Project 2. Scheme Interpreter.
COSC 252
Jeremy Bolton

You will implement a mini-scheme interpreter. Luckily, cacl has a very similar syntax to scheme, thus you do not need to create this interpreter from scratch; instead, you will add capabilities to your previously submitted calc interpreter thus effectively implementing a scheme interpreter.

You will need to update the design of the language: the tokens, the grammar, and the semantics accordingly.

When in doubt of desired behavior feel free to default to [Dy], https://scheme.cs61a.org/, and/or ask me.

Instructions:

I. Language Additions. Add to the simple calculator the following:
   a. data type: rational numbers
      - 3/4
   b. "Built-in" procedures / operations
      - Constructing lists using single quote: ‘
      - list
      - car
      - cdr
      - cons
      - lambda
      - let
      - letrec
      - define
      - null?
      - atom?
      - list?
      - cond / else
      - length
      - eval*
      - map **
      - delay and force ***
   c. Capabilities
      - High-Order Functions
        - (define make-double (lambda (f)
          (lambda (x) (f x x)))
      - Recursion
        - (define memv

```
(lambda (x ls)
        (cond
                [(null? ls) #f]
                [(eqv? (car ls) x) ls]
                [else (memv x (cdr ls))])))
```

- Appropriate scope/binding rules
  - (let ([x 1])
    (let ([x (+ x 1)])
    (+ x x)))

d. Example Usage:

- (list 'a 'b 'c)
- (define x
  (or
  (> (+ 1 2) (/ 3 2))
  (not (= 3 (+1 2)))
)
)
- (let ([x 'a])
  (let ([f (lambda (y) (list x y))])
  (f 'b)))
- (let ([list1 '(a b c)] [list2 '(d e f)])
  (cons (cons (car list1)
  (car list2))
  (cons (car (cdr list1))
  (car (cdr list2)))))

II. Design and build mini-scheme

a. Design tokens and build tokenizer
   - Updates to tokenizer will be needed.

b. Design grammar and build parser.
   - Updates to Grammar will be needed.
   - NOTE:
     - Debugging help:
       a. **<u>Test your parser before adding semantics!!!</u>**
       b. Hint: Have your parser compose a string that demonstrates the path of the function call chain during the parse, (this mimics the parse tree). Use the parse tree rendering using tool (http://mshang.ca/syntree/):

c. Implement Semantic Analysis. Incorporate semantic evaluation directly into the parser.
   - See discussion in Section I.
   - Include appropriate error messages and recovery.

- When in doubt of desired behavior feel free to default to [Dy], https://scheme.cs61a.org/, and/or ask me.

III. RUBRIC

Requirements		**Points Allocated**
mini-scheme	Parser / Tokenizer Updates	30%
	Semantic Evaluation	60%
	Error Message / Recover	5%
	map	5%
	delay and force	5%
TOTAL		**100% + 5% extra credit**