Homework Problems:

1.) **The concept of first class objects is fundamental for Scheme programming. In particular, in Scheme functions are first class objects. The main properties of functions as first class objects are exemplified by answering the following questions:**

a.) **Anonymous function:**

```
> 250
```
<u>Output:</u> 250

b.) **Named function:**

```
> (define (hello name) (display "Hello ") (display name))
> (hello “Sam”)
```
<u>Output:</u> Hello Sam

c.) **Data Structure-contained function:**

```
> (define myArray (list 0 1 2 3 4 5))
> (display (length myArray))
```
<u>Output:</u> 6

d.) **Comparing functions and list for equality:**

```
; checking to see if the given list (1 2 3 4 5) is equivalent to the array created in
; part (c). If the parts are equal, output #t (true) will show, and if they are not, then
; output #f (false) will show

> (display (equal? myArray '(1 2 3 4 5)))
```
<u>Output:</u> #f

e.) **Passing function of an argument as another function:**

```
; passing integers 5 and 10 into the multiply function
> (define (multiply x y) (* x y))
> (display (multiply 5 10))
```
<u>Output:</u> 50

f.) Returning a function as a result of another function:

```
; defining a function named 'num' that takes in two parameters and adds those
; two numbers together
> (define (num x) (lambda (y) (+ x y)))

; defining a function named 'addOne' that takes in an integer using the 'num'
; function and adds 1 to it
> (define addOne (num 1))
> (display (addOne 6))
```
Output: 7

g.) Reading function from the keyboard, a file, and displaying a function:

```
; reading from keyboard
> (define myName (read-line)) Sam Zandiasadabadi
> (display myName)
```
Output: Sam Zandiasadabadi

```
; reading from a file
; assuming a file exists and contains my name
; (this part of my solution has not been tested so I am not entirely sure if it's
accurate

> (define readFile (open-input-file "textFile.txt"))
> (read readFile)
```
Output: Sam Zandiasadabadi

2.) [*This problem was used for the final exam*]
Given the description in the assignment, write a Scheme function sigma that computes the standard deviation of any number of arguments, as in the given example.

```
; Sam Zandiasadabadi
; CSC 600-01 Homework #3.1 Scheme
; Due Date: 05/03/2024 23:59 P.M
; Problem #2
; This program writes a Scheme function sigma that computes the standard deviation of
; any number of arguments, as in the given example.
; creating a function that multiplies a number by itself
> (define (square x) (* x x))

; taking the given numbers and adds them together, given the fact that these numbers
; exist, meaning that the 'value of n numbers' has not finished and there is still numbers
; to be added/squared.
> (define (addingNums num)  (if (null? num) 0 (+ (car num) (addingNums (cdr num)))))

; after squaring the given numbers in the parameter, we essentially perform the same
; thing as the previous step, except this time, we have the value of the squared numbers
; which we need in order to compute the standard deviation
> (define (addingSquaredNums squareNum)  (if (null? squareNum) 0 (+ (square (car
squareNum)) (addingSquaredNums (cdr squareNum)))))

;  defining the sigma function that calculates the standard deviation of the
; parameters/given numbers using the formula for calculating the standard deviation.
> (define sigma (lambda value (sqrt (- (/ (addingSquaredNums value) (length value))
(square (/ (addingNums value) (length value)))))))

> (sigma 1 2 3 2 1)
Output: .7483314773547883

> (sigma 1 3 1 3 1 3)
Output: 1

> (sigma 1 3)
Output: 1

> (sigma 1)
Output: 0
```

3.) **[*This problem was used for the final exam*]**

   a.) **Given the prompt, write a recursive Scheme procedure line that prints n asterisks in a line:**

```
; Sam Zandiasadabadi
; CSC 600-01 Homework #3.1 Scheme
; Due Date: 05/03/2024 23:59 P.M
; Problem #3a
; This program writes a recursive Scheme function sigma that prints n asterisks in
; a line.

; defining a function named 'line' that prints a '*' symbol as many times as
; the user wants it. If the user inputs 0, then we just display a newline or an
; empty script/line.
> (define (line n) (if (= n 0) (newline) (begin (display "*") (line (- n 1)))))

> (line 5)
```
<u>Output: </u>*****

```
> (line 7)
```
<u>Output: </u>*******

b.) Given the prompt, write a recursive Scheme procedure histogram that uses the procedure line, and prints a histogram for a list of integers:

```
; Sam Zandiasadabadi
; CSC 600-01 Homework #3.1 Scheme
; Due Date: 05/03/2024 23:59 P.M
; Problem #3b
; This program writes a recursive Scheme function histogram that uses the
; procedure line, and prints a histogram for a list of integers

> (define (histogram nums)
     ; if the user inputs nothing or 0, we just print an empty line
     (if (null? nums) (newline)
          ; However, if not a null value, we begin to call upon the 'line' function that
          ; prints the '*' sign given how many times the user says
           (begin (line (car nums))
          ; calling upon the 'histogram' function to repeat this process.
           (histogram (cdr nums)))))

> (histogram '(1 2 3 3 2 1))
```
<u>Output:</u>

```
*
**
***
***
**
*
```

4.) **Write a Scheme program for computing a maximum of function f(x) within the interval [x1, x2]. Use the trisection method, and find the coordinate of maximum xmax with accuracy of 6 significant decimal digits.**

```
; Sam Zandiasadabadi
; CSC 600-01 Homework #3.1 Scheme
; Due Date: 05/03/2024 23:59 P.M
; Problem #4
; This program is supposed to compute a maximum of function f(x) within the interval
; [x1, x2] using the trisection method and finding the maximum xmax with accuracy of 6
; significant decimal digits.

; unable to attempt this problem due to lack of time
> (define (fxMaximum maxVal x1 x2))
```

5.) **Given the example, develop a program that computes the scalar product of two vectors. The program must not accept vectors having different size (in such a case print an error message).**

a.) **Iterative:**

```
; Sam Zandiasadabadi
; CSC 600-01 Homework #3.1 Scheme
; Due Date: 05/03/2024 23:59 P.M
; Problem #5a
; This program develops a program that iteratively computes the scalar product of
; two vectors.

; defining a temporary function to calculate the and compare the length of two
; vector lists
(define (tempScalar vector1 vector2)

; writing a do-loop which continues on comparing the values while there is
; something present in the content of each vector, meaning that the vectors are
; not empty
(do ((remainingVec1 vector1 (cdr remainingVec1))
(remainingVec2 vector2 (cdr remainingVec2))
(result 0 (+ result (* (car remainingVec1) (car remainingVec2)))))
((null? remainingVec1) result)))
```

```
; in the case that a vector finishes before another, meaning that one vector still has
; values/content to be processed, while the other vector has hit its final value (is
; full), then we display a message saying that the vectors are of different sizes.
(define (scalar-product vector1 vector2) (if (= (length vector1) (length vector2))
(tempScalar vector1 vector2) (display "ERROR: Different sizes of vectors!")))

> (scalar-product '(1 2 3) '(2 1 1))
```
<u>Output: </u>7

```
> (scalar-product '(1 2 3) '(1 2 3 4 5))
```
<u>Output: </u> ERROR: Different sizes of vectors!

b.) Recursive:

```
; Sam Zandiasadabadi
; CSC 600-01 Homework #3.1 Scheme
; Due Date: 05/03/2024 23:59 P.M
; Problem #5b
; This program develops a program that recursively computes the scalar product
; of two vectors.

; same exact logic as part (a). However, this time we do this process recursively.
(define (scalar-product vector1 vector2)

; this time around, we first check the length of both vectors, if the vectors have
; different lengths, then we state that the vectors have different sizes. If not
; then we continue to call on the function 'scalar-product'.
(cond ((not(= (length vector1)(length vector2))) "ERROR: Different sizes of
vectors!") ((null? vector1) 0) (else (+(* (car vector1) (car vector2))
(scalar-product (cdr vector1) (cdr vector2))))))

> (scalar-product '(1 2 3) '(2 1 1))
```
<u>Output: </u>7

```
> (scalar-product '(1 2 3) '(1 2 3 4 5))
```
<u>Output: </u> ERROR: Different sizes of vectors!

6.) **Given the example, the files "matrix1.dat" and "matrix2.dat" are created using a text editor and contain two rectangular matrices. In both cases the first row contains the size of the matrix (the number of rows and the number of columns). The remaining rows contain the values of elements.**

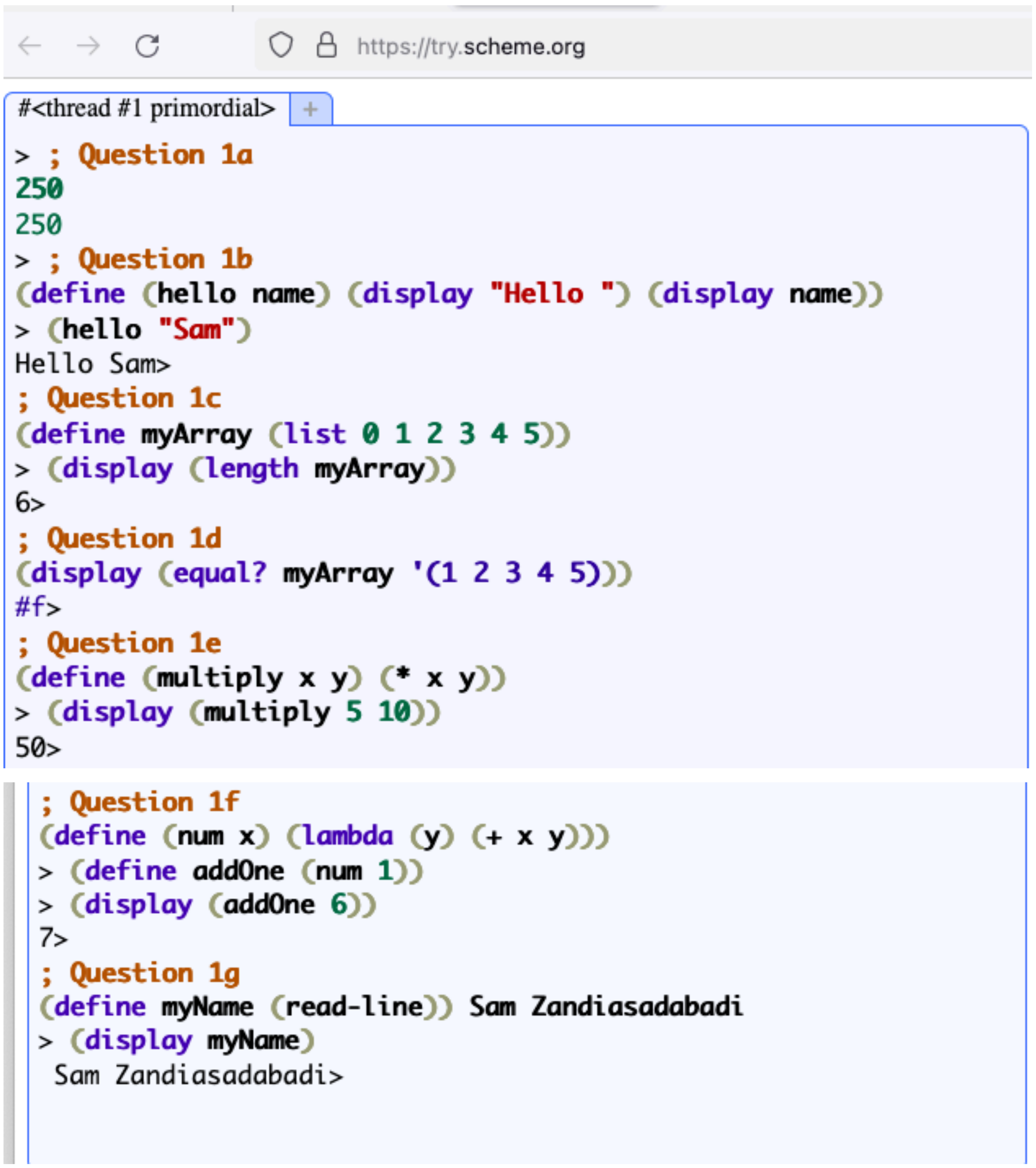
a.) ; unable to attempt this problem due to lack of time

```
; Sam Zandiasadabadi
; CSC 600-01 Homework #3.1 Scheme
; Due Date: 05/03/2024 23:59 P.M
; Problem #6a
```

b.) ; unable to attempt this problem due to lack of time

```
; Sam Zandiasadabadi
; CSC 600-01 Homework #3.1 Scheme
; Due Date: 05/03/2024 23:59 P.M
; Problem #6b
```

Screenshot of my Code:

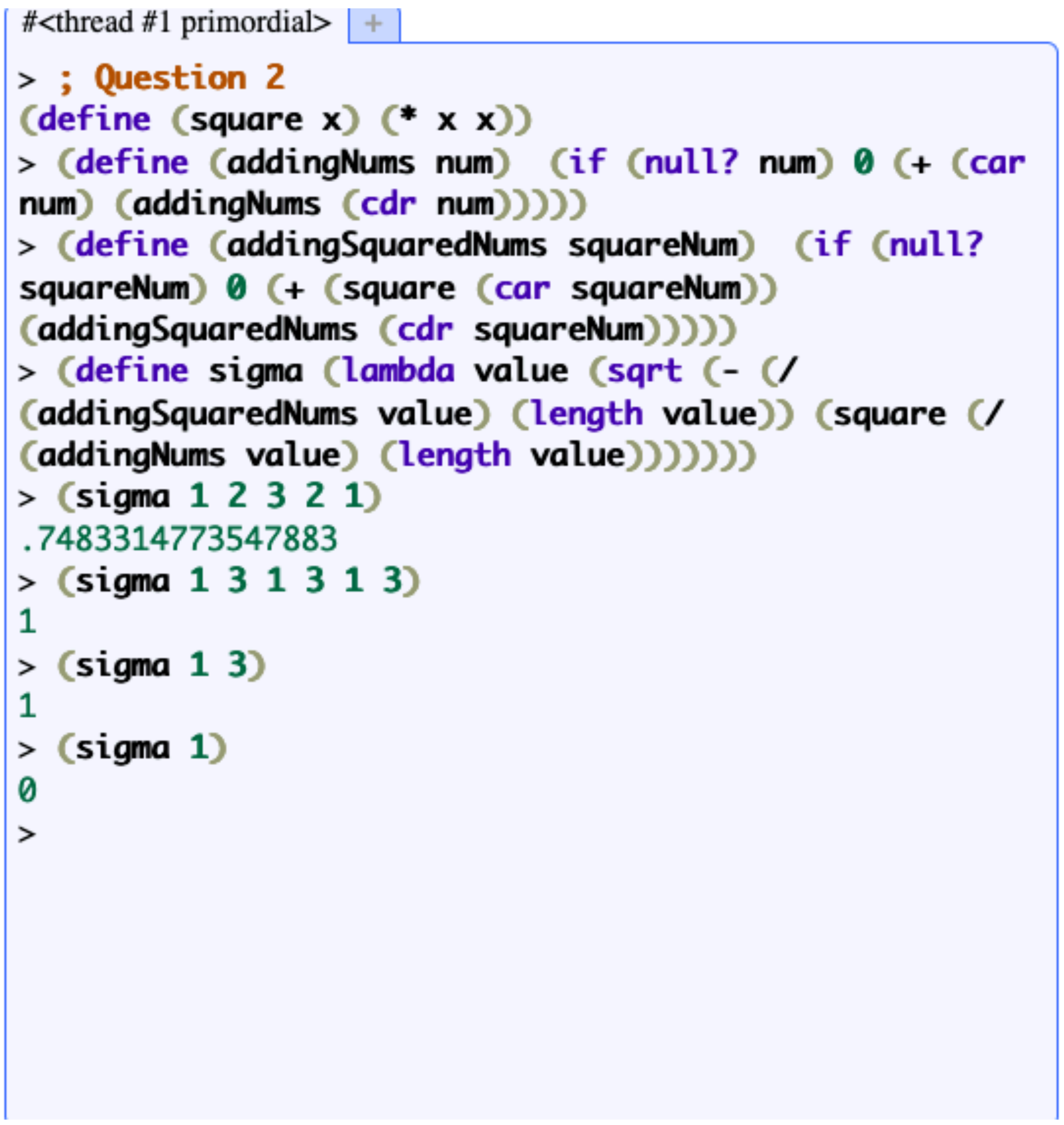


https://try.scheme.org

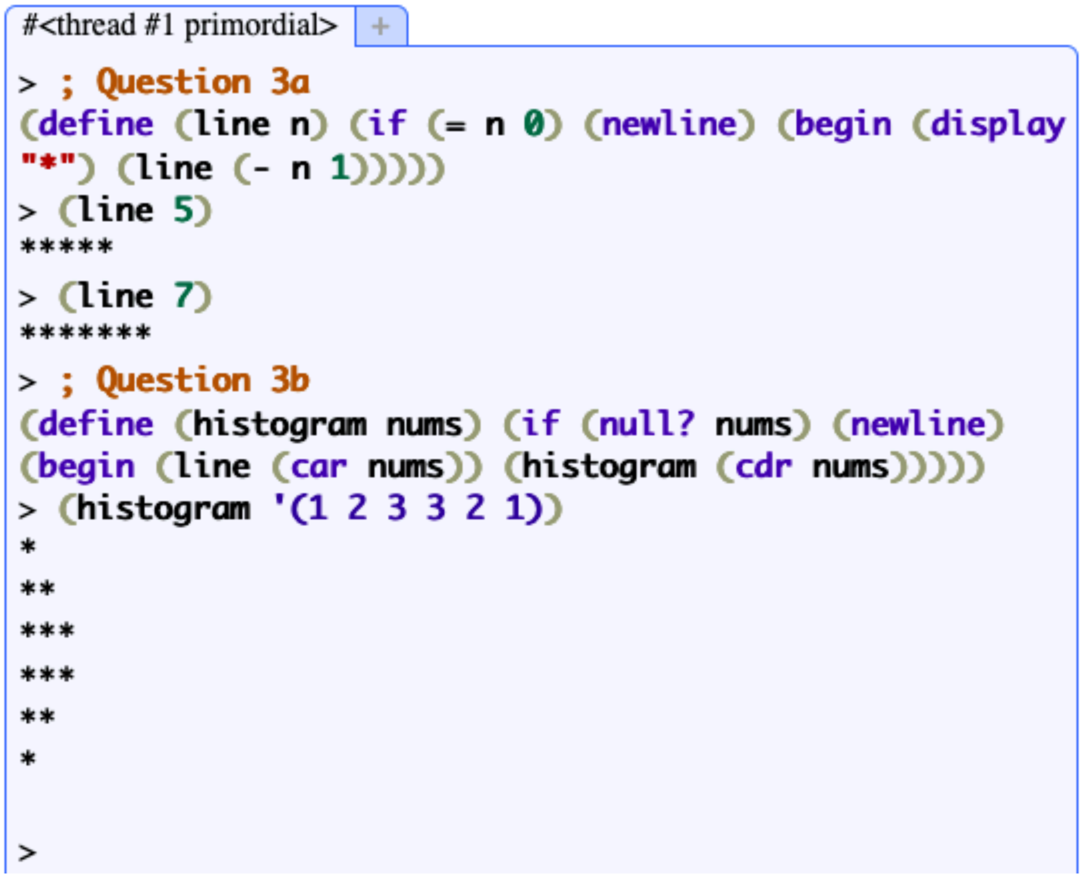
#<thread #1 primordial>

```
> ; Question 1a
250
250
> ; Question 1b
(define (hello name) (display "Hello ") (display name))
> (hello "Sam")
Hello Sam>
; Question 1c
(define myArray (list 0 1 2 3 4 5))
> (display (length myArray))
6>
; Question 1d
(display (equal? myArray '(1 2 3 4 5)))
#f>
; Question 1e
(define (multiply x y) (* x y))
> (display (multiply 5 10))
50>
```

```
; Question 1f
(define (num x) (lambda (y) (+ x y)))
> (define addOne (num 1))
> (display (addOne 6))
7>
; Question 1g
(define myName (read-line)) Sam Zandiasadabadi
> (display myName)
 Sam Zandiasadabadi>
```

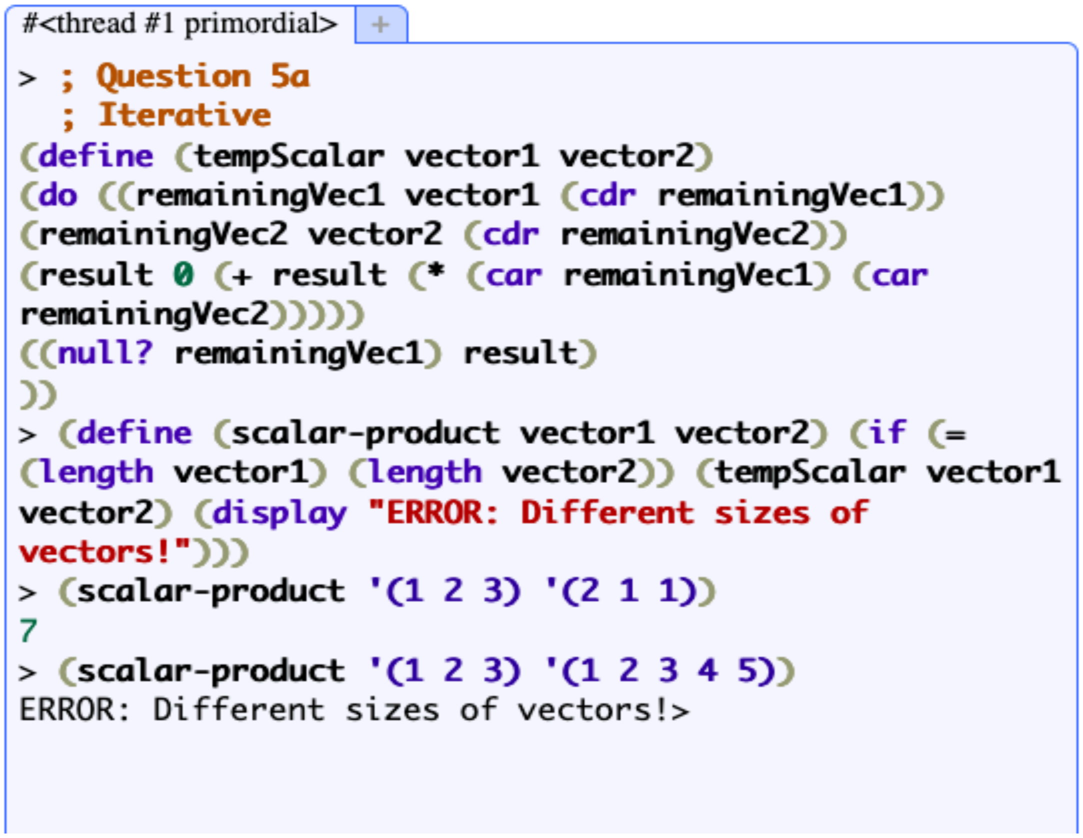


```
> ; Question 2
(define (square x) (* x x))
> (define (addingNums num)  (if (null? num) 0 (+ (car
num) (addingNums (cdr num)))))
> (define (addingSquaredNums squareNum)  (if (null?
squareNum) 0 (+ (square (car squareNum))
(addingSquaredNums (cdr squareNum)))))
> (define sigma (lambda value (sqrt (- (/
(addingSquaredNums value) (length value)) (square (/
(addingNums value) (length value)))))))
> (sigma 1 2 3 2 1)
.7483314773547883
> (sigma 1 3 1 3 1 3)
1
> (sigma 1 3)
1
> (sigma 1)
0
>
```

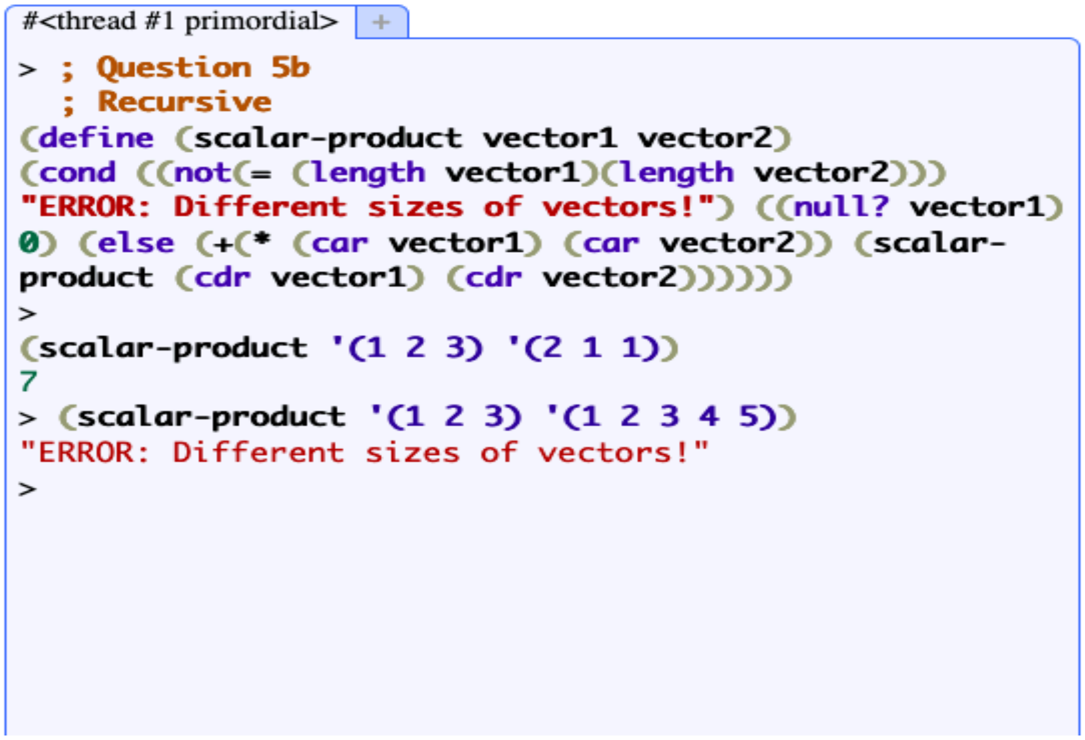

#<thread #1 primordial>

```
> ; Question 3a
(define (line n) (if (= n 0) (newline) (begin (display
"*") (line (- n 1)))))
> (line 5)
*****
> (line 7)
*******
> ; Question 3b
(define (histogram nums) (if (null? nums) (newline)
(begin (line (car nums)) (histogram (cdr nums)))))
> (histogram '(1 2 3 3 2 1))
*
**
***
***
**
*


>
```



```
> ; Question 5a
  ; Iterative
(define (tempScalar vector1 vector2)
(do ((remainingVec1 vector1 (cdr remainingVec1))
(remainingVec2 vector2 (cdr remainingVec2))
(result 0 (+ result (* (car remainingVec1) (car
remainingVec2)))))
((null? remainingVec1) result)
))
> (define (scalar-product vector1 vector2) (if (=
(length vector1) (length vector2)) (tempScalar vector1
vector2) (display "ERROR: Different sizes of
vectors!")))
> (scalar-product '(1 2 3) '(2 1 1))
7
> (scalar-product '(1 2 3) '(1 2 3 4 5))
ERROR: Different sizes of vectors!>
```



```
> ; Question 5b
  ; Recursive
(define (scalar-product vector1 vector2)
(cond ((not(= (length vector1)(length vector2)))
"ERROR: Different sizes of vectors!") ((null? vector1)
0) (else (+(* (car vector1) (car vector2)) (scalar-
product (cdr vector1) (cdr vector2))))))
>
(scalar-product '(1 2 3) '(2 1 1))
7
> (scalar-product '(1 2 3) '(1 2 3 4 5))
"ERROR: Different sizes of vectors!"
>
```