CS 331 — Clojure Activity

Mattox Beckman

Introduction

In this activity, you will look at four Clojure "sample runs" and try to guess what is going on. At the end are some questions you will do in class.

Lists

```
user> (def x1 '(1 2 3 4))
#'user/x1
user> (count x1)
4
user> (reverse x1)
(4 3 2 1)
user> x1
(1 2 3 4)
user> (first x1)
1
user> (rest x1)
(2 3 4)
user> (cons 123 x1)
(123 1 2 3 4)
user> (defn sum [xx]
        (if (empty? xx) 0 (+ (first xx) (sum (rest xx)))))
#'user/sum
user> (sum x1)
10
user> (apply + x1)
10
user> (apply * x1)
24
user> (= x1 '(1 2 3 4))
true
user> (identical? x1 '(1 2 3 4))
false
user> (inc 10)
11
user> (map inc x1)
(2 3 4 5)
```

Vectors

```
user> (def v1 [2 4 6 8 10])
#'user/v1
user> (count v1)
5
user> (v1 2)
6
user> (v1 1)
4
user> (first v1)
2
user> (rest v1)
(4 6 8 10)
user> (map inc v1)
(3 5 7 9 11)
user> (vec (map inc v1))
[3 5 7 9 11]
user> (mapv inc v1)
[3 5 7 9 11]
user> (filterv #(> % 5) v1)
[6 8 10]
user> (assoc v1 3 123)
[2 4 6 123 10]
user> v1
[2 4 6 8 10]
```

Sets

```
user> (def s1 #{10 20 30})
#'user/s1
user> (s1 0)
nil
user> (s1 10)
10
user> (s1 20)
20
user> (if (s1 20) "Has 20" "Doesn't have 20")
"Has 20"
user> (if (s1 120) "Has 120" "Doesn't have 120")
"Doesn't have 120"
user> (conj s1 13)
#{10 13 20 30}
user> (conj s1 #{1 2 3})
#{#{1 2 3} 10 20 30}
user> (union s1 #{1 2 3})
CompilerException java.lang.RuntimeException: Unable to resolve symbol: union in this contex
user> (clojure.set/union s1 #{1 2 3})
#{1 2 3 10 20 30}
user> (disj s1 20)
#{10 30}
user> (require '[clojure.set :as set])
nil
user> (set/subset? s1 #{1 10 20 30 50})
true
user> (into #{} '(1 2 3))
#{1 2 3}
user> (reduce conj #{} '(8 6 7 5 3 0 9))
#{0 3 5 6 7 8 9}
```

Hashmaps

```
user> (def h1 {:name "Gollum" :salary 340000.13 :title "Coder"})
#'user/h1
user> (h1 :name)
"Gollum"
user> (:name h1)
"Gollum"
user> (keys h1)
(:name :title :salary)
user>  (assoc h1 :hobby "Weiqi")
{:name "Gollum", :title "Coder", :hobby "Weiqi", :salary 340000.13}
user> h1
{:name "Gollum", :title "Coder", :salary 340000.13}
user> (def h2 (assoc h1 :employee {:name "Fred" :title "Lackey" :salary 19.95}))
#'user/h2
user> h2
{:employee {:name "Fred", :title "Lackey", :salary 19.95}, :name
  "Gollum", :title "Coder", :salary 340000.13}
user> (h2 :name)
"Gollum"
user> (h2 :employee)
{:name "Fred", :title "Lackey", :salary 19.95}
user> ((h2 :employee) :name)
"Fred"
user> (-> h2 :employee :name)
"Fred"
user> (assoc-in h2 [:employee :hobby] "Revenge")
{:employee {:name "Fred", :title "Lackey", :hobby "Revenge", :salary 19.95}, :name "Gollum",
user> (assoc-in h2 [:salary] 37.19)
{:employee {:name "Fred", :title "Lackey", :salary 19.95},
 :name "Gollum", :title "Coder", :salary 37.19}
user> (h2 :friends)   ; Gollum has no friends!
nil
user> (def el [{:name "Foo" :salary 100} {:name "Bar" :salary 200}
               {:name "Baz" :salary 300}])
#'user/el
user> (map :name el)
("Foo" "Bar" "Baz")
user> (defn get-who [hmap key val] (if (= (hmap key) val) hmap nil))
#'user/get-who
user> (get-who (first el) :name "Foo")
{:name "Foo", :salary 100}
user> (get-who (second el) :name "Foo")
nil
user> (map #(get-who % :name "Foo") el)
({:name "Foo", :salary 100} nil nil)
user> (filter identity  (map #(get-who % :name "Foo") el))
({:name "Foo", :salary 100})
```

Questions

Try these questions. Have the instructor check them at the end of class.

- Write a Clojure function that will take the product of the elements of a list. Do this two different ways.

- Write a Clojure function that will count how many zeros are in a given vector.

- Write a function[1] to convert a list into a set.

- Write a function that returns the salary of a given employee. The function should take the vector of employees and a name.

[1]Okay, you've figured out that it should be in Clojure by now, so I'm not going to keep specifying that.