```
;; keep-lt-5 :  list of numbers -> list of numbers
;; Purpose:  keeps all input numbers less than 5
(define (keep-lt-5 alon)
    (cond
      [(empty? alon)   empty]
      [(cons?   alon)
         (cond
            [(< (first alon) 5)
              (cons (first alon) (keep-lt-5 (rest alon)))]
            [else  (keep-lt-5 (rest alon))]
         )]
     ))
```

```
;; keep-lt-9 :  list of numbers -> list of numbers
;; Purpose:  keeps all input numbers less than 9
(define (keep-lt-9 a-lon)
    (cond
      [(empty? alon)   empty]
      [(cons?   alon)
         (cond
            [(< (first alon) 9)
              (cons (first alon) (keep-lt-9 (rest alon)))]
            [else  (keep-lt-9 (rest alon))] )
      ]  ))
```

```
;; keep-lt:  number   list-of-numbers -> list-of-numbers
;; Purpose:  keep all input numbers that are less than the
;;              given number
(define (keep-lt  num  alon)
   (cond
      [(empty? alon)  empty]
      [(cons?   alon)
      (cond
         [(< (first alon) num)
           (cons (first alon)  (keep-lt num (rest alon)))]
         [else  (keep-lt  num  (rest alon))] )
      ] ))
```

```
;; keep-lt:  number   list-of-numbers -> list-of-numbers
;; Purpose:  keep all input numbers that are less than the
;;               given number
(define (keep-lt  num  alon)
    (local
        [(define (filter-lt alon)
            (cond
                [(empty? alon)  empty]
                 [(cons? alon)
                 (cond
                     [(< (first alon) num)
                       (cons (first alon)  (filter-lt (rest alon)))]
                     [else  (filter-lt (rest alon))] ) ] ))

        ]
        (filter-lt alon)
     ))
```

```
(define (keep-lt-5 alon)
    (keep-lt  5 alon))
(define (keep-lt-9 alon)
    (keep-lt   9 alon))
```

```
;; keep-gt-5 :  list of numbers -> list of numbers
;; Purpose:  keeps all input numbers greater than 5
(define (keep-gt-5 alon)
    (cond
      [(empty? alon)   empty]
      [(cons?   alon)
         (cond
           [(> (first alon) 5)
             (cons (first alon) (keep-gt-5 (rest alon)))]
           [else  (keep-gt-5 (rest alon))]
         )]
    ))
```

```
;; keep-rel-5 : (num num -> num) list of num -> list of num
;; Purpose: keep all input numbers that have relation than 5
(define (keep-rel-5 relation alon)
    (cond
      [(empty? alon)   empty]
      [(cons?   alon)
         (cond
            [(relation (first alon) 5)
              (cons (first alon)
                       (keep-rel-5 relation (rest alon)))]
            [else  (keep-relation-5 (rest alon))]
         )]
     ))
(define (keep-lt-5 alon)
   (keep-rel-5  < alon))
(define (keep-gt-5 alon)
    (keep-rel-5 >  alon))
```

```
;; keep-rel-5 : (num num -> num) list of num -> list of num
;; Purpose: keep all input numbers that have relation than 5
(define (keep-rel-5 relation alon)
   (local
      [(define (filter-rel alon)
       (cond
          [(empty? alon)   empty]
          [(cons?   alon)
            (cond
               [(relation (first alon) 5)
                 (cons (first alon) (filter-rel (rest alon)))]
               [else  (filter-rel (rest alon))] )]  ))
     ]
     (filter-rel alon)))

       (define (keep-lt-5 alon)
   (keep-rel-5  < alon))
```

```
;; keep-rel:
;; (num num -> num) num  list-of-nums -> list-of-nums
;; Purpose: keep all the numbers in the input list that have
;;     the relation given by the function argument to the
;;     number argument  (whew!)
(define (keep-rel   relation num alon)
   (local [(define (filter-rel  alon)  ;; relation & num are invariant
             (cond
              [(empty?  alon)  empty]
              [(cons?   alon)
                   (cond
                     [(relation (first alon) num)
                       (cons (first alon) (filter-rel (rest alon)))]
                     [else (filter-rel (rest alon))] )
              ] ))
        ]
        (filter-rel alon) ))

(define (keep-gt-9 alon)
          (keep-rel  >  9 alon))
```