Shane O Grady 16357921

CT331 Assignment 2

Question 1 A):

```
#lang racket

;A cons pair of two numbers
(cons 4 5)

;A list of 3 numbers, using only the cons function.
(cons 4(cons 5 6))

;A list containing a string, a number and a nested list of three numbers, using only the cons function.
(cons "Shane"(cons 7 '(8 9 10)))

;A list containing a string, a number and a nested list of three numbers, using only the list function.
(list "Shane" '2 '(3 4 5))

;A list containing a string, a number and a nested list of three numbers, using only the append function.
(append '(Shane) '(1) '((2 3 4)))
```

```
Welcome to DrRacket, version 7.1 [3m].
Language: racket, with debugging; memory limit: 128 MB.
'(4 . 5)
'(4 5 . 6)
'("Shane" 7 8 9 10)
'("Shane" 2 (3 4 5))
'(Shane 1 (2 3 4))
>
```

B)

Cons function constructs memory objects which holds two values or pointers to values.

List function constructs a list from components.

Append collects several components from several lists into one list.

Unlike the list and cons function, the append function does not accept strings.

Question 2:

```
#lang racket

(provide ins_beg)
(provide ins_end)
(provide cout_top_level)
(provide count_instances)
(provide count_instances_tr)
(provide count_instances_deep)

;Part 1
(define (ins_beg element lst)
  (display "Hello, I'm ins_beg!\n")
  (cons element lst))

;Part 2
(define (ins_end element lst)
  (display "ins_end!\n")
  (append lst (list element)))

;Part 3
(define (cout_top_level list)
  (display "cout_top_level\n")
  (define a 0)
  (define (count lst)
      (if  (empty? lst)
           (display "Elements counted: ")
           (begin
             (set! a (+ a 1))
             (count (cdr lst)))))
  (count list)
   (display a))
```

```
;Part 4
(define (count_instances el list)
  (display "count_instances\n")
  (define a 0)
  (define (count lst)
  (if (empty? lst)
      (display "Instances counted: ")
      (begin
        (if (eq? el (car lst))
            (set! a (+ a 1))
            (set! a a))
        (count (cdr lst)))))
  (count list)
  (display a))

;Part 5
(define (count_instances_tr element list)
  (display "count_instances_tr\n")
  (define (count lst total)
    (if (empty? lst)
        (begin
        (display "Instances counted: ")
        (display total))
        (begin
          (if (eq? element(car lst))
            (count (cdr lst) (+ total 1))
            (count (cdr lst) total)))
        ))
  (count list 0))
```

```
;Part 6
(define (count_instances_deep element list)
  (display "count_instances_deep\n")
  (define a 0)
  (define (count lst)
    (if (empty? lst)
        (void)
        (begin
          (if (eq? element (car lst))
              (begin
                (set! a (+ a 1))
                (count (cdr lst)))
              (begin
                (if (number? (car lst))
                    (count (cdr lst))
                    (begin
                      (count (car lst))
                      (count (cdr lst))))))))))
  (count list)
  (display "Instances counted: ")
  (display a))
```

Question 3:

```
#lang racket

;Part 1
(define (show_tree tree)
  (begin
    (cond
      [(empty? (car tree)) (display " ")]
      [else (show_tree (car tree))]
      )
    (display (car (cdr tree)))
    (cond
      [(empty? (caddr tree)) (display " ")]
      [else (show_tree (car (cdr (cdr tree))))]
      )
    ))
(display "Part 1: ")(show_tree '((() 10 ())13 ((() 17 ()) 19 (() 21 ()))))
(display "\n");

;Part 2
(define (search item lst)
  (cond
    [(empty? lst) #t]
    [(equal? item (car (cdr lst))) #f]
    [(< item (car (cdr lst))) (search item (car lst))]
    [else (search item (car (cdr (cdr lst))))]
    ))

(display "\nPart 2: ")
(search 4 '((() 10 ()) 13 ((() 17 ()) 19 (() 21 ()))))
(display "\n");
```

```
#lang racket

;Part 3
(define (insert_value item lst)
  (cond
    [(empty? lst) (append lst (list '() item '()))]
    [(equal? item (cadr lst)) "item found"]
    [(< item (cadr lst)) (list (insert_value item (car lst)) (cadr lst) (caddr lst))]
    [else (list (car lst) (cadr lst) (insert_value item (caddr lst)))]
))

(display "Part 3:  List: ")
(define tree '((() 22 ()) 24 ((() 26 ()) 28 (() 29 ()))))
(show_tree tree)
(display "\n          Inserting item into list:  ")
(define tree4 (insert_value 30 tree))
(show_tree tree4)
(display "\n");

;Part 4
(define (insert_list lst tree)
  (cond
   [(empty? lst) tree]
   [else (insert_list (cdr lst) (insert_value (car lst) tree))]
  ))

(display "\nPart 4: ")
(insert_list '(21 34 26 19 12) '())
(display "\n");
```

```
#lang racket

;Part 5
(define (tree-sort lst)
  (show_tree (insert_list lst '()))
  )

(display "Part 5: ")
(tree-sort '(21 34 26 19 12))
(display "\n");

;Part 6
(define (ascension_descension_tree_sort lst fn)
  (show_tree (list_into_tree lst '() fn))
  )

(define (list_into_tree lst tree fn)
  (cond
   [(empty? lst) tree]
   [else (list_into_tree (cdr lst) (el_into_tree (car lst) tree fn) fn)]
  ))

(define (el_into_tree item lst fn)
  (cond
    [(empty? lst) (append lst (list '() item '()))]
    [(equal? item (cadr lst)) "item found"]
    [(fn item (cadr lst)) (list (el_into_tree item (car lst) fn) (cadr lst) (caddr lst))]
    [else (list (car lst) (cadr lst) (el_into_tree item (caddr lst) fn))]
))

(display "\nPart 6:   ascending: ")
(ascension_descension_tree_sort '(98 22 44 36 74 13 ) <)
(display "\n          descending: ")
(ascension_descension_tree_sort '(98 22 44 36 74 13) >)
(display "\n          ascending based on last digit: ")
```

Determine language from source ▾

```
Welcome to DrRacket, version 7.1 [3m].
Language: racket, with debugging; memory limit: 128 MB.
Part 1:  10 13 17 19 21

Part 2: #t

Part 3:  List:  22 24 26 28 29
         Inserting item into list:   22 24 26 28 29 30

Part 4: '((((() 12 ()) 19 ()) 21 ((() 26 ()) 34 ()))

Part 5:  12 19 21 26 34

Part 6:   ascending:  13 22 36 44 74 98
          descending:  98 74 44 36 22 13
          ascending based on last digit:
> |
```

Extra Credit Test Question 2:

```
#lang racket

(require (file "assignment_q2.rkt")
         (file "assignment_q3.rkt"))

;
;Don't worry about this file unless you are doing the extra credit tests.
;

;This structure allows a single function call
;to run every test in sequence, rather than
;calling each function separately.
(define (runTests)
  (begin
    (display "\n")
    (display "\n")
    (display "Running tests...\n")
    ;begin calling test functions
    (printf "Part 1: ~a\n" (test_ins_beg1))
    (printf "Part 2: ~a\n" (ins_end 7 '(8 9 10)))
    (printf "\n");
    (printf "Part 3: ") (cout_top_level '(15 (16 17 18) 12))
    (printf "\n");
    (printf "\nPart 4: ") (count_instances 7 '(7 8 9 7 9 10 7 8 9 7))
    (printf "\n");
    (printf "\nPart 5: ") (count_instances_tr 9 '(7 8 9 7 9 10 7 8 9 7))
    (printf "\n");
    (printf "\nPart 6: ") (count_instances_deep 9 '(7 8 9 (7 9 10) 7 8 9 7))
    (printf "\n");

    ;end calling test functions
    (display "\nTests complete!\n")))

;Begin test functions
(define (test_ins_beg1)
  (eq? (ins_beg 1 '(2 3 4)) '(1 2 3 4)))
```

Determine language from source ▾

```
Part 1: #f
ins_end!
Part 2: (8 9 10 7)

Part 3: cout_top_level
Elements counted: 3

Part 4: count_instances
Instances counted: 4

Part 5: count_instances_tr
Instances counted: 3

Part 6: count_instances_deep
Instances counted: 3

Tests complete!
> 
```