

Concepts of Programming Languages, CSCI 305, Fall 2021
Lab 2, Scheme, Sept. 3

1. Define *palindromize* that takes a list of arguments and returns a list that is twice as long and reads the same forwards and backwards.

```
(palindromize `(a b c)) returns `(a b c c b a)
(define palindromize
  (lambda (myList)
    (append
      myList
      (reverse myList)
    )
  )
)
```

2. Define *rotate-L*, a function that takes a list as its argument and returns a new list in which the former first element becomes the last. For example

```
(rotate-L `(a b c)) returns `(b c a)
```

```
(define rotate-L
  (lambda (myList)
    (append
      (cdr myList)
      (list
        (car myList)
      )
    )
  )
)
```

3. Define *rotate-R*, which is just like *rotate-L* except that it rotates in the other direction.

```
(rotate-R `(a b c)) returns `(c a b)
```

```
(define rotate-R
  (lambda (myList)
    (reverse (rotate-L (reverse myList) ) )
  )
)
```

4. Define the function *containsNested?* that takes an item and a possibly nested list and returns #t if the item appears anywhere within the list and #f not. For example

```
(containsNested? 'a '(b (c (a)) d)) => #t  
(containsNested? '(1 2 3) '(x (y (1 2 3)) (1 2))) => #t  
(containsNested? '(1) '(x y 1 2)) => #f
```

```
(define containsNested?  
  (lambda (item myList)  
    (cond  
      ((null? myList) #f)  
      ((equal? item (car myList)) #t)  
      ((list? (car myList))  
       (or (containsNested? item (car myList))  
           (containsNested? item (cdr myList))))  
      (else (containsNested? item (cdr myList))))  
    )  
  )  
)
```

5. Define the function *insertIntoSortedList* which takes a number and a list of numbers which is sorted in ascending order and returns the new list which is the same as the old list but with the new number inserted into the appropriate place.

```
(insertIntoSortedList 5 '(3 4 5 7 8)) => (3 4 5 5 7 8)  
(insertIntoSortedList 5 '()) => (5)
```

```
(define insertIntoSortedList  
  (lambda (num lst)  
    (cond  
      ((null? lst) (list num))  
      ((< num (car lst)) (cons num lst))  
      (else (cons (car lst)  
                  (insertIntoSortedList num (cdr lst))))  
    )  
  )  
)
```

6. Define the function *mySort* which takes a list and sorts it into ascending order.
 (*mySort* '(3 1 4 10 5 5 7 8)) => (1 3 4 5 5 7 8 10)

```
(define mySort
  (lambda (lst)
    (cond
      ((null? lst) lst)
      (else (insertIntoSortedList
              (car lst)
              (mySort (cdr lst))))))
  )
)
```

7. Define the function *sortBetter* which takes a list and a comparator, either '<' or '>' and sorts the list into ascending order, for '<', and descending order for '>'.
 (*sortBetter* '(3 1 4 10 5 5 7 8) <) => (1 3 4 5 5 7 8 10)
 (*sortBetter* '(3 1 4 10 5 5 7 8) >) => (10 8 7 5 5 4 3 1)

```
(define insertIntoSortedListBetter
  (lambda (num lst comparator)
    (cond
      ((null? lst) (list num))
      ((comparator num (car lst)) (cons num lst))
      (else
       (cons (car lst)
              (insertIntoSortedListBetter
               num
               (cdr lst)
               comparator))))
  )
)

(define sortBetter
  (lambda (lst comparator)
    (cond
      ((null? lst) lst)
      (else
       (insertIntoSortedListBetter
        (car lst)
        (sortBetter (cdr lst)
                    comparator)
        comparator)))
  )
)
```