```
;;
;; ***************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 04, Problem 1
;; ***************************************************
;;


(define vowels (string->list "aeiouAEIOU"))


;;(count-vowels/lst char-list) produces the number of vowels in char-list
;; count-vowels/lst: (listof Char) -> Nat
;; Examples and tests: see wrapper function count-vowels

(define (count-vowels/lst char-list)
  (cond
    [(empty? char-list) 0]
    [(member? (first char-list) vowels)
     (add1 (count-vowels/lst (rest char-list)))]
    [else (count-vowels/lst (rest char-list))]))


;;(count-vowels str) produces the number of vowels in str
;; count-vowels: Str -> Nat
;; Examples:
(check-expect (count-vowels "") 0)
(check-expect (count-vowels "abc") 1)

(define (count-vowels str)
  (count-vowels/lst (string->list str)))

;; Tests:
(check-expect (count-vowels "a") 1)
(check-expect (count-vowels "b") 0)
(check-expect (count-vowels "bbbbbbbbbabbbbbb") 1)
(check-expect (count-vowels "bbbbbbbbbbbbbbbb") 0)
(check-expect (count-vowels "aeiou") 5)
(check-expect (count-vowels "aeioubbbbbubobibeba") 10)



;;
;; ***************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 04, Problem 2
;; ***************************************************
;;


;;(remove-duplicates lst) removes all but the last
;;  occurrence of each number in lst
;; remove-duplicates: (listof Num) -> (listof Num)
;; Examples:
(check-expect (remove-duplicates empty) empty)
(check-expect (remove-duplicates (cons 1 (cons 2 (cons 2 (cons 1 empty)))))
              (cons 2 (cons 1 empty)))

(define (remove-duplicates lst)
  (cond
    [(empty? lst) empty]
    [(member? (first lst) (rest lst))
     (remove-duplicates (rest lst))]
    [else (cons (first lst)
                (remove-duplicates (rest lst)))]))

;; Tests:
(check-expect (remove-duplicates (cons 1 (cons 2 (cons 3 (cons 4 (cons 5 empty))))))
              (cons 1 (cons 2 (cons 3 (cons 4 (cons 5 empty))))))
(check-expect (remove-duplicates (cons 1 (cons 1 (cons 1 (cons 1 empty)))))
              (cons 1 empty))
(check-expect (remove-duplicates (cons 1 (cons 1 empty)))
              (cons 1 empty))
(check-expect (remove-duplicates (cons 1 (cons 2 (cons 3 (cons 1 empty)))))
              (cons 2 (cons 3 (cons 1 empty))))



;;
;; ***************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 04, Problem 3
;; ***************************************************
;;


(define unrecognized-val 0)
(define nickel-val 5)
(define dime-val 10)
(define quarter-val 25)
(define loonie-val 100)
(define toonie-val 200)


;;(coin-value coin) determines the value of coin
;; coin-value: Sym -> Nat
;; Examples:
(check-expect (coin-value 'dime) 10)
(check-expect (coin-value 'button) 0)

(define (coin-value coin)
  (cond
    [(symbol=? coin 'nickel) nickel-val]
    [(symbol=? coin 'dime) dime-val]
    [(symbol=? coin 'quarter) quarter-val]
    [(symbol=? coin 'loonie) loonie-val]
    [(symbol=? coin 'toonie) toonie-val]
    [else unrecognized-val]))


;;(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 (cons 'dime (cons 'quarter (cons 'loonie empty))))
              135)

(define (count-change coin-lst)
  (cond
    [(empty? coin-lst) 0]
    [else (+ (coin-value (first coin-lst))
             (count-change (rest coin-lst)))]))

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


(define rounding-breakpoint 3)


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

(define (make-change total)
  (cond
    [(>= (remainder total nickel-val) rounding-breakpoint)
     (make-change (+ total (- nickel-val (remainder total nickel-val))))]
    [(>= total toonie-val) (cons 'toonie (make-change (- total toonie-val)))]
    [(>= total loonie-val) (cons 'loonie (make-change (- total loonie-val)))]
    [(>= total quarter-val) (cons 'quarter (make-change (- total quarter-val)))]
    [(>= total dime-val) (cons 'dime (make-change (- total dime-val)))]
    [(>= total nickel-val) (cons 'nickel (make-change (- total nickel-val)))]
    [else empty]))

;; Tests:
(check-expect (make-change 2) empty)
(check-expect (make-change 3) (cons 'nickel empty))
(check-expect (make-change 4) (cons 'nickel empty))
(check-expect (make-change 5) (cons 'nickel empty))
(check-expect (make-change 9) (cons 'dime empty))
(check-expect (make-change 10) (cons 'dime empty))
(check-expect (make-change 11) (cons 'dime empty))
(check-expect (make-change 15) (cons 'dime (cons 'nickel empty)))
(check-expect (make-change 22) (cons 'dime (cons 'dime empty)))
(check-expect (make-change 23) (cons 'quarter empty))
```

```
(check-expect (make-change 97)
              (cons 'quarter (cons 'quarter (cons 'quarter
                                                  (cons 'dime (cons 'dime empty))))))
(check-expect (make-change 98) (cons 'loonie empty))
(check-expect (make-change 197)
              (cons 'loonie (cons 'quarter (cons 'quarter (cons 'quarter
                                                  (cons 'dime (cons 'dime empty)))))))
(check-expect (make-change 198) (cons 'toonie empty))
(check-expect (make-change 600) (cons 'toonie (cons 'toonie (cons 'toonie empty))))
(check-expect (make-change 342)
              (cons 'toonie (cons 'loonie (cons 'quarter (cons 'dime (cons 'nickel empty))))))
(check-expect (make-change 1200)
              (cons 'toonie (cons 'toonie
                                  (cons 'toonie
                                        (cons 'toonie (cons 'toonie (cons 'toonie empty)))))))


;;
;; **************************************************
;; Sample Solutions
;; CS 135 Fall 2019
;; Assignment 04, Problem 4
;; **************************************************
;;



;;4a)

;;(has-divisor? n div) determines if n has any non-1
;;  divisors less than or equal to div
;; has-divisor?: Nat Nat -> Bool
;; requires: 1 <= div < n
;; Examples:
(check-expect (has-divisor? 2 1) false)
(check-expect (has-divisor? 4 2) true)
(check-expect (has-divisor? 29 15) false)

(define (has-divisor? n div)
  (cond
    [(= 1 div) false]
    [(= 0 (remainder n div)) true]
    [else (has-divisor? n (sub1 div))]))


;;(prime? n) determines if n is prime
;; prime?: Nat -> Bool
;; Examples:
(check-expect (prime? 1) false)
(check-expect (prime? 4) false)
(check-expect (prime? 37) true)

(define (prime? n)
  (cond
    [(>= 1 n) false]
    [else (not (has-divisor? n (sub1 n)))]))
```

```
;; Tests:
(check-expect (prime? 0) false)
(check-expect (prime? 36) false)
(check-expect (prime? 37) true)



;;4b)

;;(next-prime n) determines the next prime
;; strictly larger than n
;; next-prime: Nat -> Nat
;; Examples:
(check-expect (next-prime 1) 2)
(check-expect (next-prime 37) 41)

(define (next-prime n)
  (cond
    [(prime? (add1 n)) (add1 n)]
    [else (next-prime (add1 n))]))

;; Tests:
(check-expect (next-prime 0) 2)
(check-expect (next-prime 2) 3)
(check-expect (next-prime 36) 37)



;;4c)

;;(prime-range start end) produces all primes between
;;  start and end, inclusive
;; prime-range: Nat Nat -> (listof Nat)
;; Examples:
(check-expect (prime-range 1 10) (cons 2 (cons 3 (cons 5 (cons 7 empty)))))
(check-expect (prime-range 10 1) empty)

(define (prime-range start end)
  (cond
    [(> start end) empty]
    [(prime? start) (cons start (prime-range (add1 start) end))]
    [else (prime-range (add1 start) end)]))

;; Tests:
(check-expect (prime-range 10 10) empty)
(check-expect (prime-range 11 11) (cons 11 empty))
(check-expect (prime-range 14 16) empty)
(check-expect (prime-range 10 13) (cons 11 (cons 13 empty)))
(check-expect (prime-range 13 16) (cons 13 empty))
```