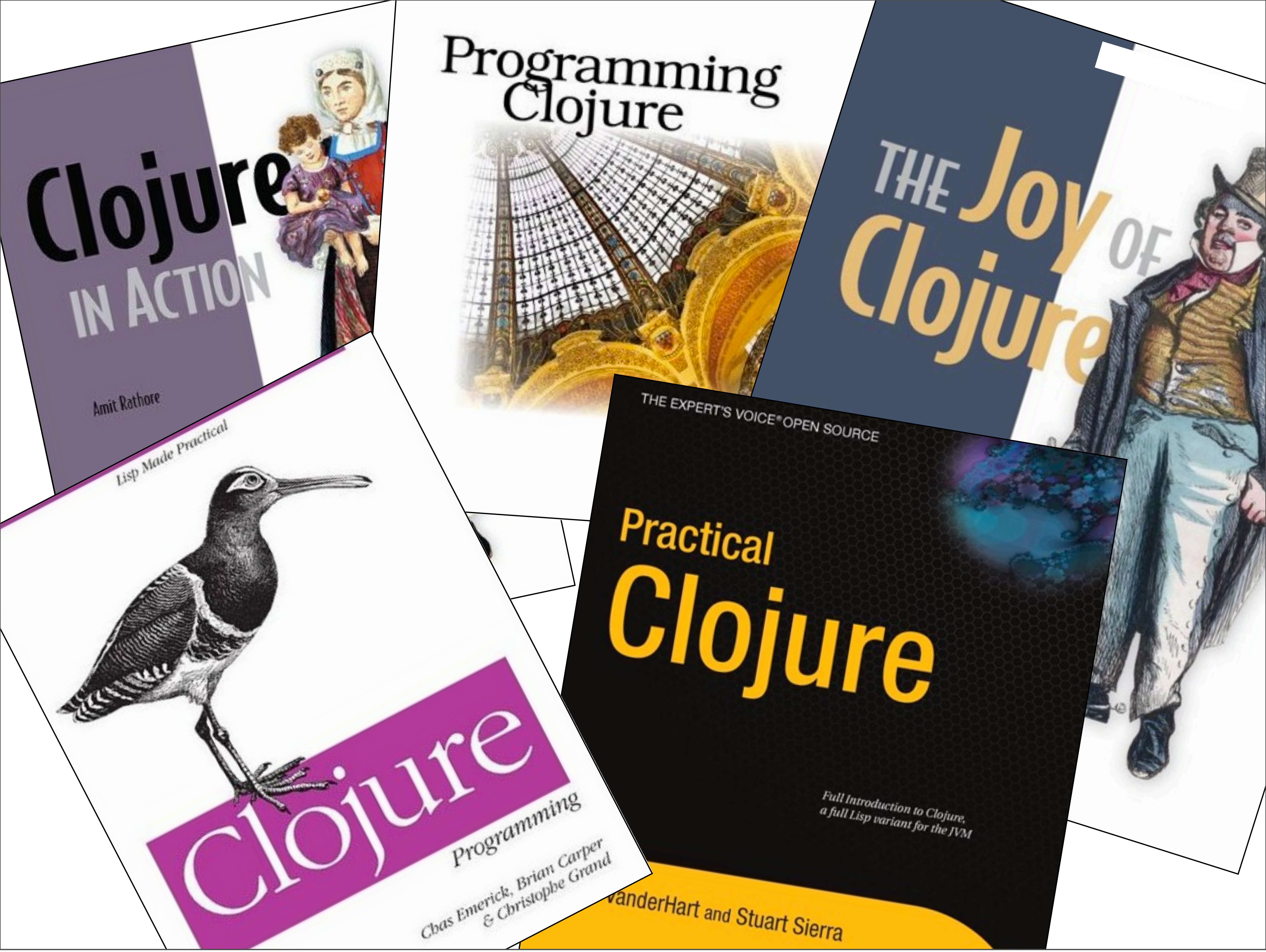
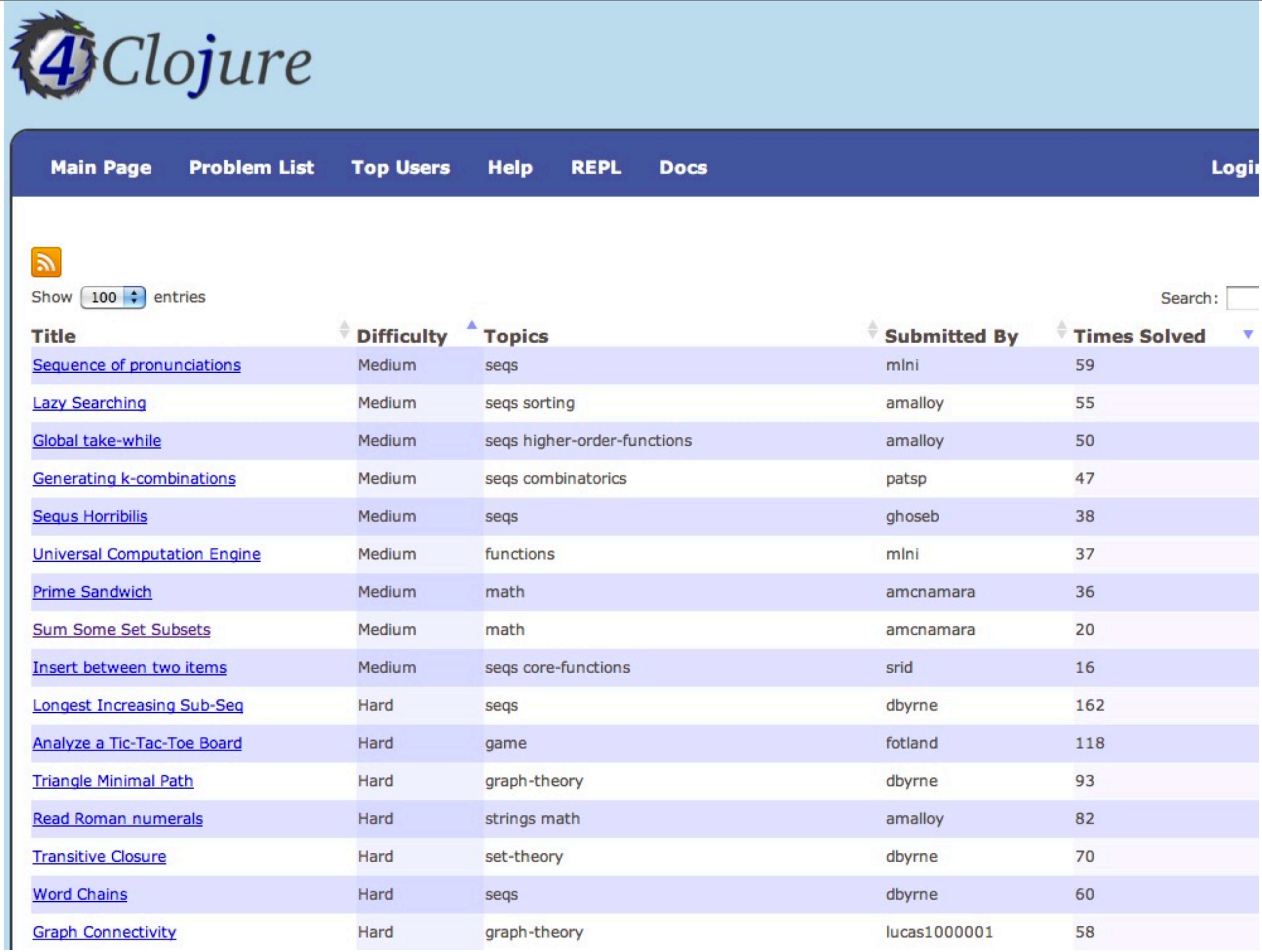
Clojure: Next Steps

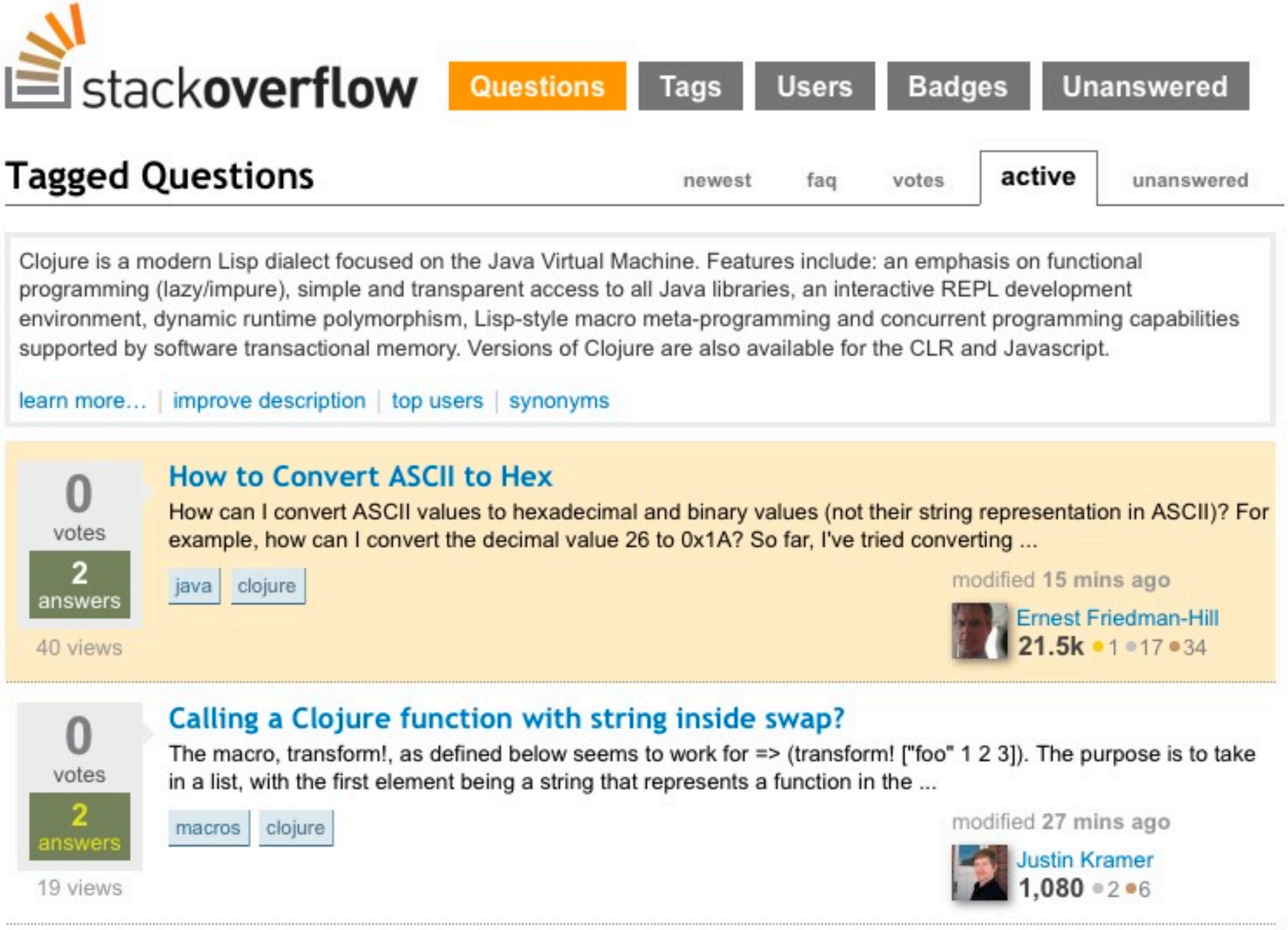
@stuartsierra
#clojure_conj

2


Clojure IN ACTION
Amit Rathore
Programming Clojure
THE Joy OF Clojure
Lisp Made Practical
Clojure Programming
Chas Emerick, Brian Carper & Christophe Grand
THE EXPERT'S VOICE® OPEN SOURCE
Practical Clojure
Full Introduction to Clojure, a full Lisp variant for the JVM
vanderHart and Stuart Sierra



Title	Difficulty	Topics	Submitted By	Times Solved
Sequence of pronunciations	Medium	seqs	mlni	59
Lazy Searching	Medium	seqs sorting	amalloy	55
Global take-while	Medium	seqs higher-order-functions	amalloy	50
Generating k-combinations	Medium	seqs combinatorics	patsp	47
Sequs Horribilis	Medium	seqs	ghoseb	38
Universal Computation Engine	Medium	functions	mlni	37
Prime Sandwich	Medium	math	amcnamara	36
Sum Some Set Subsets	Medium	math	amcnamara	20
Insert between two items	Medium	seqs core-functions	srid	16
Longest Increasing Sub-Seq	Hard	seqs	dbyrne	162
Analyze a Tic-Tac-Toe Board	Hard	game	fotland	118
Triangle Minimal Path	Hard	graph-theory	dbyrne	93
Read Roman numerals	Hard	strings math	amalloy	82
Transitive Closure	Hard	set-theory	dbyrne	70
Word Chains	Hard	seqs	dbyrne	60
Graph Connectivity	Hard	graph-theory	lucas1000001	58



Tagged Questions

Clojure is a modern Lisp dialect focused on the Java Virtual Machine. Features include: an emphasis on functional programming (lazy/impure), simple and transparent access to all Java libraries, an interactive REPL development environment, dynamic runtime polymorphism, Lisp-style macro meta-programming and concurrent programming capabilities supported by software transactional memory. Versions of Clojure are also available for the CLR and Javascript.

learn more… | improve description | top users | synonyms

0 votes
2 answers
40 views

How to Convert ASCII to Hex

How can I convert ASCII values to hexadecimal and binary values (not their string representation in ASCII)? For example, how can I convert the decimal value 26 to 0x1A? So far, I've tried converting ...

java clojure

modified 15 mins ago
Ernest Friedman-Hill
21.5k ●1 ●17 ●34

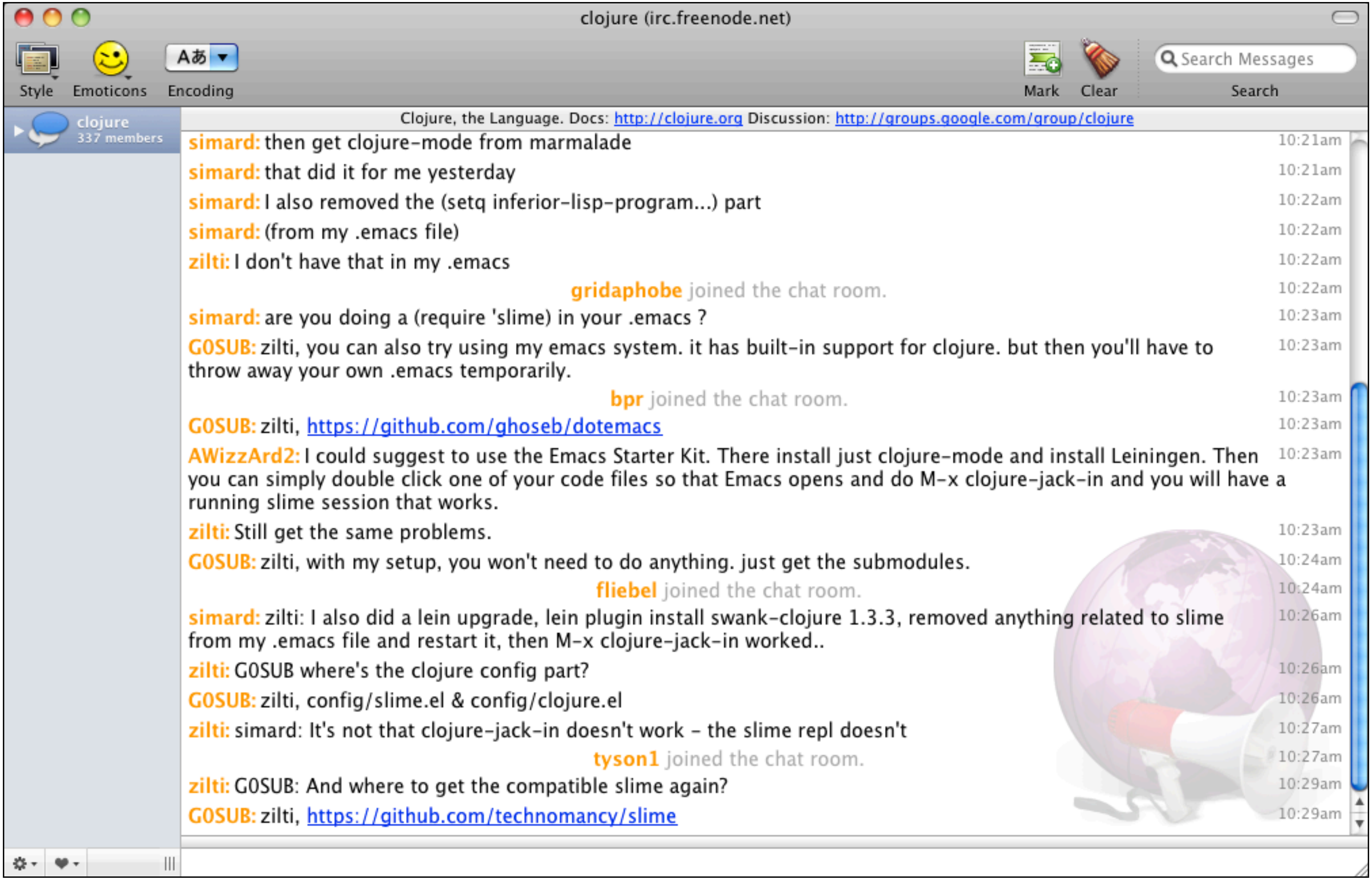
0 votes
2 answers
19 views

Calling a Clojure function with string inside swap?

The macro, transform!, as defined below seems to work for => (transform! ["foo" 1 2 3]). The purpose is to take in a list, with the first element being a string that represents a function in the ...

macros clojure

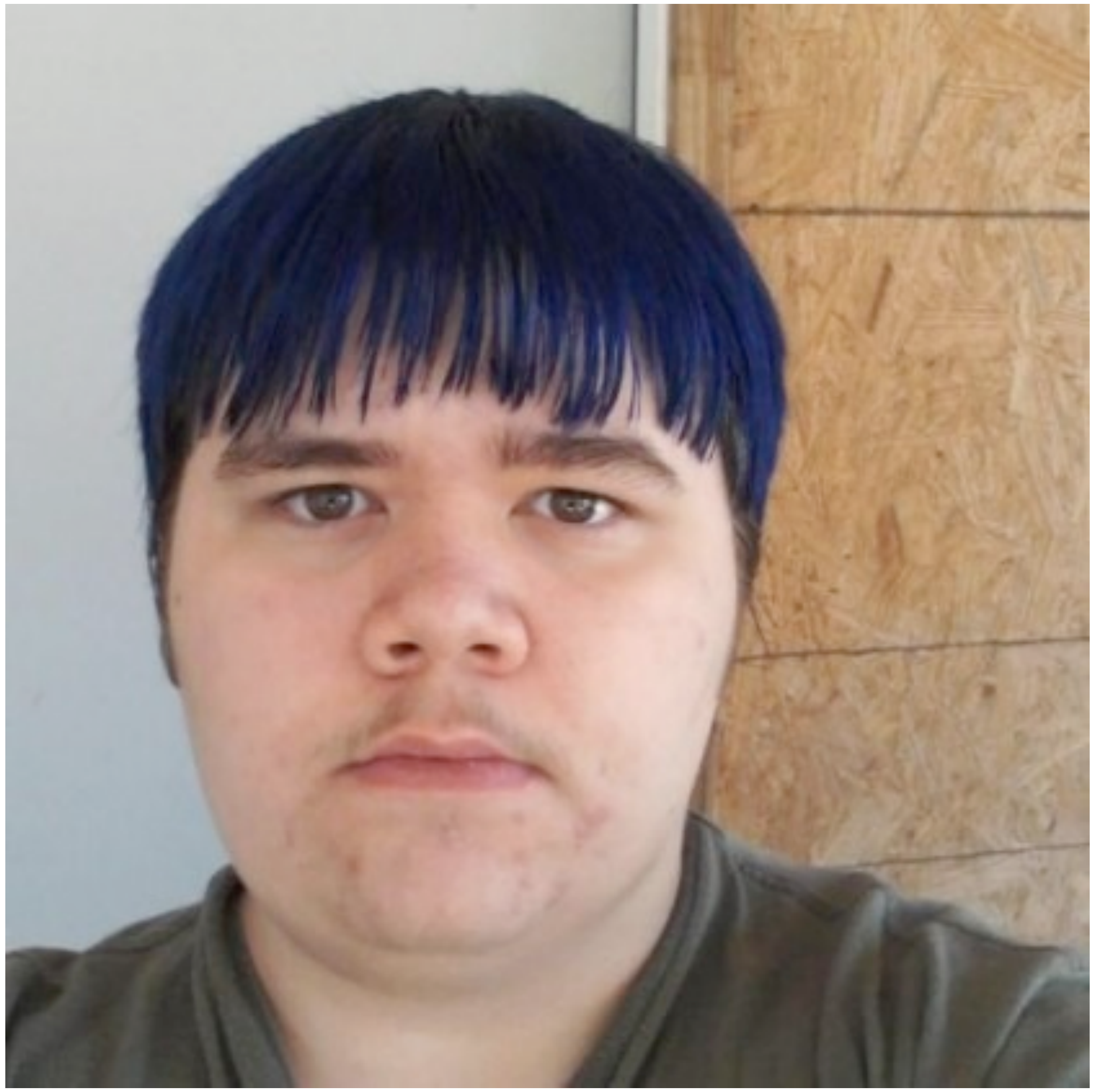
modified 27 mins ago
Justin Kramer
1,080 ●2 ●6



Clojure, the Language. Docs: http://clojure.org Discussion: http://groups.google.com/group/clojure

```
simard: then get clojure-mode from marmalade                                   10:21am
simard: that did it for me yesterday                                           10:21am
simard: I also removed the (setq inferior-lisp-program...) part                10:22am
simard: (from my .emacs file)                                                  10:22am
zilti: I don't have that in my .emacs                                          10:22am
                  gridaphobe joined the chat room.                             10:22am
simard: are you doing a (require 'slime) in your .emacs ?                      10:23am
G0SUB: zilti, you can also try using my emacs system. it has built-in support for clojure. but then you'll have to
throw away your own .emacs temporarily.                                        10:23am
                  bpr joined the chat room.                                    10:23am
G0SUB: zilti, https://github.com/ghoseb/dotemacs                               10:23am
AWizzArd2: I could suggest to use the Emacs Starter Kit. There install just clojure-mode and install Leiningen. Then
you can simply double click one of your code files so that Emacs opens and do M-x clojure-jack-in and you will have a
running slime session that works.                                              10:23am
zilti: Still get the same problems.                                            10:23am
G0SUB: zilti, with my setup, you won't need to do anything. just get the submodules.   10:24am
                  fliebel joined the chat room.                                10:24am
simard: zilti: I also did a lein upgrade, lein plugin install swank-clojure 1.3.3, removed anything related to slime
from my .emacs file and restart it, then M-x clojure-jack-in worked..          10:26am
zilti: G0SUB where's the clojure config part?                                  10:26am
G0SUB: zilti, config/slime.el & config/clojure.el                              10:26am
zilti: simard: It's not that clojure-jack-in doesn't work - the slime repl doesn't   10:27am
                  tyson1 joined the chat room.                                 10:27am
zilti: G0SUB: And where to get the compatible slime again?                     10:29am
G0SUB: zilti, https://github.com/technomancy/slime                             10:29am
```

@stuartsierra 2010



@IORayne 2011

Photo by Baishampayan Ghose (G0SUB) CC BY-SA 2.0

What next?

Reader & Printer

Serialization

```
(defn serialize [x]
  (binding [*print-dup* true]
    (pr-str x)))

(defn deserialize [string]
  (read-string string))
```

print vs. pr

```
user=> (println {:a "one", :b "two"})
{:a one, :b two}
nil
user=> (prn {:a "one", :b "two"})
{:a "one", :b "two"}
nil
```

print-readably

```
user=> (source print)
(defn print
  [& more]
  (binding [*print-readably* nil]
    (apply pr more)))
```

print vs. pr

```
user=> (def m {:a "one", :b "two"})

user=> (= m (read-string (pr-str m)))
true
user=> (= m (read-string (with-out-str (print m))))
false
```

print-dup

```
user=> (prn {:a 1, :b 2})
{:a 1, :b 2}

user=> (binding [*print-dup* true]
         (prn {:a 1, :b 2}))
#=(clojure.lang.PersistentArrayMap/create {:a 1, :b 2})
```

#=()

```
user=> (read-string "[a b #=(* 3 4)]")
[a b 12]
```

read-eval

```
user=> (read-string "#=(java.lang.System/exit 0)")
```

```
user=> (binding [*read-eval* false]
         (read-string "#=(java.lang.System/exit 0)"))
RuntimeException EvalReader not allowed when *read-eval* is
false.  clojure.lang.Util.runtimeException (Util.java:156)
```

Printer/Reader Vars

Binding	Guarantees
`*print-readably* true`	Reader can read what you print
`*print-dup* true`	Same type after print & read
`*read-eval* false`	read is safe

Extending Printing

```
(in-ns 'clojure.core)

(defmulti print-method (fn [x writer]
                         (let [t (get (meta x) :type)]
                           (if (keyword? t) t (class x)))))

(defmulti print-dup (fn [x writer] (class x)))
```

core_print.clj

```
(defmethod print-dup java.util.Collection [o, ^Writer w]
 (print-ctor o #(print-sequential "[" print-dup " " "]" %1 %2)
w))

(defmethod print-dup clojure.lang.IPersistentCollection
  [o, ^Writer w]
  (print-meta o w)
  (.write w "#=(")
  (.write w (.getName ^Class (class o)))
  (.write w "/create ")
  (print-sequential "[" print-dup " " "]" o w)
  (.write w ")"))

(prefer-method print-dup
  clojure.lang.IPersistentCollection
  java.util.Collection)
```

Resources

Maven:

```
src/main/resources/
src/test/resources/
```

Leiningen:

```
resources/
```

clojure.java.io/resource

```
user=> (require '[clojure.java.io :as io])

user=> (io/resource "clojure/core.clj")
#<URL jar:file:/Users/stuart/.m2/repository/org/clojure/
clojure/1.3.0/clojure-1.3.0.jar!/clojure/core.clj>

user=> (with-open [r (java.io.PushbackReader. (io/reader *1))]
         (read r))
(ns clojure.core)
```

clojure.build.ci.generator

```
(ns clojure.build.ci.generator
  (:require [clojure.java.io :as io]))

(defn input-data-url [] (io/resource "ci_data.clj"))

(defn input-data []
  (with-open [r (java.io.PushbackReader.
                  (io/reader (input-data-url)))]
    (read r)))
```

clojure/build.ci on GitHub

ci_data.clj (1)

```
{
 ;; The versions of Clojure against which we will test
 ;; contrib libraries
 :clojure-versions
 ["1.2.0" "1.2.1" "1.3.0" "1.4.0-master-SNAPSHOT"]

 ;; Installed Java versions. If :enabled is true we will
 ;; test contrib libraries with that Java version.
 :jdks
 [{:name "Sun JDK 1.5"
   :enabled true
   :home "/var/lib/hudson/tools/Sun_JDK_1.5.0_22"}
  {:name "Sun JDK 1.6"
   :enabled true
   :home "/usr/java/jdk1.6.0_20"}
  ;; ...
```

ci_data.clj (2)

```clojure
;; ...

;; The contrib libraries. :owners are Hudson usernames of
;; people with permission to build and release each library.
:contribs
[{:name "core.logic"
  :owners ["davidnolen"]}
 {:name "data.finger-tree"
  :owners ["Chouser"]}
 {:name "data.json"
  :owners ["stuartsierra"]}
 {:name "data.priority-map"
  :owners ["markengelberg" "seancorfield"]}
 ;; ...
```

Clojure vs. JSON

	Clojure Syntax	JSON
Sets	Yes	No
Arbitrary-precision numbers	Yes	No
Keywords	Yes	No
Non-string map keys	Yes	No
Metadata	Yes	No
Eval	Yes	No

Ideas

- Custom reader, different defaults
    - e.g. BigDecimals instead of Doubles
    - Reader macros!
    - Follow ClojureScript reader
- Reader/printer for other formats

Extending Interfaces

Clojure Interfaces

Associative
Counted
Fn
IBlockingDeref
IChunk
IChunkedSeq
IDeref
IEditableCollection
IFn
IKeywordLookup
ILookup
ILookupSite
ILookupThunk
IMapEntry
IMeta
IObj
IPending
IPersistentCollection
IPersistentList
IPersistentMap
IPersistentSet
IPersistentStack
IPersistentVector
IProxy
IRecord
IReduce
IRef
IReference
ISeq
ITransientAssociative
ITransientCollection
ITransientMap
ITransientSet
ITransientVector
IType
Indexed
IndexedSeq
MapEquivalence
Named
Reversible
Seqable
Sequential
Settable
Sorted

Finding Interfaces

```
user=> (ancestors (class []))
#{java.util.concurrent.Callable java.util.Collection
java.lang.Runnable clojure.lang.Indexed
clojure.lang.IPersistentVector java.lang.Object
java.lang.Comparable java.util.List clojure.lang.Reversible
clojure.lang.Seqable clojure.lang.ILookup clojure.lang.AFn
clojure.lang.Associative clojure.lang.IEditableCollection
java.util.RandomAccess clojure.lang.Sequential
clojure.lang.IPersistentStack clojure.lang.Counted
clojure.lang.IMeta clojure.lang.IFn
clojure.lang.APersistentVector java.io.Serializable
clojure.lang.IPersistentCollection clojure.lang.IObj
java.lang.Iterable}
```

Finding Interfaces

```
(defn inheritance-tree [klass]
  (let [f (fn f [c]
            (reduce (fn [m p] (assoc m p (f p))) {}
                    (sort-by #(.getName %) (parents c))))]
    {klass (f klass)}))

(defn print-tree [tree]
  (let  [p (fn [c indent]
             (print (apply str (repeat (* 4 indent) \space)))
             (println "*" (if (.isInterface c)
                            (.getName c)
                            (str \< (.getName c) \>))))
         f (fn f [t indent]
             (if (map? t)
               (doseq [[k v] t]
                 (p k indent)
                 (f v (inc indent)))
               (p t indent)))]
    (f tree 0)))
```

```
user=> (print-tree (inheritance-tree (class {})))
* <clojure.lang.PersistentArrayMap>
    * clojure.lang.IObj
        * clojure.lang.IMeta
    * clojure.lang.IEditableCollection
    * <clojure.lang.APersistentMap>
        * java.util.Map
        * java.lang.Iterable
        * java.io.Serializable
        * clojure.lang.MapEquivalence
        * clojure.lang.IPersistentMap
            * java.lang.Iterable
            * clojure.lang.Counted
            * clojure.lang.Associative
                * clojure.lang.IPersistentCollection
                    * clojure.lang.Seqable
                * clojure.lang.ILookup
        * <clojure.lang.AFn>
            * <java.lang.Object>
            * clojure.lang.IFn
                * java.util.concurrent.Callable
                * java.lang.Runnable
```

Finding Methods

```
user=> (source assoc)

(def assoc
 (fn ^:static assoc
   ([map key val] (. clojure.lang.RT (assoc map key val)))
   ([map key val & kvs]
    (let [ret (assoc map key val)]
      (if kvs
        (recur ret (first kvs) (second kvs) (nnext kvs))
        ret)))))
```

Finding Methods

```
// src/jvm/clojure/lang/RT.java

static public Associative
assoc(Object coll, Object key, Object val){
    if(coll == null)
        return new PersistentArrayMap(new Object[]{key, val});
    return ((Associative) coll).assoc(key, val);
}
```

Finding Methods

```
// src/jvm/clojure/lang/Associative.java

public interface Associative
                 extends IPersistentCollection, ILookup{

  boolean containsKey(Object key);

  IMapEntry entryAt(Object key);

  Associative assoc(Object key, Object val);

}
```

Tuples

```
(defmacro def-tuple-type [N]
  (let [NAME (symbol (str "Tuple" N))
        FIELDS (vec (map #(symbol (str "e" %)) (range 0 N)))]
    `(deftype ~NAME ~FIELDS :as ~'this

       clojure.lang.Associative
       (~'containsKey [~'n] (and (integer? ~'n)
                                 (< -1 ~'n ~N)))
       (~'entryAt [~'k] (case ~'k
                              ~@(interleave
                                 (range 0 N)
                                 (map (fn [e]
```

http://paste.lisp.org/+208Q/4

Tuples

```
(def-tuple-type Tuple2 2)
(def-tuple-type Tuple3 3)
(def-tuple-type Tuple4 4)
(def-tuple-type Tuple5 5)

(defn tuple
  "Creates and returns a new 2, 3, 4, or 5-element tuple.
  Tuples support the same methods as vectors."
  ([a b] (Tuple2. a b))
  ([a b c] (Tuple3. a b c))
  ([a b c d] (Tuple4. a b c d))
  ([a b c d e] (Tuple5. a b c d e)))
```

http://paste.lisp.org/+208Q/4

Ideas

- Fixed-size vector (tuples)
- Collections of primitives
- Priority queue
- Matrix
- Map with different hashing strategy

Raw Concurrency

Executors

```
user=> (import '(java.util.concurrent Executors TimeUnit))

user=> (-> (Executors/newScheduledThreadPool 1)
           (.scheduleAtFixedRate #(prn :tick)
                                 1 1 TimeUnit/SECONDS))

:tick
:tick
:tick
:tick
```

AtomicReference

```
// src/jvm/clojure/lang/Atom.java

final AtomicReference state;

public Object swap(IFn f) {
    for(; ;) {
        Object v = deref();
        Object newv = f.invoke(v);
        validate(newv);
        if(state.compareAndSet(v, newv)) {
            notifyWatches(v, newv);
            return newv;
        }
    }
}
```

AtomicLong

```clojure
(let [a (atom 0)]
  (defn counter []
    (swap! a inc)))

;; Equivalent:

(import (java.util.concurrent.atomic AtomicLong))

(let [a (AtomicLong. 0)]
  (defn counter []
    (.incrementAndGet a)))
```

CountDownLatch

```
;; src/clj/clojure/core.clj

(defn promise []
  (let [d (java.util.concurrent.CountDownLatch. 1)
        v (atom d)]
    (reify
     clojure.lang.IDeref
       (deref [_] (.await d) @v)
     clojure.lang.IBlockingDeref
       (deref
        [_ timeout-ms timeout-val]
        (if (.await d timeout-ms
                    java.util.concurrent.TimeUnit/MILLISECONDS)
          @v
          timeout-val))
   ;; ...
```

More java.util.concurrent

- CyclicBarrier
- Semaphore
- Exchanger

Concurrent Collections

"thread-safe, but not governed by a single exclusion lock"

- ConcurrentHashMap
- ConcurrentLinkedQueue
- ConcurrentSkipListMap
- ConcurrentSkipListSet

Blocking Queues

- SynchronousQueue
- LinkedBlockingQueue
- LinkedBlockingDeque
- PriorityBlockingQueue
- ArrayBlockingQueue

locking

```clojure
;; Clojure
(locking object
  ... code ...)
```

```java
// Java
synchronized (object) {
   ... code ...
}
```

- Also: java.util.concurrent.locks
  - ReadWriteLock
  - ReentrantLock

Volatile

```
(deftype MyType [^:volatile-mutable v])
```

- New value can't depend on old value
- Guarantees visibility of writes
- see Brian Goetz, Managing Volatility, IBM developerWorks

Unsynchronized

```
(deftype MyType [^:unsynchronized-mutable v])
```

- No guarantees

Cljque

```
(def a (notifier))

(def b (when-ready [x a] (* x 5)))

(do-when-ready [x a, y b]
  (println "a is" x "and b is" y))

(future
  (Thread/sleep 3000)
  (supply a 7))

;; 3 seconds later...
a is 7 and b is 35
```

stuartsierra/cljque on GitHub

Ideas

- Explore Ref history
- Partial locks (C. Grand, World in a Ref)
- Distributed locks/transactions
- Eventual consistency
- Connect to other transaction systems
    - e.g. databases, Java EE

Explore

clojure.test.generative

```
(defspec quotient-and-remainder
  (fn [a b] (sort [a b]))
  [^{:tag `integer} a ^{:tag `integer} b]
  (let [[a d] %
        q (quot a d)
        r (rem a d)]
    (assert (= a
               (+ (* q d) r)
               (unchecked-add (unchecked-multiply q d) r)))))
```

clojure.core.logic

```
(defrel likes p1 p2)
(fact likes 'Bob 'Mary)
(fact likes 'John 'Martha)
(fact likes 'Ricky 'Lucy)

(defrel fun p)
(fact fun 'Lucy)

(run* [q]
  (fresh [x y]
    (fun y)
    (likes x y)
    (== q [x y]))) ; ([Ricky Lucy])
```

The JVM and Beyond

- ClassLoaders
- ASM / bytecode
- JNI / JNA
- JSVC

Make stuff.

Make apps.

Make libraries.

Make libraries.
Not frameworks.

Have fun at Clojure/Conj

@stuartsierra