Muito mais do que map, filter e reduce

Olar bebês!

Sou Eduardo Mendes (@dunossauro)

Apto fazedor de lambdas, pythonista, apaixonado por software **livre** e ciência.

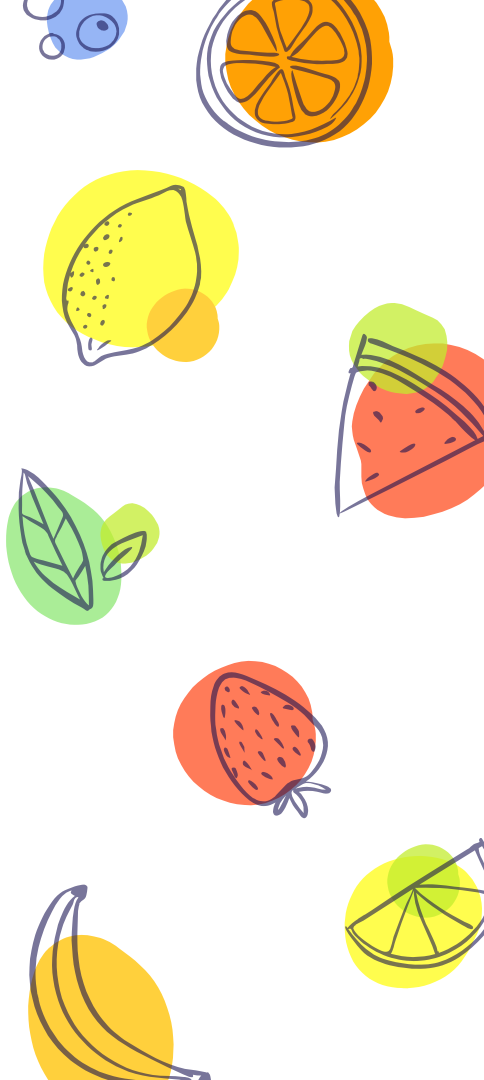
Qual o verdadeiro FP?

Nutella

- Usa libs prontas
- Não sabe matemática
- Só conhece map, filter e reduce

Raiz

- Domina categorias
- Usa tipos de dados algébricos
- Acredita no poder das monads



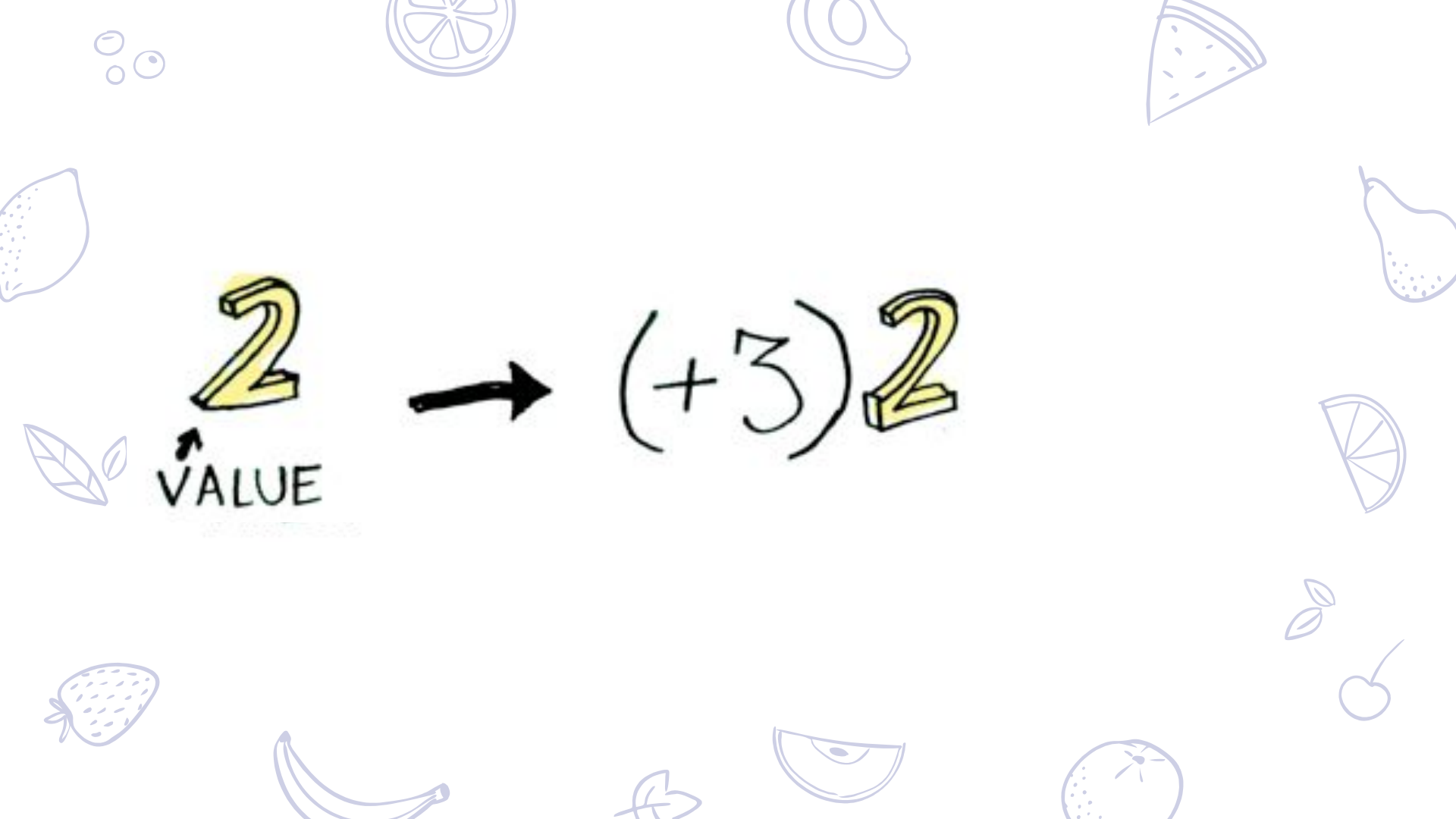
Tá, mas e dai?

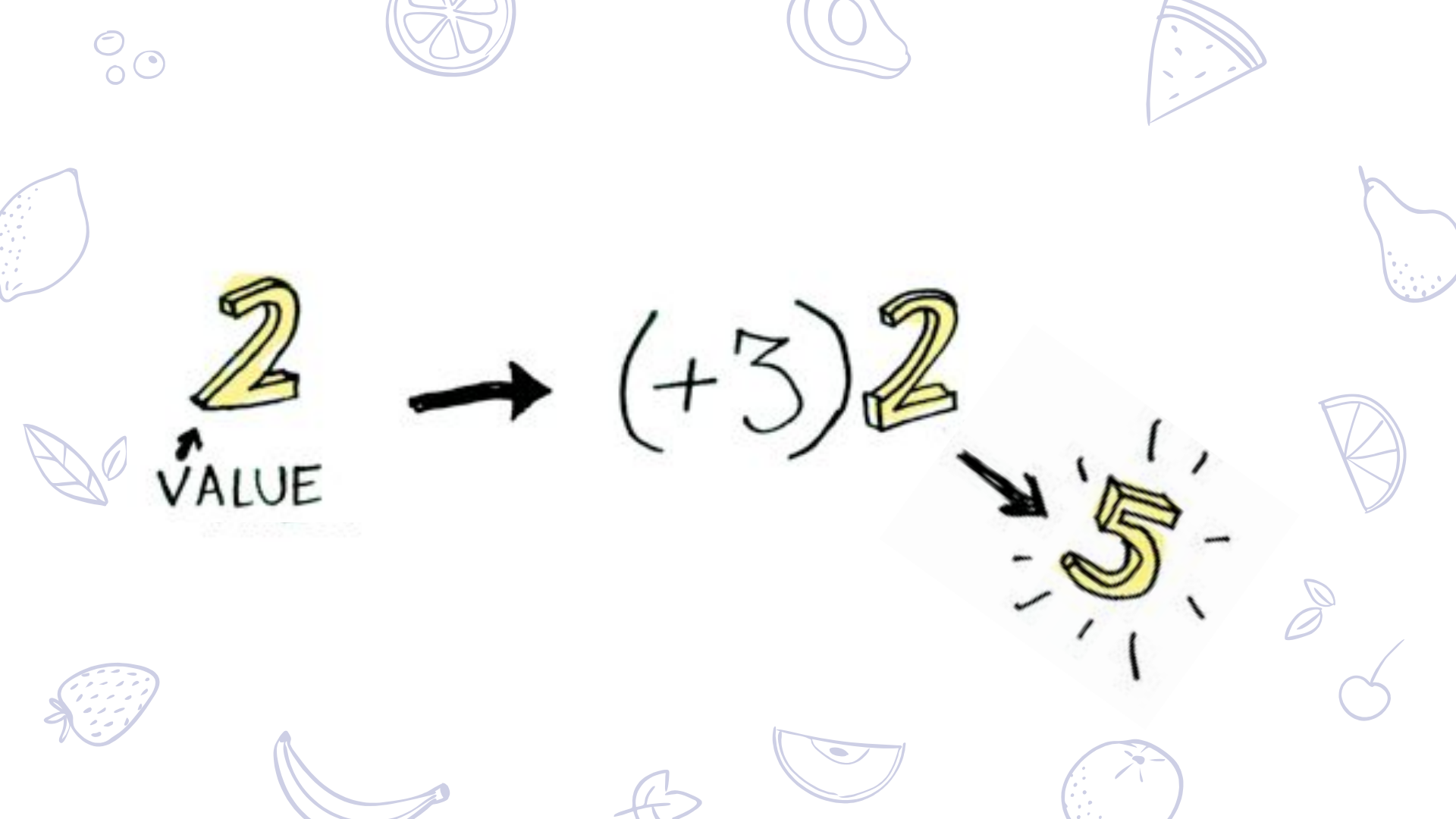
- Existem coisas que você sabe que sabe
- Coisas que sabe que não sabe
- Coisas que não sabe que não sabe

Aditya Bhargava

Thanks for the drawings, we love it <3


2
VALUE


2
VALUE
(+3)2


2
VALUE
(+3)2
5

```
In [1]: valor = 2

In [2]: def soma_3(valor):
   ...:     return valor + 3
   ...:
   ...:

In [3]: soma_3(valor)
Out[3]: 5
```

Just 2

VALUE AND CONTEXT

```
In [1]: valor = 2

In [2]: contexto = [valor]

In [3]: def soma_3(valor):
   ...:     return valor + 3
   ...:
   ...:

In [4]: soma_3(contexto)
```

```
In [4]: soma_3(contexto)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-efe19674dcf3> in <module>()
----> 1 soma_3(contexto)

<ipython-input-3-50501a3777ed> in soma_3(valor)
      1 def soma_3(valor):
----> 2     return valor + 3

TypeError: can only concatenate list (not "int") to list
```

OUCH!

```
In [1]: valor = 2

In [2]: contexto = [valor]

In [3]: def soma_3(valor):
   ...:     if isinstance(valor, list):
   ...:         valor = valor[0]
   ...:     return valor + 3
   ...:
   ...:

In [4]: soma_3(valor), soma_3(contexto)
Out[4]: (5, 5)
```

Ai você faz isso ...

Funtores

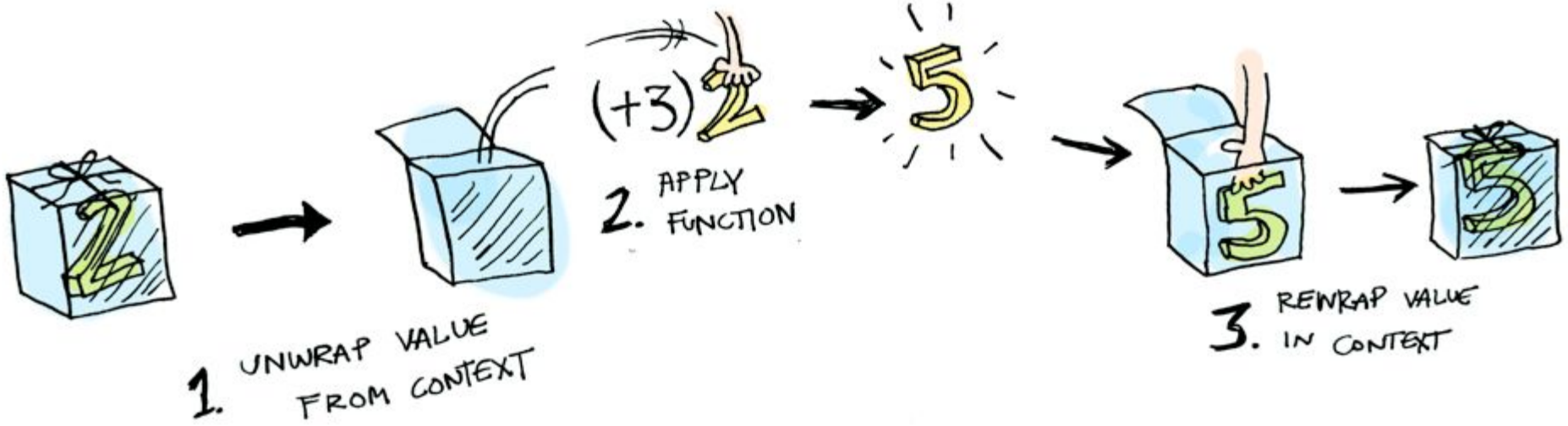
Functor, em Teoria das categorias, é um mapeamento entre categorias que preserva estruturas. Os functores podem ser entendidos como homomorfismos na categoria de todas as categorias pequenas (ou seja, a categoria que tem como objetos todas as categorias compostas por objetos que são conjuntos).

1. TO MAKE A DATA TYPE f A FUNCTOR,

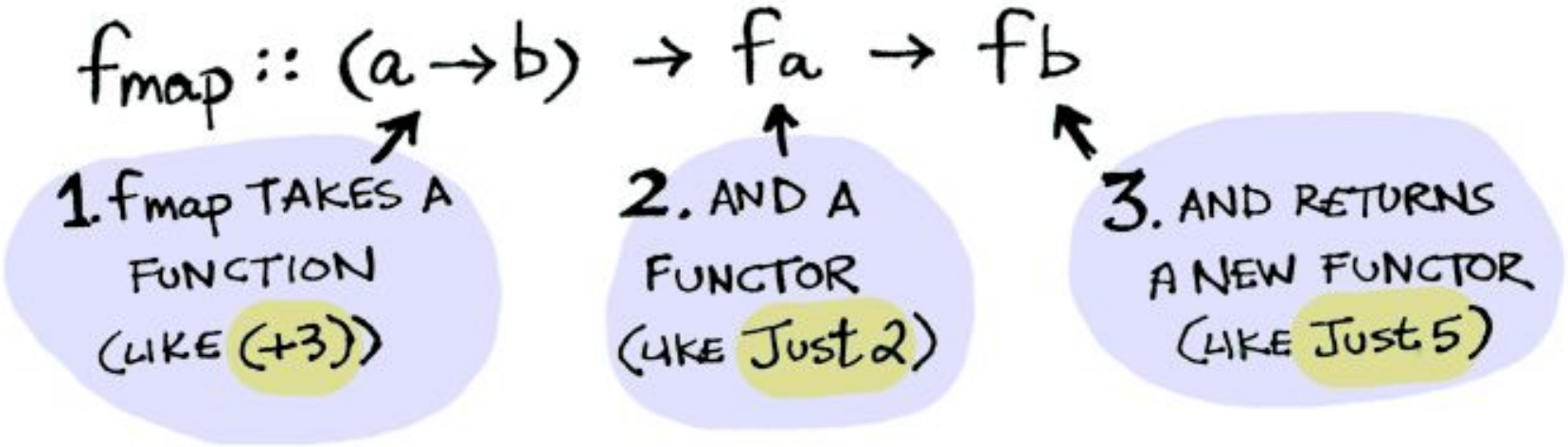
```
class Functor f where
  fmap :: (a -> b) -> f a -> f b
```

2. THAT DATA TYPE NEEDS TO DEFINE HOW fmap WILL WORK WITH IT.


fmap (+3)
2
SOME
MAGIC
HAPPENS
5


2
(+3)
2
5
5
2
1. UNWRAP VALUE FROM CONTEXT
2. APPLY FUNCTION
3. REWRAP VALUE IN CONTEXT

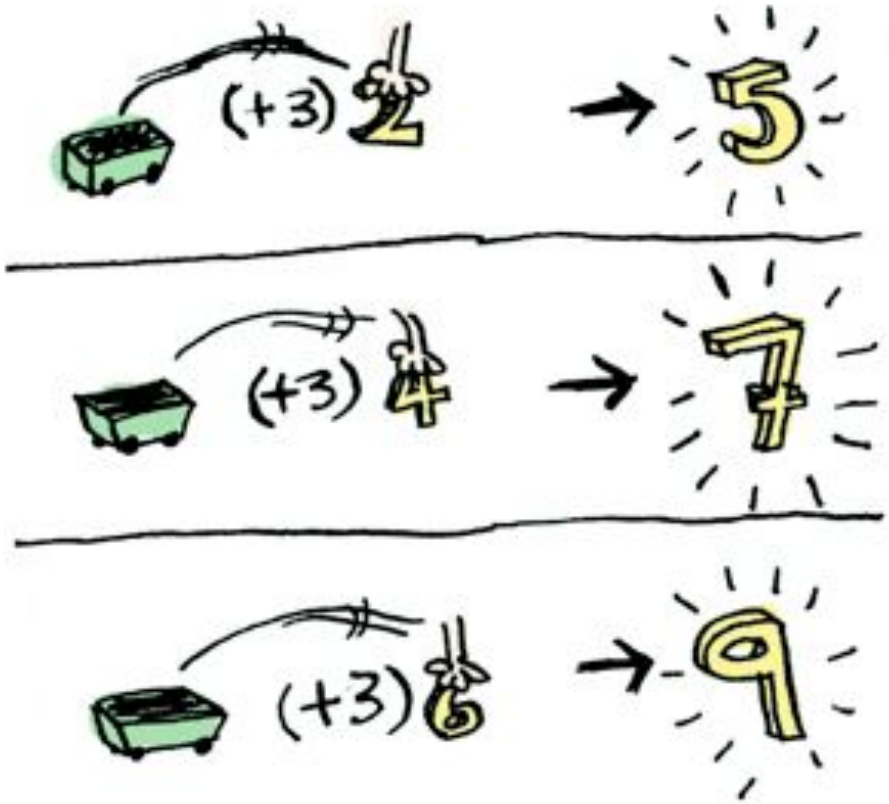
```
In [2]: class Just(int):
   ...:     def fmap(self, função):
   ...:         return função(self)
   ...:

In [3]: Just(2)
Out[3]: 2

In [4]: Just(2).fmap(soma_3)
Out[4]: 5
```



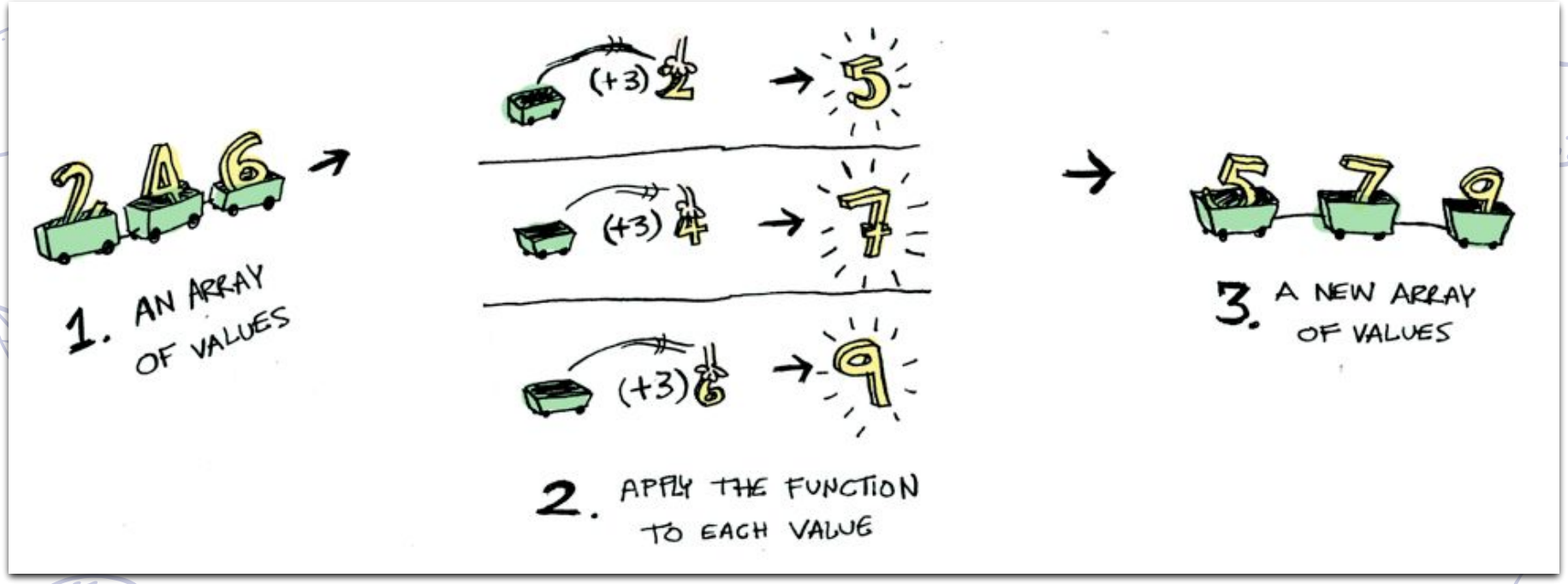
A regra é clara, Rogerinho

```
In [7]: class Just(int):
   ...:     def fmap(self: int, função: Callable) -> Just:
   ...:         return Just(função(self))
   ...:

In [8]: Just(2).fmap(soma_3).fmap(soma_3).fmap(soma_3)
Out[8]: 11
```


2 4 6


(+3) 2
5
(+3) 4
7
(+3) 6
9


5
7
9


2 4 6
1. AN ARRAY OF VALUES
(+3) 2
5
(+3) 4
7
(+3) 6
9
2. APPLY THE FUNCTION TO EACH VALUE
5 7 9
3. A NEW ARRAY OF VALUES

```
In [1]: from collections import UserList

In [2]: class List(UserList):
   ...:     def fmap(self, função):
   ...:         return List(map(função, self.data))
   ...:

In [3]: List([1,2,3,4]).fmap(lambda x: x + 1)
Out[3]: [2, 3, 4, 5]
```

```
In [9]: class List(UserList):
   ...:     def fmap(self, função):
   ...:         return List(map(função, self.data))
   ...:     def __mod__(self, função):
   ...:         return self.fmap(função)
   ...:
   ...:

In [10]: List([1,2,3,4]) % (lambda x: x + 1)
Out[10]: [2, 3, 4, 5]
```

Applicative functors

Just (+3)

Just 2

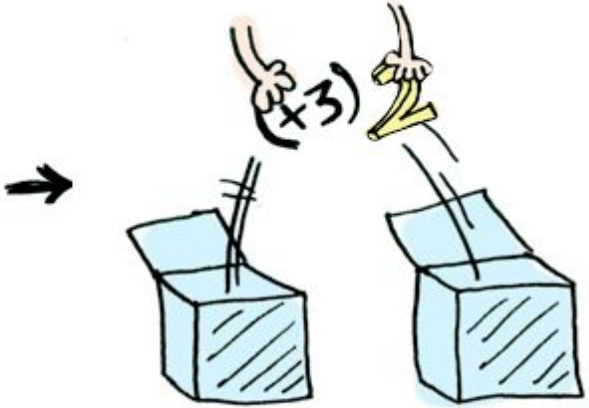
Just (+3)

<*>

Just 2

1. FUNCTION WRAPPED IN A CONTEXT

2. VALUE IN A CONTEXT

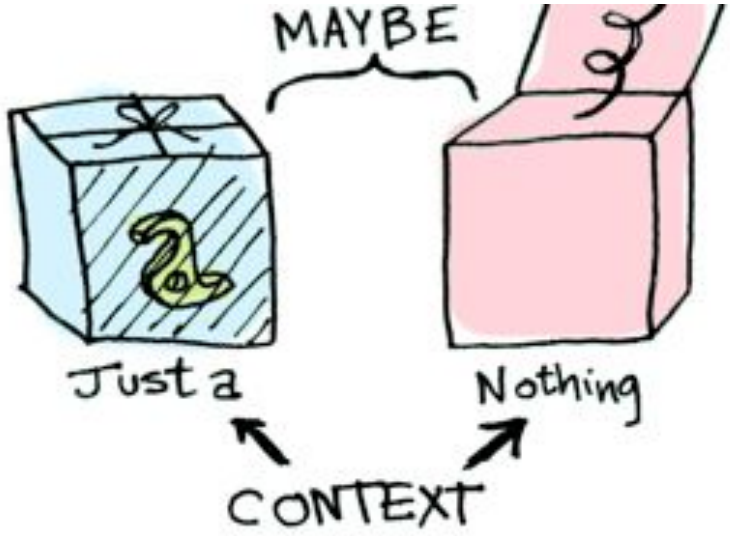


3. UNWRAP BOTH AND APPLY THE FUNCTION TO THE VALUE

4. NEW VALUE IN A CONTEXT

```
In [15]: class List(UserList):
    ...:     def fmap(self, função):
    ...:         return List(map(função, self.data))
    ...:     def __mod__(self, função):
    ...:         return self.fmap(função)
    ...:     def __mul__(self, fns):
    ...:         return [f(x) for f in self for x in fns]
    ...:

In [16]: List([lambda x: x + 1]) * List([1, 2, 3, 4])
Out[16]: [2, 3, 4, 5]
```

Monads


MAYBE
Just a
Nothing
CONTEXT

```
In [18]: class Just(int):
    ...:     def fmap(self, função):
    ...:         return Just(função(self))
    ...:
    ...:

In [19]: class Nothing:
    ...:     def fmap(self, função):
    ...:         return Nothing()
    ...:
```


```
In [72]: def metade(valor):
    ...:     if (valor % 2) == 0:
    ...:         return Just(valor //2)
    ...:     return Nothing()
```

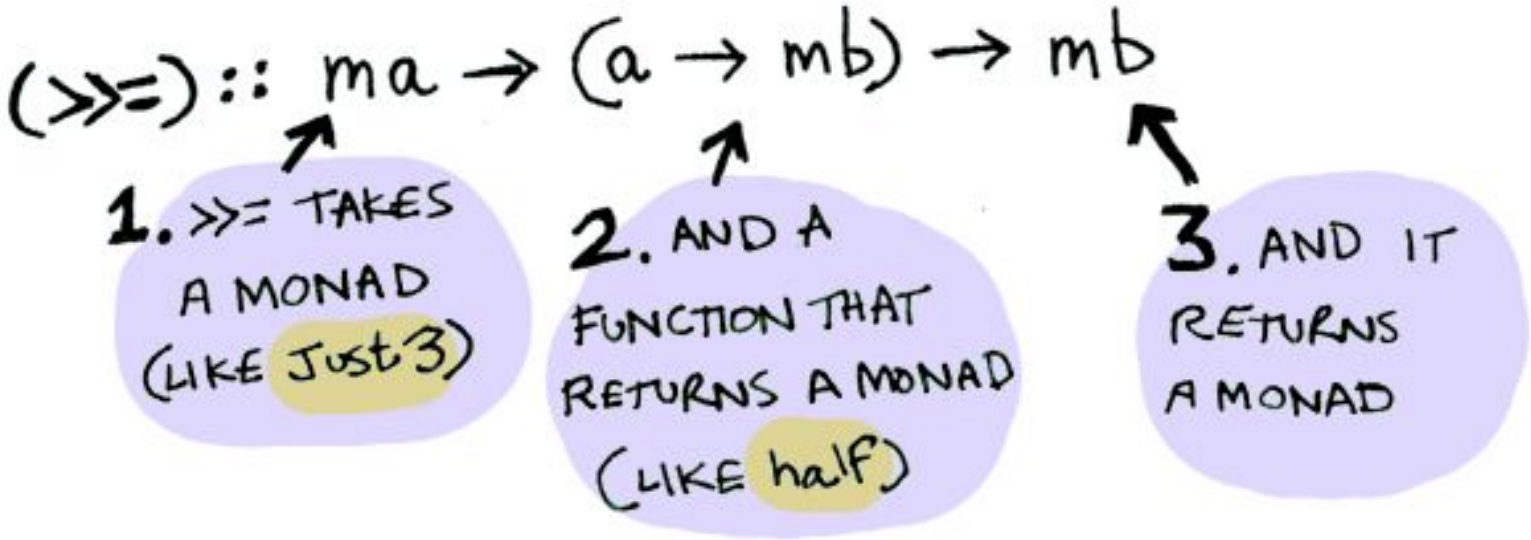

```
In [22]: metade(metade(2))
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-cb32cad46ab0> in <module>()
----> 1 metade(metade(2))

<ipython-input-21-99997e99c9f9> in metade(valor)
      1 def metade(valor):
----> 2     if valor % 2:
      3         return Just(valor //2)
      4     return Nothing()

TypeError: unsupported operand type(s) for %: 'Nothing' and '
int'
```

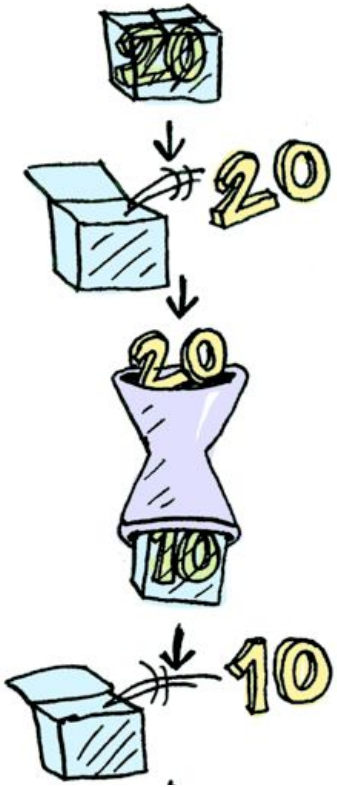
1. BIND UNWRAPS THE VALUE

2. FEEDS THE UNWRAPPED VALUE INTO THE FUNCTION

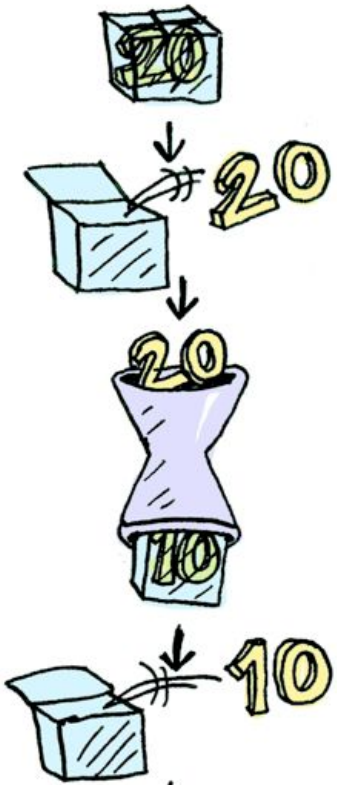
3. WRAPPED VALUE COMES OUT

```
In [57]: class Just:
    ...:     def __init__(self, value):
    ...:         self._value = value
    ...:     def fmap(self, função):
    ...:         return Just(função(self._value))
    ...:     def __or__(self, fn):
    ...:         value = self._value
    ...:         return fn(value)
    ...:     def __repr__(self):
    ...:         return f'Just({self._value})'
    ...:     def bind(self, fn):
    ...:         return self | fn
```

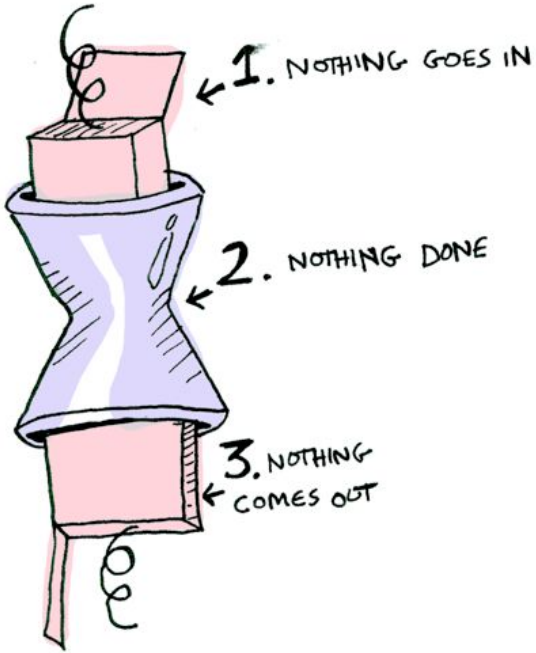

(>>=) :: ma → (a → mb) → mb
1. >>= TAKES A MONAD (LIKE Just 3)
2. AND A FUNCTION THAT RETURNS A MONAD (LIKE half)
3. AND IT RETURNS A MONAD


20
20
20
10
10


10
5
5
NOTH
ING




```
In [60]: Just(20)
Out[60]: Just(20)

In [61]: Just(20) | metade
Out[61]: Just(10)

In [62]: Just(20) | metade | metade
Out[62]: Just(5)

In [63]: Just(20) | metade | metade | metade
Out[63]: Nothing()
```




```
In [68]: Nothing() | metade | metade | metade
Out[68]: Nothing()

In [69]: Nothing() | metade | metade
Out[69]: Nothing()

In [70]: Nothing() | metade
Out[70]: Nothing()

In [71]: Nothing()
Out[71]: Nothing()
```

```
25    print(
26        # File('tabacaria.txt') |
27        Just(10) |
28        clean |
29        split |
30        count |
31        top_10
32    )
33
```

```
~/pybr ❯❯❯ ipython counter.py
Nothing()
```

```
25    print(
26        File('tabacaria.txt') |
27        # Just(10) |
28        clean |
29        split |
30        count |
31        top_10
32    )
```

```
~/pybr ❯❯❯ ipython counter.py                                    07:03:19
Nothing()
~/pybr ❯❯❯ ipython counter.py                                    07:03:34
Just([('que', 59), ('o', 44), ('a', 42), ('e', 42), ('de', 35), ('E', 29), ('não', 28), ('como', 19), ('da', 19), ('em', 15)])
```

https://github.com/dbrattli/OSlash/wiki

XOXO

Perguntas?

Você pode me achar em

@dunossauro

mendesxeduardo@gmail.com

youtube.com/c/eduardomendes