Concepts of Programming Languages, CSCI 305, Fall 2018
Extra Scheme Problems

1. Define the function *suffixes* that takes a list and produces a list of all of the suffixes of the original list. For example

   (suffixes '(a b c)) => ((a b c) (b c) (c) ())

```
(define suffixes
       (lambda (l)
              if (null? l)
                     (list l)      ; Same as '(())
                     (cons l (suffixes (cdr l))))
        )
)
```

2. Define the function *myComplex?* that takes three arguments a, b and c and return true if $b^2-4ac<0$.

```
(define myComplex?
       (lambda (a b c)
              (if (>= (* 4 (* a c)) (* b b)) #t #f)
       )
 )
```

3. Define a function *roots* that takes three parameters, a, b and c and returns a list of the two roots of the polynomial ax^2+bx+c using this formula:

   $$x = [-b +/- (b^2 - 4ac)^{1/2}]/(2a)$$

   If the roots will be complex it says so.

   For example:

   (roots 1 2 1) => (-1. -1)
   (roots 2 -6 4) => (2. 1.)
   (roots 1 4 5) => ((-2 1.) (-2 -1.))

   The final solution represents the roots -2+i and -2-i.

   Here is some sample data to test your function:

Discriminate is 0:	(roots 1 2 1) => (-1. -1.)
	(roots 1 8 16) => (-4. -4.)
	(roots 1 -4 4) => (2. 2.)
Discriminate > 0:	(roots 2 -6 4) => (2. 1.)
	(roots 1 1 -30) => (5. -6.)
	(roots 1 4 -1) => (0.236068 -4.23607)
Discriminate < 0:	(roots 1 4 5) => ((-2 1.) (-2 -1.))
	(roots 4 4 2) => ((-1/2 0.5) (-1/2 -0.5))
	(roots 1 1 1) => ((-1/2 0.866025) (-1/2 -0.866025))
	(roots 4 1 1) => ((-1/8 0.484123) (-1/8 -0.484123)

Create a library of Scheme functions that use lists to simulate sets. Remember that for sets:

- Order doesn't matter
- No duplicates

While we know that the following sets are equal {0 1 2 } = {0 2 1} = {0 0 2 1 1 1} the final set may be difficult to work with. Therefore, your library should not allow duplicate elements in any list that represents a set.

4. Create a Scheme function `element-of?` which takes an element and a list representing a set, and returns true if the element is in the list and false otherwise.

5. Create a Scheme function `validateSet` which takes a list and returns true if the list represents an acceptable set, and false otherwise.
   The following lists represent acceptable sets:
   () – represents the empty set
   (apple 1 4) – represents a set with three elements
   ((3 4 4 4) (1 2 3)) – represents a set with two elements (It doesn't matter if the elements themselves represent lists.)

6. Create a Scheme function `union` which takes two lists representing sets and returns a list representing the union of the sets (making sure that no duplicates exist in the results).

7. Create a Scheme function `intersection` which takes two lists representing sets and returns a list representing the intersection of the sets

8. Create a Scheme function `difference` which takes two lists representing sets and returns a list representing a set containing the elements in the first set which are not also in the second set.

   Feel free to create other supporting functions for your set library.