Problem 1-B.) Two selector

A screen shot of a computer program

Description automatically generated

Problem 1-A.) Multi-way selector

A screenshot of a computer program

Description automatically generated

Problem 2: Remove 2nd and third element

A screen shot of a computer

Description automatically generated

Problem 3: Delete atom from nested list

A screenshot of a computer program

Description automatically generated

Problem 1 -B code

#lang racket

; area of equilateral triangle - sqrt 3 /4 \* a\*\*2 where a = length of side

; area of hexagon - 3 \* sqrt(3) / 2 \* a\*\*2 where a = length of side

; helper functions

; square func

(define (squared x) (\* x x))

; func for calculating eq triangle

(define (calc\_eq\_triangle sqrt3 a) (\* (/ sqrt3 4) (squared a)))

; func for calculating are of hexagon

(define (calc\_hexagon sqrt3 a) (\* (/ (\* sqrt3 3) 2) (squared a)))

; identifier determines if calculating area of triangle or hexagon

; 1 for triangle

; 2 for hexagon

; a = length of sides

(define (my\_calc identifier a)

(let (

(sqrt3 1.732)

)

(if (not(integer? a))

#f

(if (not(positive? a))

#f

(if (eq? identifier 1)

(calc\_eq\_triangle sqrt3 a)

(if (eq? identifier 2)

(calc\_hexagon sqrt3 a)

#f

)

)

)

)

)

)

'(Running my\_calc two-way selection structure: 1 and 5 as parameters)

(my\_calc 1 5)

'(Running my\_calc two-way selection structure: 1 and -2 as parameters)

(my\_calc 1 -2)

'(Running my\_calc two-way selection structure: atom a as both parameters)

(my\_calc 'a 'a)

'(Running my\_calc two-way selection structure: 3 and 5 parameters)

(my\_calc 3 5)

'(Running my\_calc two-way selection structure: 2 and 5 as parameters)

(my\_calc 2 5)

Problem 1-A code

#lang racket

; area of equilateral triangle - sqrt 3 /4 \* a\*\*2 where a = length of side

; area of hexagon - 3 \* sqrt(3) / 2 \* a\*\*2 where a = length of side

; helper functions

; square func

(define (squared x) (\* x x))

; func for calculating eq triangle

(define (calc\_eq\_triangle sqrt3 a) (\* (/ sqrt3 4) (squared a)))

; func for calculating are of hexagon

(define (calc\_hexagon sqrt3 a) (\* (/ (\* sqrt3 3) 2) (squared a)))

; identifier determines if calculating area of triangle or hexagon

; 1 for triangle

; 2 for hexagon

; a = length of sides

(define (my\_calc identifier a)

(let (

(sqrt3 1.732)

)

(cond

; if a is not integer

[(not(integer? a)) #f]

; if a length of side isn't positive return false

[(not(positive? a)) #f]

; calculate area of eq triangle

[(eq? identifier 1) (calc\_eq\_triangle sqrt3 a)]

; calculate area of hexagon

[(eq? identifier 2) (calc\_hexagon sqrt3 a)]

; catch invalid input and return false

[else #f]

)

)

)

'(Running my\_calc multi-way selection structure: 1 and 5 as parameters)

(my\_calc 1 5)

'(Running my\_calc multi-way selection structure: 1 and -2 as parameters)

(my\_calc 1 -2)

'(Running my\_calc multi-way selection structure: atom a as both parameters)

(my\_calc 'a 'a)

'(Running my\_calc two-way selection structure: 3 and 5 parameters)

(my\_calc 3 5)

'(Running my\_calc multi-way selection structure: 2 and 5 as parameters)

(my\_calc 2 5)

Problem 2 code

#lang racket

; remove second and third element from list and return the list

(define (remove\_second\_and\_third\_element lst)

(cond

[(not(list? lst)) '(Error please pass a list)]

[(null? lst) '()]

[(null? (cdr lst)) '()]

[(null? (cddr lst)) '() ]

[else (cons (car lst) (cdddr lst))]

)

)

'(type passed is not a list)

(remove\_second\_and\_third\_element 'a)

'(list less than 3 elements)

(remove\_second\_and\_third\_element '(1 3))

'(list of 3 elements (1 2 3))

(remove\_second\_and\_third\_element '(1 2 3))

'(list of 5 elements (1 2 3 4 5))

(remove\_second\_and\_third\_element '(1 2 3 4 5))

Problem 3 code

#lang racket

; take a list and atom

; delete occurrences of the atom from the list

; returns the list result

(define (delete-atom lst atom)

(cond

; make sure the list is a list

[(not(list? lst)) '(ERROR: pass a list please)]

; if its null base case

[(null? lst) '()]

; check if list is nested and trigger recursion on the nested list

[(list? (car lst)) (cons (delete-atom (car lst) atom) (delete-atom (cdr lst) atom))]

; chop the first element and check it

[(eq? (car lst) atom) (delete-atom (cdr lst) atom)]

[else (cons (car lst) (delete-atom (cdr lst) atom))]

)

)

'(Error for when list is not a list)

(delete-atom 1 1)

'(delete 1 from list (1))

(delete-atom '(1) 1)

'(delete 1 from list with nested list (1 2 (1 3) 3 4))

(delete-atom '(1 2 '(1 3) 3 4 ) 1)

'(delete 1 from INCEPTION - a list within a list within a list...the world isnt real)

(delete-atom '(1 '(1 '(1) 2) 2 3) 1)