Clojure Basic Training - Module 3

Working with Clojure data structures

First, common types

- String: “hello world!”
- Number: 3, 2/3, 0.66, 3456N, 123M
- Boolean: true, false
- Keyword: :article-id or :created-date
- Symbol: (def **s** 3) or (let [**x** 4] (+ x 3)) or (defn **my-fn**[**x**] (println “Ignore me”))
- Nil: equivalent to java null

Usual suspects

- lists - ‘(:x :y :z :x) or (list x y z x)
- sets - #{:x :y :z} or (set :x :y :z)
- vector - [:x :y :z] or (vector :x :y :z)
- maps - {:a 1 :b 45} or (hash-map :a 1 :b 45)

Persistent (Immutable)

In computing, a persistent data structure is a data structure that always preserves the previous version of itself when it is modified. Such data structures are effectively immutable, as their operations do not (visibly) update the structure in-place, but instead always yield a new updated structure.

Collection operations

- **conj** : adds an item to the collection
- **seq** : gets a sequence of a collection
- **count** : returns count of items in collection
- **empty** : returns empty collection of same type
- **=** : value equality

Sequences

- applicable on all collections, iterables and strings
- create a sequence: (seq *collection*)
- a sequence is not a list (can be lazy and infinite)
- first, second, rest, take, empty?

Associative

- Maps are associative: they associate a key and a value
- Vectors are associative: they associate an index and a value

- assoc/dissoc
- get
- contains?

Accessing data

- get / first / second / last / nth / find
- keywords are functions
- collections are functions

Common operations

- **map**: (map inc [1 2 5 -5 9])
- **filter** / **remove**: (filter pos? [1 2 5 -5 9])
- **empty?**
- **into**
- **range**