CSC240
Lab 3 – Recursion

```
;basic comparison of 2 strings

(define (longer-string str1 str2)

    (if (> (string-length str1) (string-length str2))

        str1

        str2))


;longest in a list of strings using a helper function to

compare (define (longest-in-list ls)

    (if (null? (cdr ls))

        (car ls)

        (longer-string (car ls)

                       (longest-in-list (cdr ls)))))
```

1. Type the above definitions of the `longer-string` and `longest-in-list` functions and confirm that it correctly finds the longest string in on the list `("red" "white" "and" "blue")`.

2. Explain each of the recursive calls for the list `("red" "white" "and" "blue")` and how the function produces its result.

3. Find out what happens if `longest-in-list` is applied to an empty list.
   a. Explain what happens.
   b. How can this be fixed

When writing the following recursive functions, use only **`car/first, cdr/rest`** and **`cons`** functions. Do not use any built-in functions, except **`expt`**.

4. Write a recursive function, **`mult5`**, that will return how many numbers in the list are multiples of 5.

```
(mult5 '(60 22 13 25) ===> 2
(mult5 '(5) ===> 1
(mult5 '()  ===> 0
```

5. Write a recursive function, **`sumAdj`**, that will return the sum of all adjacent pairs in a list.

```
(sumAdj '(6 2 3 4) ===> 20          ; 8 + 5 + 7
(sumAdj '(5) ===> 0
(sumAdj '()  ===> 0
```

6. Write a recursive function, **`getlast`**, that will return the last element in a list. If the list is empty, return the string "empty list".

7. Write a recursive function, **`removeLast`**, that will remove the last element from a list of numbers. If the list is empty, return an empty list. Do NOT use any built-in list functions.

8. Write a recursive function, **`series`**, that will return a list of the first n values in the series. The series beginning with a_1 and a change of delta between terms.
For example: $a_1 = 2$, delta = 3 would be the series 2, 5, 8, 11, 14, …

```
(sumSeries 2 3 5)   ===> (2 5 8 11 14)
(sumSeries 5 8 1)   ===> (5)
(sumSeries 0 10 10) ===> (0 10 20 30 40 50 60 70 80 90)
(sumSeries -2 0 3)   ===> (-2 -2 -2)
```

9. Write a recursive function, **`remove-large`**, that takes a list of numbers and returns a list where any value larger than the second parameter has been removed from the original list:

```
(remove-large '(1 3 5 7 9 11) 5) ===> (1 3 5)
(remove-large '(10.5 3 8 12.7 6.2) 10) ===> (3 8 6.2)
(remove-large '(1 3 5 7 9 11) 0) ===> ()
(remove-large '() 0)             ===> ()
```

10. Write a function, **`sqr-each`**, that takes a list of numbers and returns a list where each value has been squared:

```
(sqr-each '(3 -62 41.4 17/4)) ===> (9 3844 1713.96 289/16)
(sqr-each '(0)) ===> (0)
(sqr-each '())  ===> ()
```

11. Write a function, **`repeat`**, that takes two arguments, `size` and `item`, and returns a list of `size` elements, each of which is `item`:

```
(repeat 6 'foo)   ===> (foo foo foo foo foo foo)
(repeat 2 '())    ===> (() ())
(repeat 1 15)     ===> (15)
(repeat  3 '(alpha beta)) ===> ((alpha beta) (alpha beta) (alpha beta))
(repeat  0 'help)  ===> ()
```

Submit a well documented Racket source file (.rkt) with answers to questions #2-3 in comments and Racket functions for questions #4-11 through Canvas.