Problem 1.

The following is a definition of a sort function

```scheme
;; sort : Listof[Number]  ->  Listof[Number]
;; to construct a list with all items from alon in increasing order
(define (sort-a alon)
  (local ((define (insert an alon)
         (cond
          [(empty? alon) (list an)]
          [else (cond
                  [(< an (first alon)) (cons an alon)]
                  [else (cons (first alon)
                               (insert an (rest alon)))]))))

  (cond
   [(empty? alon) empty]
   [else (insert (first alon) (sort-a (rest alon)))])))
```

Design an abstracted version of the `sort-a` function which consumes the comparison as an additional argument and uses a loop function. Use this function to design `sort-ascending` and `sort-descending`, which sort a `Listof[Number]` in ascending and descending order, respectively.

Problem 2:
DrRacket has lots of great abstract functions for processing lists (pg. 313, Sect. 21.2, or the [online version](#)).

Given the following data definitions:

```scheme
;; A Grade is: (make-grade Symbol Number)
(define-struct grade (letter num))
```
The Symbol in a Grade represents

'A >= 90
'B >= 80
'C >= 70
'D >= 60
'F < 60

A [Listof Grades] ...
(define grades
  (list (make-grade 'D 62) (make-grade 'C 79) (make-grade 'A 93)
        (make-grade 'B 84) (make-grade 'F 57) (make-grade 'F 38)
        (make-grade 'A 90) (make-grade 'A 95) (make-grade 'C 76)
        (make-grade 'A 90) (make-grade 'F 55) (make-grade 'C 74)
        (make-grade 'A 92) (make-grade 'B 86) (make-grade 'F 43)
        (make-grade 'C 73)))

Design the requested functions to manipulate Grades. You must use the given list as one of your tests.

For each you may use a local function or an anonymous (lambda) function.

Note: if you do not use a DrRacket loop function, you will not receive credit for the sub-problem!

1. Design the function log->los that converts a [listof Grade] into a [Listof Symbol] that contains just the letter grade.
2. Design the function average-grade that finds the average (number) Grade in a [Listof Grade].
3. Design a function all-above-79 that returns a list of only the grades that are above 79.
4. Design the function all-pass? that checks to see if all the Grades in a given list are not 'F.
5. Finally design the function bonus that adds 5 to all of the Grades in a given list, and updates the letter portion of the Grade if it changes. Your function must return a [Listof Grade]!

HtDP Problems:
26.1.1, 26.1.2