

Open your Clojure

<https://www.dropbox.com/s/5ge6ppbcgchc02v/clojure-nick.pdf>

Николай Линкер

Clojure - это

- Лисп
- Динамический строго типизированный
- Функциональный
- Под JVM
- Тёплое ламповое коммьюнити

Самый быстрый способ погружения

```
(defn create-user
  "Create a new user with a given username and password"
  [username password]
  (let [salt (sha-256 (str (java.util.UUID/randomUUID)))
        password (sha-256 (str password salt))]
    (oyako/insert :users {:username username
                          :password password
                          :salt salt})))

(defn add-tags-to-post
  "Given a post and a seq of tag titles, adds tags as appropriate."
  [post tag-titles]
  (doseq [title tag-titles
          :let [wanted-tag (or (oyako/fetch-one :tags :where {:title title})
                               (tag-from-title title))
                tag (oyako/insert-or-select :tags wanted-tag)]]
    (oyako/insert-or-select :post_tags
      {:post_id (:id post)
       :tag_id (:id tag)})))
```

О, б-жечки, скобки!

What I see

```
define (sym-add augend addend carry)
  (if not (and nil? augend nil? addend)
    (let (ag (car-or-zero augend))
      (ad (car-or-zero addend)
        (cond (= 1 ag ad) (recurse carry augend addend 1))
        (any-nonzero ag ad)
        (recurse (opposite carry) augend addend carry))
      #t (recurse carry augend addend 0)))
    (if (= 1 carry) (cons carry '()) '()))
```

What the non-Lisper sees

```
(define (sym-add augend addend carry)
  (if (not (and (nil? augend) (nil? addend)))
    (let ((ag (car-or-zero augend))
          (ad (car-or-zero addend)))
      (cond ((= 1 ag ad) (recurse carry augend addend 1))
            ((any-nonzero ag ad)
             (recurse (opposite carry) augend addend carry))
            (#t (recurse carry augend addend 0))))
    (if (= 1 carry) (cons carry '()) '())))
```

OH GOD! :-;

Мы – рабы привычек

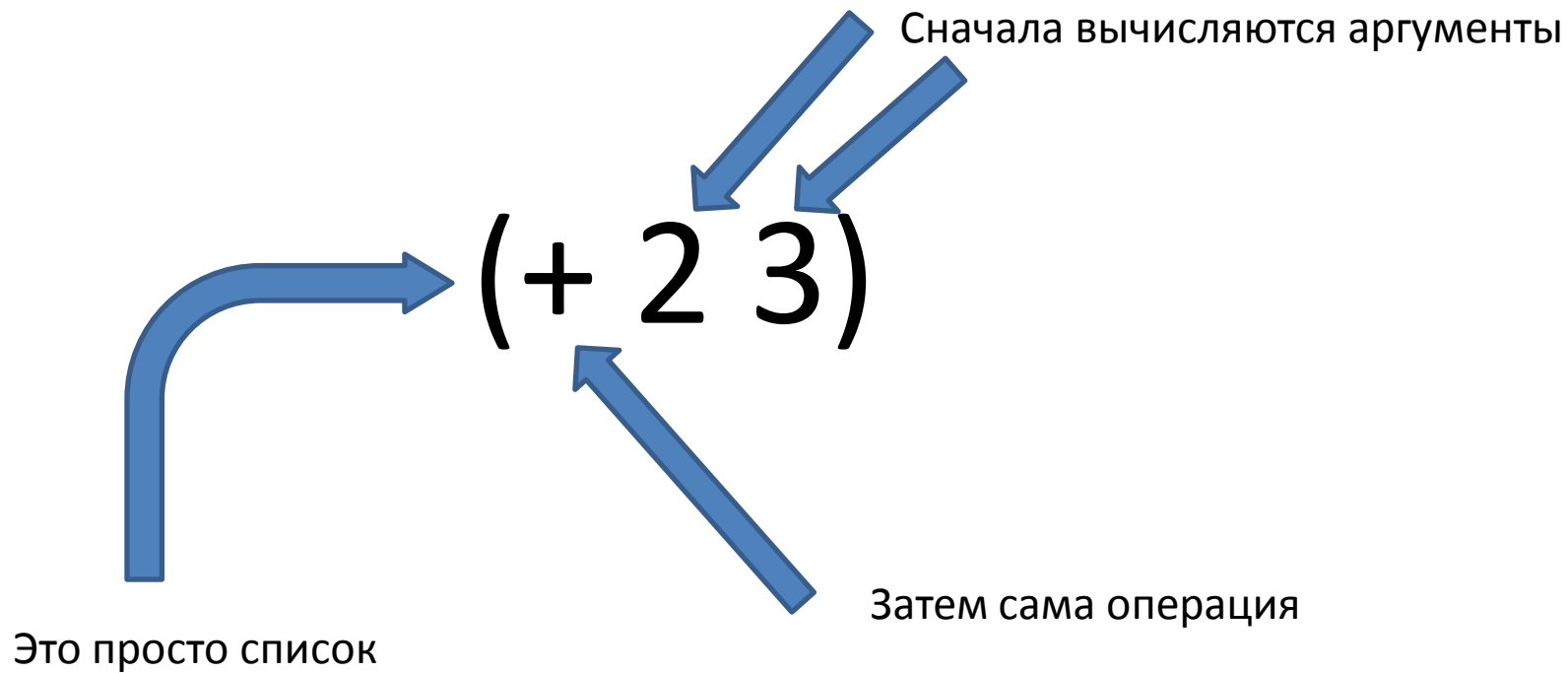
- Обычные калькулятры
- МК-61



Но мы можем выработать новые привычки!

- Java
- !k, ++a, a++
- !a ? b + 1: b – 1
- new java.util.ArrayList(10)
- Math.pow(2, 10)
- “hello”.substring(1, 3)
- Integer.MAX_VALUE
- obj.field
- alist instanceof java.util.List
- Clojure
- (not k), (inc a), ...
- (if (not a) (inc b) (dec b))
- (new java.util.ArrayList 10)
- (Math/pow 2 10)
- (.substring “hello” 1 3)
- Integer/MAX_VALUE
- (.field obj)
- (instance? java .util.List alist)

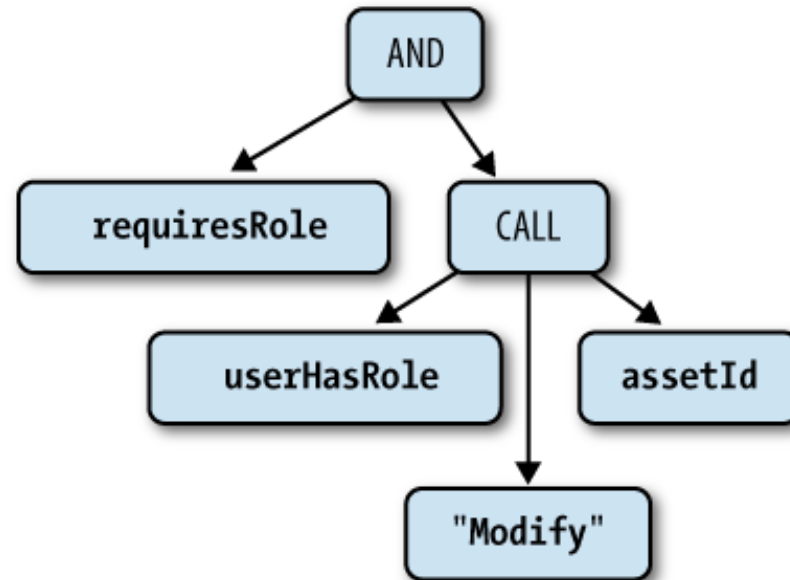
Гармония, как она есть



Code = data = code

- Код = абстрактное синтаксическое дерево

```
(and requiresRole (userHasRole "MODIFY" assetId))
```



непревзойдённая лёгкость оперирования с самой программой

Динамический строго типизированный

(count [1 2 3])

(count {:a 1 :b 2 :c 3})

(count #{1 2 3})

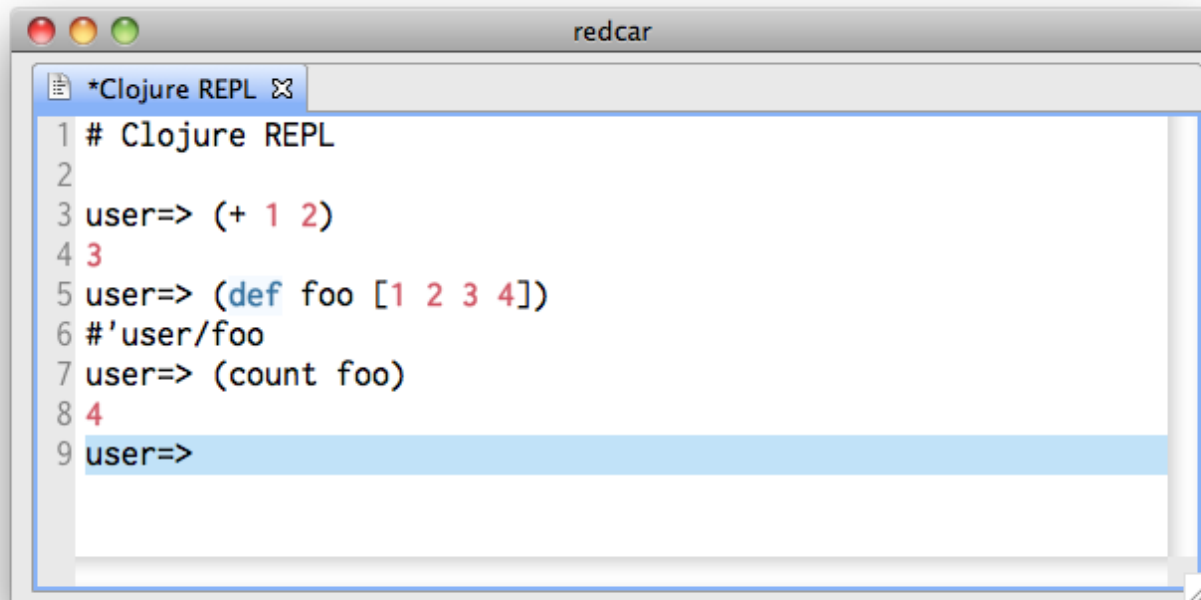
(count '(1 2 3))

(:a {:a 1 :b 2 :c 3}) ; => 1

({:a 1 :b 2 :c 3} :c) ; => 3

REPL

Умная командная строка + поддержка IDE



```
*Clojure REPL ☒  
1 # Clojure REPL  
2  
3 user=> (+ 1 2)  
4 3  
5 user=> (def foo [1 2 3 4])  
6 #'user/foo  
7 user=> (count foo)  
8 4  
9 user=>
```

Android users: https://play.google.com/store/apps/details?id=com.sattvik.clojure_repl

Функциональный

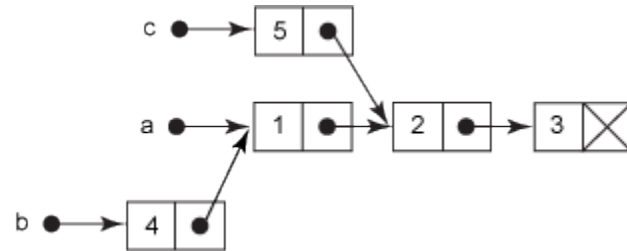
- Функции – главная движущая сила
- неизменяемые коллекции
- Структуры данных объявляются, не присваиваются
- Контролируемое изменение данных (atoms, STM)

Функциям здесь хорошо

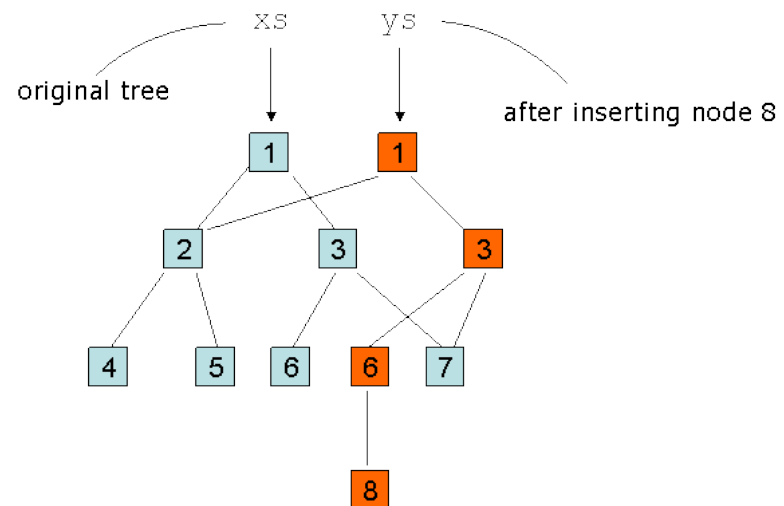
- `(*42 42)`
- `(fn [x] (* x x))`
- `((fn [x] (* x x)) 42) ; => выдаст число`
- `(def sq (fn [x] (* x x)))`
- `(defn sq [x] (* x x))`
- `#(* % %)` ; лямбдочка, 1 параметр
- `#(* %1 %2)` ; лямбдочка, 2 параметр

Изменение неизменяемого

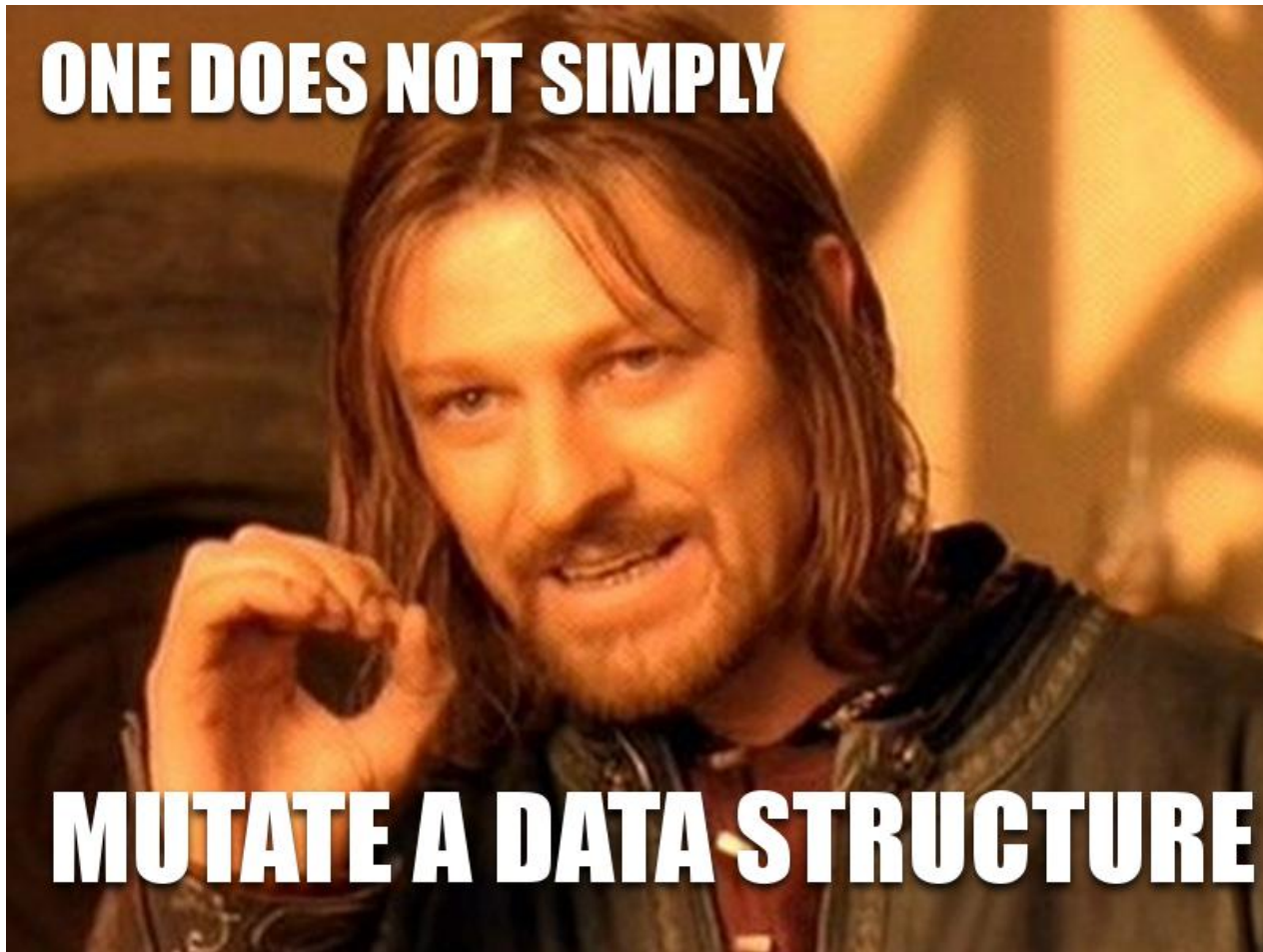
«изменение» списка



«изменение» дерева



Atoms, Refs, Agents



Как выглядит программа на Clojure

- Вначале идёт неймспейс
- Потом декларация(-ии) require

```
(ns blog.db
  "This is the DB layer, using Oyako."
  (:require (clojure.contrib [sql :as sql]
                              [string :as s])
            (oyako [core :as oyako]
                   [query :as query])
            (blog [config :as config]
                  [util :as util]
                  [gravatar :as gravatar]
                  [time :as time]
                  [markdown :as markdown]))
  (:refer-clojure :exclude [comment type]))
```

Как выглядит программа на Clojure

- Потом определяются функции, константы и может быть макросы

```
(defn sha-256 [s]
  (let [md (java.security.MessageDigest/getInstance "SHA-256")]
    (.update md (.getBytes s))
    (s/join ""
      (mapcat #(Integer/toHexString (bit-and 0xff %))
        (into [] (.digest md))))))
```

```
(def schema
  (oyako/make-datamap config/DB
    [:posts
     [belongs-to :categories as :category]
     [belongs-to :users as :user]
     [has-many :comments]
     [habtm :tags via :post_tags]
     [belongs-to :posts as :parent parent-key :parent_id]]
    [:comments
     [belongs-to :posts as :post]]
    [:categories [has-many :posts]]
    [:tags [habtm :posts via :post_tags]]))
```


Тёплое ламповое коммьюнити

- Замечательный доклад, программирование в реальном времени
<http://jokerconf.com/#ryzhikov>
- <http://clojure.org>
- <http://tryclj.org>
- Коммьюнити менее фрагментировано по сравнению со Scala

Have fan!



www.shutterstock.com · 60105112

Спасибо

