Unknown (or less known) pearls from the Clojure Standard Library

ClojuTRE 2015
@reborg

https://www.flickr.com/photos/lizsaldana/4334547080/

Who

- @reborg
- https://github.com/reborg
- Clojure Weekly http://reborg.net
- SICP http://tinyurl.com/sicp-mailonline
- www.DailyMail.co.uk

Freshwater mussels

- Very rare in southern Finland
- More common in the north
- Capable of making fine-quality pearls
- Oldest animal in Finland! Can live up to 250 years

The oldest recorded animal in Finland: ontogenetic age and growth in *Margaritifera margaritifera* (L. 1758) based on internal shell increments

Samuli Helama* & Ilmari Valovirta

*Helama, S., Department of Geology, P.O. Box 64, FI-00014 University of Helsinki, Finland. *Corresponding author; e-mail: samuli.helama@helsinki.fi*

Our definition

- Live in clojure.core, just use straight
- Or in other clojure ns but no “require” required
- Or in other clojure ns and requires “require”
- No external deps should be required
- Excluding usual suspects: juxt, group-by, frequencies etc.

destructure

```
(destructure '[[x y & others] v])

output=> [v2 v
          x (nth v2 0 nil)
          y (nth v2 1 nil)
          others (nthnext v2 2)]
```

Some cleanup, but essentially:

- Shows how destructuring behaves
- Useful for debugging

reductions

```
(reductions + (range 10))
;(0 1 3 6 10 15 21 28 36 45)
```

the last is the normal reduce result: 45

- Show the inner reduction steps
- Useful to debug a complex **`reduce`**

test

```
(defn add+
 {:test #(do
     (assert (= (add+ 2 3) 5))
     (assert (= (add+ 4 4) 8)))}
  [x y] (+ x y))

(test #'add+)
```

trigger

- Tests can be “embedded” in the var metadata
- Useful for small quick assertions
- Tests are visible with (:test (meta #’add+))

clojure.pprint/cl-format

```
(clojure.pprint/cl-format nil "~:r" 1234)
; "one thousand, two hundred thirty-fourth”
(clojure.pprint/cl-format nil “~@r" 1234)
; "MCCXXXIV"
```

- The glorious common lisp (format) function
- It can do much more: pluralization, number auto-scaling etc.

clojure.java.browse/**browse-url**

```
(def url "http://localhost:3000")
(clojure.java.browse/browse-url url)
```

- Open up a browser pointing at url
- Useful to call into the system browser programmatically

clojure.java.javadoc/javadoc

```
(clojure.java.javadoc/javadoc (list* 1 []))
;; open clojure.lang.Cons Javadoc
```

- Configurable with *local-javadocs* or *remote-javadocs*
- Useful quick peek into Java classes at the REPL

clojure.reflect/reflect

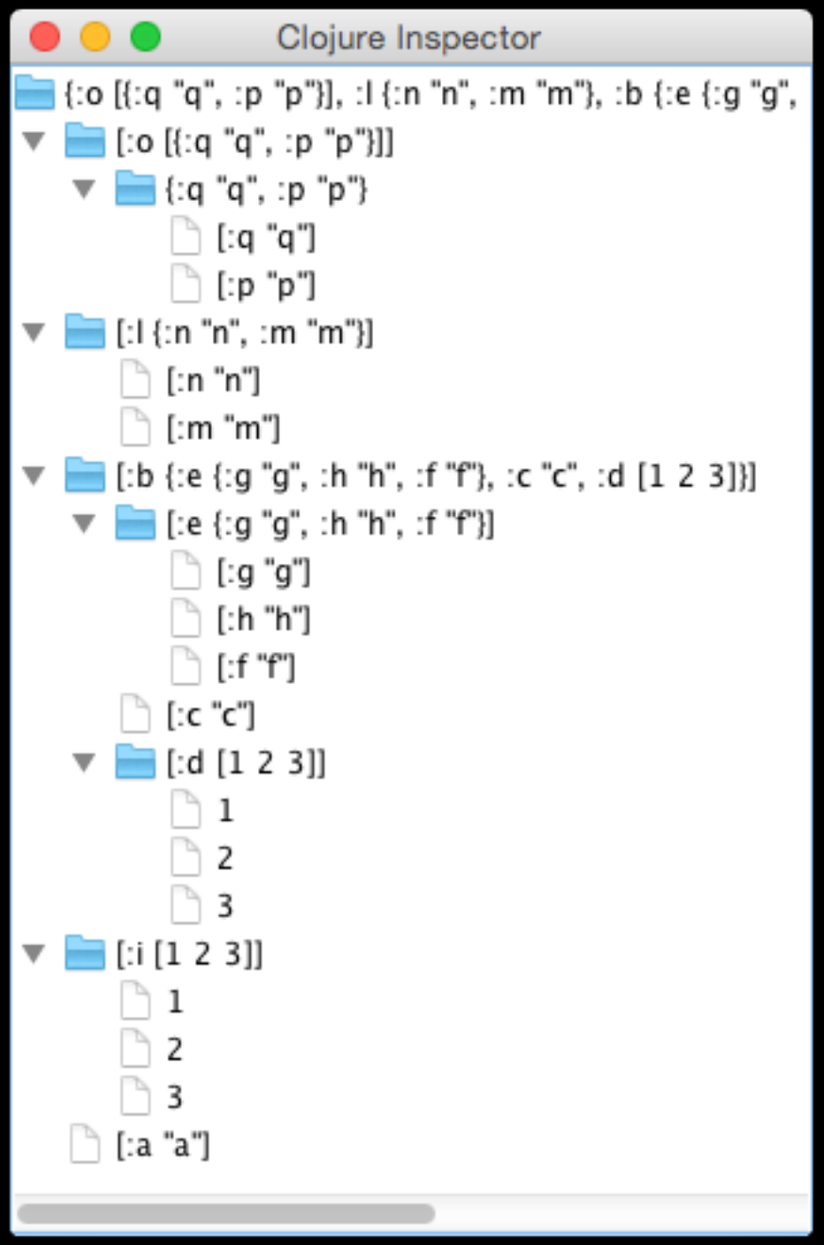
```
(require '[clojure.reflect :refer [reflect]])
(println (with-out-str (clojure.pprint/write (reflect :a))))

;; extract from a typical output:
{:name invoke,
 :return-type java.lang.Object,
 :declaring-class clojure.lang.Keyword,
 :parameter-types [java.lang.Object],
 :exception-types [],
 :flags #{:public :final}}
```

- Java reflection on steroids
- Easy to process output as a Clojure map

clojure.inspector/inspect-tree

```clojure
(require '[clojure.inspector :as i])
(def m {:a "a"
        :b {:c "c"
            :d [1 2 3]
            :e {:f "f"
                :g "g"
                :h "h"}}
        :i [1 2 3]
        :l {:m "m"
            :n "n"}
        :o [{:p "p" :q "q"}]})
(i/inspect-tree m)
```



- Visualizing complex data structures
- Also table and list views
- For exmple: json visualization

clojure.lang.PersistentQueue

```
(def e (clojure.lang.PersistentQueue/EMPTY))
(def buf (reduce conj e (range 10)))
(peek buf)
; 0
(peek (pop buf))
; 1
(peek (pop (pop buf)))
; 2
```

- Persistent FIFO queue
- Buffers, schedulers, etc. (the functional way)

fnil

```
(def m {:host "127.0.0.1" :port nil})
(update m :port (fnil #(Integer/parseInt %) "80"))
; {:port 80, :host “http://localhost"}

(def m {:host "127.0.0.1" :port "8008"})
(update m :port (fnil #(Integer/parseInt %) "80"))
; {:port 8008, :host "http://localhost"}
```

- Decorate other functions with a nil-handler
- Use for defaults in environment maps

Honorable mentions

- counted? reversible? O(1) operation check
- vector-of (unboxed vectors of primitives)
- clojure.set/rename-keys (but it's about maps!)
- clojure.data/diff (dead easy diffing)
- munge, gensym, seque, zippers, OMG!

~ Fin ~

What are your favourite pearls?

-Bonus-
every? and the vacuous truth

```
(def nums [])
(every? even? nums)
; true
(every? odd? nums)
; true
```

- Some bizarre logic to close.
- All numbers in the empty set are both even and odd simultaneously!