Introduction to Dr. Racket

Debugging in Dr. Racket

1. Consider the codes below for the function “remove-item.” Consider that “remove-item”, given an atom and a list as parameters, will, no matter how deep the atom, delete all atoms and return a list without them. Type in the code and use tools in Dr. Racket to detect and debug any erroneous expression in the codes. **Show that your fixed code returns the correct results.**

```
(define (remove-item atom lst)
      (cond
            ((null? lst) '())
            ((list? (car lst)) (cons (remove-item atom (car lst))
                  (remove-item (cdr lst))))
            ((= atom (car lst)) (remove-item (cdr lst)))
            (else (cons (car lst)(remove-item atom (cdr lst))))
      )
)

(remove-item 1 (cons '(1 1 (1 1 (1))) (1 2 (1 3)) -> ((((()))) 2 (3))
```

Debugging in Dr. Racket #2

2. Consider the code for the function “trafficLight” below. Type in the code and use tools in Dr. Racket to detect and debug any erroneous expression in the code. **Show that your code returns the correct results.** (Tip: You can enter Debug mode and press Stop in order to exit from runtime and reload the interactive mode of Dr. Racket)

```
(define (trafficLight color)
      (cond
            ((trafficLight “green”) (display “It’s green\n”))
            ((trafficLight “yellow”) (display “It’s yellow!\n”))
            ((trafficLight “red”) (display “stop\n”))
            (else (display “wrong color\n”))
            )
      )

(trafficLight “blue”)
(trafficLight “green”)
```

Higher-Order Function

3. Write a higher-order function **cal** which applies a function to a list as values and produces a list results. The function **cal** should apply the passed function to each element in the list and return a same-size list as a result.

   Example:
   `(cal sqrt'(1 2 3 4 5))` ⇒ `(1 1.4142135623730951 1.7320508075688772 2 2.23606797749979)`

Higher-Order Function #2

4. Write a higher-order defined function not-prime that gets only non-prime numbers of an input list. You will also have to create a separate function that can determine if a given number is a prime number or not.

Intersection

5. Consider the code below, for the function "intersection" from last week's Flipped Lab. Recall that "intersection", given two lists represented as sets, will return the intersection of both sets. Type in the code and use tools in Dr. Racket to detect and debug any erroneous expression in the code. Show that your fixed code returns correct results (NOTE: order of elements in returned set DOESN'T matter!).

```
(define (intersection lst1 lst2)
      (cond
            ((null? lst2) '()) ;base case
            ((exist? (car lst1) lst2) (cons (car lst1)
                  (intersection lst1 lst2))
            ((intersection (cdr lst1) lst2))
      )
)

(define (exist? X lst)
      (cond
            ((null? lst) #f)
            ((eq? X (car lst)) #t)
            (#t (exist? X (cdr lst)))
      )
)

(define (main)
      (display (intersection '(1 2 3 4) '(4 -1 2 5))) ; => '(2 4) or '(4 2)
      (newline)
```

```
    (display (intersection '(5 6 7 8) '(1 2 3))) ; => '()
    (newline)
)

(main)
```