Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

*CSI 2120 Programming Paradigms*

# Assignemnt-3

# Mini-Sudoku using Scheme

*Course Coordinator*: ***Wassim El Ahmar***

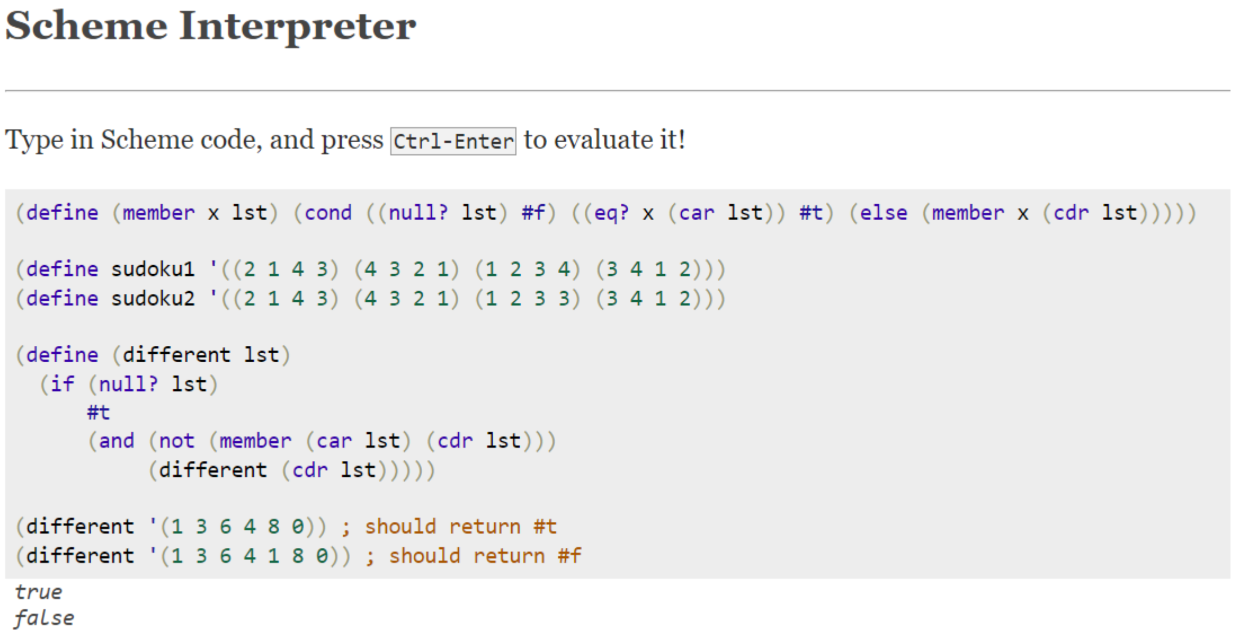
*Name: Rakshita Mathur*

*Student Number: 300215340*

The archive submitted contain this and the assignment3.scm scheme file.

The online interpreter used to run and check the code :https://inst.eecs.berkeley.edu/~cs61a/fa14/assets/interpreter/scheme.html

Following are the screenshots of the code



## Scheme Interpreter

Type in Scheme code, and press Ctrl-Enter to evaluate it!

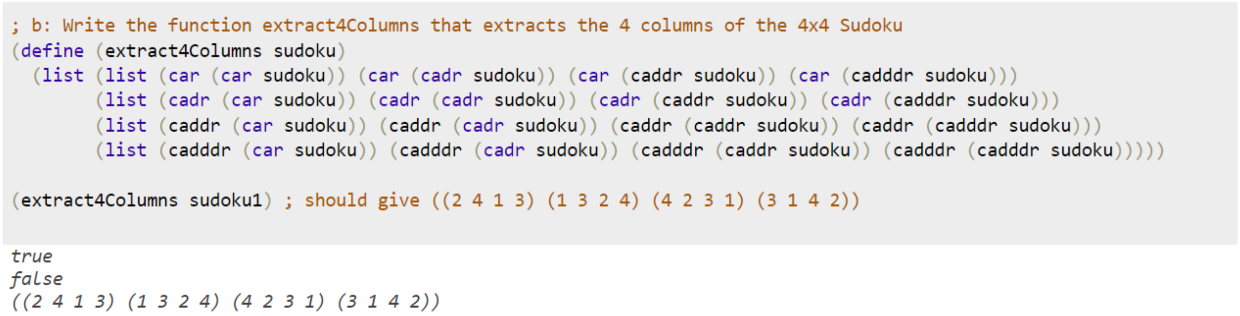
```
(define (member x lst) (cond ((null? lst) #f) ((eq? x (car lst)) #t) (else (member x (cdr lst)))))

(define sudoku1 '((2 1 4 3) (4 3 2 1) (1 2 3 4) (3 4 1 2)))
(define sudoku2 '((2 1 4 3) (4 3 2 1) (1 2 3 3) (3 4 1 2)))

(define (different lst)
  (if (null? lst)
      #t
      (and (not (member (car lst) (cdr lst)))
           (different (cdr lst)))))

(different '(1 3 6 4 8 0)) ; should return #t
(different '(1 3 6 4 1 8 0)) ; should return #f
```

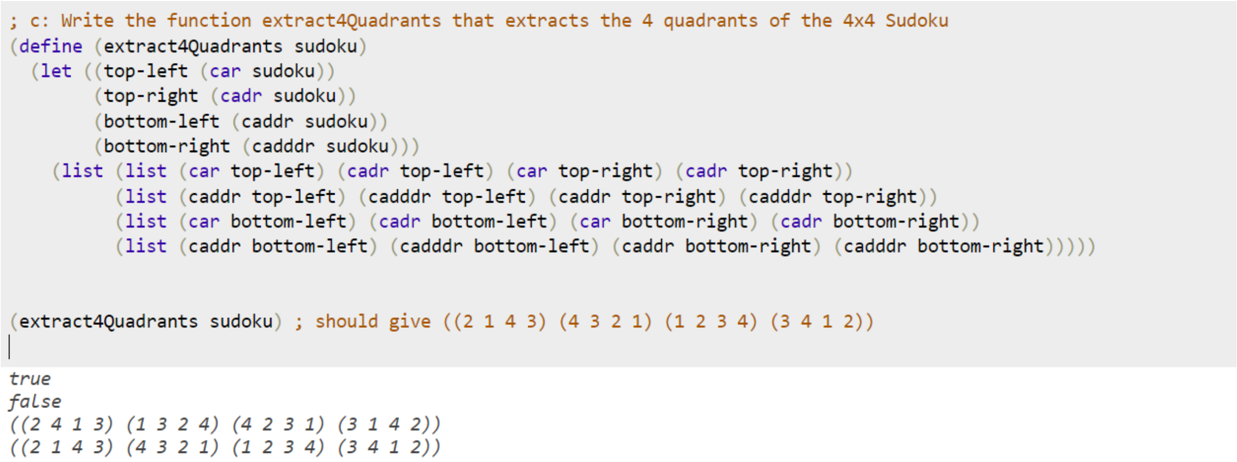
```
true
false
```



```
; b: Write the function extract4Columns that extracts the 4 columns of the 4x4 Sudoku
(define (extract4Columns sudoku)
  (list (list (car (car sudoku)) (car (cadr sudoku)) (car (caddr sudoku)) (car (cadddr sudoku)))
        (list (cadr (car sudoku)) (cadr (cadr sudoku)) (cadr (caddr sudoku)) (cadr (cadddr sudoku)))
        (list (caddr (car sudoku)) (caddr (cadr sudoku)) (caddr (caddr sudoku)) (caddr (cadddr sudoku)))
        (list (cadddr (car sudoku)) (cadddr (cadr sudoku)) (cadddr (caddr sudoku)) (caddr (cadddr sudoku)))))

(extract4Columns sudoku1) ; should give ((2 4 1 3) (1 3 2 4) (4 2 3 1) (3 1 4 2))
```

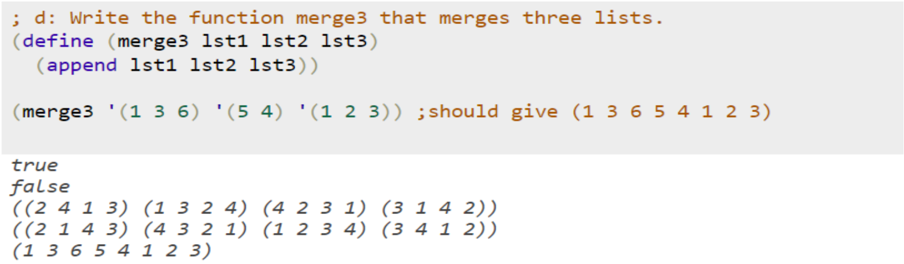
```
true
false
((2 4 1 3) (1 3 2 4) (4 2 3 1) (3 1 4 2))
```



```
; c: Write the function extract4Quadrants that extracts the 4 quadrants of the 4x4 Sudoku
(define (extract4Quadrants sudoku)
  (let ((top-left (car sudoku))
        (top-right (cadr sudoku))
        (bottom-left (caddr sudoku))
        (bottom-right (cadddr sudoku)))
    (list (list (car top-left) (cadr top-left) (car top-right) (cadr top-right))
          (list (caddr top-left) (cadddr top-left) (caddr top-right) (cadddr top-right))
          (list (car bottom-left) (cadr bottom-left) (car bottom-right) (cadr bottom-right))
          (list (caddr bottom-left) (cadddr bottom-left) (caddr bottom-right) (cadddr bottom-right)))))


(extract4Quadrants sudoku) ; should give ((2 1 4 3) (4 3 2 1) (1 2 3 4) (3 4 1 2))
```

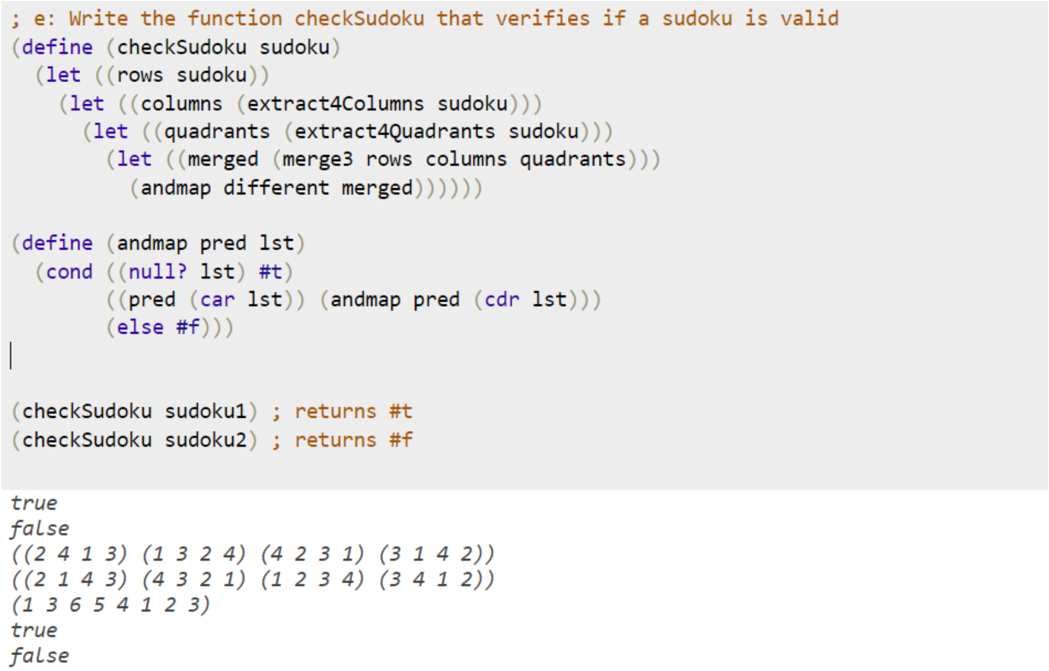
```
true
false
((2 4 1 3) (1 3 2 4) (4 2 3 1) (3 1 4 2))
((2 1 4 3) (4 3 2 1) (1 2 3 4) (3 4 1 2))
```



```
; d: Write the function merge3 that merges three lists.
(define (merge3 lst1 lst2 lst3)
  (append lst1 lst2 lst3))

(merge3 '(1 3 6) '(5 4) '(1 2 3)) ;should give (1 3 6 5 4 1 2 3)
```

```
true
false
((2 4 1 3) (1 3 2 4) (4 2 3 1) (3 1 4 2))
((2 1 4 3) (4 3 2 1) (1 2 3 4) (3 4 1 2))
(1 3 6 5 4 1 2 3)
```



```
; e: Write the function checkSudoku that verifies if a sudoku is valid
(define (checkSudoku sudoku)
  (let ((rows sudoku))
    (let ((columns (extract4Columns sudoku)))
      (let ((quadrants (extract4Quadrants sudoku)))
        (let ((merged (merge3 rows columns quadrants)))
          (andmap different merged))))))

(define (andmap pred lst)
  (cond ((null? lst) #t)
        ((pred (car lst)) (andmap pred (cdr lst)))
        (else #f)))


(checkSudoku sudoku1) ; returns #t
(checkSudoku sudoku2) ; returns #f
```

```
true
false
((2 4 1 3) (1 3 2 4) (4 2 3 1) (3 1 4 2))
((2 1 4 3) (4 3 2 1) (1 2 3 4) (3 4 1 2))
(1 3 6 5 4 1 2 3)
true
false
```