HW #3.1 – Functional programming and schemeFile

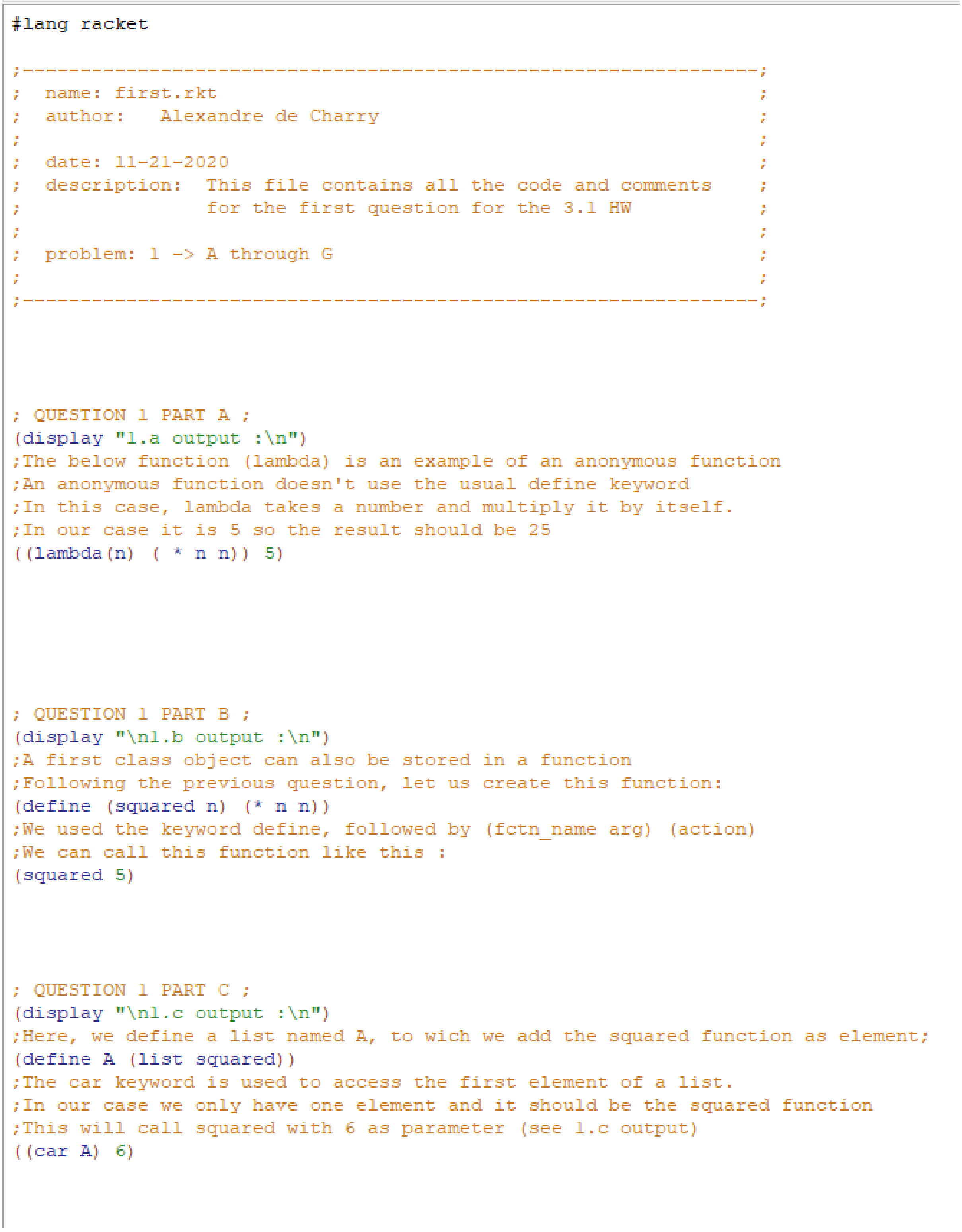
Alexandre de Charry

CSC 600
By Jozo Dujmovic

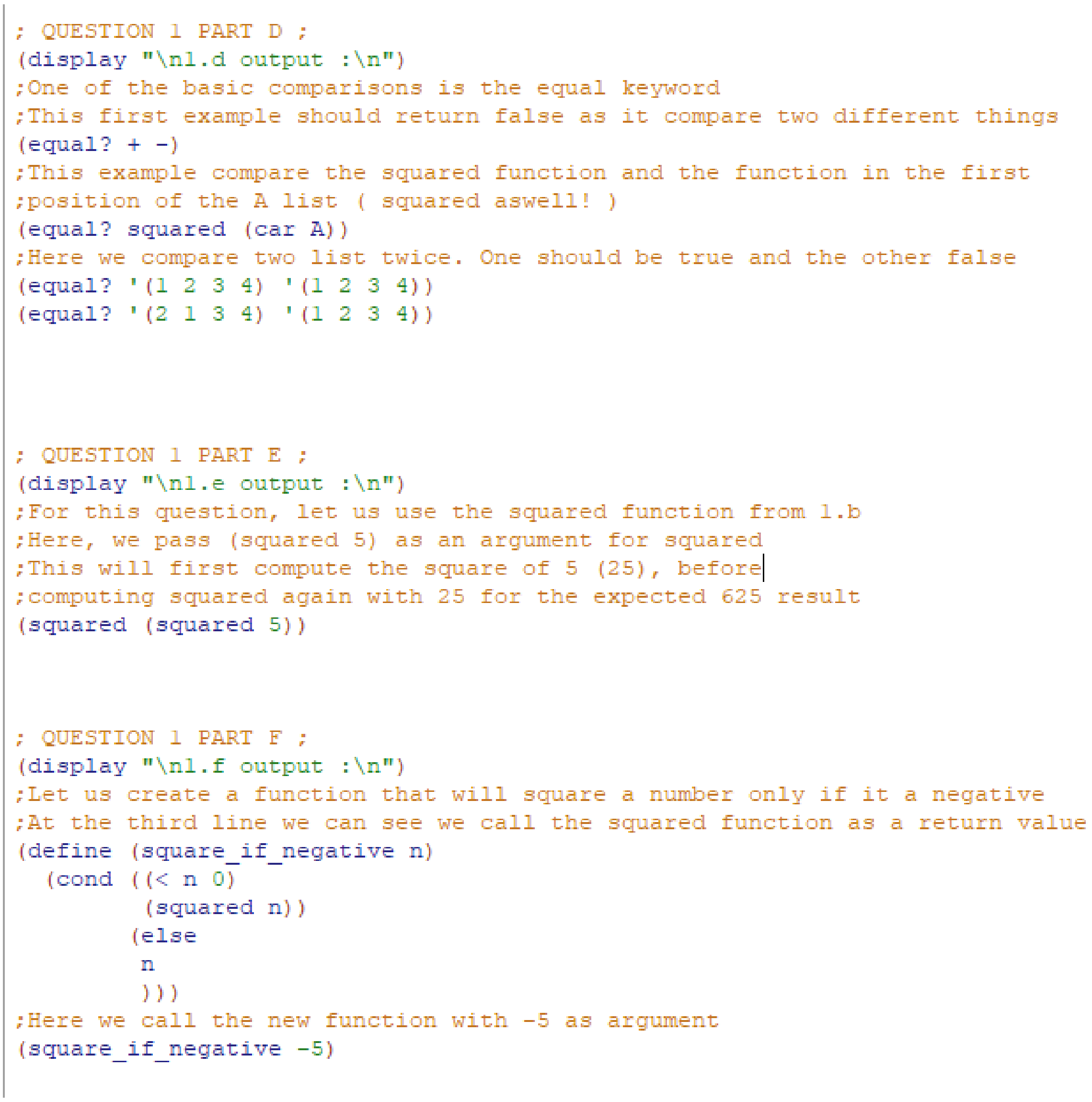
Notes: The code shown will be screenshots (for readability purposes)
*A transcript you can copy paste will be available **at the end** of the file.*

You can also get all this code on my github :
https://github.com/alex7090/CSC600-racket

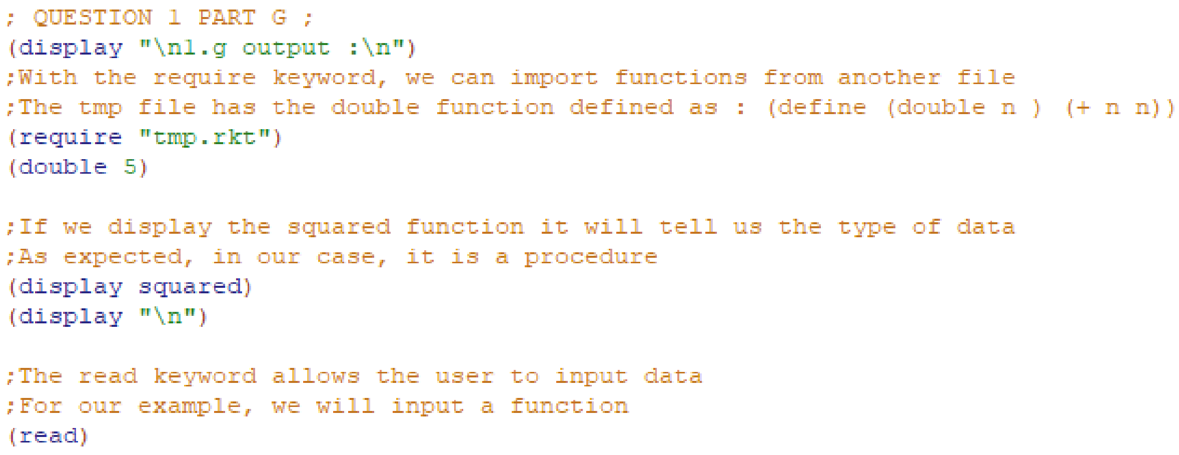
1. Concept of First Class Objects - Scheme



```
#lang racket

;----------------------------------------------------------------;
;  name: first.rkt                                               ;
;  author:   Alexandre de Charry                                 ;
;                                                                ;
;  date: 11-21-2020                                              ;
;  description:  This file contains all the code and comments    ;
;                for the first question for the 3.1 HW           ;
;                                                                ;
;  problem: 1 -> A through G                                     ;
;                                                                ;
;----------------------------------------------------------------;




; QUESTION 1 PART A ;
(display "1.a output :\n")
;The below function (lambda) is an example of an anonymous function
;An anonymous function doesn't use the usual define keyword
;In this case, lambda takes a number and multiply it by itself.
;In our case it is 5 so the result should be 25
((lambda(n) ( * n n)) 5)






; QUESTION 1 PART B ;
(display "\n1.b output :\n")
;A first class object can also be stored in a function
;Following the previous question, let us create this function:
(define (squared n) (* n n))
;We used the keyword define, followed by (fctn_name arg) (action)
;We can call this function like this :
(squared 5)




; QUESTION 1 PART C ;
(display "\n1.c output :\n")
;Here, we define a list named A, to wich we add the squared function as element;
(define A (list squared))
;The car keyword is used to access the first element of a list.
;In our case we only have one element and it should be the squared function
;This will call squared with 6 as parameter (see 1.c output)
((car A) 6)
```

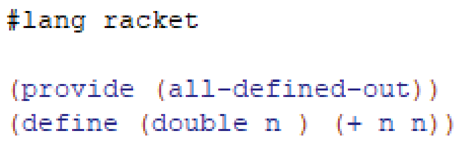


```
; QUESTION 1 PART D ;
(display "\n1.d output :\n")
;One of the basic comparisons is the equal keyword
;This first example should return false as it compare two different things
(equal? + -)
;This example compare the squared function and the function in the first
;position of the A list ( squared aswell! )
(equal? squared (car A))
;Here we compare two list twice. One should be true and the other false
(equal? '(1 2 3 4) '(1 2 3 4))
(equal? '(2 1 3 4) '(1 2 3 4))




; QUESTION 1 PART E ;
(display "\n1.e output :\n")
;For this question, let us use the squared function from 1.b
;Here, we pass (squared 5) as an argument for squared
;This will first compute the square of 5 (25), before
;computing squared again with 25 for the expected 625 result
(squared (squared 5))



; QUESTION 1 PART F ;
(display "\n1.f output :\n")
;Let us create a function that will square a number only if it a negative
;At the third line we can see we call the squared function as a return value
(define (square_if_negative n)
  (cond ((< n 0)
         (squared n))
        (else
         n
         )))
;Here we call the new function with -5 as argument
(square_if_negative -5)
```

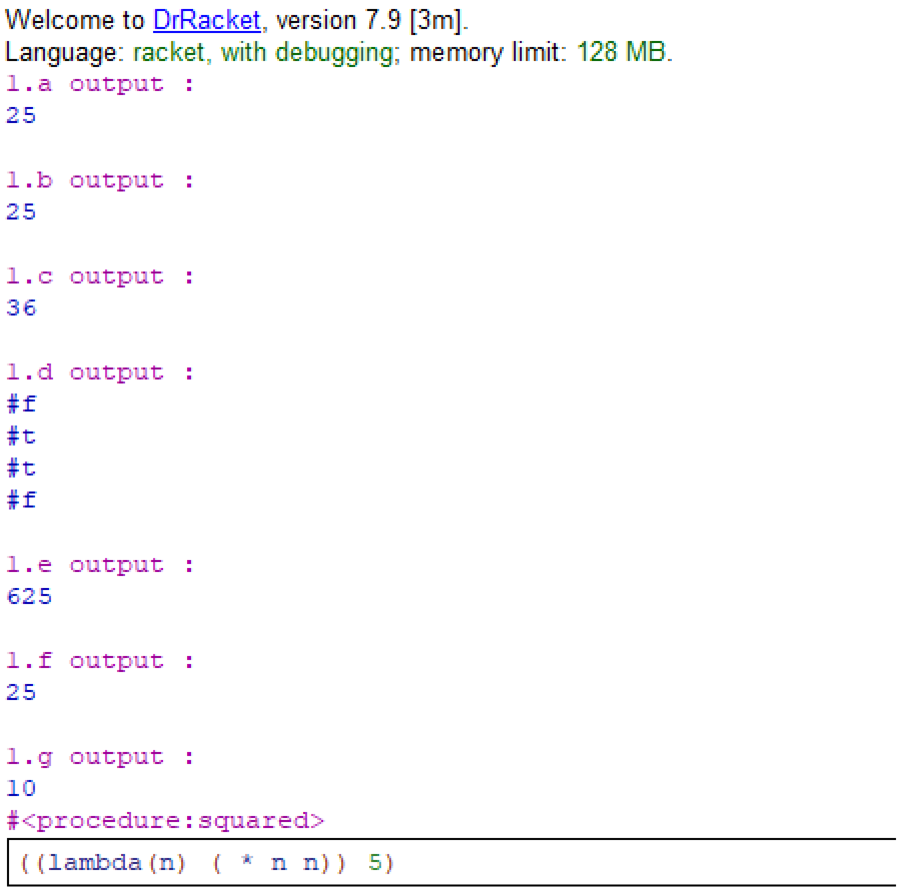


```
; QUESTION 1 PART G ;
(display "\n1.g output :\n")
;With the require keyword, we can import functions from another file
;The tmp file has the double function defined as : (define (double n ) (+ n n))
(require "tmp.rkt")
(double 5)

;If we display the squared function it will tell us the type of data
;As expected, in our case, it is a procedure
(display squared)
(display "\n")

;The read keyword allows the user to input data
;For our example, we will input a function
(read)
```

This is the tmp.rkt file ->

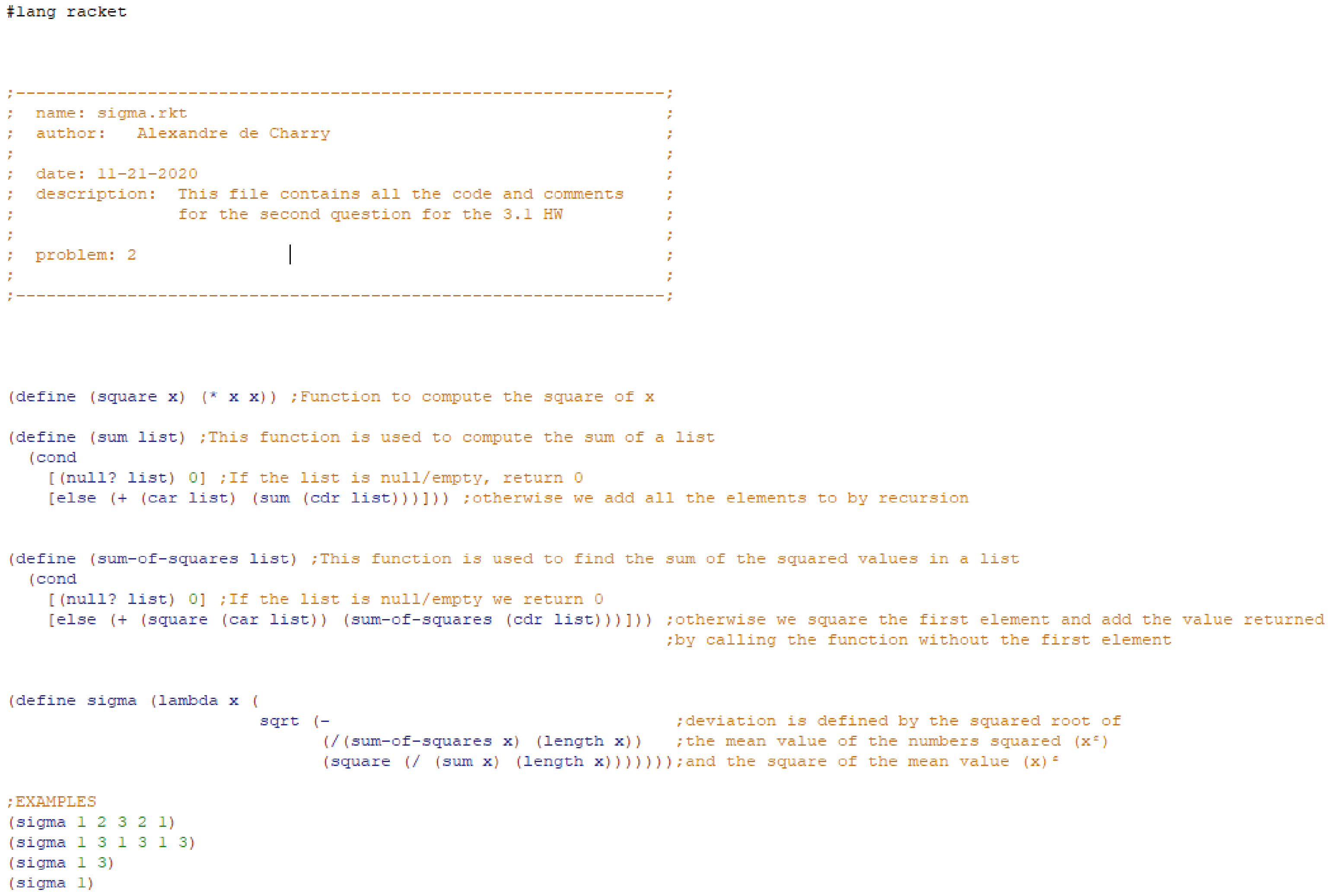


```
#lang racket

(provide (all-defined-out))
(define (double n ) (+ n n))
```

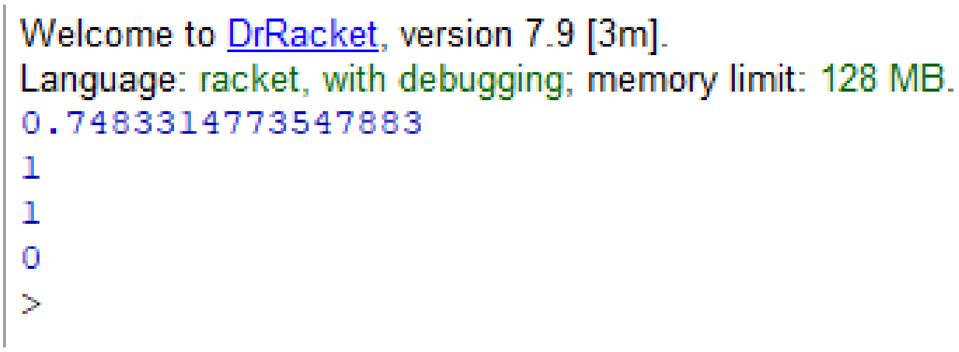


```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
1.a output :
25

1.b output :
25

1.c output :
36

1.d output :
#f
#t
#t
#f

1.e output :
625

1.f output :
25

1.g output :
10
#<procedure:squared>
((lambda(n)  ( * n n))  5)
```

2. Sigma

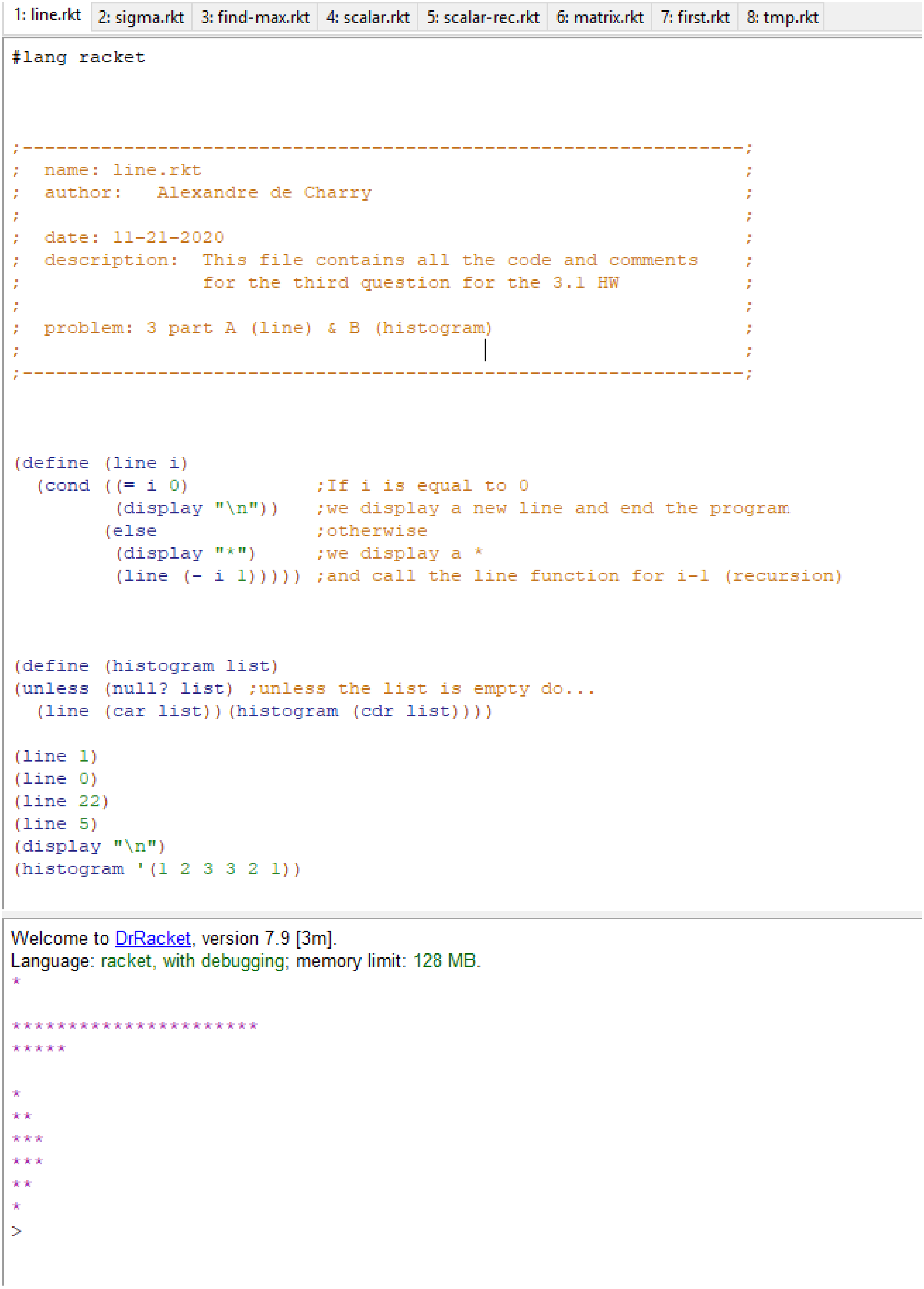


```
#lang racket



;-----------------------------------------------------------------;
;  name: sigma.rkt                                                ;
;  author:   Alexandre de Charry                                  ;
;                                                                 ;
;  date: 11-21-2020                                               ;
;  description:  This file contains all the code and comments     ;
;                for the second question for the 3.1 HW           ;
;                                                                 ;
;  problem: 2                                                     ;
;                                                                 ;
;-----------------------------------------------------------------;




(define (square x) (* x x)) ;Function to compute the square of x

(define (sum list) ;This function is used to compute the sum of a list
  (cond
    [(null? list) 0] ;If the list is null/empty, return 0
    [else (+ (car list) (sum (cdr list)))])) ;otherwise we add all the elements to by recursion


(define (sum-of-squares list) ;This function is used to find the sum of the squared values in a list
  (cond
    [(null? list) 0] ;If the list is null/empty we return 0
    [else (+ (square (car list)) (sum-of-squares (cdr list)))])) ;otherwise we square the first element and add the value returned
                                                                  ;by calling the function without the first element


(define sigma (lambda x (
                         sqrt (-                                  ;deviation is defined by the squared root of
                               (/(sum-of-squares x) (length x))   ;the mean value of the numbers squared (x²)
                               (square (/ (sum x) (length x)))))));and the square of the mean value (x)²

;EXAMPLES
(sigma 1 2 3 2 1)
(sigma 1 3 1 3 1 3)
(sigma 1 3)
(sigma 1)
```



```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
0.7483314773547883
1
1
0
>
```

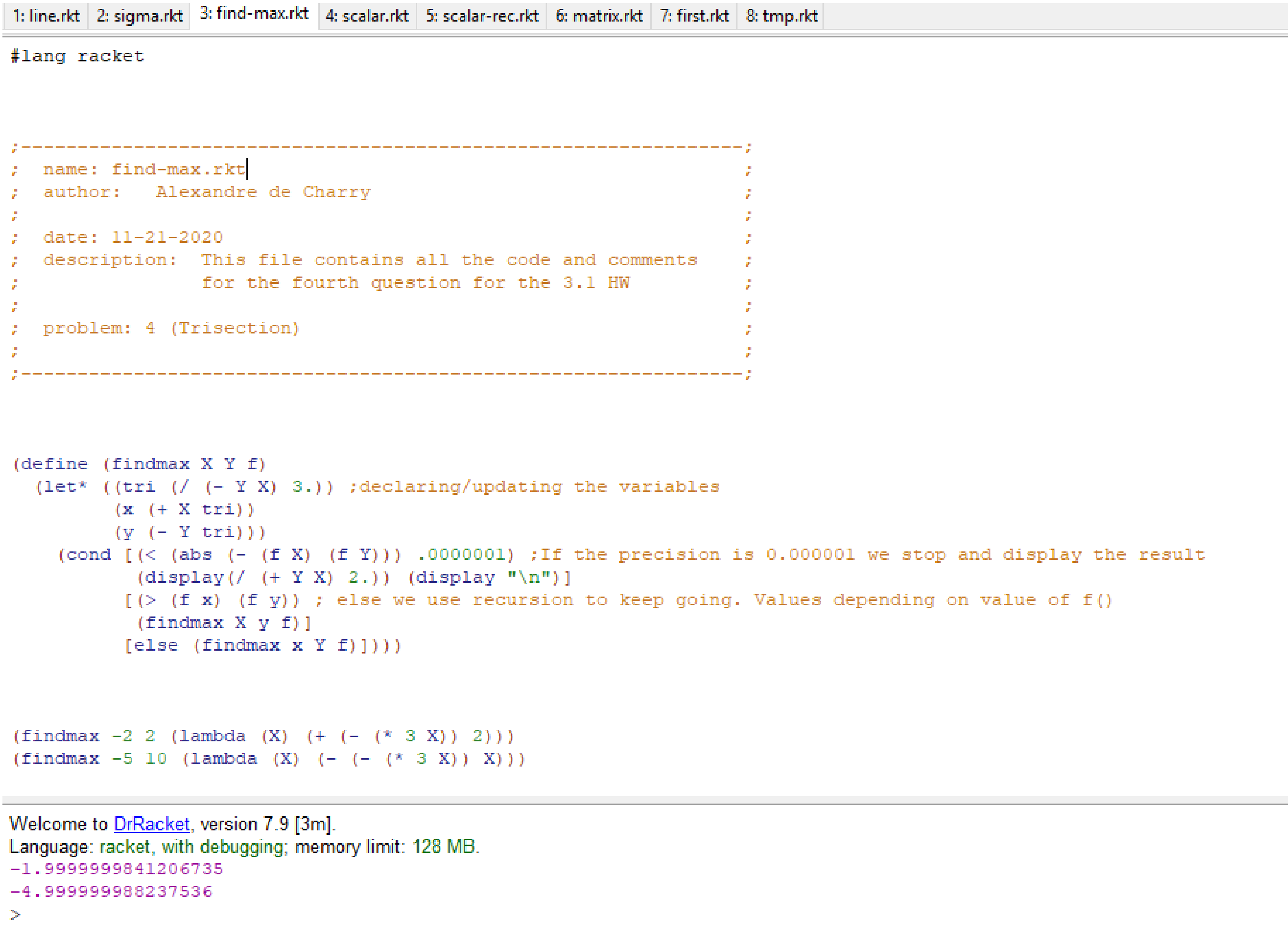
3. Line & Histogram



```
#lang racket



;----------------------------------------------------------------;
;  name: line.rkt                                                ;
;  author:   Alexandre de Charry                                 ;
;                                                                ;
;  date: 11-21-2020                                              ;
;  description:  This file contains all the code and comments    ;
;                for the third question for the 3.1 HW           ;
;                                                                ;
;  problem: 3 part A (line) & B (histogram)                      ;
;                                                                ;
;----------------------------------------------------------------;



(define (line i)
  (cond ((= i 0)              ;If i is equal to 0
         (display "\n"))      ;we display a new line and end the program
        (else                 ;otherwise
         (display "*")        ;we display a *
         (line (- i 1)))))    ;and call the line function for i-1 (recursion)



(define (histogram list)
(unless (null? list) ;unless the list is empty do...
  (line (car list))(histogram (cdr list))))

(line 1)
(line 0)
(line 22)
(line 5)
(display "\n")
(histogram '(1 2 3 3 2 1))
```

```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
*

**********************
*****

*
**
***
***
**
*
>
```

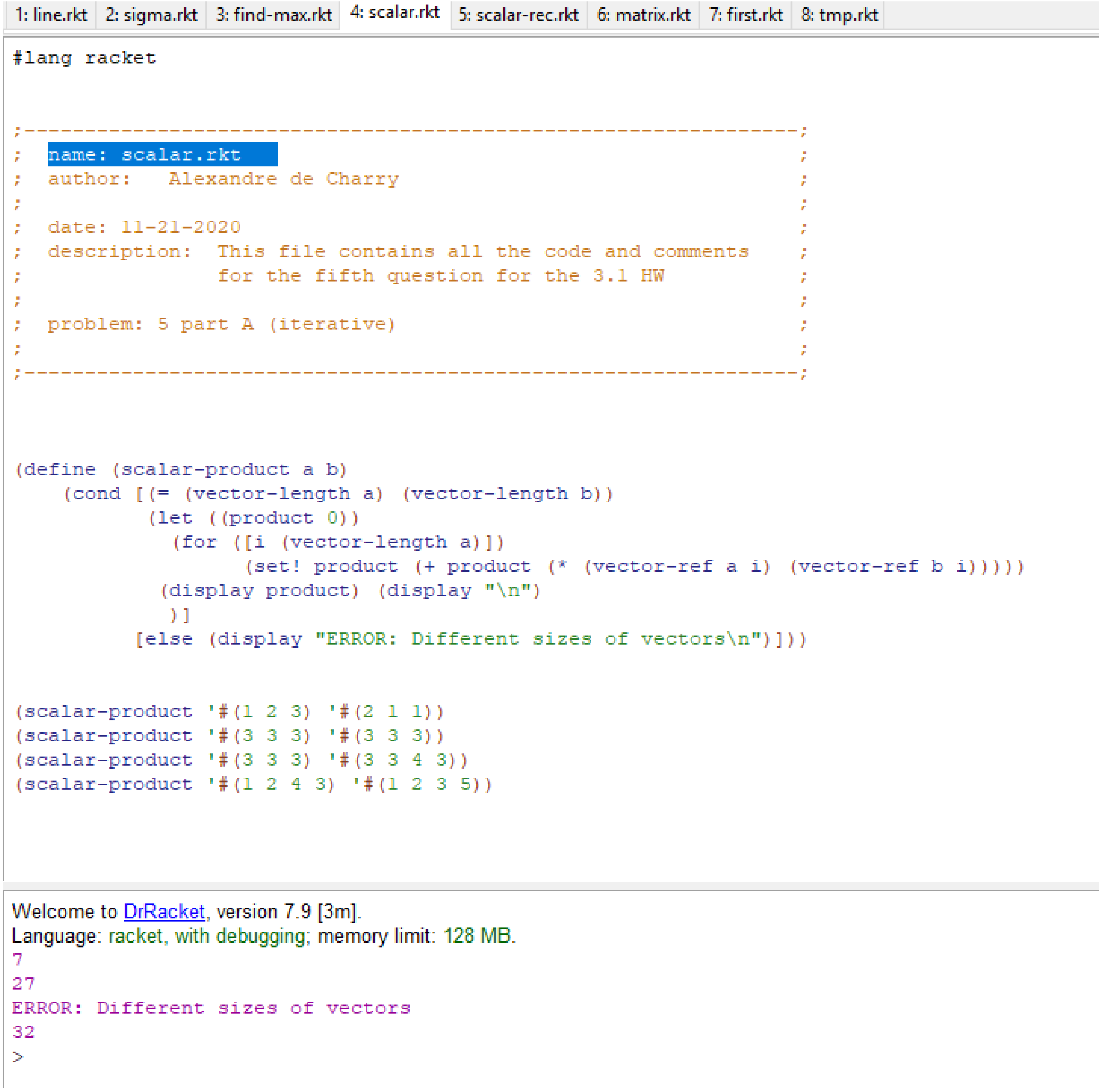
4. Findmax (trisection method)



```
#lang racket



;----------------------------------------------------------------;
;  name: find-max.rkt                                            ;
;  author:   Alexandre de Charry                                 ;
;                                                                ;
;  date: 11-21-2020                                              ;
;  description:  This file contains all the code and comments    ;
;                for the fourth question for the 3.1 HW          ;
;                                                                ;
;  problem: 4 (Trisection)                                       ;
;                                                                ;
;----------------------------------------------------------------;



(define (findmax X Y f)
  (let* ((tri (/ (- Y X) 3.)) ;declaring/updating the variables
         (x (+ X tri))
         (y (- Y tri)))
    (cond [(< (abs (- (f X) (f Y))) .0000001) ;If the precision is 0.000001 we stop and display the result
           (display(/ (+ Y X) 2.)) (display "\n")]
          [(> (f x) (f y)) ; else we use recursion to keep going. Values depending on value of f()
           (findmax X y f)]
          [else (findmax x Y f)])))



(findmax -2 2 (lambda (X) (+ (- (* 3 X)) 2)))
(findmax -5 10 (lambda (X) (- (- (* 3 X)) X)))
```

```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
-1.9999999841206735
-4.999999988237536
>
```

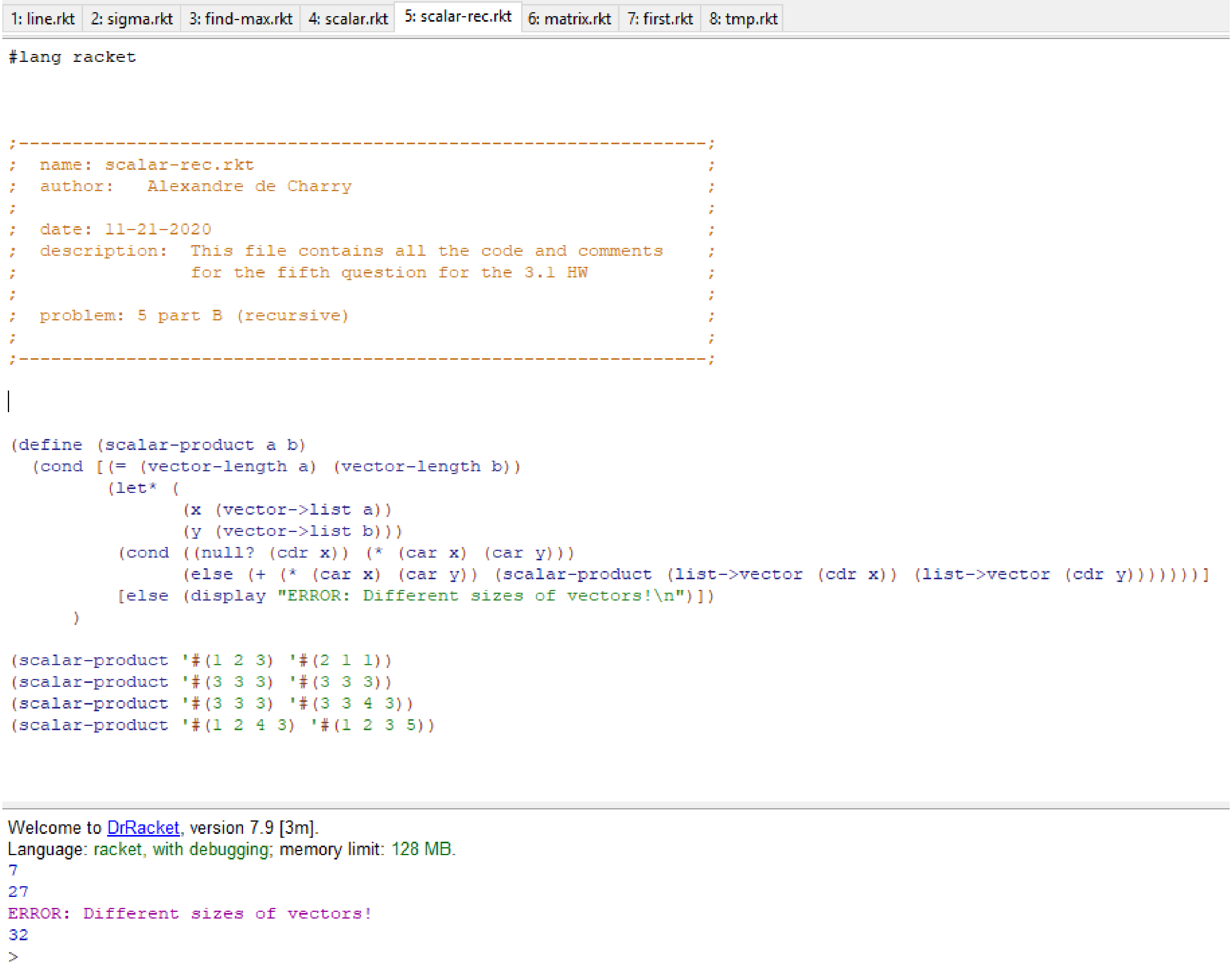
5.A Scalar Product Iterative



```
#lang racket


;-----------------------------------------------------------------;
;  name: scalar.rkt                                               ;
;  author:   Alexandre de Charry                                  ;
;                                                                 ;
;  date: 11-21-2020                                               ;
;  description:  This file contains all the code and comments     ;
;                for the fifth question for the 3.1 HW            ;
;                                                                 ;
;  problem: 5 part A (iterative)                                  ;
;                                                                 ;
;-----------------------------------------------------------------;



(define (scalar-product a b)
    (cond [(= (vector-length a) (vector-length b))
           (let ((product 0))
             (for ([i (vector-length a)])
                   (set! product (+ product (* (vector-ref a i) (vector-ref b i)))))
            (display product) (display "\n")
             )]
          [else (display "ERROR: Different sizes of vectors\n")]))


(scalar-product '#(1 2 3) '#(2 1 1))
(scalar-product '#(3 3 3) '#(3 3 3))
(scalar-product '#(3 3 3) '#(3 3 4 3))
(scalar-product '#(1 2 4 3) '#(1 2 3 5))
```

```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
7
27
ERROR: Different sizes of vectors
32
>
```

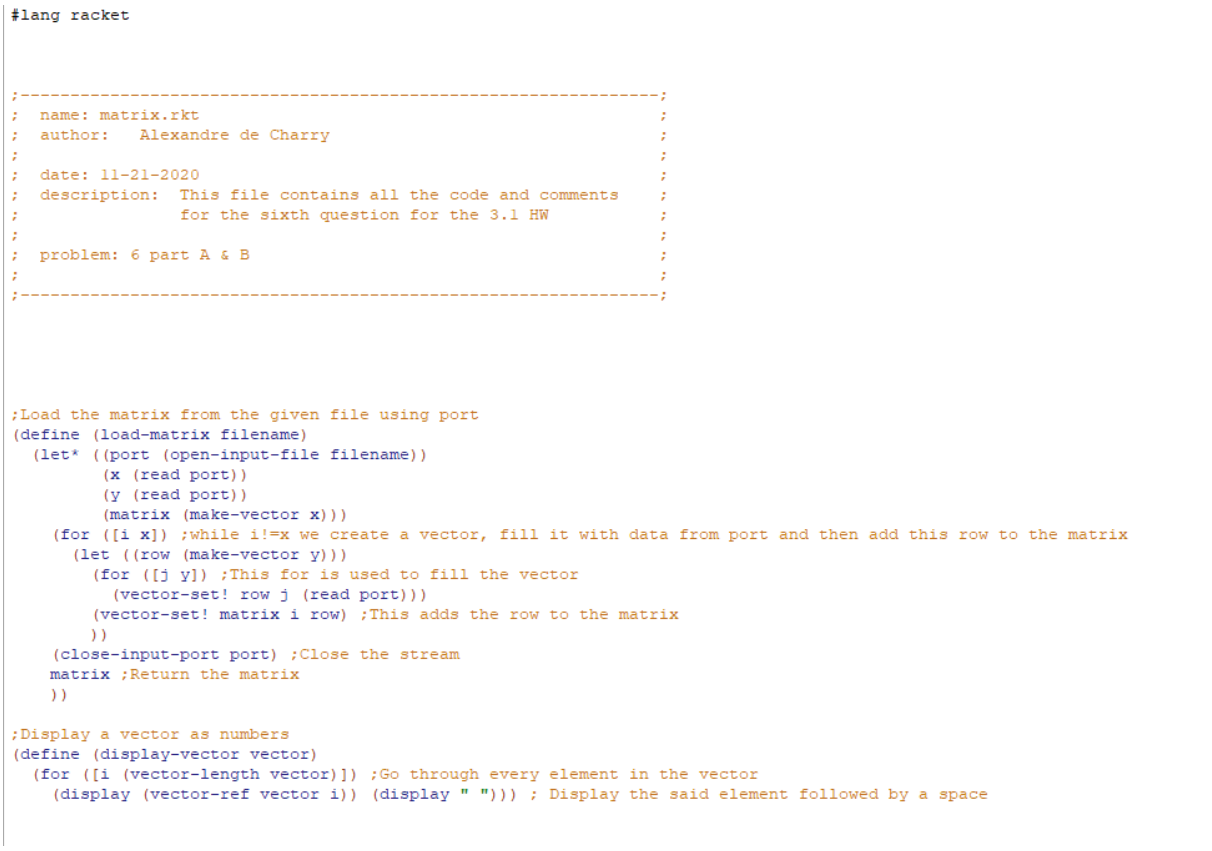
5.B Scalar Product Recursive



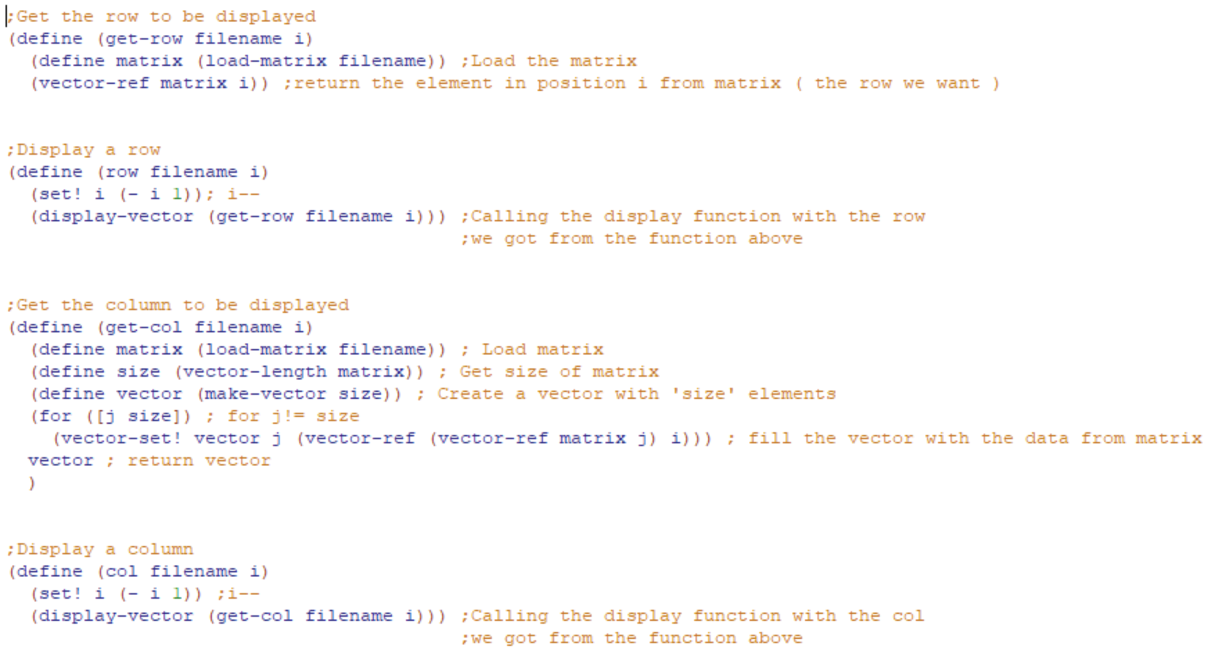
```
#lang racket



;----------------------------------------------------------------;
;  name: scalar-rec.rkt                                          ;
;  author:   Alexandre de Charry                                 ;
;                                                                ;
;  date: 11-21-2020                                              ;
;  description:  This file contains all the code and comments    ;
;                for the fifth question for the 3.1 HW           ;
;                                                                ;
;  problem: 5 part B (recursive)                                 ;
;                                                                ;
;----------------------------------------------------------------;



(define (scalar-product a b)
  (cond [(= (vector-length a) (vector-length b))
         (let* (
                (x (vector->list a))
                (y (vector->list b)))
          (cond ((null? (cdr x)) (* (car x) (car y)))
                (else (+ (* (car x) (car y)) (scalar-product (list->vector (cdr x)) (list->vector (cdr y))))))))]
          [else (display "ERROR: Different sizes of vectors!\n")])
      )

(scalar-product '#(1 2 3) '#(2 1 1))
(scalar-product '#(3 3 3) '#(3 3 3))
(scalar-product '#(3 3 3) '#(3 3 4 3))
(scalar-product '#(1 2 4 3) '#(1 2 3 5))
```

```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
7
27
ERROR: Different sizes of vectors!
32
>
```

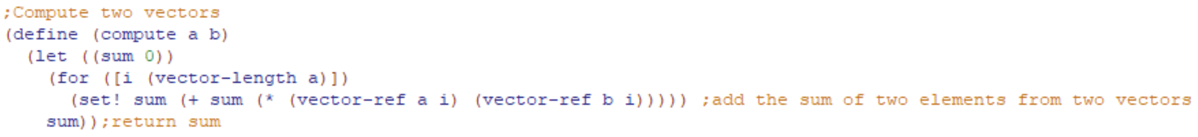
6. Matrix display & multiplication



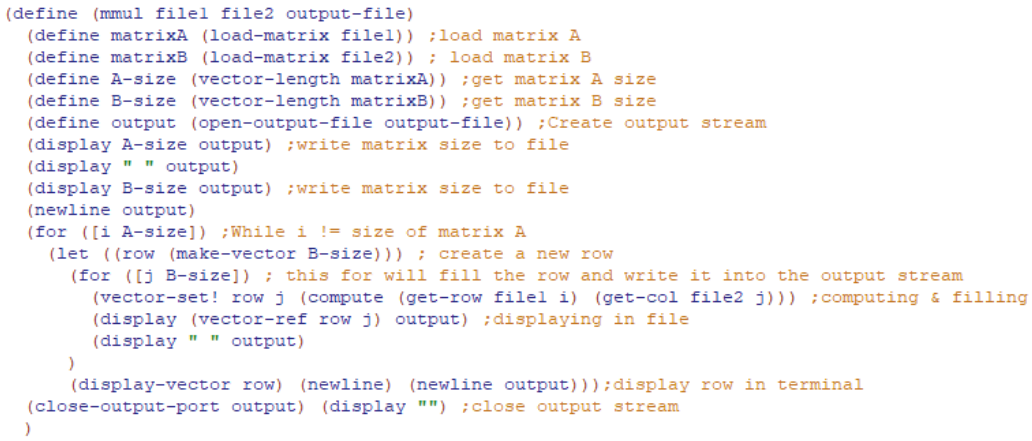
```
#lang racket



;-----------------------------------------------------------------;
;  name: matrix.rkt                                               ;
;  author:   Alexandre de Charry                                  ;
;                                                                 ;
;  date: 11-21-2020                                               ;
;  description:  This file contains all the code and comments    ;
;                for the sixth question for the 3.1 HW            ;
;                                                                 ;
;  problem: 6 part A & B                                          ;
;                                                                 ;
;-----------------------------------------------------------------;




;Load the matrix from the given file using port
(define (load-matrix filename)
  (let* ((port (open-input-file filename))
         (x (read port))
         (y (read port))
         (matrix (make-vector x)))
    (for ([i x]) ;while i!=x we create a vector, fill it with data from port and then add this row to the matrix
      (let ((row (make-vector y)))
        (for ([j y]) ;This for is used to fill the vector
          (vector-set! row j (read port)))
        (vector-set! matrix i row) ;This adds the row to the matrix
        ))
    (close-input-port port) ;Close the stream
    matrix ;Return the matrix
    ))

;Display a vector as numbers
(define (display-vector vector)
  (for ([i (vector-length vector)]) ;Go through every element in the vector
    (display (vector-ref vector i)) (display " "))) ; Display the said element followed by a space
```



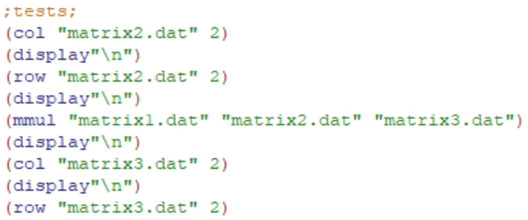
```
;Get the row to be displayed
(define (get-row filename i)
  (define matrix (load-matrix filename)) ;Load the matrix
  (vector-ref matrix i)) ;return the element in position i from matrix ( the row we want )


;Display a row
(define (row filename i)
  (set! i (- i 1)); i--
  (display-vector (get-row filename i))) ;Calling the display function with the row
                                         ;we got from the function above


;Get the column to be displayed
(define (get-col filename i)
  (define matrix (load-matrix filename)) ; Load matrix
  (define size (vector-length matrix)) ; Get size of matrix
  (define vector (make-vector size)) ; Create a vector with 'size' elements
  (for ([j size]) ; for j!= size
    (vector-set! vector j (vector-ref (vector-ref matrix j) i))) ; fill the vector with the data from matrix
  vector ; return vector
  )


;Display a column
(define (col filename i)
  (set! i (- i 1)) ;i--
  (display-vector (get-col filename i))) ;Calling the display function with the col
                                         ;we got from the function above
```



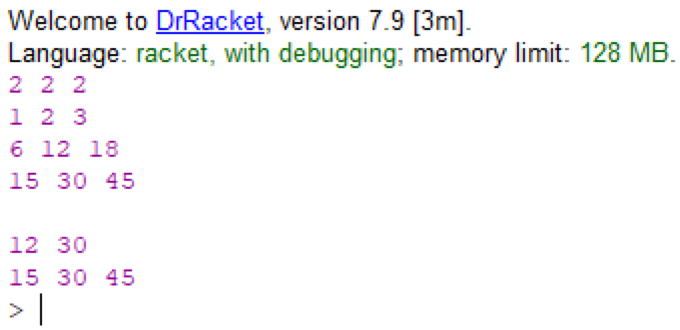
```
;Compute two vectors
(define (compute a b)
  (let ((sum 0))
    (for ([i (vector-length a)])
      (set! sum (+ sum (* (vector-ref a i) (vector-ref b i))))) ;add the sum of two elements from two vectors
    sum));return sum
```



```
(define (mmul file1 file2 output-file)
  (define matrixA (load-matrix file1)) ;load matrix A
  (define matrixB (load-matrix file2)) ; load matrix B
  (define A-size (vector-length matrixA)) ;get matrix A size
  (define B-size (vector-length matrixB)) ;get matrix B size
  (define output (open-output-file output-file)) ;Create output stream
  (display A-size output) ;write matrix size to file
  (display " " output)
  (display B-size output) ;write matrix size to file
  (newline output)
  (for ([i A-size]) ;While i != size of matrix A
    (let ((row (make-vector B-size))) ; create a new row
      (for ([j B-size]) ; this for will fill the row and write it into the output stream
        (vector-set! row j (compute (get-row file1 i) (get-col file2 j))) ;computing & filling
        (display (vector-ref row j) output) ;displaying in file
        (display " " output)
      )
      (display-vector row) (newline) (newline output)));display row in terminal
  (close-output-port output) (display "") ;close output stream
  )
```



```
;tests;
(col "matrix2.dat" 2)
(display"\n")
(row "matrix2.dat" 2)
(display"\n")
(mmul "matrix1.dat" "matrix2.dat" "matrix3.dat")
(display"\n")
(col "matrix3.dat" 2)
(display"\n")
(row "matrix3.dat" 2)
```



```
Welcome to DrRacket, version 7.9 [3m].
Language: racket, with debugging; memory limit: 128 MB.
2 2 2
1 2 3
6 12 18
15 30 45

12 30
15 30 45
> 
```

FULL TRANSCRIPT

QUESTION 1

```
#lang racket

;-----------------------------------------------------------------;
;  name: first.rkt                                  ;
;  author:   Alexandre de Charry                              ;
;                                                   ;
;  date: 11-21-2020                                 ;
;  description:  This file contains all the code and comments    ;
;               for the first question for the 3.1 HW          ;
;                                                   ;
;  problem: 1 -> A through G                             ;
;                                                   ;
;-----------------------------------------------------------------;




; QUESTION 1 PART A ;
(display "1.a output :\n")
;The below function (lambda) is an example of an anonymous function
;An anonymous function doesn't use the usual define keyword
;In this case, lambda takes a number and multiply it by itself.
;In our case it is 5 so the result should be 25
((lambda(n) ( * n n)) 5)




; QUESTION 1 PART B ;
(display "\n1.b output :\n")
;A first class object can also be stored in a function
;Following the previous question, let us create this function:
(define (squared n) (* n n))
```

```scheme
;We used the keyword define, followed by (fctn_name arg) (action)
;We can call this function like this :
(squared 5)



; QUESTION 1 PART C ;
(display "\n1.c output :\n")
;Here, we define a list named A, to wich we add the squared function as element;
(define A (list squared))
;The car keyword is used to access the first element of a list.
;In our case we only have one element and it should be the squared function
;This will call squared with 6 as parameter (see 1.c output)
((car A) 6)



; QUESTION 1 PART D ;
(display "\n1.d output :\n")
;One of the basic comparisons is the equal keyword
;This first example should return false as it compare two different things
(equal? + -)
;This example compare the squared function and the function in the first
;position of the A list ( squared aswell! )
(equal? squared (car A))
;Here we compare two list twice. One should be true and the other false
(equal? '(1 2 3 4) '(1 2 3 4))
(equal? '(2 1 3 4) '(1 2 3 4))



; QUESTION 1 PART E ;
(display "\n1.e output :\n")
;For this question, let us use the squared function from 1.b
;Here, we pass (squared 5) as an argument for squared
;This will first compute the square of 5 (25), before
;computing squared again with 25 for the expected 625 result
(squared (squared 5))
```

```
; QUESTION 1 PART F ;
(display "\n1.f output :\n")
;Let us create a function that will square a number only if it a negative
;At the third line we can see we call the squared function as a return value
(define (square_if_negative n)
  (cond ((< n 0)
         (squared n))
        (else
         n
         )))
;Here we call the new function with -5 as argument
(square_if_negative -5)
```

```
; QUESTION 1 PART G ;
(display "\n1.g output :\n")
;With the require keyword, we can import functions from another file
;The tmp file has the double function defined as : (define (double n ) (+ n n))
(require "tmp.rkt")
(double 5)

;If we display the squared function it will tell us the type of data
;As expected, in our case, it is a procedure
(display squared)
(display "\n")

;The read keyword allows the user to input data
;For our example, we will input a function
(read)
```

QUESTION 2

```
#lang racket



;--------------------------------------------------------------;
;  name: sigma.rkt                              ;
;  author:   Alexandre de Charry                        ;
;                                               ;
;  date: 11-21-2020                             ;
;  description:  This file contains all the code and comments    ;
;              for the second question for the 3.1 HW         ;
;                                               ;
;  problem: 2                                   ;
;                                               ;
;--------------------------------------------------------------;




(define (square x) (* x x)) ;Function to compute the square of x

(define (sum list) ;This function is used to compute the sum of a list
  (cond
    [(null? list) 0] ;If the list is null/empty, return 0
    [else (+ (car list) (sum (cdr list)))])) ;otherwise we add all the elements to by recursion



(define (sum-of-squares list) ;This function is used to find the sum of the squared values in a
list
  (cond
    [(null? list) 0] ;If the list is null/empty we return 0
    [else (+ (square (car list)) (sum-of-squares (cdr list)))])) ;otherwise we square the first
element and add the value returned
                                                 ;by calling the function without the first element



(define sigma (lambda x (
                    sqrt (-                          ;deviation is defined by the squared root of
                         (/(sum-of-squares x) (length x))   ;the mean value of the numbers squared
(x²)
                         (square (/ (sum x) (length x))))))));and the square of the mean value (x)²
```

```
;EXAMPLES
(sigma 1 2 3 2 1)
(sigma 1 3 1 3 1 3)
(sigma 1 3)
(sigma 1)
```

QUESTION 3

```
#lang racket



;-------------------------------------------------------------;
;  name: line.rkt                                    ;
;  author:   Alexandre de Charry                            ;
;                                                  ;
;  date: 11-21-2020                                    ;
;  description:  This file contains all the code and comments    ;
;               for the third question for the 3.1 HW          ;
;                                                  ;
;  problem: 3 part A (line) & B (histogram)                   ;
;                                                  ;
;-------------------------------------------------------------;



(define (line i)
  (cond ((= i 0)          ;If i is equal to 0
        (display "\n"))   ;we display a new line and end the program
       (else            ;otherwise
        (display "*")     ;we display a *
        (line (- i 1))))) ;and call the line function for i-1 (recursion)
```

```
(define (histogram list)
(unless (null? list) ;unless the list is empty do...
  (line (car list))(histogram (cdr list))))

(line 1)
(line 0)
(line 22)
(line 5)
(display "\n")
(histogram '(1 2 3 3 2 1))
```

QUESTION 4

```
#lang racket


;----------------------------------------------------------------;
;  name: find-max.rkt                                  ;
;  author:   Alexandre de Charry                         ;
;                                                 ;
;  date: 11-21-2020                                   ;
;  description:  This file contains all the code and comments    ;
;              for the fourth question for the 3.1 HW          ;
;                                                 ;
;  problem: 4 (Trisection)                               ;
;                                                 ;
;----------------------------------------------------------------;

(define (findmax X Y f)
  (let* ((tri (/ (- Y X) 3.)) ;declaring/updating the variables
        (x (+ X tri))
        (y (- Y tri)))
    (cond [(< (abs (- (f X) (f Y))) .0000001) ;If the precision is 0.000001 we stop and display the
result
          (display(/ (+ Y X) 2.)) (display "\n")]
         [(> (f x) (f y)) ; else we use recursion to keep going. Values depending on value of f()
          (findmax X y f)]
         [else (findmax x Y f)])))
```

```
(findmax -2 2 (lambda (X) (+ (- (* 3 X)) 2)))
(findmax -5 10 (lambda (X) (- (- (* 3 X)) X)))
```

QUESTION 5.A

```
#lang racket


;-------------------------------------------------------------;
;  name: scalar.rkt                                  ;
;  author:   Alexandre de Charry                          ;
;                                               ;
;  date: 11-21-2020                                  ;
;  description:  This file contains all the code and comments    ;
;              for the fifth question for the 3.1 HW          ;
;                                               ;
;  problem: 5 part A (iterative)                         ;
;                                               ;
;-------------------------------------------------------------;



(define (scalar-product a b)
   (cond [(= (vector-length a) (vector-length b))
        (let ((product 0))
          (for ([i (vector-length a)])
               (set! product (+ product (* (vector-ref a i) (vector-ref b i)))))
         (display product) (display "\n")
          )]
        [else (display "ERROR: Different sizes of vectors\n")]))


(scalar-product '#(1 2 3) '#(2 1 1))
(scalar-product '#(3 3 3) '#(3 3 3))
(scalar-product '#(3 3 3) '#(3 3 4 3))
(scalar-product '#(1 2 4 3) '#(1 2 3 5))
```

QUESTION 5.B

```
#lang racket



;----------------------------------------------------------------;
;  name: scalar-rec.rkt                                  ;
;  author:   Alexandre de Charry                             ;
;                                              ;
;  date: 11-21-2020                                  ;
;  description:  This file contains all the code and comments    ;
;              for the fifth question for the 3.1 HW          ;
;                                              ;
;  problem: 5 part B (recursive)                         ;
;                                              ;
;----------------------------------------------------------------;



(define (scalar-product a b)
  (cond [(= (vector-length a) (vector-length b))
       (let* (
            (x (vector->list a))
            (y (vector->list b)))
        (cond ((null? (cdr x)) (* (car x) (car y)))
            (else (+ (* (car x) (car y)) (scalar-product (list->vector (cdr x)) (list->vector (cdr
y)))))))]
        [else (display "ERROR: Different sizes of vectors!\n")])
    )

(scalar-product '#(1 2 3) '#(2 1 1))
(scalar-product '#(3 3 3) '#(3 3 3))
(scalar-product '#(3 3 3) '#(3 3 4 3))
(scalar-product '#(1 2 4 3) '#(1 2 3 5))
```

QUESTION 6

```
#lang racket



;-----------------------------------------------------------------;
;  name: matrix.rkt                                   ;
;  author:   Alexandre de Charry                             ;
;                                                  ;
;  date: 11-21-2020                                   ;
;  description:  This file contains all the code and comments    ;
;              for the sixth question for the 3.1 HW          ;
;                                                  ;
;  problem: 6 part A & B                              ;
;                                                  ;
;-----------------------------------------------------------------;




;Load the matrix from the given file using port
(define (load-matrix filename)
  (let* ((port (open-input-file filename))
        (x (read port))
        (y (read port))
        (matrix (make-vector x)))
    (for ([i x]) ;while i!=x we create a vector, fill it with data from port and then add this row to
the matrix
      (let ((row (make-vector y)))
        (for ([j y]) ;This for is used to fill the vector
          (vector-set! row j (read port)))
        (vector-set! matrix i row) ;This adds the row to the matrix
        ))
    (close-input-port port) ;Close the stream
    matrix ;Return the matrix
    ))

;Display a vector as numbers
(define (display-vector vector)
  (for ([i (vector-length vector)]) ;Go through every element in the vector
    (display (vector-ref vector i)) (display " "))) ; Display the said element followed by a space
```

```
;Get the row to be displayed
(define (get-row filename i)
  (define matrix (load-matrix filename)) ;Load the matrix
  (vector-ref matrix i)) ;return the element in position i from matrix ( the row we want )


;Display a row
(define (row filename i)
  (set! i (- i 1)); i--
  (display-vector (get-row filename i))) ;Calling the display function with the row
                                        ;we got from the function above


;Get the column to be displayed
(define (get-col filename i)
  (define matrix (load-matrix filename)) ; Load matrix
  (define size (vector-length matrix)) ; Get size of matrix
  (define vector (make-vector size)) ; Create a vector with 'size' elements
  (for ([j size]) ; for j!= size
    (vector-set! vector j (vector-ref (vector-ref matrix j) i))) ; fill the vector with the data from matrix
  vector ; return vector
  )


;Display a column
(define (col filename i)
  (set! i (- i 1)) ;i--
  (display-vector (get-col filename i))) ;Calling the display function with the col
                                        ;we got from the function above



;Compute two vectors
(define (compute a b)
  (let ((sum 0))
    (for ([i (vector-length a)])
      (set! sum (+ sum (* (vector-ref a i) (vector-ref b i))))) ;add the sum of two elements from two vectors
    sum));return sum
```

```
(define (mmul file1 file2 output-file)
  (define matrixA (load-matrix file1)) ;load matrix A
  (define matrixB (load-matrix file2)) ; load matrix B
  (define A-size (vector-length matrixA)) ;get matrix A size
  (define B-size (vector-length matrixB)) ;get matrix B size
  (define output (open-output-file output-file)) ;Create output stream
  (display A-size output) ;write matrix size to file
  (display " " output)
  (display B-size output) ;write matrix size to file
  (newline output)
  (for ([i A-size]) ;While i != size of matrix A
    (let ((row (make-vector B-size))) ; create a new row
      (for ([j B-size]) ; this for will fill the row and write it into the output stream
        (vector-set! row j (compute (get-row file1 i) (get-col file2 j))) ;computing & filling
        (display (vector-ref row j) output) ;displaying in file
        (display " " output)
      )
      (display-vector row) (newline) (newline output)));display row in terminal
  (close-output-port output) (display "") ;close output stream
  )
```

```
;tests;
(col "matrix2.dat" 2)
(display"\n")
(row "matrix2.dat" 2)
(display"\n")
(mmul "matrix1.dat" "matrix2.dat" "matrix3.dat")
(display"\n")
(col "matrix3.dat" 2)
(display"\n")
(row "matrix3.dat" 2)
```