Laboratory: Clojure (1)

Summary: We begin our exploration of the Clojure programming language.

Prerequisites: Tate, Sections 7.1 and 7.2.

Contents:

- Preparation
- Exercises
  - Exercise 1: Examining Structures
  - Exercise 2: Fun with Quotes
  - Exercise 3: Fun with Scope
  - Exercise 4: Tate’s Problems
- For Those with Extra Time

Preparation

a. Create a directory for the lab.

b. Open a browser window on Tate’s examples, in case you want to try any of them.

c. Make sure that you know how to load files into Clojure. (Hint: Give them a .clj suffix and use (load "prefix"). If that’s not enough, make sure that your CLASSPATH is set correctly.)

Exercises

Exercise 1: Examining Structures

What do you expect the results of each of the following to be? After predicting each, check your answer experimentally.

a. \(\text{first (list 'a 'b 'c)}\)

b. \(\text{rest (list 'a 'b 'c)}\)

c. \(\text{first ['a 'b 'c]}\)

d. \(\text{rest ['a 'b 'c]}\)

e. \(\text{first #\{a 'b 'c\}}\)
f. (rest #\{a 'b 'c\})
g. (first #\{a "Alpha" 'b "Beta"\})
h. (rest (list 'a))
i. (rest ['a])
j. (rest #{'a})
k. (rest {'a "Alpha"})
l. (first nil)
m. (rest nil)
n. (last nil)
o. (cons 'a (list 'b 'c 'd))
p. (cons 'a ['b 'c 'd])
q. (cons 'a #{'b 'c 'd})
r. (cons 'a #{'b "Beta" 'c 'Charlie'})

Exercise 2: Fun with Quotes

a. What do you expect the result of the following to be?

(first (first #\{(a b) '(c d)\}))

b. Check your answer experimentally.

c. What does this example suggest about why we discourage the use of quotes to create lists in 151?

Exercise 3: Fun with Scope

Consider the following code

(def x 5)
(defn f [y] (+ x y))

a. What do you expect (f 3) to be?

b. Check your answer experimentally.

c. Suppose we redefined x, as in
(def x 23)

What do you expect \( f \ 3 \) to be?

d. Check your answer experimentally.

e. Suppose we locally defined \( x \), as in

(let \[ x \ 100 \] \( f \ 3 \))

What do you expect the result to be?

f. Check your answer experimentally.

g. What do you expect the result of the following code to be?

(defn g [a] (+ y a))
(def y 10)
(g 3)

h. Check your answer experimentally.

i. What do you expect the result of the following code to be?

(def y (list 1 2 3))
(defn g [a] (+ y a))
(g 3)
(def y 4)
(g 3)

j. Check your answer experimentally.

k. What have these exercises taught you about Clojure’s scoping and type checking?

**Exercise 4: Tate’s Problems**

*These problems taken from the Tate Day 1 Self-Study.*

a. Implement a function called \( \text{big st n} \) that returns true if a string \( \text{st} \) is longer than \( n \) characters.

b. Write a function called \( \text{collection-type col} \) that returns :list, :map, or :vector based on the type of collection \( \text{col} \).

**For Those with Extra Time**

If you find yourself with extra time, further explore the differences between Clojure and Scheme.