**Lab 4 Clojure**

4)

(defn greater-num [x y]

(if ( > x y)

(println (format "%d is bigger" x))

(println (format "%d is bigger" y))))

(greater-num 96 33)

Graphical user interface, text, application

Description automatically generated

5)

1. takes a positive integer and prints that many dots

a. Recursive:

(defn recur-dot [num]

(loop [i num]

(when (> i 0)

(print ".")

(recur (- i 1))))) //can even change out the "recur" with "recur-dot"

(recur-dot 5)

Graphical user interface, text, application

Description automatically generated

Iterative:

(defn iter-dot [num]

(def x 0)

(while (< x num)

(do

(print ".")

(def x (+ x 1)))))

(iter-dot 7)

Graphical user interface, text, application

Description automatically generated

2. takes a list and returns the number of times the symbol a

occurs in it.

a. Iterative

(defn iter-list [lst]

(def x 0)

(def cnt 0)

(def ls lst)

(while (< x (count lst))

(if (= (first ls) 'a)

(do

(def ls (rest ls))

(def x (+ x 1))

(def cnt (+ cnt 1)))

(do

(def ls (rest ls))

(def x (+ x 1))

(def cnt (+ cnt 0)))))

(println cnt))

(def ls (list 'a 'b 'c 'd 'a 'a))

(iter-list ls)

Graphical user interface, text

Description automatically generated

b. Recursive

(defn recur-list [lst OgCnt]

(def cnt OgCnt)

(def ls lst)

(if (= 0 (count lst))

(println cnt)

(if (= (first ls) 'a)

(do

(def cnt (+ OgCnt 1))

(recur (rest ls) cnt))

(do

(def cnt (+ OgCnt 0))

(recur (rest ls) cnt)))))

(def ls (list 'a 'b 'c 'd 'a 'j))

(recur-list ls 0)

Graphical user interface, text

Description automatically generated

6)

(defn list-of-num [num]

(def x 1)

(while (<= x num)

(do

(print x "")

(def x (+ x 1)))))

(list-of-num 11)

Graphical user interface, text, application

Description automatically generated

7)

(defn remove-multiples [lst num newLst]

(def ls lst)

(def rtrnLst newLst)

(if (= 0 (count lst))

(print rtrnLst)

(if (not= (mod (first ls) num) 0)

(do

(def rtrnLst (cons (first ls) rtrnLst))

(recur (rest ls) num rtrnLst))

(do

(recur (rest ls) num rtrnLst)))))

(def ls (list 1 5 10 6 95 82))

(remove-multiples ls 5 (list))

Graphical user interface, text

Description automatically generated

8)

(defn sieve [n]

(if (< 1 n)

(empty? (filter #(= 0 (mod n %)) (range 2 n)))

false))

(defn prime-seq [from to]

(print (filter sieve (range from (inc to)))))

(prime-seq 2 16)

Graphical user interface, text, application

Description automatically generated

9)

(defn fizzbuzz [lst newLst]

(def ls lst)

(def rtrnLst newLst)

(if (= 0 (count lst))

(print rtrnLst)

(if (= (mod (first ls) 6) 0)

(do

(def rtrnLst (cons "MeWantCookie" rtrnLst))

(recur (rest ls) rtrnLst))

(if (= (mod (first ls) 3) 0)

(do

(def rtrnLst (cons "cookie" rtrnLst))

(recur (rest ls) rtrnLst))

(if (= (mod (first ls) 2) 0)

(do

(def rtrnLst (cons "want" rtrnLst))

(recur (rest ls) rtrnLst))

(do

(def rtrnLst (cons (first ls) rtrnLst))

(recur (rest ls) rtrnLst)))))))

(def ls (list 2 3 6 5 19 11))

(fizzbuzz ls (list))

Text

Description automatically generated