

Assignment 1 — Lisp in Twelve Lessons Due 18 February

For this assignment, you will go through the material in Chapter 3 of *Successful Lisp* <http://www.psg.com/~dlamkins/sl/contents.html>, subject to the following exceptions.

Lesson 1. All reasonable, nothing to run however.

Lesson 2. Try the code in your Lisp environment. Ignore “A function can return any number of values” section.

Lesson 3. All good, but SET unnecessary.

Lesson 4. Good—CONS, FIRST, and REST very important.

Lesson 5. Good (although the use of function names as variable names is a bad idea).

Lesson 6. Key point: you can change the value of a variable within a local scope (variable in a LET, or parameter within a function body, but it does not affect the variable outside of that scope. LET examples are quite artificial here.

Lesson 7. LAMBDA forms will be more useful with mappers.

Lesson 8. The only useful thing here (to you) is BACKQUOTE—you will not need to define macros, but you might use BACKQUOTE later.

Lesson 9. Skip this.

Lesson 10. Mostly unnecessary, worth a skim at most. Structures and property-lists may be useful later, but are treated too superficially here.

Lesson 11. Poor job, not useful. We will discuss READ and FORMAT as ways to read and write to and from files, as well as keyboard input.

Lesson 12. Skip this.

After having done gone through the lessons, you will need to implement the following:

my-equal-p A general equality tester: takes two arguments, returns `t` if they are equal, `nil` otherwise. Two objects are equal if either 1) they are `eq`, 2) they are numbers that are `=`, or 3) they are lists whose elements are equal (using `my-equal-p`). The only predicates allowed here are `eq`, `numberp`, `=`, and `consp`; specifically you may *not* use `equal` or `equalp`.

prefix-p takes list two arguments. The first argument is a prefix of the second if 1) it is the empty list or 2) its first element is equal to the first element of the second and its tail is a prefix of the second argument’s tail. Return `nil` if the first element is not a prefix of the second, something else otherwise. Use `my-equal-p` for the equality test.

e.g., `(prefix-p '(a b c) '(a (b c) d e f)) => nil`

top-count-symbols takes a list, returns the number of the list’s members that are symbols. You should not worry about nested lists.

e.g., `(top-count-symbols '(a (b c) a ((d) a))) => 2`

nested-count-symbols takes a list, returns the number of symbols in the list (including the symbols in any sublists).

e.g., `(nested-count-symbols '(a (b c) a ((d) a))) => 6`

include-if-true takes a list and a function, returns a list containing all of the elements for which the function applied to the element is true.

e.g., `(include-if-true '(a (b c) a ((d) a)) #'listp) => ((B C)((D) A))`

reversit takes a list, returns a list with the same elements in reverse order. *Note what functions are allowed in list below!*

e.g., `(reversit '(a (b c) ((d) e ((f)))))` => `((D) E ((F))) (B C) A`

flattenit takes a list, returns the “flat” list (that is, one with no nested sublists) of the same elements in the same order.

e.g., `(flattenit '(a (b c) ((d) e ((f)))))` => `(A B C D E F)`

The allowable functions and special forms that you may use in your function definitions are `first`, `rest`, `cons`, `cond`, `defun`, `eq`, `=`, `numberp`, `consp`, `and`, `or`, `not`, `null`, `if`, `append`, `symbolp`, `listp`, `funcall` (as well as those you define using these).

The functions should be tested on data you supply to verify that they work. Keep it clean and simple!

Turn in a transcript and a listing. Good luck.

[NOTE: This assignment requires that you get a lisp environment to work. Clisp is on the lab machines, you can run it in a “command prompt” shell if nothing else. Try the examples we did in lecture. Modify them. Trace them! (See Chapter 16 in Successful Lisp for `TRACE` and `STEP`.)]

Good luck. Enjoy!