Nhat Doan
CSC 135 -Meilu
PL-HW 2

1>volume = 4 * 3.14 * r3 / 3: function sphere-volume takes in a radius as parameter and return the volume of the sphere

(define (sphere-volume radius)
(exact->inexact (/ (* 4 3.14 (expt radius 3))3))); return the volume of the sphere with the given radius
(define (shell-volume inner-radius outer-radius)
(- (exact->inexact (/ (* 4 3.14 (expt outer-radius 3))3)) (exact->inexact (/ (* 4 3.14 (expt inner-radius 3))3)))) ; return the volume of the ring with the given inner and outer radiuses

Show case 1:

```
> (shell-volume 1 2)
#i29.306666666666665
>
```

Show case 2:

```
> (shell-volume 2 4)
#i234.45333333333332
>
```

2> Function close-with-limit takes three arguments -- number-1, number-2, and limit -- and returns #t if the two numbers are within a distance of limit from each other
(define (close num1 num2)
(or (< (abs (- num1 num2)) 0.001) (= (abs (- num1 num2)) 0.001)); return true if the distance of 2 number is less than or equal to 0.001
)
(define (close-with-limit number-1 number-2 limit)
(or (< (abs (- number-1 number-2)) limit) (= (abs (- number-1 number-2)) limit)); return true if the distance of 2 number is less than or equal to the limit
)

Show case 1:

```
> (close-with-limit 1 1.001 0.0001)
#false
>
```

Show case 2:

```
> (close-with-limit 1 1.001 0.01)
#true
>
```

3> Function how-many, which consumes the coefficients a, b, and c of a proper quadratic equation and determines how many solutions the equation has

```
(define (how-many a b c)
        (if (> (* b b) (* 4 a c))          ; check if b^2 > 4ac
        2                                  ;return 2 if so
        (if(< (* b b) (* 4 a c))           ;if not, check if b^2 < 4ac
        0                                  ;return 0 if so
        1))                                ;return 1 if not
)
```

Show case 1:

```
> (how-many -1 2 3)
2
```

Show case 2:

```
> (how-many 1 2 3)
0
>
```

4> Function filter-out-symbol take 2 argument a list and a symbol, then return the list with all the matching symbols removed

```
(define (filter-out-symbol list symbol)
        (cond ((null? list) '())           ;check if the list is null, return empty list if so
        ((eq? symbol (car list))           ; else check if the first element of the list equal to the symbol
        (filter-out-symbol (cdr list) symbol))          ;if so, recurse the function by passing a new list without the first symbol
        (else (cons (car list) (filter-out-symbol (cdr list) symbol))))          ; else concatenate the first element with the recursive function by passing a new list without the first symbol
)
```

Show case 1:

```
> (filter-out-symbol '(no no a thousand times no) 'no)
(list 'a 'thousand 'times)
>
```

Show case 2:

```
> (filter-out-symbol '(1 2 3 4 1 75 1 2 1 5) '1)
(list 2 3 4 75 2 5)
>
```

5> Function pMinMax take in an argument a list of integer, then return the min and max

(define (Min L)

(cond ((null? (cdr L)) (car L)) ; check if the list without the first symbol is the last symbol, return that symbol if so

((<(car L)(Min (cdr L)))(car L));else check if the first symbol is less than the recursive of the next elements on the list, return that element if it is less than the rest of the element

(else (Min (cdr L))))); else recurse to find the next smallest integer.

(define (Max L)

(cond ((null? (cdr L))(car L));check if the list without the first symbol is the last symbol, return that symbol if so

((> (car L)(Max (cdr L))) (car L));else check if the first symbol is greater than the recursive of the next elements on the list, return that element if it is greater than the rest of the element

(else(Max(cdr L))))); else recurse to find the next largest integer.

(define (pMinMax L)

(list(Min L)(Max L))); return the list of min and max

Show case 1:

```
> (pMinMax '(4 71 2 56 22 125 -1))
(list -1 125)
>
```

Show case 2:

```
> (pMinMax '(4 71 2 956 22 -125 -1))
(list -125 956)
>
```

;6>Higher-order function incnth that takes an integer n as a parameter and returns an n-th increment function, which increments its parameter by n

(define (incnth n)

(lambda (x) ; take 2 parameter n and x

(+ x n))) ; return the sum of them

Language: Intermediate Student with lambda; memory limit: 128 MB.

```
> ((incnth 1) 2)
3
> ((incnth -2) 7)
5
>
```