CS 296 — Clojure's Built-in Data Structures, Part 1

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]
```

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.