CS35L: Week 1 Lisp Worksheet Solutions

Elisp and more

1. What do the following Lisp expressions output, when evaluated?
   a. `(quote (1 2 3))`
      `(1 2 3)`
   b. `'(1 2 3)`
      `(1 2 3)`
   c. `(list (+ 1 2) '(+ 1 2))`
      `(3 (+ 1 2))`
   d. `(cons (+ 1 2) '(3 4))`
      `(3 3 4)`
   e. `(+ 10 (car '(1 2 3)))`
      `11`
   f. `(reverse (append '(1 2) '(3 4)))`
      `(4 3 2 1)`
   g. `(cdddar (1 2 3 4 5 6 7))`
      This is illegal!
2. Dr. Eggert mentioned in lecture that you can edit your Emacs source code to customize it to your liking. One source of confusion is how the indenting works with tabs. Can you adjust your Emacs settings to set tab indenting to 4 spaces?
   (setq-default indent-tabs-mode nil)
   (setq-default tab-width 4)
   (setq indent-line-function 'insert-tab)
   a. Where are the configuration files located for Emacs?
      Check for init.el, possible inside .emacs.d
   b. How do we create a configuration file if none exist?
      Touch, emacs, and other commands can create files.

Elisp Functions

3. Write a function called `is-even` that takes one argument and returns whether it is even.
   a. Example: `(is-even 4)` should evaluate to `t`.
   b. Example: `(is-even 3)` should evaluate to `nil`.

```
(defun is-even (number)
 "Return whether a number is even"
 (interactive "Return whether a number is even")
 (= (% number 2) 0)
 )
```

4. Write a function called `is-current-line-even` that takes no arguments, and uses the `message` function to print out whether the current line number is even or odd.

a. Example: If the current line is 11, `(is-current-line-even)` should print "even" to the echo area.
b. Example: If the current line is 4, `(is-current-line-even)` should print "odd" to the echo area.
c. Add the `interactive` keyword to make your function callable from Emacs.

```
(defun is-current-line-even ()
 "Print out whether the current line number is even or odd"
 (interactive "Print out whether the current line number is even or odd")
 (is-even (line-number-at-pos))
 )
;; this is broken ^^
```

5. Write a function is-in that takes in an integer and a list of integers as parameters and returns whether or not the integer is in the list.

```
(defun is-in (a l)
 (cond ((null l) nil)
       ((= a (car l)) t)
       (t (is-in a (cdr l)))
       )
 )
```

6. Write a function that searches deeply for an integer in a list of integers and lists of integers and lists of lists of integers… and returns whether the integer is found.

```
(defun is-in-deep (a l)
 (cond ((null l) nil)
       ((listp (car l))
            (or (is-in-deep a (car l))
                (is-in-deep a (cdr l))))
       ((equal a (car l)) t)
       (t (is-in-deep a (cdr l)))
       )
 )
```