Ray Dela Cruz
5/3/24
CSC 600
Prof Dujmović
HW 3.1

```
#lang scheme

;1.a)
((lambda (x) (* 2 x)) 2)
;b
(define double (lambda (x) (* x 2)))
(double 2)
;c
(define doubletriplelist (list (lambda (x) (* x 2))
                               (lambda (x) (* x 3))))
((car doubletriplelist) 2)
((cadr doubletriplelist) 2)
;d
(equal? '(lambda (x) (* x 2)) '(lambda (x) (* x 2)))
;e
(define funcParam (lambda (f x) (f x)))
(funcParam (lambda (x) (* 2 x)) 2)
;f
(define adder (lambda (x) (lambda (y) (+ x y))))
(define add-five (adder 5))
(add-five 5)
;g
(display adder)
(newline)
;2
(define (sigma . nums)
  (if (= (length nums) 1) ; check if single element
      0
      (let* ((n (length nums)) ; Number of elements in list
             (sum (apply + nums)) ; Summation of all the elements
             (sum-of-squares (apply + (map (lambda (x) (* x x)) nums))) ; Summation of all the
squares of list
             (mean (/ sum n)) ; Average of elements
             (mean-square (/ sum-of-squares n)) ; Average of square of elements
             (variance (- mean-square (* mean mean)))) ; Variance
        (sqrt variance)))) ; Standard deviation
(sigma 1 2 3 2 1)
(sigma 1 3 1 3 1 3)
(sigma 1 3)
;3a
(define (line n)
  (cond ; Set conditions
    ((<= n 0) (newline))  ; Check if n <= 0
    (else
      (begin
        (display "*")       ; Print one asterisk
        (line (- n 1)))))   ; Recursive call n-1
)
```

```
(line 5)
(newline)
;b
(define (histogram list)
  (if (null? list)
      'done ; Check if the list is empty
      (begin
        (line (car list)) ; Call 'line' with the first element of the list
        (histogram (cdr list))))) ; Recersive call to histogram
(histogram '(1 2 3 3 2 1))
(newline)
;4
(define (f x)
  (exact->inexact (- (- (* x x)) (* 2 x) 2)))  ; f(x) = -x^2 - 2x - 2

; Trisection method definition to find the maximum of f over an interval [x1, x2]
(define (find-maximum f x1 x2 tolerance)
  (let loop ((xL (exact->inexact x1)) (xR (exact->inexact x2)))  ; Floating-point calculations
    (let* ((d (/ (- xR xL) 3))  ; Divide the interval into three parts
           (x1 (+ xL d))        ; Calculate first trisection point
           (x2 (- xR d)))       ; Calculate second trisection point
      (if (< (- xR xL) tolerance)  ; Check if the current interval is smaller than tolerance
          (/ (+ xL xR) 2)          ; Return the midpoint as maximum
          (let ((fx1 (f x1))
                (fx2 (f x2)))
            (if (> fx1 fx2)
                (loop xL x2)   ; If f(x1) is greater, continue in the left subinterval
                (loop x1 xR)))))))  ; Otherwise, continue in the right subinterval

(define maximum-x (find-maximum f -3 3 1e-10))
(display "The coordinate of the maximum, xmax: ")  ; Display the x-coordinate of the maximum
(display maximum-x)  ; Print the maximum
(newline)
(display "Maximum value at xmax: ")  ; Display the maximum value
(display (f maximum-x))  ; Calculate and print the maximum value of f at xmax
(newline)
;5a
(define (scalar-product v1 v2)
  (if (= (vector-length v1) (vector-length v2))
      (let ((sum 0))
        (do ((i 0 (+ i 1)))  ; Initialize i to 0 and increment by 1 each iteration
            ((= i (vector-length v1)) sum)  ; Loop until i equals the length of v1, then return sum
          (set! sum (+ sum (* (vector-ref v1 i) (vector-ref v2 i)))))  ; Multiply corresponding elements
and add to sum
        )
      (display "ERROR: Different sizes of vectors!")))
(scalar-product '#(1 2 3) '#(2 1 1))
(scalar-product '#(1 2 3) '#(1 2 3 4 5))
(newline)
;b
(define (scalar-product2 v1 v2)
  (if (= (vector-length v1) (vector-length v2))
      (letrec ((recursive-sum (lambda (i sum)
                                (if (= i (vector-length v1))
```

```
                        sum
                        (recursive-sum (+ i 1)
                                (+ sum (* (vector-ref v1 i) (vector-ref v2 i))))))))
      (recursive-sum 0 0))
    (display "ERROR: Different sizes of vectors!")))
(scalar-product2 '#(1 2 3) '#(2 1 1))
(scalar-product2 '#(1 2 3) '#(1 2 3 4 5))
(newline)
```

```
Welcome to DrRacket, version 8.12 [cs].
Language: scheme, with debugging; memory limit: 128 MB.
4
4
4
6
#t
4
10
#<procedure:adder>
0.7483314773547883
1
1
*****

*
**
***
***
**
*
done

The coordinate of the maximum, xmax: -0.999999987129955
Maximum value at xmax: -1.0
7
ERROR: Different sizes of vectors!
7
ERROR: Different sizes of vectors!
>
```