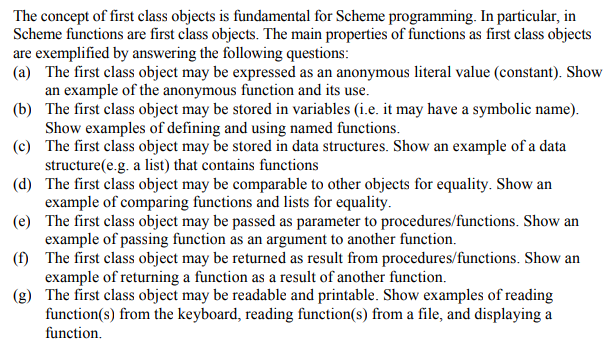
Scheme Programming Language Design – Documentation

Csc 600-01

Keawa Rozet

Code available at: [https://github.com/krozet/](https://github.com/krozet/PROLOG-List-Operations-1)Scheme-Programming-Language-Design

Question #1 Prompt



About my code

For my coding environment I am using MIT/GNU Scheme running under a Linux environment from my terminal. This is why it may appear a bit different from what was demonstrated in class.

Code

; CSc 600: PROGRAMMING LANGUAGE DESIGN

; Scheme

;

; Keawa Rozet

; Homework 4: Question #1

;;;(a) The first class object may be expressed as an anonymous

;;;literal value (constant). Show an example of the anonymous function

;;;and its use.

1 ]=> 600

;Value: 600

;;;(b) The first class object may be stored in variables (i.e. it may

;;;have a symbolic name). Show examples of defining and using

;;;named functions.

1 ]=> (define class 600)

;Value: class

1 ]=> class

;Value: 600

;;;(c) The first class object may be stored in data structures. Show

;;;an example of a data structure(e.g. a list) that contains functions

1 ]=> (length (list 1 2 3 4 5))

;Value: 5

;;;(d) The first class object may be comparable to other objects

;;;for equality. Show an example of comparing functions and lists

;;;for equality.

1 ]=> (eqv? (+ 1 4) (length (list 1 2 3 4 5)))

;Value: #t

;;;(e) The first class object may be passed as parameter to

;;;procedures/functions. Show an example of passing function as an

;;;argument to another function.

1 ]=> ((lambda (x y) (\* x y)) 10 60)

;Value: 600

;;;(f) The first class object may be returned as result from

;;procedures/functions. Show an example of returning a function as a

;;;result of another function.

1 ]=> (define cubed (lambda (x) (\* x (sqrd x))))

;Value: cubed

1 ]=> (define sqrd (lambda (x) (\* x x)))

;Value: sqrd

1 ]=> (cubed 2)

;Value: 8

;;;(g) The first class object may be readable and printable. Show

;;;examples of reading function(s) from the keyboard, reading function(s)

;;;from a file, and displaying a function.

1 ]=> (define name (read-line))

Keawa Rozet

;Value: name

1 ]=> (display name)

Keawa Rozet

1 ]=> (define fn (open-input-file "matrix1.dat"))

;Value: fn

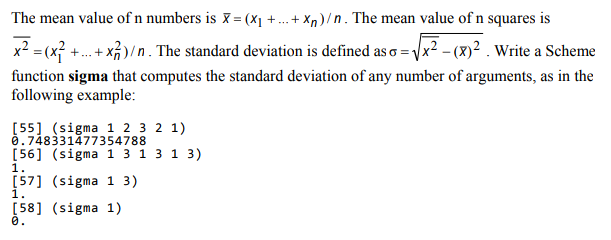
1 ]=> (read fn)

;Value: 2

1 ]=> (read fn)

;Value: 3

Question #2 Prompt



About my code

For my coding environment I am using MIT/GNU Scheme running under a Linux environment from my terminal. This is why it may appear a bit different from what was demonstrated in class.

My solution involves using a function to square a value, a function to sum a list, and a function to square the values in a list and sum them to calculate the standard deviation.

Code

; CSc 600: PROGRAMMING LANGUAGE DESIGN

; Scheme

;

; Keawa Rozet

; Homework 4: Question #2

(define (square x) (\* x x))

(define (sum-list list)

(if (null? list)

0

(+ (car list) (sum-list (cdr list)))))

(define (square-sum-list list)

(if (null? list)

0

(+ (square (car list)) (square-sum-list (cdr list)))))

(define sigma (lambda x (sqrt (- (/ (square-sum-list x) (length x))

(square (/ (sum-list x) (length x)))))))

Example

1 ]=> (load "std.scm")

;Loading "std.scm"... done

;Value: sigma

1 ]=> (sigma 1 2 3 2 1)

;Value: .7483314773547883

1 ]=> (sigma 1 3 1 3 1 3)

;Value: 1

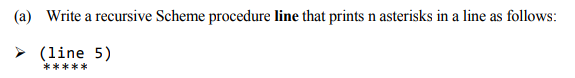
1 ]=> (sigma 1 3)

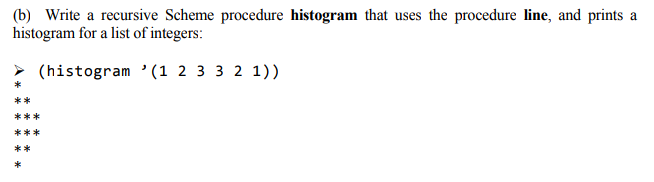
;Value: 1

1 ]=> (sigma 1)

;Value: 0

Question #3 Prompt





About my code

For my coding environment I am using MIT/GNU Scheme running under a Linux environment from my terminal. This is why it may appear a bit different from what was demonstrated in class.

My solution involves recursion to print n number of ‘\*’.

Code

; CSc 600: PROGRAMMING LANGUAGE DESIGN

; Scheme

;

; Keawa Rozet

; Homework 4: Question #3

(define (line n)

(cond ((< n 1) (newline))

(else (display "\*") (line (- n 1)))))

; recursive call to print the histogram

(define (histogram list)

(if (null? list) (newline)

(begin (line (car list))

(histogram (cdr list)))))

Example

1 ]=> (load "histogram.scm")

;Loading "histogram.scm"... done

;Value: histogram

1 ]=> (line 5)

\*\*\*\*\*

1 ]=> (histogram '(1 2 3 3 2 1))

\*

\*\*

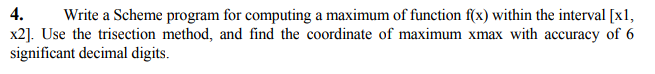
\*\*\*

\*\*\*

\*\*

\*

Question #4 Prompt



About my code

For my coding environment I am using MIT/GNU Scheme running under a Linux environment from my terminal. This is why it may appear a bit different from what was demonstrated in class.

My solution involves the use of trisection to locate the maximum value in a function. In my code I provided a test function (test-fn) which I use to check to make sure the maximum values are found.

Code

; CSc 600: PROGRAMMING LANGUAGE DESIGN

; Scheme

;

; Keawa Rozet

; Homework 4: Question #4

(define (find-max x y fn)

; implementation of the trisection

(let\* ((trisection (/ (- y x) 3.))

(x-trisection (+ x trisection))

(y-trisection (- y trisection)))

(cond ((> x y) (display "first value cannot be larger than the second value."))

; sets the precision to 6 significant digits

((< (abs (- (fn x) (fn y))) .0000001) (/ (+ y x) 2.))

((> (fn x-trisection) (fn y-trisection)) (find-max x y-trisection fn))

(else (find-max x-trisection y fn)))))

; x\*x - 2\*x + 1

(define (test-fn x) (+ 1 (- (\* x x) (\* 2 x))))

Example

1 ]=> (load "max.scm")

;Loading "max.scm"... done

;Value: test-fn

1 ]=> (find-max 0 2 test-fn)

;Value: 1.

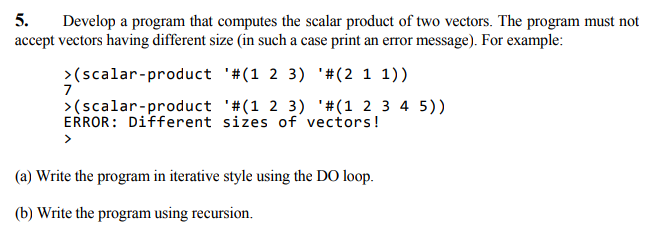
1 ]=> (find-max 0 4 test-fn)

;Value: 3.999999992942522

1 ]=> (find-max -2 2 test-fn)

;Value: -1.9999999929425214

Question #5 Prompt



About my code

For my coding environment I am using MIT/GNU Scheme running under a Linux environment from my terminal. This is why it may appear a bit different from what was demonstrated in class.

My solution involves two solutions to the scalar product. The iterative solution uses a do loop to extract each value within the vectors and multiply the values together, and sum the results. The recursive solution converts the vectors into a list and passes those lists to the dot-list function. The dot-list function then recursively multiplies and adds the two lists together.

Code

; CSc 600: PROGRAMMING LANGUAGE DESIGN

; Scheme

;

; Keawa Rozet

; Homework 4: Question #5

; iterative scalar product function

(define (scalar-iter vector1 vector2)

(cond ((zero? (vector-length vector1)) (display "vector1 is empty."))

((zero? (vector-length vector2)) (display "vector2 is empty."))

((= (vector-length vector1) (vector-length vector2))

(let ((s 0))

; here the do loop is implemented

(do ((i 0 (add1 i)))

((>= i (vector-length vector1)) (display s))

(set! s (+ s (\* (vector-ref vector1 i) (vector-ref vector2 i)))))))

(else (display "vector1 and vector2 are different sizes."))))

(define (add1 x) (+ x 1))

; recursive scalar product function

(define (scalar-recur vector1 vector2)

(cond ((not (equal? (vector-length vector1) (vector-length vector2)))

(display "vector1 and vector2 are different sizes."))

((zero? (vector-length vector1)) (display "vector is empty."))

(else (dot-list (vector->list vector1) (vector->list vector2)))))

(define (dot-list list1 list2)

(cond((null? (cdr list1)) (\* (car list1) (car list2)))

(else (+ (\* (car list1) (car list2)) (dot-list (cdr list1) (cdr list2))))))

Example

1 ]=> (load "scalar-product.scm")

;Loading "scalar-product.scm"... done

;Value: dot-list

1 ]=> (scalar-iter '#(1 2 3) '#(2 1 1))

7

1 ]=> (scalar-recur '#(1 2 3) '#(2 1 1))

;Value: 7

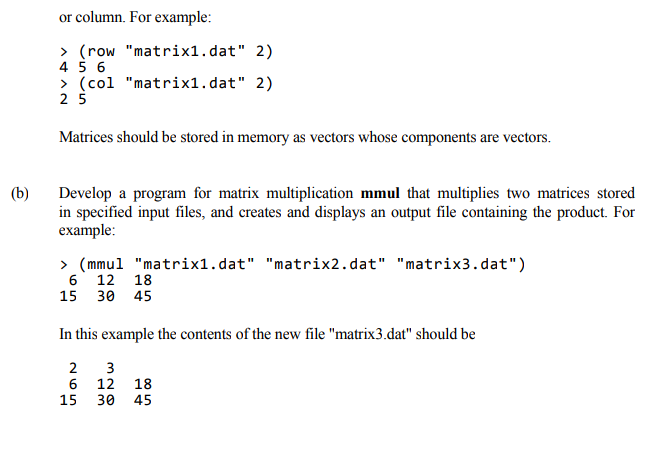
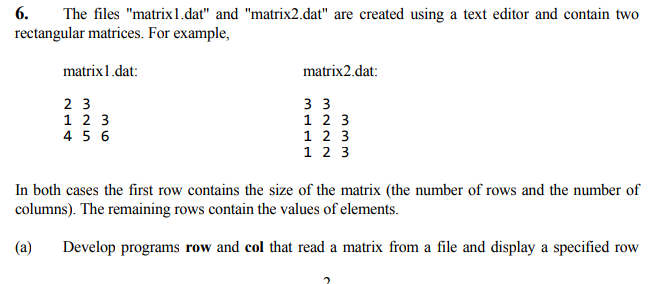
1 ]=> (scalar-iter '#(1 2 3) '#(1 2 3 4 5))

vector1 and vector2 are different sizes.

1 ]=> (scalar-recur '#(1 2 3) '#(1 2 3 4 5))

vector1 and vector2 are different sizes.

Question #6 Prompt



About my code

For my coding environment I am using MIT/GNU Scheme running under a Linux environment from my terminal. This is why it may appear a bit different from what was demonstrated in class.

My solution involves multible functions working together to achieve matrix multiplication. The read-matrix function relies on a do loop to create vecotrs out of the provided files. The dot-product also relies on a do loop to multiply two vectors values together and add them all up. row and col are both used to retrieve the actual values within the newly formed vectors (by read-matrix). mmul uses all of these functions to create vectors out of the two matrix files, write the new matrix’s number of rows and columns on the first line, and display the new matrix by iterating through all the values of the two vectors and calling dot-product on them.

Code

; CSc 600: PROGRAMMING LANGUAGE DESIGN

; Scheme

;

; Keawa Rozet

; Homework 4: Question #6

; create add1 function

(define (add1 x) (+ x 1))

; reads in matrix from user input

(define (read-matrix filename)

(let\* ((input-file (open-input-file filename))

(max-row (read input-file))

(max-col (read input-file))

(mat (make-vector max-row)))

(do ((i 0 (add1 i)))

((>= i max-row) (close-input-port input-file) mat)

(let ((display-row (make-vector max-col)))

(do ((j 0 (add1 j)))

((>= j max-col) (vector-set! mat i display-row))

(vector-set! display-row j (read input-file)))))))

(define (display-vector v)

(do ((i 0 (add1 i)))

((>= i (vector-length v)) (display ""))

(display (vector-ref v i)) (display " ")))

(define (dot-product v1 v2)

(do

((i 0 (add1 i)) (sum 0 (+ sum (\* (vector-ref v1 i) (vector-ref v2 i)))))

((>= i (vector-length v1)) sum)))

; get the row

(define (row filename i)

(define mat (read-matrix filename))

(vector-ref mat i))

; display the row

(define (display-row filename i)

(display-vector (row filename (- i 1))))

; get the column

(define (col filename j)

(define mat (read-matrix filename))

(define max-row (vector-length mat))

(define column (make-vector max-row))

(do ((i 0 (add1 i)))

((>= i max-row) column)

(vector-set! column i (vector-ref (vector-ref mat i) j))))

; matrix multiplication

(define (mmul matrix-data1 matrix-data2 matrix-output-file)

(define matrix1 (read-matrix matrix-data1))

(define matrix2 (read-matrix matrix-data2))

(define max-row (vector-len0gth matrix1))

(define max-col (vector-length matrix2))

(define output-file (open-output-file matrix-output-file))

; add the num of rows and cols to the first line

(display max-row output-file) (display " " output-file)

(display max-col output-file) (newline output-file)

; displays the columns row by row

(do ((i 0 (add1 i)))

((>= i max-row) (close-output-port output-file) (display ""))

(let ((display-row (make-vector max-col)))

(do ((j 0 (add1 j)))

((>= j max-col) (display-vector display-row) (newline) (newline output-file))

(vector-set! display-row j (dot-product (row matrix-data1 i) (col matrix-data2 j)))

(display (vector-ref display-row j) output-file) (display " " output-file)))))

Example

1 ]=> (load "matrix.scm")

;Loading "matrix.scm"... done

;Value: mmul

1 ]=> (display-row "matrix1.dat" 2)

4 5 6

1 ]=> (display-col "matrix1.dat" 2)

2 5

1 ]=> (mmul "matrix1.dat" "matrix2.dat" "matrix3.dat")

6 12 18

15 30 45

1 ]=> (display-vector (read-matrix "matrix3.dat"))

#(6 12 18) #(15 30 45)

;;;contents of matrix3.dat:

;;;2 3

;;;6 12 18

;;;15 30 45