



TEACH YOUR EYE TO EAT (CLOJURE)

kcdc14

OO

Nouns verbing other nouns


```
user.setEmail("mario@outpace.com")
```

```
pitcher.pitch(:curve, batter)
```

```
pitcher.pitch(:curve, ball, batter)
```

OO

Nouns verbing other nouns



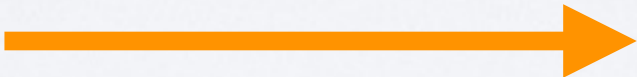
`pitcher.pitch(:curve, ball, batter)`

OO

Nouns verbing other nouns

```
pitcher.pitch(:curve, ball, batter)
```

The pitcher pitches a curve ball to the batter

Left  **Right**

OO

Objects

Messages

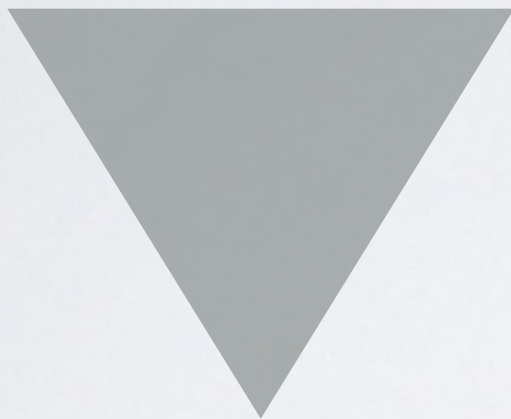


Relationships

OO

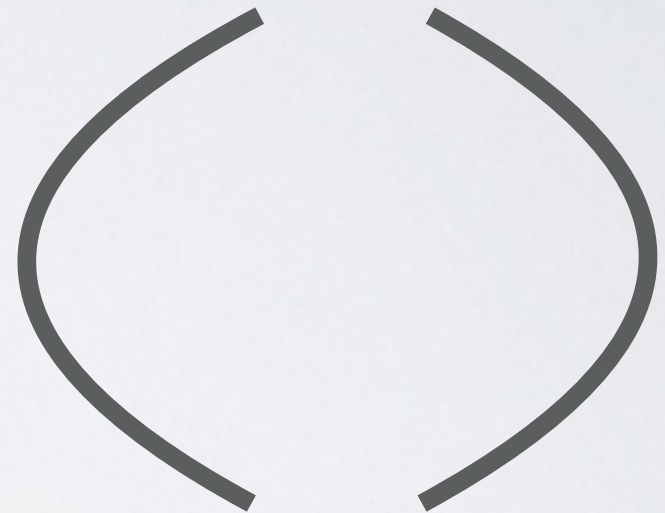
Objects

Messages



Relationships

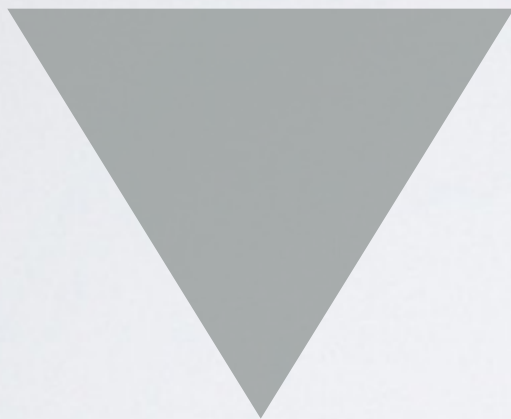
Clojure => LISP



OO

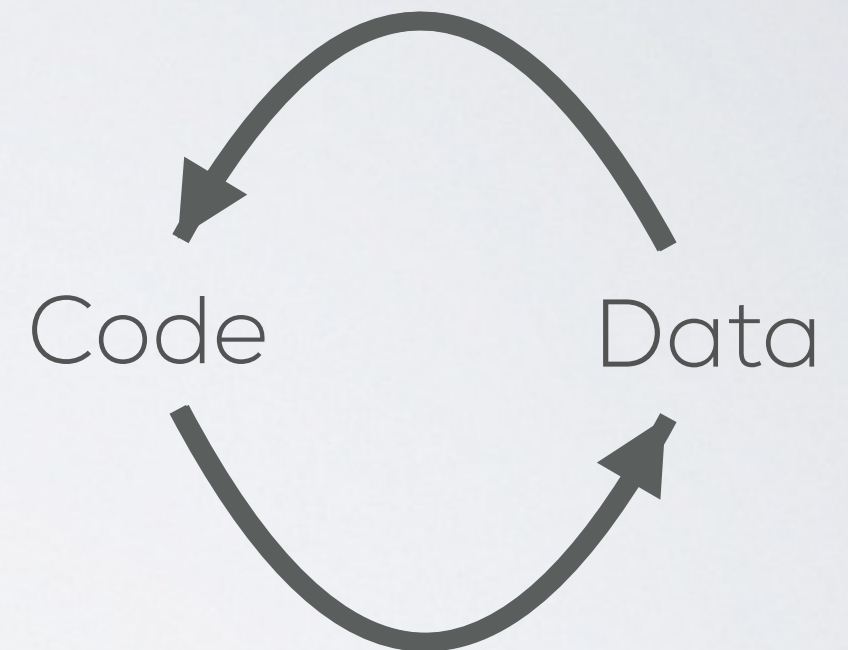
Objects

Messages



Relationships

Clojure => LISP



Clojure deals in lists of things

Clojure deals in lists of things

`'(1 2 3 4)`

List

`[1 2 3 4]`

Vector

`#{1 2 3 4}`

Set

`{1 2 3 4}`

Map



**No
Commas
Needed!**

Clojure deals in lists of things

`'(1 2 3 4)` List

`[1 2 3 4]` Vector

`#{1 2 3 4}` Set

`{:hands 2 :limbs 4}` Map



**No
Commas
Needed!**

Collections are Persistent Data Structures

1

3

(concat [1 2] [3]) => (1 2 3)

2

Data Structures are **Declared** not **Assigned**

```
(def marios-favorite-langs  
  ["ruby"  
   "coffeescript"])
```

Data Structures are **Declared** not **Assigned**

```
(def marios-favorite-langs  
  ["ruby"  
   "coffeescript"])
```

```
=> (marios-favorite-languages)  
[ruby coffeescript]
```

```
=> (assoc marios-favorite-langs 1 "clojure")  
[ruby clojure]
```

```
=> (marios-favorite-langs)  
[ruby coffeescript]
```


Data structure manipulation yields a
new data structure

Clojure is a functional programming language

Clojure is a functional programming language

`(pitch ball batter)`

(pitch ball batter)

Reads/Evaluates Left → Right

(pitch ball batter)

function

parameters

Prefix Notation

parameters

(+ 2 5)

function

Nested function calls

5

4

6

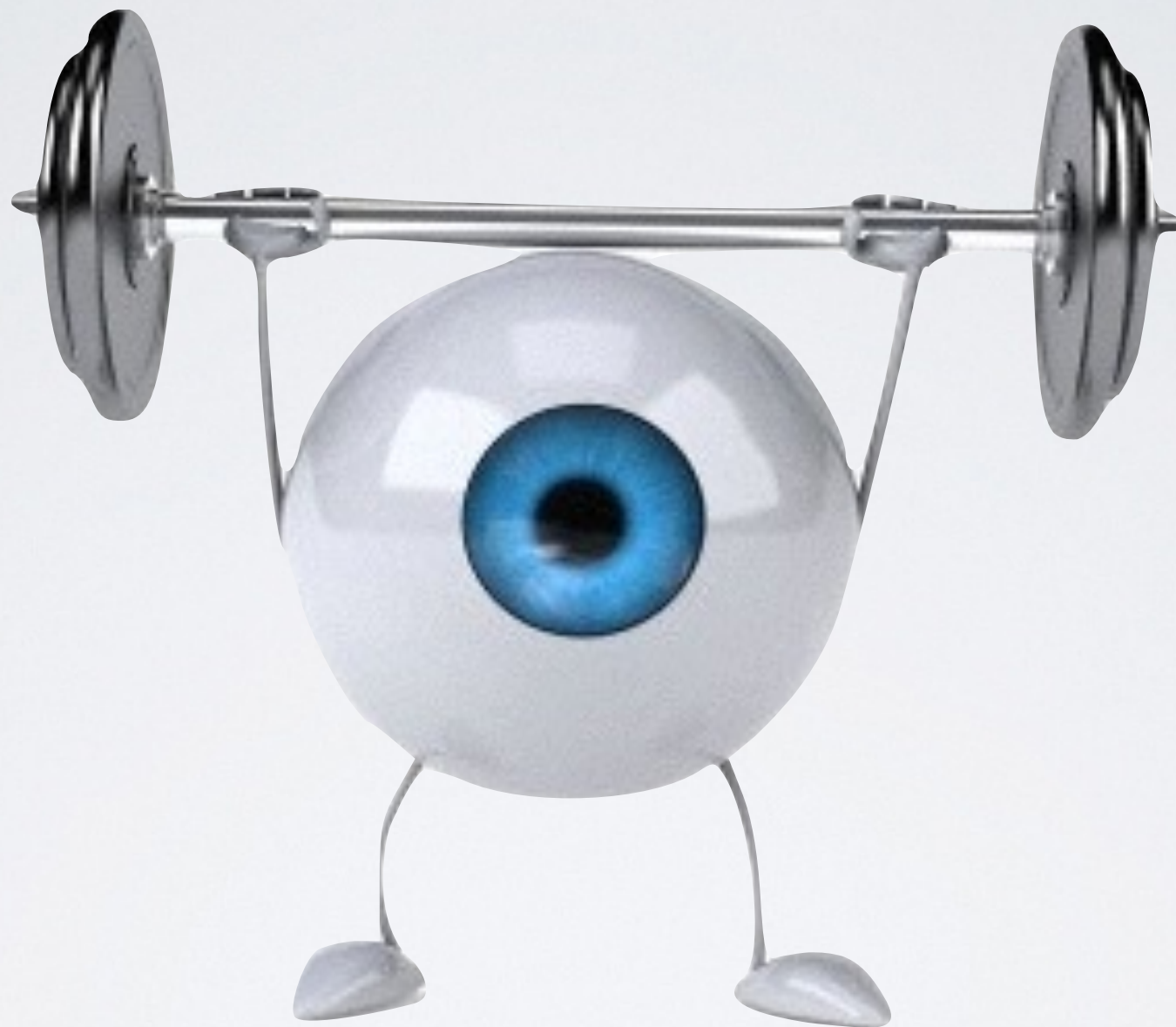
(generate-count (pitch ball batter) inning)

1

2

3

Reads/Evaluates Inside Out



Things that can go in lists...

Numbers

Keywords

Symbols

Other lists

Numbers

Integers

1 2 3...

Floating-point

1.234

Rational Numbers

22/7

Can (*sometimes) auto-promote (`int` \rightarrow `BigInt`)

A photograph of a white plate filled with several skewers of kebabs. The kebabs are made of chunks of white cheese, red onions, and red bell peppers. The plate is set on a light-colored wooden table. Overlaid on the image are three text boxes: a white box with a black border at the top center containing the word 'Keywords', a white box with a black border in the middle containing the text ':are-a-thing-in-clojure', and a white box with a black border at the bottom center containing the text '{:some-key valuetron}'. In the bottom left corner, there is an orange starburst shape containing the text 'kebab-case-all-the-things'. In the bottom right corner, there is a white box with a black border containing the text '@chrisdlugosz'.

Keywords

:are-a-thing-in-clojure

{:some-key valuetron}

kebab-
case-all-
the-things

@chrisdlugosz

:keywords & maps

function



```
(get { :some-key 123 } :some-key) => 123
```



map



key

:keywords & maps

({ : some-key 123 } : some-key) => 123

:keywords & maps

(:some-key {:some-key 123}) => 123

Maps & keywords implement **IFn**

Symbols

are-the-names-for-values-and-functions

Values

`(def marios-twitter` ← Symbol
 `“@marioaquino”)`

3 ways to declare functions

3 ways to declare functions

```
(defn mix [&foods]  
  {:name mixture  
   :contents foods  
   :taste interesting})
```

```
(mix popcorn butter salt parmesan tabasco)
```

3 ways to declare functions

```
(map (fn [email]  
      (lower-case email))  
  ["Mario@thestrangeloop.com"  
   "Mario.E.Aquino@gmail.com"])
```

3 ways to declare functions

```
(map (fn [email]  
      (lower-case email))  
  ["Mario@thestrangeloop.com"  
   "Mario.E.Aquino@gmail.com"])
```

```
=> (mario@thestrangeloop.com  
    mario.e.aquino@gmail.com)
```


3 ways to declare functions

```
(apply  
  (fn power [n e]  
    (if (zero? e)  
        1  
        (* n (power n (dec e)))))  
  3 4 [])
```

=> 81

Same as: (* 3 3 3 3)

3 ways to declare functions

```
=>(reduce #(str %1 " " %2)  
          ["Mario" "Enrique" "Aquino"])
```

"Mario Enrique Aquino"

Constructing algorithms with **let**

```
(defn get-rand-threshold [threshold]
  (let [min (int (* threshold 0.75))
        range (int (* threshold 0.5))]
    (+ min (rand-int range))))
```

Macros

Compile-time code generation/substitution



Reader Macro

$(+ \ 2 \ 5 \ 9) \Rightarrow 16$

Reader Macro

$(+ \ 2 \ 5 \ \#_9) \Rightarrow 7$

Reader Macro

Clojure doesn't have language keywords

Threading Macro

```
(defn read-resource [path]
  (read-string (slurp (io/resource path))))
```

3

2

1

Threading Macro

```
(defn read-resource [path]
  (read-string (slurp (io/resource path))))
```

;; using the threading macro

```
(defn read-resource [path]
  (-> path
      io/resource
      slurp
      read-string))
```

Clojure is

A LISP

A functional programming language

A JVM language

Sufficiently different from imperative, OO langs

Find out more...

<http://clojure.org>

<http://www.braveclojure.com/>

<http://4clojure.com>

<http://www.infoq.com/presentations/Value-Values>



Mario Aquino
@marioaquino



mario@outpace.com