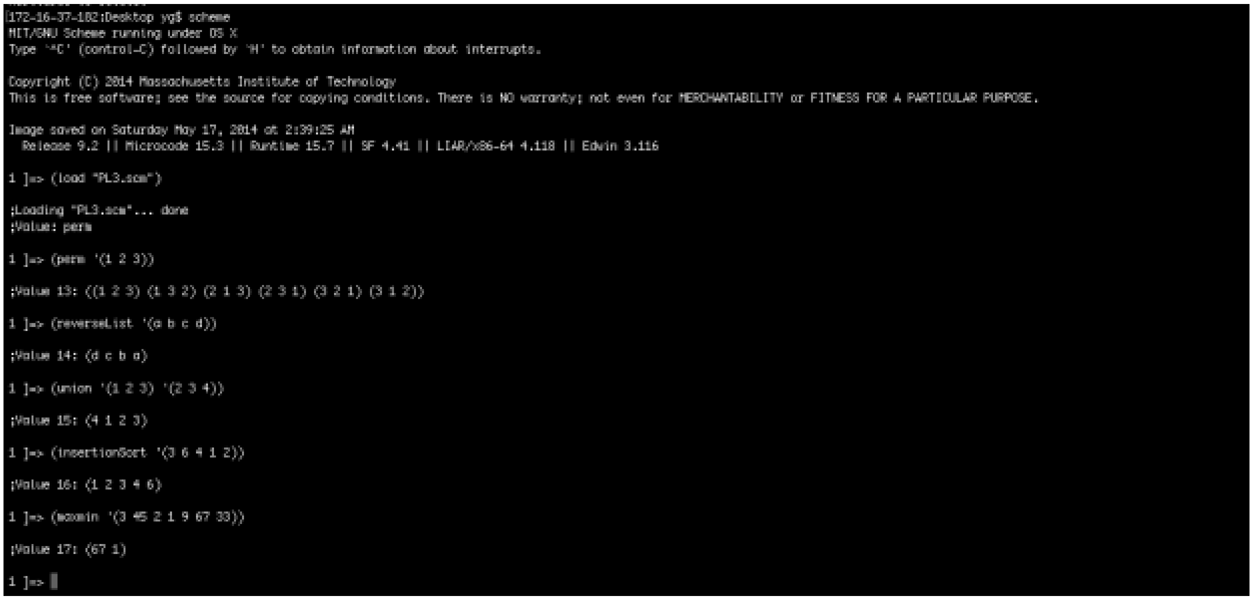
# PROGRAMING LANGUAGES ASSIGNMENT 4

## PART 1:SCHEME

```
(define (reverseList lst)
(COND
((NULL? lst) lst)
(ELSE (APPEND (reverseList(CDR lst)) (LIST (CAR lst))))
))

(define (union l1 l2)
 (cond ((null? l2) l1)
    ((member (car l2) l1)
     (union l1 (cdr l2)))
    (else (union (cons (car l2) l1) (cdr l2)))))


(DEFINE (insert item list)
  (COND
      ((NULL? list) (CONS item '()))
      ((< item (CAR list)) (CONS item list))
      (ELSE (CONS (CAR list) (insert item (CDR list))))
  )
)

(DEFINE (insertionSort list)
 (IF (NULL? list)'()
  (insert (CAR list) (insertionSort (CDR list)))
 )
)

(define (maxmin L)
 (COND
  ((null? L) '())
  ((null? (cdr L)) (list (car L) (car L)))
  (else (let((maxmintemp(maxmin (cdr L))) (temp (car L)) )
      (cond(( > temp (car maxmintemp)) (cons temp(cdr maxmintemp)))
        (( < temp (cadr maxmintemp)) (list (car maxmintemp) temp))
        (else maxmintemp))))))
```

```
(define (perm s)
  (cond ((null? s) '())
           ((null? (cdr s)) (list s))
           (else (let splice ((l '()) (m (car s)) (r (cdr s)))
              (append
                (map (lambda (l1) (cons m l1)) (perm (append l r)))
                (if (null? r) '()
                      (splice (cons m l) (car r) (cdr r))))))))
```

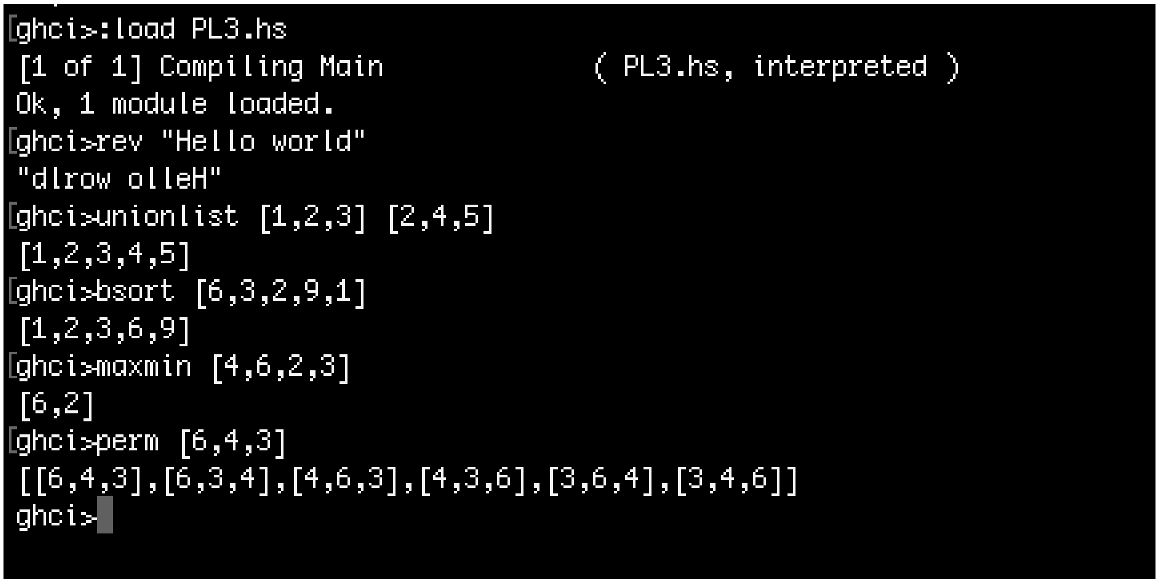


```
172-16-37-182:Desktop yg$ scheme
MIT/GNU Scheme running under OS X
Type `^C' (control-C) followed by `H' to obtain information about interrupts.

Copyright (C) 2014 Massachusetts Institute of Technology
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Image saved on Saturday May 17, 2014 at 2:39:25 AM
  Release 9.2 || Microcode 15.3 || Runtime 15.7 || SF 4.41 || LIAR/x86-64 4.118 || Edwin 3.116

1 ]=> (load "PL3.scm")

;Loading "PL3.scm"... done
;Value: perm

1 ]=> (perm '(1 2 3))

;Value 13: ((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 2 1) (3 1 2))

1 ]=> (reverseList '(a b c d))

;Value 14: (d c b a)

1 ]=> (union '(1 2 3) '(2 3 4))

;Value 15: (4 1 2 3)

1 ]=> (insertionSort '(3 6 4 1 2))

;Value 16: (1 2 3 4 6)

1 ]=> (maxmin '(3 45 2 1 9 67 33))

;Value 17: (67 1)

1 ]=> 
```

## PART 2:HASKELL

```
import Data.List


rev ::[a]->[a]
rev [] = []
rev (l1:xs) = rev(xs)++[l1]


unionlist :: (Eq a) => [a] -> [a] -> [a]
unionlist xs ys = xs ++ unionlist ys xs
   where unionlist [] _ = []
        unionlist (a:as) first = if (a `elem` first) then unionlist as first else a : unionlist as (a:first)
```

```
bsort :: (Ord a) => [a] -> [a]
bsort [] = []
bsort (x:xs) =
   let smallerSorted = bsort [a | a <- xs, a <= x]
       biggerSorted = bsort [a | a <- xs, a > x]
   in  smallerSorted ++ [x] ++ biggerSorted



maxf s a
     | null s = a
     | (>) (head s) a = maxf (tail s) (head s)
     | otherwise = maxf (tail s) a

minf s a
     | null s = a
     | (<) (head s) a = minf (tail s) (head s)
     | otherwise = minf (tail s) a

maxmin s
     | null s = []
     | otherwise = [maxf s (head s)] ++ [minf s (head s)]



perm :: [a] -> [[a]]
perm [] = [[]]
perm xs = [ y:zs | (y,ys) <- select xs, zs <- perm ys]
  where select []     = []
        select (l1:xs) = (l1,xs) : [ (y,l1:ys) | (y,ys) <- select xs ]
```



```
[ghci>:load PL3.hs
 [1 of 1] Compiling Main             ( PL3.hs, interpreted )
 Ok, 1 module loaded.
[ghci>rev "Hello world"
 "dlrow olleH"
[ghci>unionlist [1,2,3] [2,4,5]
 [1,2,3,4,5]
[ghci>bsort [6,3,2,9,1]
 [1,2,3,6,9]
[ghci>maxmin [4,6,2,3]
 [6,2]
[ghci>perm [6,4,3]
 [[6,4,3],[6,3,4],[4,6,3],[4,3,6],[3,6,4],[3,4,6]]
 ghci>
```