From OOP to FP

The journey

cabify

Hi!

Iñaki Quesada
@piedresybarro

cabify

github.com/iquesada/from-oop-to-fp

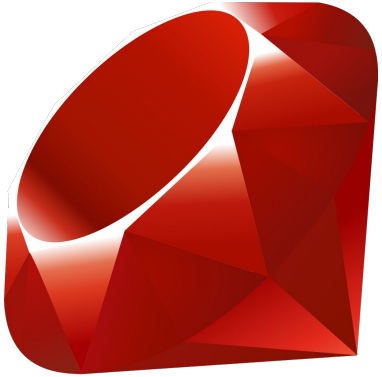
cabify

Smart enough?

Is anyone going to pay me?

cabify

Object oriented programming



cabify

cabify ❤ iñaki

cabify

cabify

cabify

cabify

“A programming paradigm is a philosophy, style, or general approach to writing code”

cabify

Basic concepts

- **Declarative programming**
- Functions as first-class citizens
- High-order functions
- Pure functions - Idempotent
- Immutability - Stateless
- Pattern-matching rocks!

Declarative vs imperative

What vs How

Imperative

JS - OOP

```
invoices = Invoices.fetch_all()
paid_invoices = []

for(i = 0; i < invoices.length; i++) {
    if (invoices[i].paid) paid_invoices.push(invoices[i])
}
```

Declarative

Elixir - FP

```
invoices = Invoices.fetch_all()

paid_invoices = Enum.filter(invoices, fn invoice -> invoice.paid end)
```

Imperative

JS - OOP

```
invoices = Invoices.fetch_all()
paid_invoices = []

for(i = 0; i < invoices.length; i++) {
    if (invoices[i].paid) paid_invoices.push(invoices[i])
}
```

Declarative vs imperative

Order?

cabify

Declarative vs imperative

Sequentially?

Declarative vs imperative

How build return value?

Declarative vs imperative

SRP

cabify

Declarative vs imperative

Don’t reveal implementation details

Basic concepts

- Declarative programming
- **Functions as first-class citizens**
- High-order functions
- Pure functions - Idempotent
- Immutability - Stateless
- Pattern-matching rocks!

Can be passed as an argument

Elixir - FP

```
def is_available?(car) do
  car.free_seats > 0
end

available_cars = Enum.filter(cars, &is_available?/1)
```

Can be returned from a function

Elixir - FP

```
def is_available do
  fn car -> car.free_seats > 0 end
end
```

Can be assigned to a variable

Elixir - FP

```
car_available = is_available()

car_available.(car)
```

Basic concepts

- Declarative programming
- Functions as first-class citizens
- **High-order functions**
- Pure functions - Idempotent
- Immutability - Stateless
- Pattern-matching rocks!

Higher-order functions

Elixir - FP

```
def my_func(x), do: x*2

Enum.map([0, 1, 2], &my_func/1)
```

Higher-order functions

Elixir - FP

```
def my_func(x), do: x*2

Enum.map([0, 1, 2], &my_func/1)
```

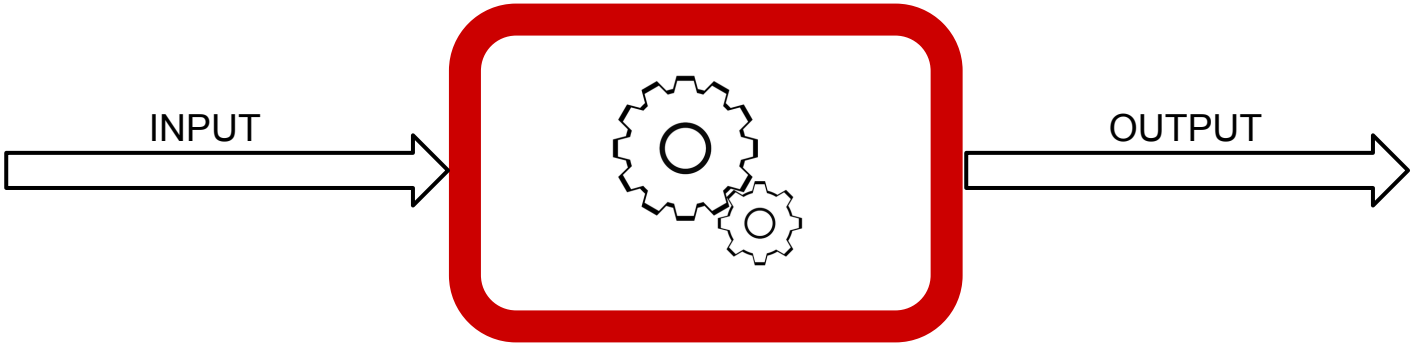
Basic concepts

- Declarative programming
- Functions as first-class citizens
- High-order functions
- **Pure functions - Idempotent**
- Immutability - Stateless
- Pattern-matching rocks!

Pure functions - Idempotent

Same output from the same input

Pure functions - Idempotent



cabify

Pure functions - Idempotent

No side effects

Pure functions - Idempotent

Parallelizable

Pure functions - Idempotent

cabify

Pure functions - Idempotent

Referentially transparent

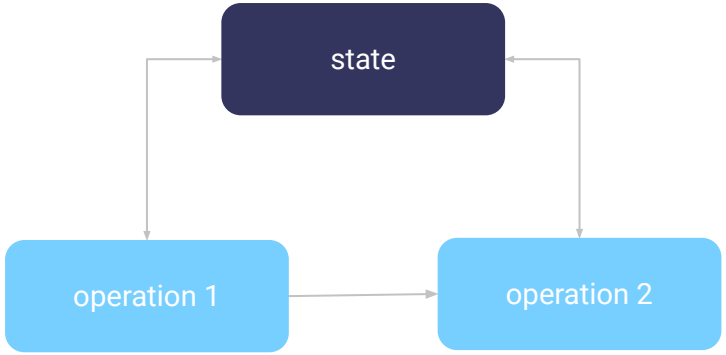
Basic concepts

- Declarative programming
- Functions as first-class citizens
- High-order functions
- Pure functions - Idempotent
- **Immutability - Stateless**
- Pattern-matching rocks!

Stateful

JavaScript - OO

```
cart = new Cart()
cart.add_product("voucher")
cart.add_product("tshirt")
total = cart.total()
```

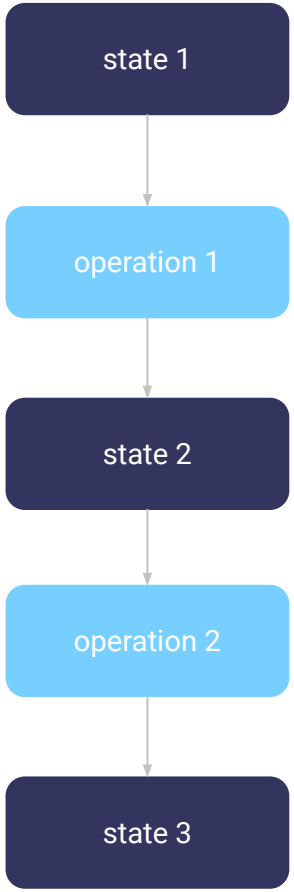


cabify

Stateless

Elixir - FP

```
total =
 cart
 |> Cart.add_product("voucher")
 |> Cart.add_product("tshirt")
 |> Cart.total()
```



cabify

Immutability - Stateless

~~variables~~ value labels

Immutability - Stateless

Avoid side effects

Immutability - Stateless

Easier to test

Recap

No state?

cabify

And yes, we use databases!!

cabify

Basic concepts

- Declarative programming
- Functions as first-class citizens
- High-order functions
- Pure functions - Idempotent
- Immutability - Stateless
- **Pattern-matching rocks!**

Pattern-matching rocks!!

Let’s code!

Pattern-matching rocks!!

Elixir - FP

```
iex> x = 1
iex> 1 = x
Ok
iex> 2 = x
Error
iex> [a, b, c] = [1, 2, 3] # a = 1, b = 2, c = 3

iex> {_, message} = {:ok, "Hello"} # message="Hello"
```

Pattern-matching rocks!!

Ruby - NO PM

```
def display_name(user)
  if user.first_name.length > 0
    if user.last_name.length > 0
      "#{user.first_name} #{user.last_name}".strip
    else
      "#{user.first_name}".strip
    end
  elsif user.username.length > 0
    user.username
  else
    “New User”
  end
end
```

Pattern-matching rocks!!

Elixir - PM

```
def display_name(%{first_name: first, last_name: last}) do
    String.trim("#{first} #{last}")
end
```

Pattern-matching rocks!!

Elixir - PM

```
def display_name(%{first_name: first, last_name: last}) do
    String.trim("#{first} #{last}")
end

def display_name(%{first_name: first}) do
    String.trim("#{first}")
end
```

Pattern-matching rocks!!

Elixir - PM

```
def display_name(%{first_name: first, last_name: last}) do
    String.trim("#{first} #{last}")
end

def display_name(%{first_name: first}) do
    String.trim("#{first}")
end

def display_name(%{username: username}), do: "#{username}"
```

Pattern-matching rocks!!

Elixir - PM

```
def display_name(%{first_name: first, last_name: last}) do
    String.trim("#{first} #{last}")
end

def display_name(%{first_name: first}) do
    String.trim("#{first}")
end

def display_name(%{username: username}), do: "#{username}"

def display_name(_), do: “New User”
```

Quick recap

Declarative programming

Functions as first-class citizens

High-order functions

Pure functions - Idempotent

Immutability - Stateless

Pattern-matching rocks!

Quick recap

~~Declarative programming~~	Don’t iterate (map, reduce…)
~~Functions as first class citizens~~	Function as a value
~~High order functions~~	Functions accepts functions
~~Pure functions - Idempotent~~	Functions without side effects
~~Immutability - Stateless~~	Data doesn’t change
~~Pattern matching rocks!~~	Yeah, it rocks

Your turn

~~OOP~~

Design patterns

SOLID

cabify


goto;
copenhagen
GOTO Copenhagen 2018
Conference Nov. 19 - 21
SOLID Elixir
Georgina McFadyen
Follow us @gotocph

Language?

But I’m frontend!

cabify


JSUnconf 2016
Learning Functional
Programming with JS
@AnjanaVakil
JS
UNCONF

JS libraries

- Ramdajs - https://ramdajs.com
- Immutable-js - https://github.com/immutable-js/immutable-js

Ramda

```
const R = require('ramda');

const a = [4, 5, 6];

const b = R.concat(a, [7, 8, 9]); // Instead of: a.push(7, 8, 9);

const g = R.reverse(a); // instead a.reverse()

const double = x => x * 2;

R.map(double, [1, 2, 3]); // [2, 4, 6]
```

Immutable-js

```
const list1 = Immutable.List(['A', 'B', 'C']);

const list2 = list1.push('D', 'E');

console.log([...list1]); // ['A', 'B', 'C']

const map1 = Immutable.Map([[1, “one”], [2, “two”]]);

const map2 = map1.set('four', 4);

console.log([...map1]); // [['one', 1], ['two', 2], ['three', 3]]

console.log([...map2]); // [['one', 1], ['two', 2], ['three', 3],
['four', 4]]
```

Try ELM!

cabify

https://martinsstewart.gitlab.io/hackman/

cabify

Smart enough?

I’m with

stupid

cabify

Is anyone to pay me?

https://elixir-companies.com/en

OOP > FP

cabify

FP > OOP

cabify

“Learn at least one new language every year”

The Pragmatic Programmer - Andrew Hunt & David Thomas

Resources

https://rachelcarmena.github.io/2019/08/05/functional-programming-sparks-joy.html

Solid Elixir - https://www.youtube.com/watch?v=rmftOs2BzgU

FP with JS - https://www.youtube.com/watch?v=e-5obm1G_FY

Let's get functional with Elixir - https://www.youtube.com/watch?v=wVrnoxNbOts

ELM game - https://martinsstewart.gitlab.io/hackman/

https://medium.com/aviabird/10-amazing-open-source-elixir-phoenix-apps-e2c52ee25053

Thanks!!

Questions?

Iñaki Quesada
@piedresybarro