

# CSCE 476/876 Fall 2017

## Lisp Tutorial #2\*

Constraint Systems Laboratory  
University of Nebraska-Lincoln

August 30, 2017

---

Disclaimer: the content of this document includes material borrowed from AI and Lisp text books.

---

If you put your code in a file named `week4.lisp`, then you can first load your code into the lisp environment by the following command:

```
(load "week4.lisp") or :ld week4.lisp
```

Then you compile the file using the following command:

```
(compile-file "week4.lisp") or :cl week4.lisp
```

Some usefull functions to learn:

```
mapcar, reduce, remove-if, apply, funcall, some, every, count-if, eval
```

The first three exercises were already suggested in the previous Lisp recitation.

1. Define a function that takes a list and return the first three items and the last three items. For example, for the list `'(a b c this is a list 1 2 3)`, this function returns `'(a b c 1 2 3)`.
2. Given a list of lists, return the union of these lists. For example, for the list `'((1 2)(1 3)(1 5 6))`, this function returns `'(1 2 3 5 6)`. Do not use the CL primitive `union`.
3. Compute the summation of 1 through a specified positive integer.

---

\*Prepared by previous GTAs of this course: Yaling Zheng and Nick Zielinski.

4. Define a function, `count-letters`, that takes a list and returns the number of every distinct element in this list. Use a hash-table to store the result. For example, for the list '(1 2 1 a b a c), this function returns a hash-table with the following items:

<i>key</i>	<i>val</i>
1	2
2	1
a	2
b	1
c	1

5. Define a function, `count-letter2`, that takes a string and returns the number of every distinct letter in this string. Use a hash-table to store the result. For example, for the string THIS IS A GOOD COURSE, this function returns a hash table with the following items:

<i>key</i>	<i>val</i>
<i>g</i>	1
<i>h</i>	1
<i>i</i>	2
<i>o</i>	3
<i>r</i>	1
<i>s</i>	3
<i>t</i>	1
<i>u</i>	1
<i>Space</i>	4
<i>a</i>	1
<i>c</i>	1
<i>d</i>	1
<i>e</i>	1

6. Define a function, `reachable`, that takes three parameters: a list representing the edges of a directed graph, source vertex  $u$ , and destination vertex  $v$ . The function returns true if  $u$  can reach  $v$  and return false if  $u$  cannot reach  $v$ .

An example of a directed graph represented by edges are '(u1 v1)(u1 v3)(v1 v4)).

7. Define a predicate, `bipartite`, that determines whether or not an undirected graph is bipartite.