```
;;
;; ************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 09, Problem 2
;; ************************************************
;;

(define locoins '((nickel 5)
                  (dime 10)
                  (quarter 25)
                  (loonie 100)
                  (toonie 200)))


;; 3a)

;;(count-change coin-lst) produces the value of all coins in coin-lst
;; count-change: (listof Sym) -> Nat
;; Examples:
(check-expect (count-change empty) 0)
(check-expect (count-change '(dime quarter loonie)) 135)

(define (count-change los)
  (foldr (lambda (coin rror)
           (local
             [(define look-up-res (filter (lambda (denom)
                                            (symbol=? (first denom) coin))
                                          locoins))]
             (cond
               [(empty? look-up-res) rror]
               [else (+ (second (first look-up-res)) rror)]))) 0 los))

;; Tests:
(check-expect (count-change '(nickel)) 5)
(check-expect (count-change '(dime)) 10)
(check-expect (count-change '(quarter)) 25)
(check-expect (count-change '(loonie)) 100)
(check-expect (count-change '(toonie)) 200)
(check-expect (count-change '(button)) 0)
(check-expect (count-change '(toonie button toonie)) 400)
(check-expect (count-change '(nickel dime quarter loonie toonie)) 340)



;; 3b)

(define rounding-breakpoint 3)

;;(make-change cents) produces a list of coins adding up to cents
;; make-change: Nat -> (listof Str)
;; Examples:
(check-expect (make-change 0) empty)
(check-expect (count-change (make-change 40)) 40)
(check-expect (make-change 12) '(dime))

(define (make-change cents)
  (local
    [;;(my-append lst1 lst2) adds the elements of lst1 to lst2
     ;; my-append: (listof X) (listof Y) -> (listof (anyof X Y))
     (define (my-append lst1 lst2) (foldr cons lst2 lst1))

     (define rounded-cents
```

```
       (cond
         [(>= (remainder cents (second (first locoins))) rounding-breakpoint)
          (+ cents (- (second (first locoins))
                      (remainder cents (second (first locoins)))))]
         [else cents]))]
     (foldr (lambda (denom coins-so-far)
              (local
                [(define cents-needed (- rounded-cents (count-change coins-so-far)))
                 (define can-take (quotient cents-needed (second denom)))]
                (my-append (build-list can-take (lambda (i) (first denom)))
                           coins-so-far)))
            empty locoins)))


;; Tests:
(check-expect (make-change 2) empty)
(check-expect (make-change 3) '(nickel))
(check-expect (make-change 4) '(nickel))
(check-expect (make-change 5) '(nickel))
(check-expect (make-change 9) '(dime))
(check-expect (make-change 10) '(dime))
(check-expect (make-change 11) '(dime))
(check-expect (make-change 22) '(dime dime))
(check-expect (make-change 23) '(quarter))
(check-expect (make-change 98) '(loonie))
(check-expect (make-change 198) '(toonie))
(check-expect (make-change 600) '(toonie toonie toonie))
(check-expect (count-change (make-change 15)) 15)
(check-expect (count-change (make-change 197)) 195)
(check-expect (count-change (make-change 97)) 95)
(check-expect (count-change (make-change 342)) 340)




;;
;; ************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 09, Problem 3
;; ************************************************
;;


;;(sequence n1 n2 type) produces a function that generates a sequence
;;  of a given type whose first elements are n1, n2
;; sequence: Num Num (anyof 'arithmetic 'geometric) -> (Nat -> (listof Num))
;; requires: n1,n2 =/= 0 if type is 'geometric
;; Examples:
(check-expect ((sequence 1 2 'arithmetic) 0) empty)
(check-expect ((sequence 1 2 'arithmetic) 5) '(1 2 3 4 5))
(check-expect ((sequence 4 8 'geometric) 4) '(4 8 16 32))

(define (sequence n1 n2 type)
  (cond
    [(symbol=? type 'arithmetic)
     (lambda (len) (build-list len (lambda (i) (+ n1 (* i (- n2 n1))))))]
    [else (lambda (len) (build-list len (lambda (i) (* n1 (expt (/ n2 n1) i)))))]))

;; Tests:
(check-expect ((sequence 1 2 'arithmetic) 1) '(1))
(check-expect ((sequence 1 2 'arithmetic) 2) '(1 2))
(check-expect ((sequence 1 1 'arithmetic) 4) '(1 1 1 1))
```

```
(check-expect ((sequence 2 1 'arithmetic) 5) '(2 1 0 -1 -2))
(check-expect ((sequence 1 2 'geometric) 0) empty)
(check-expect ((sequence 1 2 'geometric) 1) '(1))
(check-expect ((sequence 1 2 'geometric) 2) '(1 2))
(check-expect ((sequence 1 1 'geometric) 4) '(1 1 1 1))
(check-expect ((sequence 2 1 'geometric) 5) '(2 1 0.5 0.25 0.125))



;;
;; *************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 09, Problem 4
;; *************************************************
;;

;; 4a)

;;(rotate-right lst) brings the last element of lst to the front
;; rotate-right: (listof X) -> (listof X)
;; Examples:
(check-expect (rotate-right empty) empty)
(check-expect (rotate-right '(1 2 3 4 5)) '(5 1 2 3 4))

(define (rotate-right lst)
  (foldr (lambda (frst rror)
           (cond
             [(empty? rror) (list frst)]
             [else (cons (first rror) (cons frst (rest rror)))]))
         empty lst))

;; Tests:
(check-expect (rotate-right '(1)) '(1))
(check-expect (rotate-right '(1 2)) '(2 1))



;; 4b)

;;(rotate-left lst) brings the first element of lst to the end
;; rotate-left: (listof X) -> (listof X)
;; Examples:
(check-expect (rotate-left empty) empty)
(check-expect (rotate-left '(1 2 3 4 5)) '(2 3 4 5 1))

(define (rotate-left lst)
  (cond
    [(empty? lst) empty]
    [else (foldr cons (list (first lst)) (rest lst))]))

;; Tests:
(check-expect (rotate-left '(1)) '(1))
(check-expect (rotate-left '(1 2)) '(2 1))



;; 4c)


;;(len lst) produces the length of lst
;; len: (listof Any) -> Nat
;; Examples:
```

```
(check-expect (len empty) 0)
(check-expect (len '(1 2 3 4 5)) 5)

(define (len lst) (foldr (lambda (frst rror) (add1 rror)) 0 lst))

;; Test:
(check-expect (len '(1)) 1)


;;(prefix n lst) produces the first n elements of lst
;; prefix: Nat (listof X) -> (listof X)
;; Examples:
(check-expect (prefix 2 empty) empty)
(check-expect (prefix 0 '(1 2 3 4)) empty)
(check-expect (prefix 3 '(1 2 3 4)) '(1 2 3))
(check-expect (prefix 5 '(1 2 3 4)) '(1 2 3 4))

(define (prefix n lst)
  (foldr (lambda (frst rror)
           (cond
             [(> n (len rror)) (cons frst rror)]
             [else (rest (rotate-right (cons frst rror)))]))
         empty lst))

;; Tests:
(check-expect (prefix 1 '(1 2 3)) '(1))
(check-expect (prefix 2 '(1 2 3)) '(1 2))
(check-expect (prefix 3 '(1 2 3)) '(1 2 3))
(check-expect (prefix 4 '(1 2 3)) '(1 2 3))



;; 4d)

;;(insert-at position new lst) inserts new after the element in
;;  position of lst
;; insert-at: Nat X (listof Y) -> (listof (anyof X Y))
;; Examples:
(check-expect (insert-at 2 'a empty) '(a))
(check-expect (insert-at 2 'a '(b c d e)) '(b c a d e))

(define (insert-at position new lst)
  (local
    [(define elems-after (max 0 (- (len lst) position)))]
  (foldr (lambda (frst rror)
           (cond
             [(< elems-after (len rror))
              (cons frst rror)]
             [else (cons (first rror) (cons frst (rest rror)))]))
         (list new) lst)))

;; Tests:
(check-expect (insert-at 0 'a '()) '(a))
(check-expect (insert-at 0 'a '(b c d)) '(a b c d))
(check-expect (insert-at 1 'a '(b c d)) '(b a c d))
(check-expect (insert-at 2 'a '(b c d)) '(b c a d))
(check-expect (insert-at 3 'a '(b c d)) '(b c d a))
(check-expect (insert-at 100 'a '(b c d)) '(b c d a))
```