CS 330 – Programming Languages
HW5Scheme 1

Name ________Stuart Lech_____________ Grade ________________

Part I. Warming up - tell the result from evaluating the following scheme expressions. You could use DrRacket (Lang = Pretty Big or similar*)* to verify your answers for this part. (2x20=40 points)

(a) (cdr (car '((a b) d (c d)))) → ___(b)_________

(b) (car (car (cdr '(cdr ((a b) (c d) e f))))) → ____’(a b)________

(c) (cadadr ' ((a b) (c d) (e f))) → ____d_______

(d) (cdr '(car (cdr (cdr ((a b) (c d) e f))))) → ____((cdr (cdr((a b) (c d) ef))))_____

(e) '(car '(car (cdr (cdr ((a b) (c d) e f))))) → _____(car '(car (cdr (cdr ((a b) (c d) e f)))))_______

(f) (cons ‘a ' (b c d)) → ___(a b c d)_____

(g) (append '(a b) ‘(c d)) → ___(a b c d)___

(h) (list '(a b) '(c d)) → ______((a b) (c d))______

(i) (member 'a '(ba b a c)) → ___(a c)_____

(j) (list '(b c d) (list 'a)) → ______((b c d) (a))______

(k) ((lambda x x) 1 2 3) → _____(1 2 3)____

(l) (symbol? 'a) → _____#t_____

(m) (null? ' ()) → ____#t______

(n) (reverse ' (a (b c) d)) → ___(d (b c) a)_____

(o) (length ' (a (b c) (d) e)) → _____4______

(p) (display “Hello World!”) → ____Hello World!____

(q) (write “Hello World!”) → __”Hello World!”____

(r) (let ((a 2)) (set! a (read)) a)
(input is *Hello World!)* → ___Hello_____

(s) (append ' (b c d) (list 'a)) → ______(b c d a)______

(t) (reverse (cdr (reverse '(x y z)))) → ______(x y)___

Part II. Working with numbers on Leetcode – define racket scheme functions for the following leetcode problems. Note that leetcode heading include a contract def using -> which should not affect your solution. The easiest way to implement your scheme solutions in racket and then simply initiate a call from the leetcode interface function to your scheme function. Submit screenshot of your source code and leetcode acceptance for this part. (10X3=30 points)

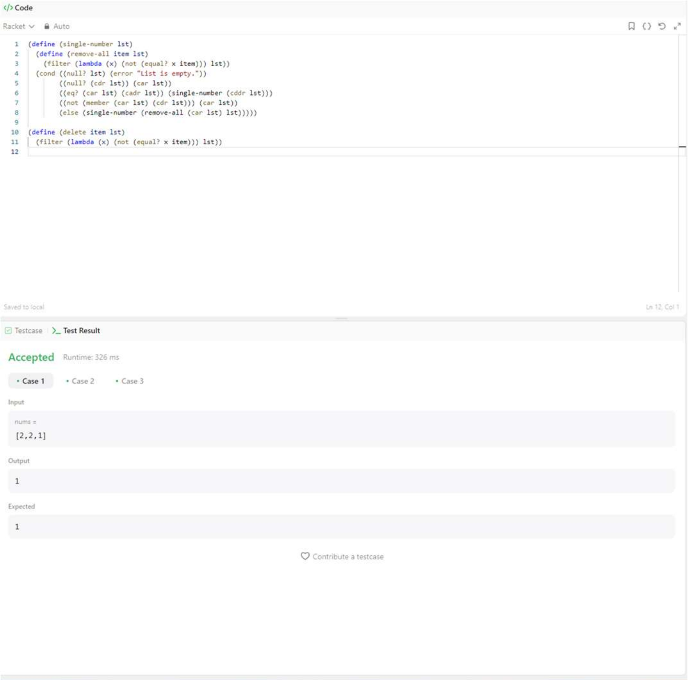
1. Problem 136 single number

   Hint:

   a. call your function (ex., sn) with a sorted list (asc),

   b. check to see if car is same as cadr, if yes, recursive call on cdr, if no, return car

   What is the base case here?



```
(define (single-number lst)
  (define (remove-all item lst)
    (filter (lambda (x) (not (equal? x item))) lst))
  (cond ((null? lst) (error "List is empty."))
        ((null? (cdr lst)) (car lst))
        ((eq? (car lst) (cadr lst)) (single-number (cddr lst)))
        ((not (member (car lst) (cdr lst))) (car lst))
        (else (single-number (remove-all (car lst) lst)))))

(define (delete item lst)
  (filter (lambda (x) (not (equal? x item))) lst))

```

Accepted Runtime: 326 ms

Input

nums =
[2,2,1]

Output

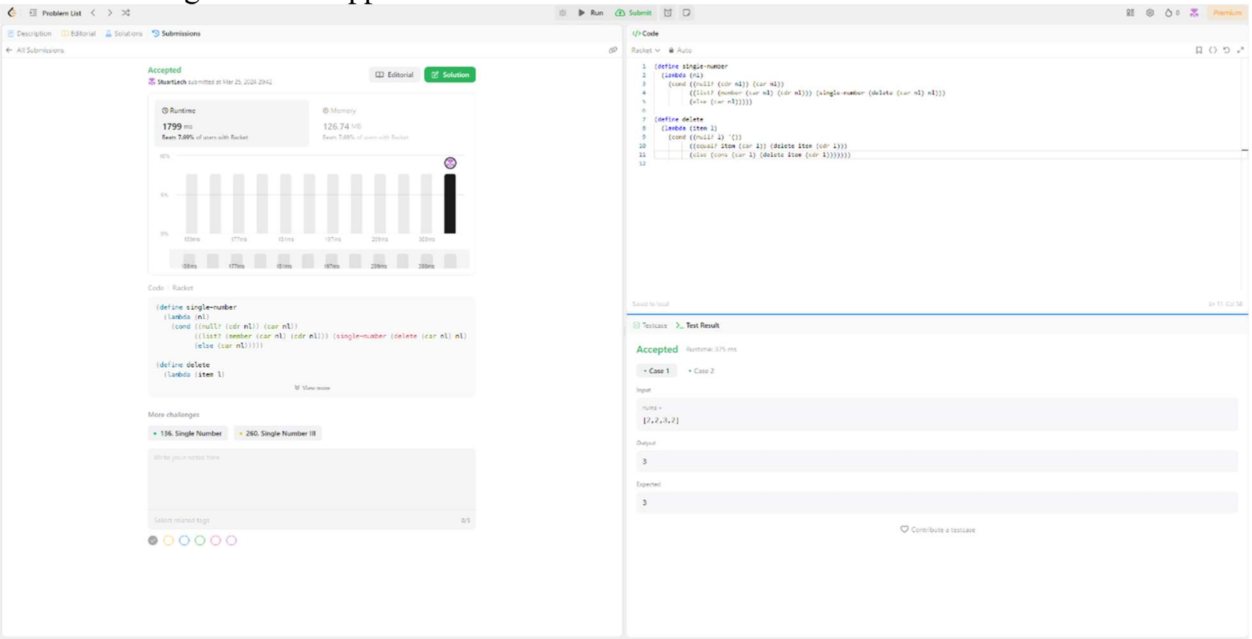
1

Expected

1

2. Problem 137 single number 2

   Hint: extending from 136 appears to be cake



Accepted

StuartLech submitted at Mar 25, 2024 20:42

Runtime 1799 ms Beats 7.69% of users with Racket

Memory 126.74 MB Beats 7.69% of users with Racket

Code | Racket

```
(define single-number
  (lambda (nl)
    (cond ((null? (cdr nl)) (car nl))
          ((list? (member (car nl) (cdr nl))) (single-number (delete (car nl) nl)
          (else (car nl)))))

(define delete
  (lambda (item l)
```

```
(define single-number
  (lambda (nl)
    (cond ((null? (cdr nl)) (car nl))
          ((list? (member (car nl) (cdr nl))) (single-number (delete (car nl) nl)))
          (else (car nl)))))

(define delete
  (lambda (item l)
    (cond ((null? l) '())
          ((equal? item (car l)) (delete item (cdr l)))
          (else (cons (car l) (delete item (cdr l)))))))

```

Accepted Runtime: 375 ms

Input

nums =
[2,2,3,2]

Output

3

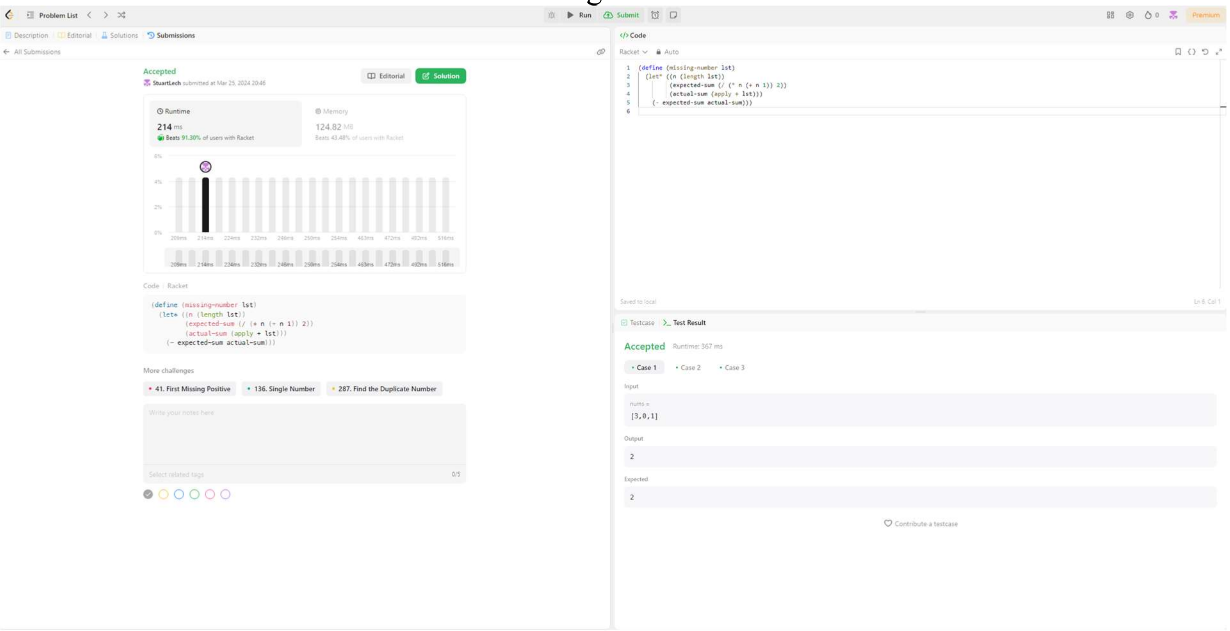
Expected

3

3. Problem 268 missing number

   Hint:

a. call your main function with the list sorted in DSC order.
b. At reverse order, car should equal list length
c. How do we know the list is missing a 0?



Accepted
StuartLech submitted at Mar 25, 2024 20:46

Runtime 214 ms
Beats 91.30% of users with Racket

Memory 124.82 MB
Beats 43.48% of users with Racket

Code | Racket

```
(define (missing-number lst)
  (let* ((n (length lst))
         (expected-sum (/ (* n (+ n 1)) 2))
         (actual-sum (apply + lst)))
    (- expected-sum actual-sum)))
```

More challenges
41. First Missing Positive
136. Single Number
287. Find the Duplicate Number

Write your notes here
Select related tags 0/5

```
(define (missing-number lst)
  (let* ((n (length lst))
         (expected-sum (/ (* n (+ n 1)) 2))
         (actual-sum (apply + lst)))
    (- expected-sum actual-sum)))
```

Saved to local
Ln 6, Col 1

Accepted Runtime: 367 ms

Input
nums =
[3,0,1]

Output
2

Expected
2

Part III. Working with list – Define any three of the following scheme functions and store them in a single scheme file. Defining helper function as needed. Be sure to include your name(s) as well as other necessary comments in your scheme file. (3x10=30 points each)

1. removeHT – a function that accepts a list argument and returns a list with the first (i.e., Head) and last elements (i.e., Tail) removed. It returns ‘() if original list has fewer than 3 elements. if Ex.,

   (removeHT ‘((a b) c d e)) → (c d)
   (removeHT ‘(a b)) → ( )

2. level - a procedure that takes a list argument and returns a list that contains all the original atoms as top-level elements. For example, (level '(a (“bb” c) d (e (4 g)))) → (a “bb” c d e 4 g).

3. insert – a procedure that accepts an object obj, a nonnegative integer n, and a list L. It inserts obj into L at position n. Note that n is between 0 and (length L). For example, (insert ‘a 0 ‘(b c d)) → (a b c d), (insert ‘a 3 ‘(b c d)) → (b c d a), and (insert ‘a 0 ‘()) → (a)

4. permutation - a procedure that accepts a list argument L and returns a list of all the permutations of L. For example, (permutations '(a b c)) should return something like the following:

   ((a c b) (c a b) (c b a) (a b c) (b a c) (b c a))

   Hint: You could get permutation of ‘(a b c d) by inserting ‘a (the car of list) into every position of each top-level element (i.e., sublist) of the permutation of ‘(a b c) (the cdr of list).

This assignment is worth 100 points and is due on Monday (3/25) at 10 PM. Please submit two documents on moodle: a document file (word or pdf) for part 1 and 2, and a scheme file for part 3.