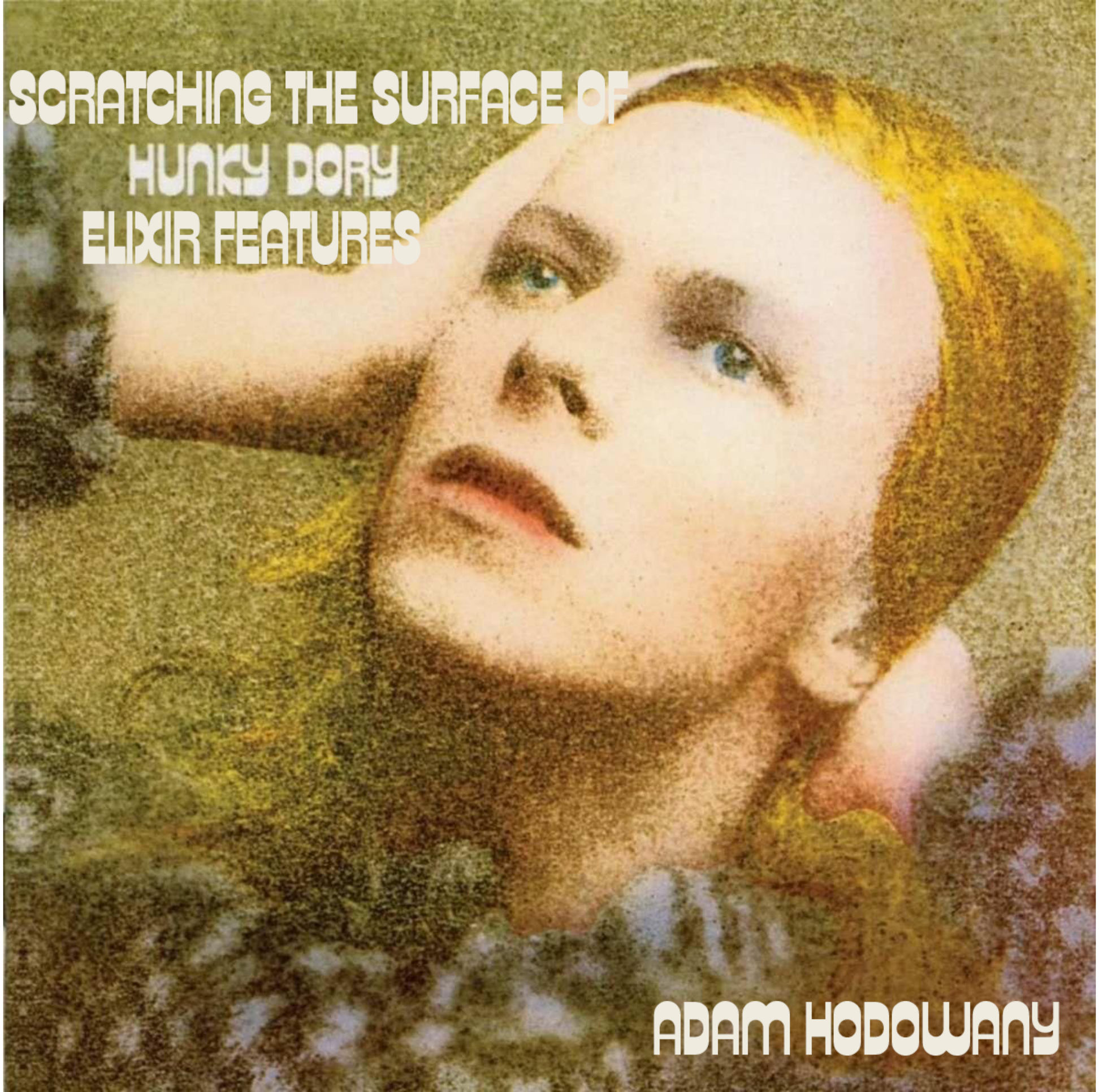
SCRATCHING THE SURFACE OF HUNKY DORY ELIXIR FEATURES



ADAM HODOWANY

ABOUT ME

Adam Hodowany

hodak.pl

@adhodak

Elixir Dev at Recruitee


WHAT

CONTROL FLOW

if / else

```
if hour < 18 do
  IO.puts("G'day mate")
else
  IO.puts("Hooroo")
end
```

"ternary" operator

```
if hour < 18, do: "G'day", else: "Hooroo"
```

else if


IF
ELSE
ELSE IF
ELSE
imgflip.com

else if

```
time_of_day =
  if hour < 13 do
    "Morning"
  else
    if hour < 18 do
      "Afternoon"
    else
      if hour < 21 do
        "Evening"
      else
        "Night"
      end
    end
  end
```

cond

```
cond do
  hour < 13 ->
    "Morning"
  hour < 18 ->
    "Afternoon"
  hour < 21 ->
    "Evening"
  true ->
    "Night"
end
```

```
hour = 22

cond do
  hour < 13 ->
    "Morning"
  hour < 18 ->
    "Afternoon"
  hour < 21 ->
    "Evening"
end
```

```
** (CondClauseError) no cond clause evaluated to a true value
```

pattern matching

```
def get_port, do: 443
```

```
port = get_port()
443 = port
```

```
def get_port, do: 443
port = get_port()
```

```
iex(13)> 443 = port
443
iex(14)> 444 = port
** (MatchError) no match of right hand side value: 443
```

```
iex(30)> %{port: port} = URI.parse("https://onet.pl")
%URI{
  authority: "onet.pl",
  fragment: nil,
  host: "onet.pl",
  path: nil,
  port: 443,
  query: nil,
  scheme: "https",
  userinfo: nil
}
iex(31)> port
443
```

Me: what is an array?

Elixir:

Lists

```
iex(37)> [1, 2, 3]
[1, 2, 3]
iex(38)> [1 | [2 | [3 | []]]]
[1, 2, 3]
iex(39)> [1, 2, 3] == [1 | [2 | [3 | []]]]
true
```

```
iex(40)> [head | tail] = [1, 2, 3]
[1, 2, 3]
iex(41)> head
1
iex(42)> tail
[2, 3]
```

```
empty_list = []
one_element_list = [1]
two_elements_list = [1, 2]
```

```
iex(49)> [_] = empty_list
** (MatchError) no match of right hand side value: []

iex(49)> [_] = one_element_list
[1]
iex(50)> [_] = two_elements_list
** (MatchError) no match of right hand side value: [1, 2]
```

How to match a non-empty list?

```
iex(53)> [_ | _] = one_element_list
[1]
iex(54)> [_ | _] = two_elements_list
[1, 2]
iex(55)> [_ | _] = empty_list
** (MatchError) no match of right hand side value: []
```

case

```
case Repo.insert(changeset) do
  {:ok, user} ->
    render_user(user)

  {:error, error} ->
    render_error(error)
end
```

```elixir
case URI.parse("https://onet.pl") do
  %{port: 443} ->
    IO.puts("Safe!")
  %{port: 80} ->
    IO.puts("Not safe")
end
```

```elixir
case URI.parse("https://onet.pl:4000") do
  %{port: 443} ->
    IO.puts("Safe!")

  %{port: 80} ->
    IO.puts("Not safe")
end
```

```
** (CaseClauseError) no case clause matching: %URI{authority: "onet.pl:4000", fragment: nil, host: "onet.pl", path: nil
, port: 4000, query: nil, scheme: "https", userinfo: nil}
```

```
case URI.parse("https://onet.pl:4000") do
  %{port: 443} ->
    IO.puts("Safe!")
  _ ->
    IO.puts("Not safe")
end
```

```
case URI.parse("https://onet.pl:4000") do
  %{port: 443} ->
    IO.puts("Safe!")
  port ->
    IO.puts("Port #{port} is not safe")
end
```

pattern matching with functions

```
def safe_uri?(%{port: 443}), do: true

def safe_uri?(_), do: false

safe_uri?(URI.parse("https://onet.pl"))
```

guards

```
def time_of_day(hour) when hour < 13, do:
  "Morning"
def time_of_day(hour) when hour < 18, do:
  "Afternoon"
def time_of_day(hour) when hour < 21, do:
  "Evening"
def time_of_day(_hour), do:
  "Night"

"Good #{time_of_day(DateTime.utc_now().hour)}"
```

pipe operator

```
foo(bar(baz(new(other()))))
```

```
other() |> new() |> baz() |> bar() |> foo()
```

```
other()
|> new()
|> baz()
|> bar()
|> foo()
```

```
Enum.reverse(
  String.split(
    String.upcase("hunky dory")
  )
)
```

```
"hunky dory"
|> String.upcase()
|> String.split()
|> Enum.reverse()
```

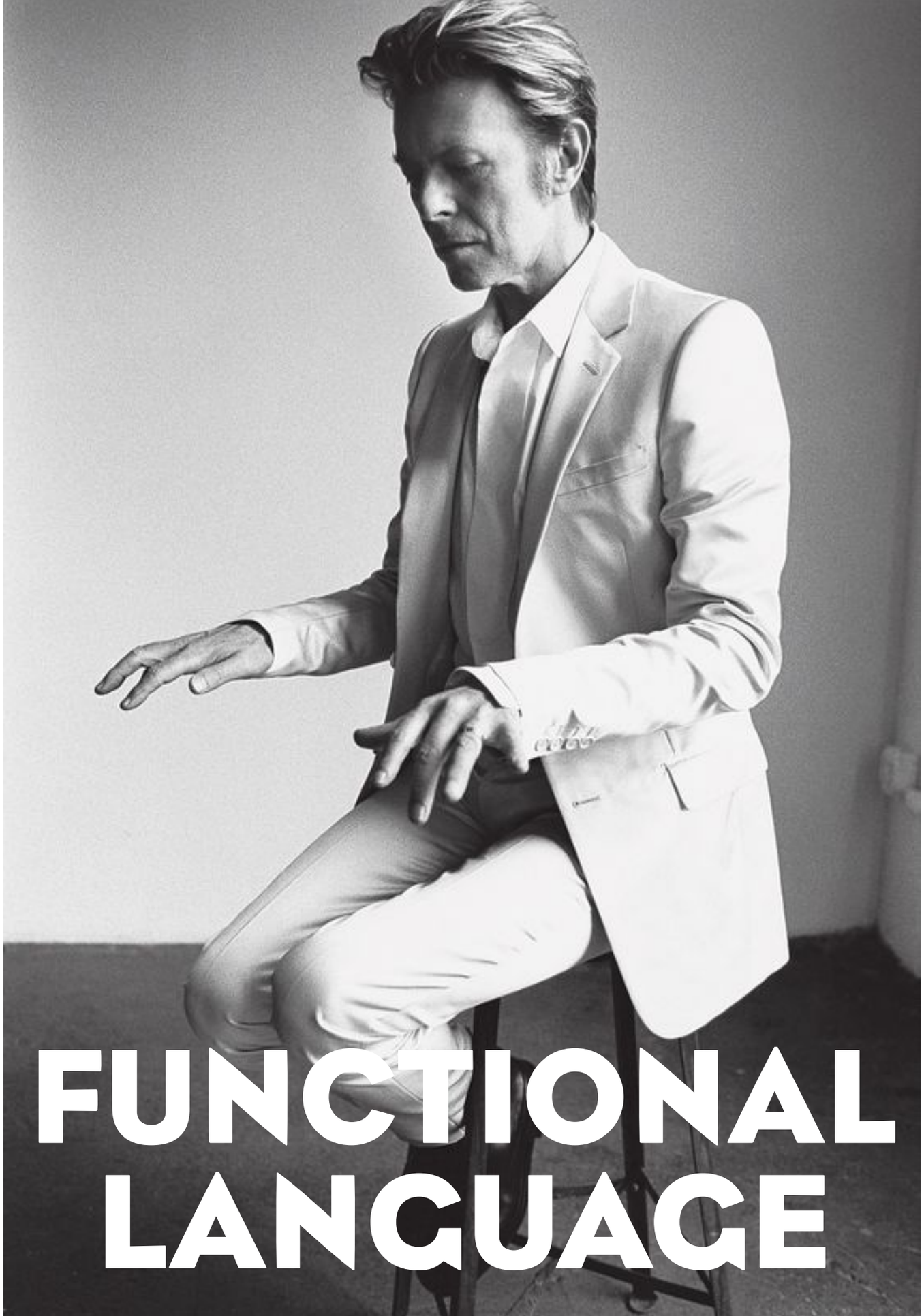
```
iex(60)> "hunky dory" |> String.upcase() |> String.split() |> Enum.reverse()
["DORY", "HUNKY"]
```

```
File.read("text.txt")
|> extract_email_addresses()
|> Enum.at(0)
|> User.find_by_email()
|> notify()
```

```
case File.read("text.txt") do
  {:ok, text} ->
    case extract_email_addresses(text) do
      [email | _] ->
        case Repo.get_by(User, email: email) do
          %User{} = user ->
            notify(user)
            :ok
          nil ->
            :error
        end
      [] ->
        :error
    end
  _ ->
    :error
end
```

with statement

```
with {:ok, text} <- File.read("text.txt"),
     [email | _] <- extract_email_addresses(text),
     %User{} = user <- Repo.get_by(email: email) do
  notify(user)
else
  :error
end
```


FUNCTIONAL
LANGUAGE

```
fn ->
  IO.puts("I'm inside a function")
end
```

```
iex(3)> fn -> IO.puts("I'm inside a function") end.()
I'm inside a function
:ok
```

```
yell = fn wat ->
  IO.puts(String.upcase(wat))
end
```

```
iex(7)> ["a", "b", "c"] |> Enum.each(yell)
A
B
C
:ok
```

```
defmodule Utils do
  def yell(wat) do
    IO.puts(String.upcase(wat))
  end
end
```

```
iex(13)> Utils.yell
iex(13)> &Utils.yell
** (CompileError) iex:13:
 capture containing at le
    Utils.yell()
```

```
iex(14)> yell = &Utils.yell/1
&Utils.yell/1
iex(15)> yell.("a")
A
:ok
iex(16)> ["a", "b", "c"] |> Enum.each(yell)
A
B
C
:ok
```

```
/arity, &local/arity or a
```

```
["a", "b", "c"] |> Enum.each(&Utils.yell/1)
```

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

```
[1, 2, 3] |> Enum.map(&(&1 * 2))
```

```
iex(20)> [1, 2, 3] |> Enum.map(&(&1 * 2))
[2, 4, 6]
```

```
[
  %{id: 1, active: true},
  %{id: 2, active: false},
  %{id: 3, active: true}
]
|> Enum.filter(& &1.active)
```

```
[%{active: true, id: 1}, %{active: true, id: 3}]
```

```
[
  %{id: 1, active: true},
  %{id: 2, active: false},
  %{id: 3, active: true}
]
|> Enum.filter(fn
  %{active: true} ->
    true

  _ ->
    false
end)
```


PROCESSES

```
spawn(fn ->
  IO.puts("I'm inside a process")
end)
```

```
[1, 2, 3, 4, 5]
|> Enum.each(
  &spawn(fn ->
    :timer.sleep(:rand.uniform(1000))
    IO.puts(&1)
  end)
)
```

```
:ok
2
4
5
1
3
```

```
[
  "https://www.onet.pl",
  "https://www.wp.pl",
  "https://www.o2.pl"
]
|> Enum.map(&Tesla.get(&1).body)
```

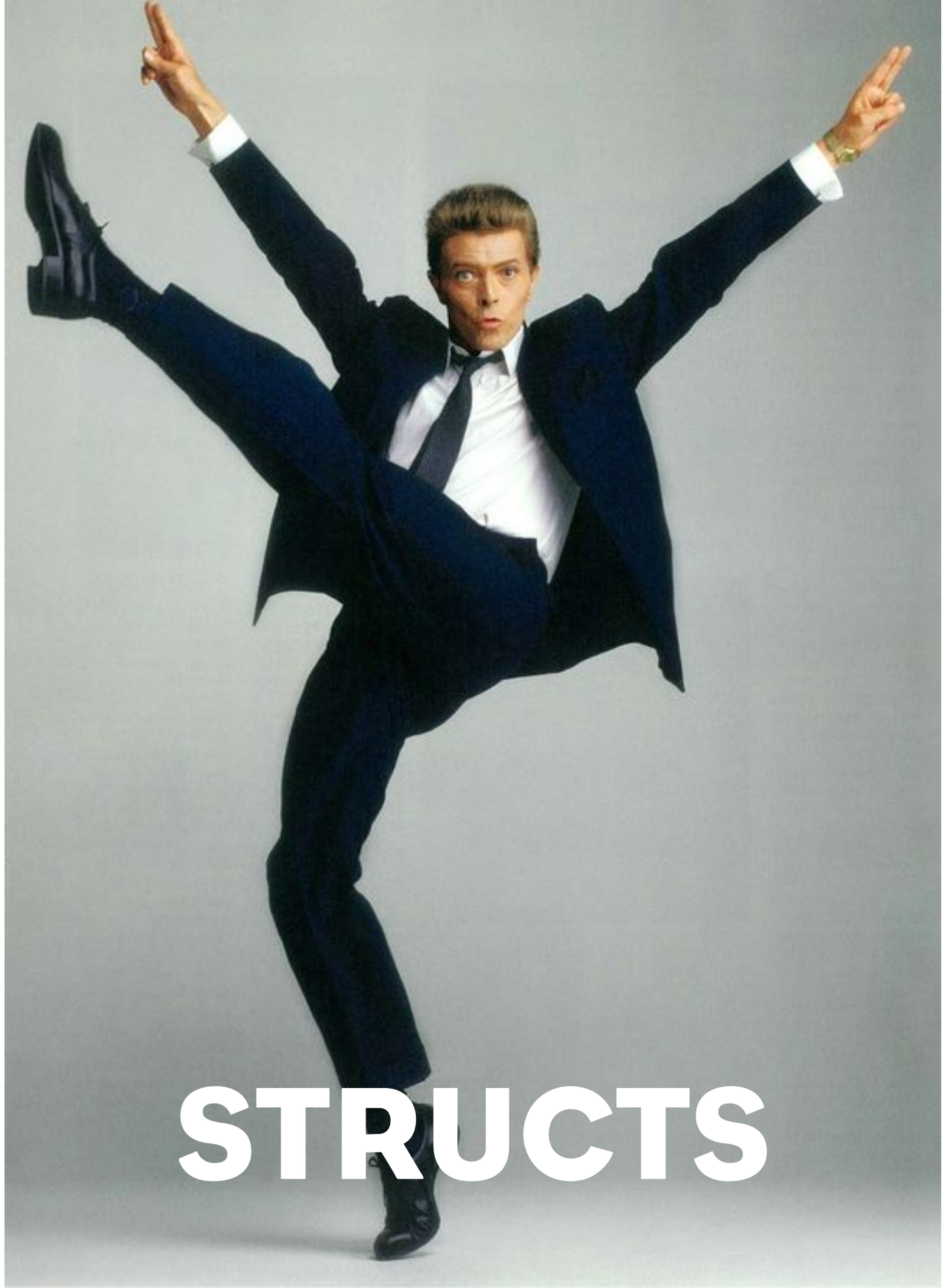
```
[
  "https://www.onet.pl",
  "https://www.wp.pl",
  "https://www.o2.pl"
]
|> Enum.map(&Task.async(fn ->
  Tesla.get(&1).body
end))
|> Enum.map(&Task.await/1)
```

```
:timer.tc(fn ->
  urls |> Enum.map(&Tesla.get(&1))
end)
```

```
{3305784,
```

```
:timer.tc(fn ->
  urls
  |> Enum.map(
    &Task.async(fn ->
      Tesla.get(&1).body
    end)
  )
  |> Enum.map(&Task.await/1)
end)
```

```
{920455,
```


STRUCTS

```
user = %{
  id: 1,
  email: "david@bowie.com",
  name: "David",
  age: 69
}
```

```
user = %{
  id: 1,
  email: "david@bowie.com",
  name: "David",
  age: 69,
  lord_of_winterfell: true
}
```


GOT
HBO

```
defmodule User do
  defstruct [:id, :email, :name, :age]
end

user = %User{
  id: 1,
  email: "david@bowie.com",
  name: "David",
  age: 69
}
```

STRUCTS

```
iex(68)> user = %User{
...(68)>   id: 1,
...(68)>   email: "david@bowie.com",
...(68)>   name: "David",
...(68)>   age: 69,
...(68)>   lord_of_winterfell: true
...(68)> }
** (KeyError) key :lord_of_winterfell not found in: %User{age: 69, email: "david@bowie.com", id: 1, name: "David"}
    (stdlib) :maps.update(:lord_of_winterfell, true, %User{age: 69, email: "david@bowie.com", id: 1, name: "David"})
    iex:67: anonymous fn/2 in User.__struct__/1
    (elixir) lib/enum.ex:1899: Enum."-reduce/3-lists^foldl/2-0-"/3
```

```elixir
defmodule User do
  @enforce_keys [:id, :email]
  defstruct [:id, :email, :name, :age]
end
```

```
iex(70)> %User{id: 2, name: "Jon", age: 18}
** (ArgumentError) the following keys must also be given when building struct User: [:email]
    expanding struct: User.__struct__/1
    iex:70: (file)
```

```elixir
defmodule User do
  @enforce_keys [:id]
  defstruct [:id, :first_name, :last_name]

  def full_name(%User{} = user) do
    [user.first_name, user.last_name]
    |> Enum.reject(&is_nil/1)
    |> List.join(" ")
  end
end
```

```
iex(77)> %User{first_name: "Jon", last_name: "Snow"} |> User.full_name()
"Jon Snow"
iex(78)> %{first_name: "Jon", last_name: "Snow"} |> User.full_name()
** (FunctionClauseError) no function clause matching in User.full_name/1

    The following arguments were given to User.full_name/1:

        # 1
        %{first_name: "Jon", last_name: "Snow"}

    iex:78: User.full_name/1
```

```
defmodule HunkyDory.User do
  use Ecto.Schema
  import Ecto.Changeset

  schema "users" do
    field(:first_name, :string)
    field(:last_name, :string)
    field(:email, :string)
    field(:age, :integer)

    timestamps()
  end
end
```

STRUCTS

hos > hunky_dory_dev > users Connected.

id	first_name	last_name	email	age	inserted_at	updated_at
1	David	Bowie	david@bowie.com	69	2019-02-25 20:11:46.483622	2019-02-25 20:11:46.483622
2	Jon	Snow	NULL	18	2019-02-25 20:11:46.483622	2019-02-25 20:11:46.483622

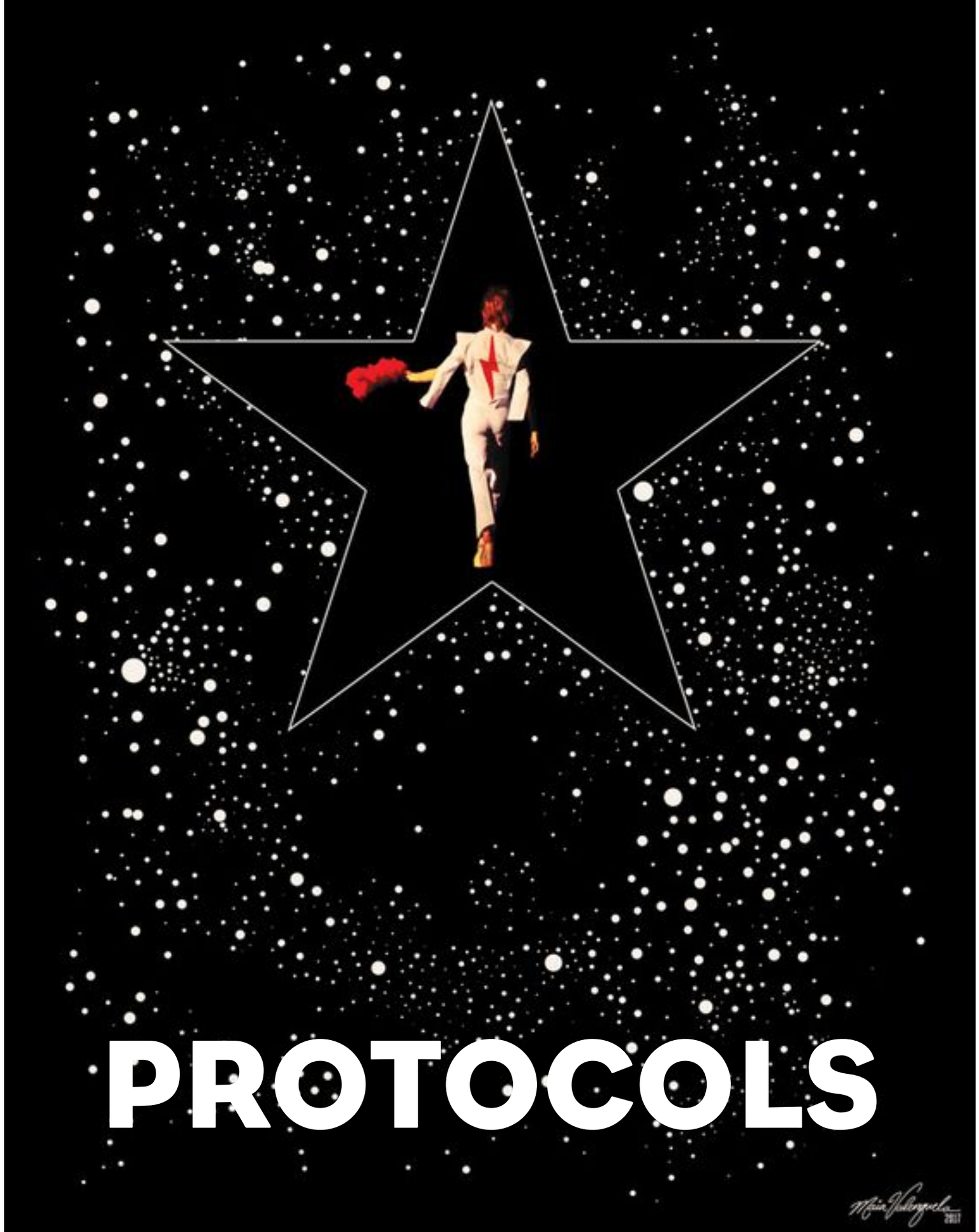
```
iex(3)> %HunkyDory.User{}
%HunkyDory.User{
  __meta__: #Ecto.Schema.Metadata<:built, "users">,
  age: nil,
  email: nil,
  first_name: nil,
  id: nil,
  inserted_at: nil,
  last_name: nil,
  updated_at: nil
}
```

STRUCTS

hos | hunky_dory_dev | users | Connected.

id	first_name	last_name	email	age	inserted_at	updated_at
1	David	Bowie	david@bowie.com	69	2019-02-25 20:11:46.483622	2019-02-25 20:11:46.483622
2	Jon	Snow	NULL	18	2019-02-25 20:11:46.483622	2019-02-25 20:11:46.483622

```
iex(5)> Repo.get(HunkyDory.User, 1)
[debug] QUERY OK source="users" db=8.0ms
SELECT u0."id", u0."age", u0."email", u0."first_name", u0."last_name", u0."inserted_at", u0."updated_at" FROM "users"
AS u0 WHERE (u0."id" = $1) [1]
%HunkyDory.User{
  __meta__: #Ecto.Schema.Metadata<:loaded, "users">,
  age: 69,
  email: "david@bowie.com",
  first_name: "David",
  id: 1,
  inserted_at: ~N[2019-02-25 20:11:46.483622],
  last_name: "Bowie",
  updated_at: ~N[2019-02-25 20:11:46.483622]
}
```


PROTOCOLS

POLYMORPHISM

in JAVA programming

```
defprotocol Size do
  def size(data)
end

    defimpl Size, for: BitString do
      def size(string), do: byte_size(string)
    end

    defimpl Size, for: Map do
      def size(map), do: map_size(map)
    end

    defimpl Size, for: Tuple do
      def size(tuple), do: tuple_size(tuple)
    end
```

```
iex(11)> Size.size("foo")
3
iex(12)> Size.size({:ok, "hello"})
2
iex(13)> Size.size(%{label: "some label"})
1
```

```
iex(10)> Size.size([1, 2, 3])
** (Protocol.UndefinedError) protocol Size not implemented for [1, 2, 3]
    iex:1: Size.impl_for!/1
    iex:2: Size.size/1
```

```
defmodule User do
  defstruct [:first_name, :last_name]
end

defmodule Admin do
  defstruct [:name]
end


defprotocol DisplayName do
  def full(entity)
end
```

```
defimpl DisplayName, for: User do
  def full(user) do
    "#{user.first_name} #{user.last_name}"
  end
end

defimpl DisplayName, for: Admin do
  def full(admin), do: admin.name
end
```

```
iex(25)> DisplayName.full(%User{first_name: "Jon", last_name: "Snow"})
"Jon Snow"
iex(26)> DisplayName.full(%Admin{name: "Ziggy"})
"Ziggy"
```

THANK YOU.