Rails 3 en… — 466 ★
- **easy_http_cache** — Allows Rails controllers to use HTTP cache sp… — 78 ★
- **xgen** — Integrating Elixir, Mix and OTP — 45 ★

Repositories contributed to

- **elixir-lang/elixir** — Elixir is a dynamic, functional language design… — 3,206 ★
- **elixir-lang/elixir-lang.github.com** — Website for Elixir — 60 ★
- **elixir-lang/plug** — A specification and conveniences for composa… — 202 ★
- **phoenixframew…/phoenix** — Elixir Web Framework — 1,711 ★
- **elixir-lang/ex_doc** — ExDoc produces HTML and online documenta… — 53 ★

Public contributions

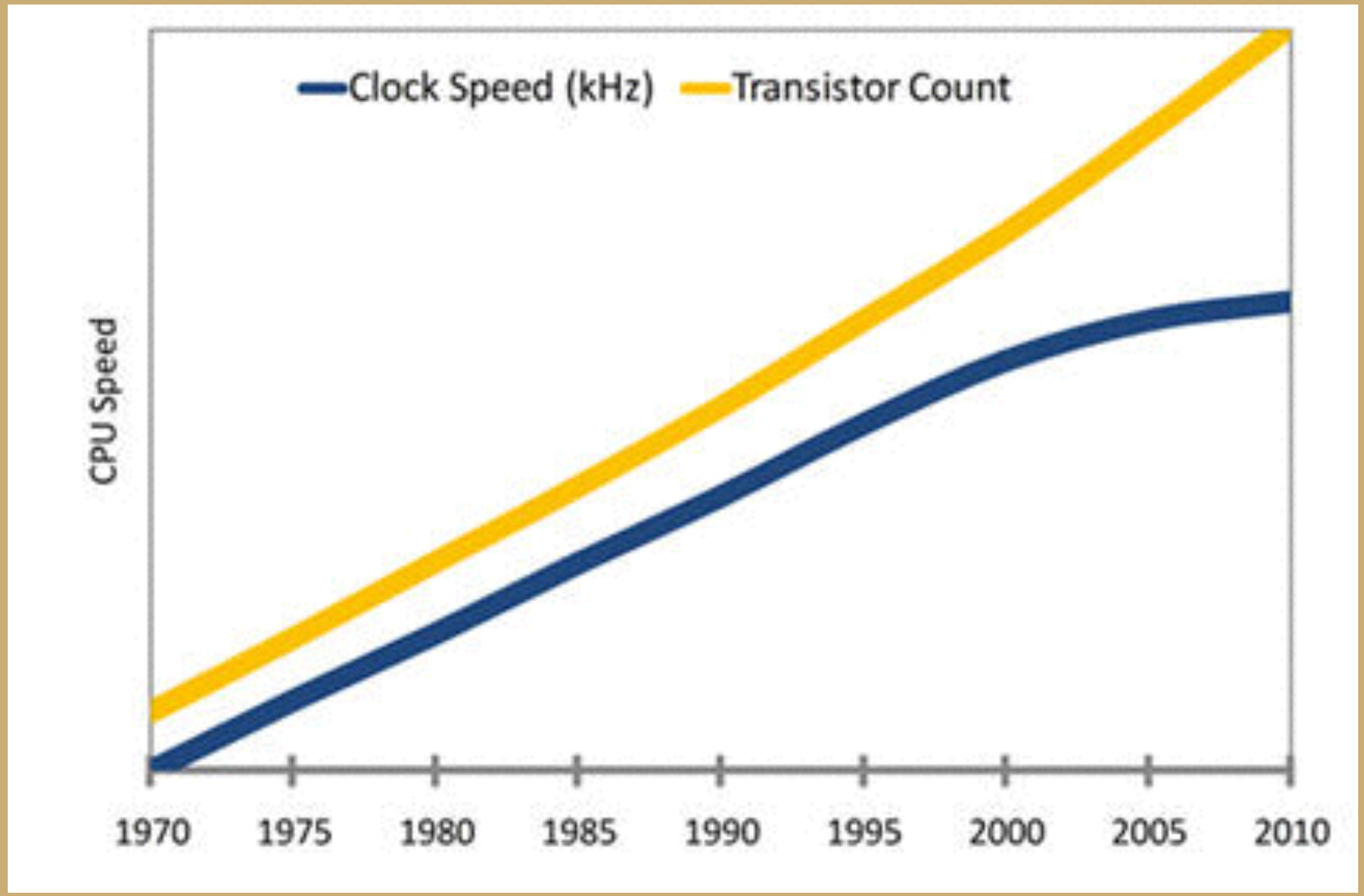


Summary of Pull Requests, issues opened, and commits. Learn more.

Less More

Year of contributions	Longest streak	Current streak
3,192 total	103 days	19 days
Dec 12, 2013 – Dec 12, 2014	April 14 – July 25	November 24 – December 12


the free lunch is over



the free lunch is over

the free lunch is over

the free lunch is over


rails is thread-safe! hah!

seven languages in seven weeks

seven languages in seven weeks

Io

seven languages in seven weeks

Io

seven languages in seven weeks

Io

ProLog

seven languages in seven weeks

Io

ProLog

seven languages in seven weeks

Io

ProLog

seven languages in seven weeks

Io

ProLog

seven languages in seven weeks

Erlang

what is Elixir?

***Elixir** is a functional, meta-programming aware language built on top of the Erlang VM*

productivity

productivity

natural syntax

```elixir
defmodule Queries do
  def rainy_days do
    query = from w in Weather,
            where: w.prcp > 0 or is_nil(w.prcp),
            select: w
  end
end
```

```elixir
defmodule Queries do
  def rainy_days do
    query = from w in Weather,
            where: w.prcp > 0 or is_nil(w.prcp),
            select: w
  end
end
```

productivity

language tooling

```
> iex

> mix deps.get

> mix build

> mix test

> mix hex.publish
```

```
> iex

> mix deps.get

> mix build

> mix test

> mix hex.publish
```

extensibility

extensibility

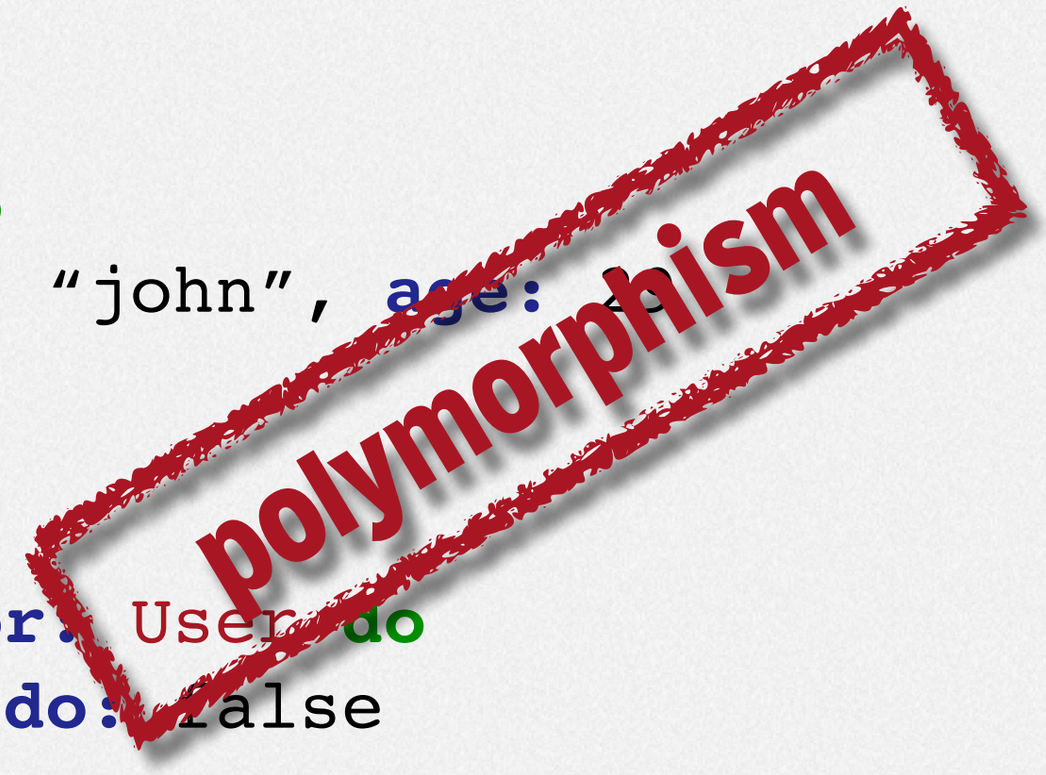
polymorphism

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

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


defmodule User do
  defstruct name: “john”, age: 29
end
```

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


defmodule User do
  defstruct name: “john”, age: 29
end


defimpl Blank, for: User do
  def blank?(_), do: false
end
```

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


defmodule User do
  defstruct name: "john", age: 25
end


defimpl Blank, for: User do
  def blank?(_), do: false
end
```



extensibility

meta-programming

```elixir
defmodule Router do
  use Phoenix.Router, port: 4000

  get  "/pages/:page", Ctrls.Pages, :show
  get  "/files/*path", Ctrls.Files, :show
  post "/files",       Ctrls.Files, :upload

  resources "users", Ctrls.Users do
    resources "comments", Ctrls.Comments
  end
end
```

```
defmodule Router do
  use Phoenix.Router, port: 4000

  get  "/pages/:page", Ctrls.Pages, :show
  get  "/files/*path", Ctrls.Files, :show
  post "/files",       Ctrls.Files, :upload

  resources "users", Ctrls.Users do
    resources "comments", Ctrls.Comments
  end
end
```

> *Lisps traditionally empowered developers because* ***you can eliminate anything that's tedious through macros, and that power is really what people keep going back for.***
>
> *— Rich Hickey*

compatibility

compatibility

interoperabilty

compatibility

interoperabilty

compatibility

parallelism

compatibility

distribution

OTP

DISTRIBUTED
FAULT-TOLERANT
APPLICATIONS
WITH HOT-CODE
SWAPPING

it’s functional!

it’s functional!

(a(b(c(d(e)))))

it’s functional!

it’s  nal!

pattern matching

```elixir
def foo do
  {"bar", "baz"}
end
```

```
iex > {str1, str2} = foo()
iex > str1
"bar"
iex > str2
"baz"
```

pattern matching

```elixir
def foo do
 {"bar", "baz"}
end
```

```
iex > {str1, str2} = foo()
iex > str1
"bar"
iex > str2
"baz"
```

```elixir
def foo({var1, var2, "bar"}) do
  var1
end

def foo({_, var2, "baz"}) do
  var2
end
```

```
iex > foo {1,2,"bar"}
1
iex > foo {1,2,"baz"}
2
```

lists

```
iex > [ head | tail ] = [1, 2, 3, 4, 5]
iex > head
1
iex > tail
[2, 3, 4, 5]
```

lists & recursion

```elixir
defmodule MyList
  def max([x]), do: x
  def max([ head | tail ]), do: Kernel.max(head, max(tail))
end
```

recursion

```elixir
defmodule Fib do
  def fib(0) do 0 end
  def fib(1) do 1 end
  def fib(n) do fib(n-1) + fib(n-2) end
end
```

pipe |> operator

```
iex > Enum.map(["a", "b", "c"], fn(x) -> x <> "LEM" end)
```

pipe |> operator

```
iex > Enum.map(["a", "b", "c"], fn(x) -> x <> "LEM" end)
["aLEM", "bLEM", “cLEM"]
```

pipe |> operator

```
iex > Enum.map(["a", "b", "c"], fn(x) -> x <> "LEM" end)
["aLEM", "bLEM", “cLEM"]

iex > ["a", "b", "c"] |> Enum.map fn(x) -> x <> "LEM" end
```

pipe |> operator

```
iex > Enum.map(["a", "b", "c"], fn(x) -> x <> "LEM" end)
["aLEM", "bLEM", “cLEM"]

iex > ["a", "b", "c"] |> Enum.map fn(x) -> x <> "LEM" end
["aLEM", "bLEM", "cLEM"]
```

pipe |> operator

```
iex > Enum.map(["a", "b", "c"], fn(x) -> x <> "LEM" end)
["aLEM", "bLEM", “cLEM"]

iex > ["a", "b", "c"] |> Enum.map fn(x) -> x <> "LEM" end
["aLEM", "bLEM", "cLEM"]

# OR

iex > ["a", "b", "c"] |> Enum.map &(&1 <> "LEM")
["aLEM", "bLEM", "cLEM"]
```

pipe |> operator

```
iex > Enum.map(["a", "b", "c"], fn(x) -> x <> "LEM" end)
["aLEM", "bLEM", “cLEM"]

iex > ["a", "b", "c"] |> Enum.map fn(x) -> x <> "LEM" end
["aLEM", "bLEM", "cLEM"]

# OR

iex > ["a", "b", "c"] |> Enum.map &(&1 <> "LEM")
["aLEM", "bLEM", "cLEM"]

# OR

iex > add_lem = fn(x) -> x <> "LEM" end
iex > ["a", "b", "c"] |> Enum.map add_lem
```

pipe |> operator

```
iex > Enum.map(["a", "b", "c"], fn(x) -> x <> "LEM" end)
["aLEM", "bLEM", “cLEM"]

iex > ["a", "b", "c"] |> Enum.map fn(x) -> x <> "LEM" end
["aLEM", "bLEM", "cLEM"]

# OR

iex > ["a", "b", "c"] |> Enum.map &(&1 <> "LEM")
["aLEM", "bLEM", "cLEM"]

# OR

iex > add_lem = fn(x) -> x <> "LEM" end
iex > ["a", "b", "c"] |> Enum.map add_lem

iex > ["a", "b", "c"] |> Enum.map add_lem |> Enum.count
3
```

polymorphism

protocols

```elixir
defprotocol Blank do
  @doc "Returns true if data is considered blank/empty"
  def blank?(data)
end

defimpl Blank, for: Integer do
  def blank?(_), do: false
end

defimpl Blank, for: List do
  def blank?([]), do: true
  def blank?(_),  do: false
end

defimpl Blank, for: Map do
  def blank?(map), do: map_size(map) == 0
end

defimpl Blank, for: Atom do
  def blank?(false), do: true
  def blank?(nil),   do: true
  def blank?(_),     do: false
end
```

protocols

```
iex> Enum.map [1, 2, 3], fn(x) -> x * 2 end
[2,4,6]
iex> Enum.reduce 1..3, 0, fn(x, acc) -> x + acc end
6

iex> to_string :hello
"hello"
```

```
iex> "age: #{25}"
"age: 25"

iex> tuple = {1, 2, 3}
{1, 2, 3}
iex> "tuple: #{tuple}"
** (Protocol.UndefinedError) protocol String.Chars
 not implemented for {1, 2, 3}

iex> "tuple: #{inspect tuple}"
"tuple: {1, 2, 3}"
```

meta-programming

macros

```
iex> quote do: (1 + 2) - 5 * 10
{:-, [],
 [{:+, [], [1, 2]},
  {:*, [], [5, 10]}]}
```

macros

```
defmodule TestCase do
  use ExUnit.Case

  test "the truth is true" do
    assert 1 == 1
  end

  test "pretty error messages are pretty" do
    assert 1 == 2
  end
end
```

macros

```elixir
defmodule TestCase do
  use ExUnit.Case

  test "the truth is true" do
    assert 1 == 1
  end

  test "pretty error messages are pretty" do
    assert 1 == 2
  end
end
```

```
1) test pretty error messages are pretty (TestCase)
   Assertion with == failed
   code: 1 == 2
   lhs:  1
   rhs:  2
```

macros

```
defmodule Assertions do
  defmacro assert({operator, _, [lhs, rhs]}) do
    quote do
      do_assert unquote(operator),
                unquote(lhs),
                unquote(rhs)
      end
    end
  end

  def do_assert(:==, lhs, rhs) when lhs == rhs do
    IO.write(".")
  end

  def do_assert(:==, lhs, rhs) do
    IO.puts """
    FAIL: Expected: #{lhs}
    to be equal to: #{rhs}
    """
  end
end
```

macros

```
defmodule Router do
  use Phoenix.Router, port: 4000

  get  "/pages/:page", Ctrls.Pages, :show, as: :page
  get  "/files/*path", Ctrls.Files, :show
  post "/files",       Ctrls.Files, :upload

  resources "users", Ctrls.Users do
    resources "comments", Ctrls.Comments
  end
end
```

macros

```
defmodule Router do
  def match(conn, :get,    ["pages", page])
  def match(conn, :get,    ["files" | path])
  def match(conn, :post,   ["files"])
  def match(conn, :get,    ["users"])
  def match(conn, :get,    ["users", id, "edit"])
  def match(conn, :get,    ["users", id])
  def match(conn, :get,    ["users", "new"])
  def match(conn, :post,   ["users"])
  def match(conn, :put,    ["users", id])
  def match(conn, :patch,  ["users", id])
  def match(conn, :delete, ["users", id])
  def match(conn, :get,    ["users", user_id, "comments"])
  def match(conn, :get,    ["users", user_id, "comments", id, "edit"])
  def match(conn, :get,    ["users", user_id, "comments", id])
  def match(conn, :get,    ["users", user_id, "comments", "new"])
  def match(conn, :post,   ["users", user_id, "comments"])
  def match(conn, :put,    ["users", user_id, "comments", id])
  def match(conn, :patch,  ["users", user_id, "comments", id])
  def match(conn, :delete, ["users", user_id, "comments", id])
end
```

it’s parallel!

spawn

```
iex > spawn fn -> IO.puts "long running calcs" end
long running calcs
#PID<0.67.0>
```

spawn

```
iex > spawn fn -> IO.puts "long running calcs" end
long running calcs
#PID<0.67.0>
```

```
defmodule Foo do
  def bar do
    spawn fn -> raise "boom" end
  end
end

Foo.bar
```

```
$ elixir spawn.ex
$
```

spawn

```elixir
defmodule Foo do
  def bar do
    spawn_link fn -> raise "boom" end
  end
end

Foo.bar
```

```
$ elixir spawn_link.ex
```

spawn

```elixir
defmodule Foo do
  def bar do
    spawn_link fn -> raise "boom" end
  end
end

Foo.bar
```

```
$ elixir spawn_link.ex

02:05:52.531 [error] Error in process <0.53.0> with exit value:
{#{'__exception__'=>true,'__struct__'=>'Elixir.RuntimeError',
message=><<4 bytes>>},[{'Elixir.Foo','-bar/0-fun-0-',0,
[{file,"bla.ex"},{line,3}]}]}
** (EXIT from #PID<0.47.0>) an exception was raised:
    ** (RuntimeError) boom
        bla.ex:3: anonymous fn/0 in Foo.bar/0
```

tasks

```
long_operation = fn ->
:timer.sleep(5*1000) "Let's pretend this is a
 time expensive operation"
end

task = Task.async(long_operation)

foo = "a,b,c,d"
      |> String.split(“,")
      |> Enum.map(&String.upcase/1)
      |> Enum.join

res = Task.await(task)
IO.puts foo <> res
```

agents

```elixir
defmodule Cache do
  def start_link do
    Agent.start_link(fn -> HashDict.new end)
  end

  def put(pid, key, value) do
    Agent.update(pid, &Dict.put(&1, key, value))
  end

  def get(pid, key) do
    Agent.get(pid, &Dict.get(&1, key))
  end
end
```

tooling

Mix

jack of all trades

Hex

package manager

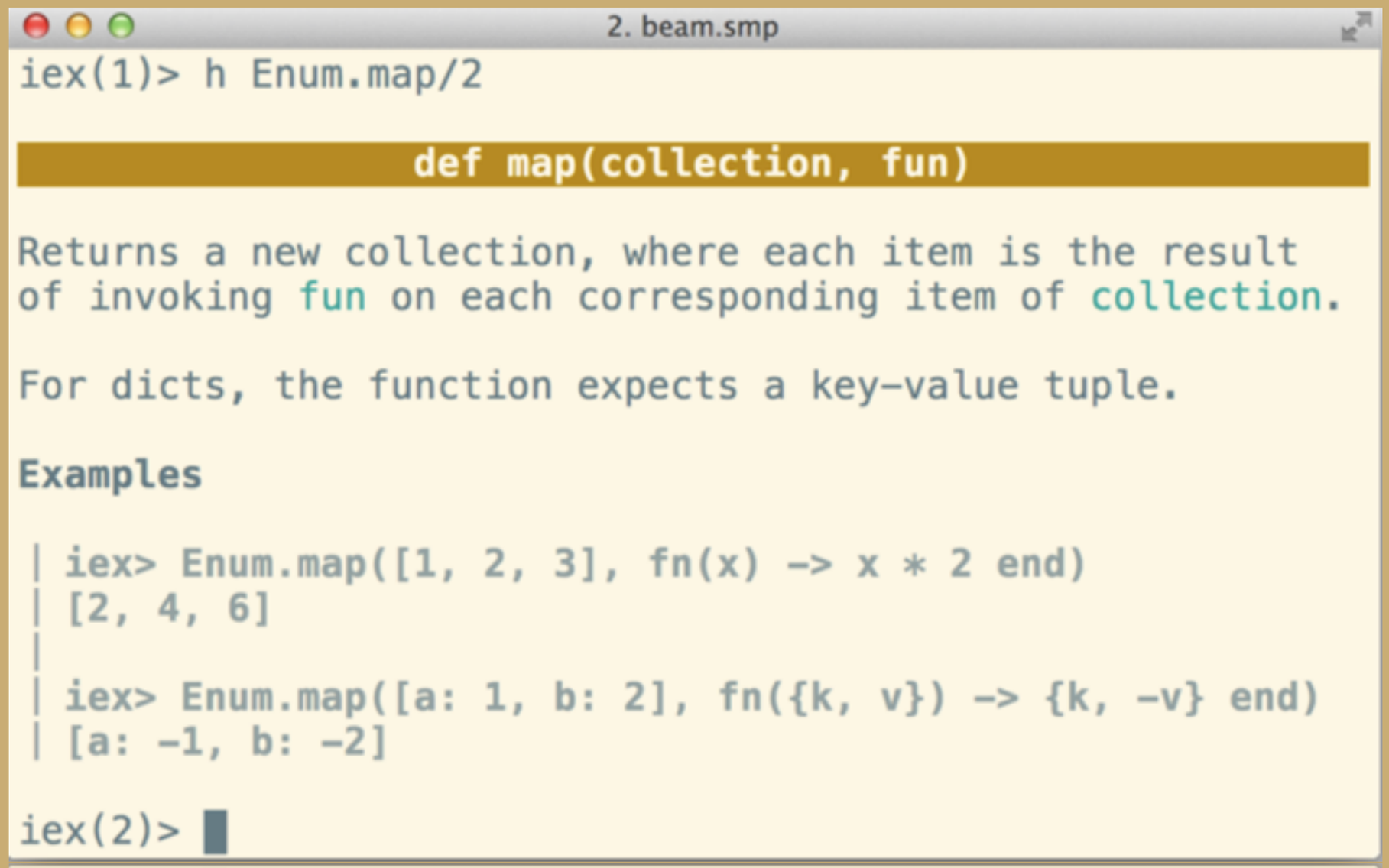
IEx

elixir's very own REPL

IEx.pry

similar to Ruby's Pry

documentation



documentation

```
defmodule Foo do


  def bar() do
    "hello"
  end
end
```

documentation

```
defmodule Foo do
@moduledoc """
Provides Foo related functions
## Examples
    iex > Foo.bar
    “hello”
"""



  def bar() do
    "hello"
  end
end
```

documentation

```elixir
defmodule Foo do
@moduledoc """
Provides Foo related functions
## Examples
    iex > Foo.bar
    “hello”
"""
  @doc """
  Evals the string \”hello\"
  """
  def bar() do
    "hello"
  end
end
```

documentation

```
Erlang/OTP 17 [erts-6.2] [source-aaaefb3] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-po

Interactive Elixir (1.0.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> h Foo

                                    Foo

Provides Foo related functions

Examples

|  iex > Foo.bar
|  "hello"

iex(2)> h Foo.bar

                                  def bar()

Evals the string "hello"

iex(3)>
```

documentation & tests

```elixir
defmodule FooTest do
  use ExUnit.Case, async: true
  doctest Foo
end
```

```elixir
ExUnit.start
```

demo time

murphy time...

murphy time...

http://polar-mountain-5415.herokuapp.com/

resources

learning more

- **Programming Elixir by Dave Thomas**
- **The Little Elixir and OTP Guidebook**
- **Seven More Languages in Seven Weeks**
- **Elixir Sips**

projects

- **elixir/plug**
- **phoenixframework/phoenix**
- **elixir-lang/ecto**
- **HashNuke/hound**
- **parroty/excheck**
- **BennyHallet/obelisk**