

## C SCI 316 (Kong): Lisp Assignment 2 [HW Exercises: Not for submission]

For problems 6 – 13, use a text editor to create a file *solutions.lsp* that contains your function definitions. For more information, see the [Working with Lisp Files](#) section of this document on pp. 3 – 4. Some recommended text editors are mentioned on p. 4.

Lisp's comment character is “;”—this makes Lisp ignore the rest of the line, and is used like “//” in C++ or Java. See the Lisp Toolkit subsection after sec. 5.7 of Touretzky for more about comments.

**IMPORTANT: Do NOT use SETF except when you are answering questions which explicitly indicate that you may use them!**

1. (a) Can an atom be a variable? (b) Can a number be a variable?  
(c) Can a list be a variable? (d) Can an atom have no value?  
(e) Can an atom have more than one value at the same time?  
(f) Can a variable have itself as a value?  
(g) Can a list (A B C) be the value of two different variables?
2. Each of the following may be a single atom, a single list, or neither. Identify each accordingly. For the lists, also say how many elements the list has. [You are ***not*** asked to evaluate any expression—e.g., (+ 1 9) would be a ***list of 3 elements***, even though this list would evaluate to a numeric atom (i.e., 10).]  
(a) ATOMS (b) (THIS IS AN ATOM) (c) ) (  
(d) ((A B) (C D)) 3 (3) (e) (( ) ( )  
(f) ((A B C (g) (/ (+ 3 1) (- 3 1)) (h) (LIST 3)
3. [Exercise 4 on p. 37 of Wilensky] Use SETF to assign X the value (A B C), and then use X to produce the list (A B C A B C).
4. [Exercise 5 on p. 37 of Wilensky] Write the expression ' '(A) using QUOTE rather than '. What is the data type of the expression 'A ?
5. (a) Use SETF to give Y the value (A B). But, instead of writing '(A B), you must use a Lisp function call to create the list (A B).  
(b) Write code that makes the list (D A). However, you must get A from the variable Y, which has the value (A B) from part (a).
6. Define a function called SQR that returns a list of the perimeter and the area of a square, given the length of one side. Thus (SQR 2) should return (8 4).

7. Define QUADRATIC, a function with three parameters A, B and C that returns a list of the two roots of the equation  $Ax^2 + Bx + C = 0$ . You should use the built-in function Sqrt. Recall that the two roots are given by:

$$\frac{-B + \sqrt{B^2 - 4AC}}{2A} \quad \text{and} \quad \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

8. **[Exercise 1 on p. 52 of Wilensky]** Write a Lisp function that computes the area of a circle given its radius. Use the predefined constant PI.
9. Define a function called FTOC which takes as its argument a degree reading in Fahrenheit and returns its Celsius equivalent. (The Celsius equivalent of a Fahrenheit temperature is obtained by subtracting 32 and multiplying by 5/9.)
10. Define a function ROTATE-LEFT which takes a list as argument and returns a new list in which the former first element has become the last element. Thus (ROTATE-LEFT '(A B C D)) should return (B C D A).
11. **[Exercise 4 on pp. 52 – 3 of Wilensky]** A point  $(x, y)$  in the plane can be represented as a two-element list  $(x\ y)$ . Write a Lisp function that takes two such lists as arguments and returns the distance between the corresponding points. Recall that the distance between two points  $(x_1, y_1)$  and  $(x_2, y_2)$  is given by  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ .
12. **[Exercise 5 on pp. 52 – 3 of Wilensky]** Define Lisp functions HEAD and TAIL that behave just like CAR and CDR, respectively.
13. **[Exercise 6 on pp. 52 – 3 of Wilensky]** Define a Lisp function SWITCH that takes as its argument a two-element list and returns a list consisting of the same two elements, but in the opposite order. **Example:** (switch '(A B)) => (B A).
14. Suppose you have just entered the following three Lisp expressions at successive Clisp > prompts (with no spaces before or after \* and + in 8\*7 and 8+7):
- ```
(setf 8*7 5)
(defun 8+7 (x y) (+ x y))
(defun 8*7 () 9)
```
- If you now enter the expression (8+7 (\* 8 7) (8+7 (8\*7) 8\*7)) what value will be printed by Clisp? Check your answer using Clisp.

**The next six questions are important.** Be sure to check your answers using Clisp!

15. [Exercise 1 on pp. 36 – 7 of Wilensky] For each of (a), (b), and (c) below, suppose SETF has just been used to give the variable E the specified value. (E.g., for (a) we suppose (setf e '(a b x d)) has just been entered at Clisp's > prompt.) In each case, write an expression that involves only E, car, and cdr, and which evaluates to the symbol X. [Hint: For a specified value of (A X C), you would be expected to write the expression (car (cdr E)) because the car of the cdr of (A X C) is X.]
- (a) (A B X D)      (b) (A (B (X D)))      (c) (((A (B (X) D))))

16. [Exercise 2 on pp. 36 – 7 of Wilensky] For each of the three lists in exercise 15, write an expression that involves only quoted symbols, NIL, and calls of CONS, and which evaluates to that list. [Hint: For a list (A X C), you would be expected to write the expression (cons 'a (cons 'x (cons 'c nil))).]

**Note:** One way to solve such problems is to first write the list using calls of LIST, and then rewrite the expression using appropriate calls of CONS. (Another approach is to do a preorder traversal of the tree-representations of the given lists—read p. 393 in Sethi for more on the tree-representation of an S-expression.)

For questions 17 – 20, suppose E has been given a value as follows:

```
(setf E
  '((90 91 92 93 94 95 96 97 98 99) (+ 3 4 -) (9 19 29 39 49 59 69 79 89 99)))
```

For each question, write an expression with the specified properties; your expressions may involve E, 'A, 'B and Lisp functions. LIST is a good function to use.

**Example** Write an expression that *does not involve any numbers*, but which evaluates to the following list:

```
(92 (29 39 49 59 69 79 89 99 + 3 4 -))
```

**Solution** (list (third (first E)) (append (rest (rest (third E))) (second E)))  
[Or, equivalently, (list (caddr (car E)) (append (caddr (caddr E)) (cadr E))).]


17. Write an expression that *does not involve any numbers*, but which evaluates to the list ((90 91) 92 93 94 95 96 97 98 99) (A B 29 39 49 59 69 79 89 99)).
18. Write an expression that *does not involve any numbers*, but which evaluates to the list ((90 A 92 93 94 95 96 97 98 99) 3 29 (4 29 39 49 59 69 79 89 99)).
19. Write an expression that *does not involve any numbers*, but which evaluates to the list ((90 91 92 93 94 95 96 97 98 99 3) (+ 3 4 - 29 39 49 59 69 79 89 99)).
20. Write an expression that *does not involve any numbers*, but which evaluates to the list ((A 91 92 93 94 95 96 97 98 99) (90 (19 29) 39 49 59 69 79 89 99)).

### Working with Lisp Files


Suppose xyz.lsp is a file, in your current working directory, that contains one or more Lisp function definitions. Then you can "load" xyz.lsp into Clisp by entering (load "xyz.lsp") at Clisp's > prompt. When you do this, the function definitions in xyz.lsp will be read into Clisp as if you had entered them one by one at Clisp's prompt. [If (load "xyz.lsp") produces an error, there is a syntax error in the function definitions in xyz.lsp.] Unless you are running Clisp as a subprocess of emacs (see below), each time you add a new lisp function definition to a file xyz.lsp or modify an existing definition in the file, it's a good idea to immediately save the file, load the saved file into Clisp, and then test the new or modified function. If you work this way, then whenever (load "xyz.lsp") gives an error message it must be because of a syntax error in the new or modified function.

***For problems 6 – 13, write your function definitions in a file named `solutions.lsp`.*** On `venus` or `euclid` you should create the file in your current working directory—you could use `nano solutions.lsp` to do this.

If you are working on your PC (and have already installed Clisp on the PC), create a folder—`c:\316lisp`, say—on the PC to use as your working directory for Lisp. This can be done, e.g., as follows:


1. Type **Win-r** (i.e., hold down the Windows key and type `r`) to open the Run dialog box.
2. Type `powershell` into the Open: textbox and press  to open a powershell window.
3. In the powershell window, enter `md c:\316lisp` to create the folder `c:\316lisp`.

All the Lisp files you create for this course (such as `solutions.lsp`) should be saved in this folder. After the folder has been created, do the following whenever you want to start Clisp on your PC:











1. Type **Win-r** to open the Run dialog box, type `powershell` into the Open: textbox, and press .
2. In the powershell window, enter `cd c:\316lisp` and enter `clisp` to start Clisp.










Then, assuming you have saved `solutions.lsp` in `c:\316lisp`, you will be able to load `solutions.lsp` into Clisp by entering `(load "solutions.lsp")` at Clisp's prompt.

You should ***not*** use Notepad to create `solutions.lsp` on a PC: It is much better to use an editor that matches parentheses for you. Some examples of such editors are listed at the bottom of this page. I recommend you configure Windows to ***not*** hide file name extensions (such as `.lsp`); this can be done as follows:

1. Type **Win-r** (i.e., hold down the Windows key and type `r`) to open the Run dialog box.
2. Type `control folders` into the Open: textbox and press ; this opens a folder options dialog box.
3. Click on the View tab at the top of the Folder Options dialog box.
4. In the "Advanced settings" window, find the "Hide extensions for known file types" checkbox. If that checkbox is unchecked, click Cancel; otherwise, ***uncheck the checkbox*** and click OK.


### Emacs is a Good Editor for Writing Lisp Code on `euclid` or `venus`

To learn to use emacs on `euclid`, connect to `euclid` using the native ssh client on a PC (in a cmd or powershell window) or a Mac (in a terminal window) and enter `emacs` at the shell prompt. Then (once emacs has started up) type the two characters  `h t` to bring up a tutorial. This editor *automatically* matches parentheses—if you place the cursor at an opening parenthesis or *immediately after* a closing parenthesis, that and the matching parenthesis (if there is one) will be highlighted. Type the two characters   `f` to move the cursor forward across one S-expression—when the cursor is at an opening parenthesis, this moves the cursor to the position that immediately follows the matching closing parenthesis. Conversely, typing   `b` will move the cursor backward across one S-expression—when the cursor immediately follows a closing parenthesis, this moves the cursor back to the matching opening parenthesis. When the cursor is at the start of an S-expression, typing   `k` will delete that S-expression (but typing  `y` will bring it back), whereas typing   `t` will transpose it with the previous S-expression.

You can run Clisp as a subprocess (an “inferior” process) within emacs. To try this, start emacs on `euclid` by entering `emacs test.lsp` at the shell prompt. (Any filename could be used instead of `test.lsp`.) Split the emacs window into two by typing  `x 2` and then enter  `x run-lisp` to get a Clisp `[1]>` prompt in one of the two windows. After that you can switch back-and-forth between editing `test.lsp` and using Clisp simply by typing  `O`. As soon as you have written a new function definition or made a change to a function definition in `test.lsp`, you can load that new/changed definition into Clisp by typing   `X` while the cursor is within or immediately to the right of the definition. This allows you to test new/changed definitions before deciding whether to save the current edited version of `test.lsp`. In the Clisp window, you can cycle backward and forward through previously entered expressions by typing  `p` and  `n`. To “unsplit” the window, type  `x 1` (after which you can switch between the editing and Clisp windows by entering  `b`).

The above instructions also work on `venus` if you connect to `venus` using the native ssh client on a PC (in a cmd or powershell window), provided you have entered `/home/faculty/ykong/316setup` on `venus` at some time this semester as the Lisp Assignment 1 document asked you to do.

### Some Other Editors That Match Parentheses

The `vim` editor on `venus` and `euclid` matches parentheses automatically; also, if when `vim` is in command mode you type `%` at a parenthesis then the cursor will jump to the matching parenthesis. (Enter `vimtutor` on `venus` or `euclid` to bring up a tutorial that teaches beginners to use `vim`.) The `nano` editor on `venus` and `euclid` can match parentheses, but ***not*** automatically: In `nano`, typing  `]` when the cursor is at a parenthesis moves the cursor to the matching parenthesis. Free editors for PCs that support automatic parenthesis matching include:

Notepad++ <https://notepad-plus-plus.org>

VS Code <https://code.visualstudio.com> (***Don't*** use the remote-ssh extension to remotely edit files on `euclid`.)

Atom <https://atom.io>

As with emacs, the `vim` and `nano` editors are preinstalled on Macs for use in a terminal window. VS Code and Atom are also available for Macs. Sublime Text (<https://www.sublimetext.com>) is another good editor for PCs and Macs that automatically matches parentheses; this editor is not free, but can be evaluated for free.