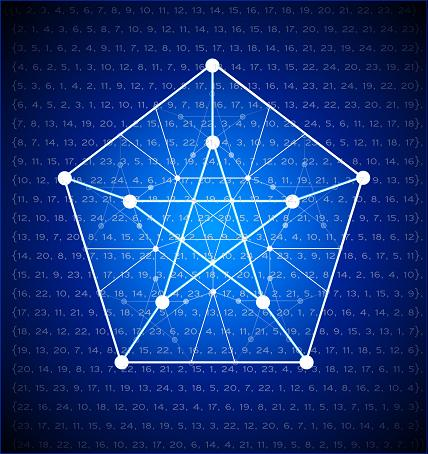
****

**Tecnológico de Monterrey**

**TC2037: Implementación de métodos computacionales**

Actividad 2.1: Programación funcional – Parte 2

****

* Carlos Estrada Ceballos A01638214
* Abigail Velasco García A01638095
* Natalia Velasco García A01638047

1. La función insert toma dos entradas: un número n y una lista lst que contiene números en orden ascendente. Devuelve una nueva lista con los mismos elementos de lst pero con n insertado en su lugar correspondiente.

> (define (insert n listA)

(insert2 n listA '())

)

> (define (insert2 n listA listB)

(if (null? listA)

(append listB (cons n '()))

(if(< (car listA) n)

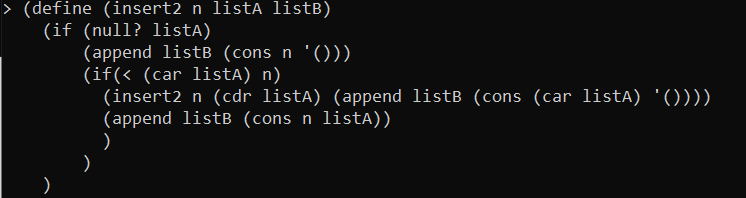
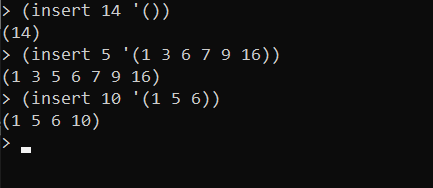
(insert2 n (cdr listA) (append listB (cons (car listA) '())))

(append listB (cons n listA))

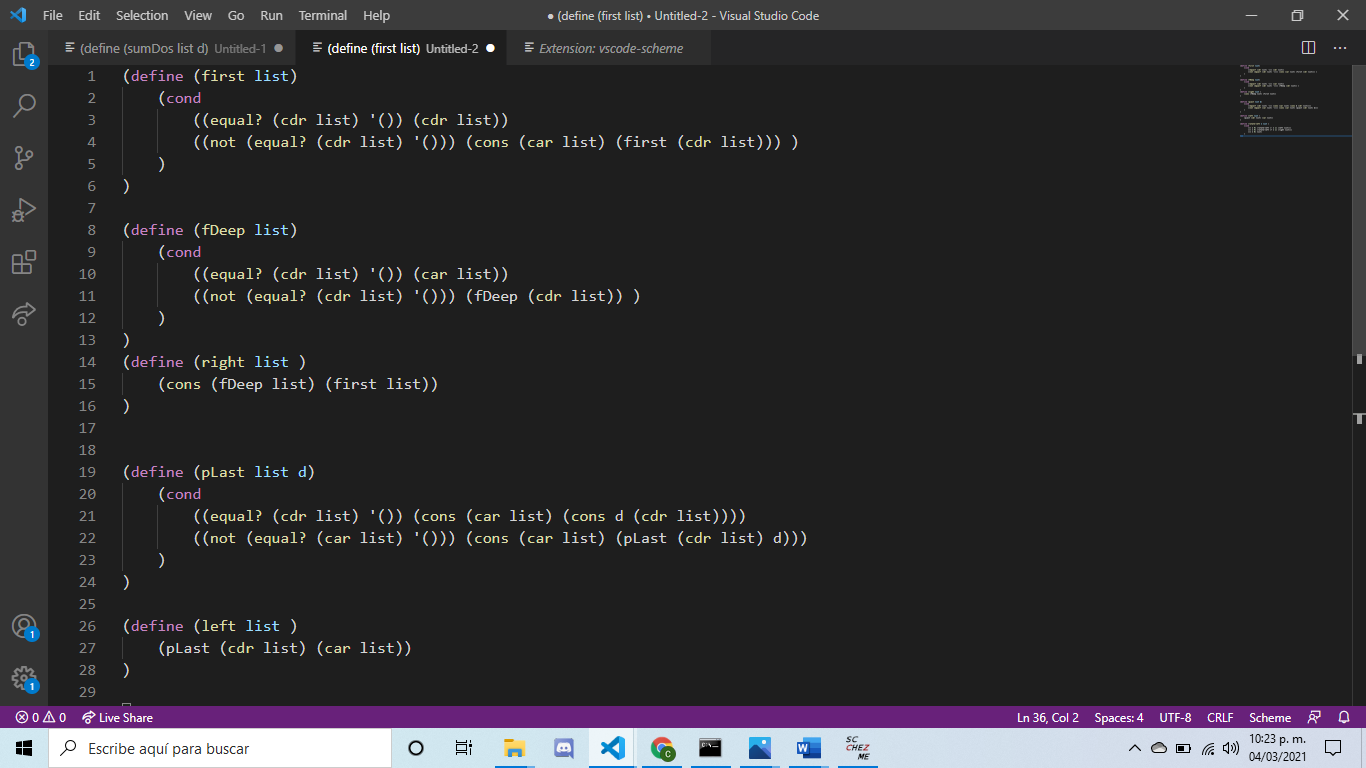
)

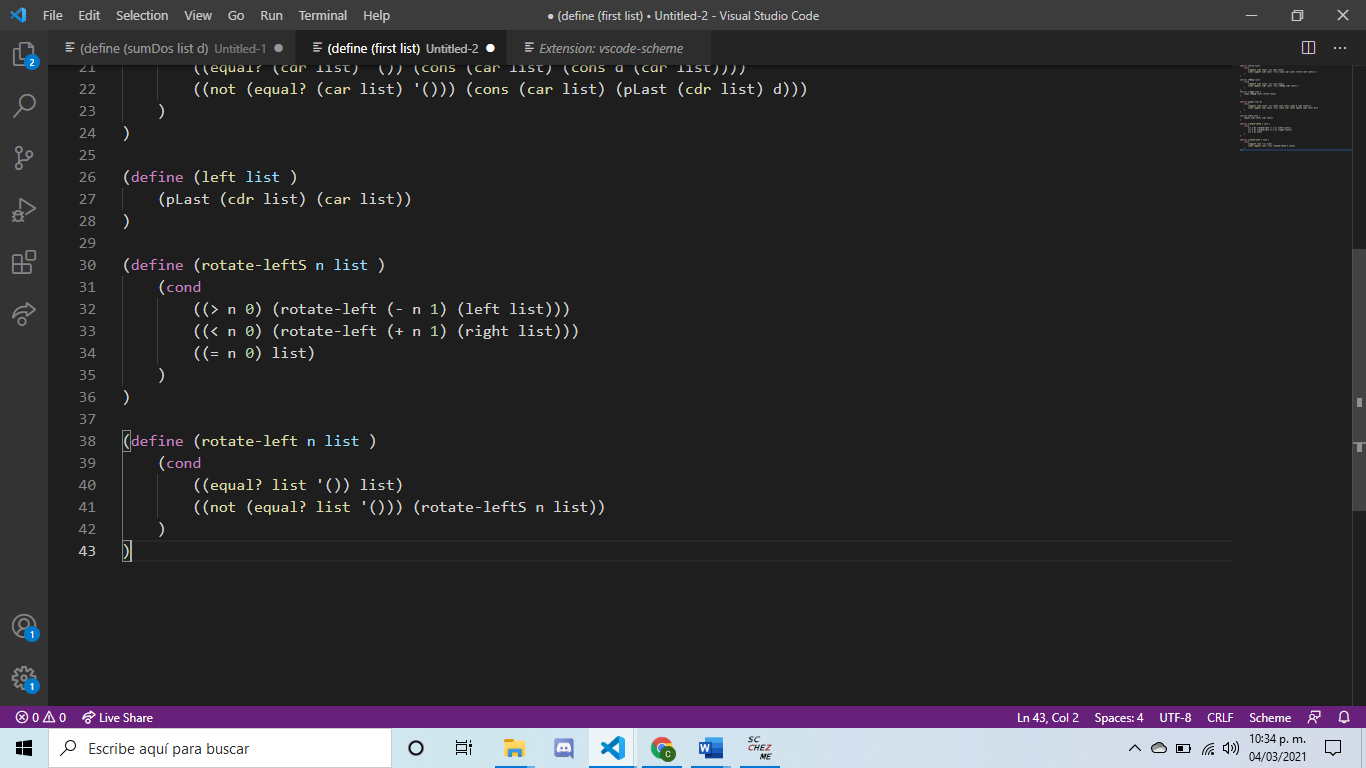
)

)



1. La función ‘rotate-left’ toma dos entradas: un número entero n y una lista lst. Devuelve la lista que resulta de rotar ‘lst’ un total de n elementos a la izquierda. Si n es negativo, rota hacia la derecha.





(define (first list)

(cond

((equal? (cdr list) '()) (cdr list))

((not (equal? (cdr list) '())) (cons (car list) (first (cdr list))) )

)

)

(define (fDeep list)

(cond

((equal? (cdr list) '()) (car list))

((not (equal? (cdr list) '())) (fDeep (cdr list)) )

)

)

(define (right list )

(cons (fDeep list) (first list))

)

(define (pLast list d)

(cond

((equal? (cdr list) '()) (cons (car list) (cons d (cdr list))))

((not (equal? (car list) '())) (cons (car list) (pLast (cdr list) d)))

)

)

(define (left list )

(pLast (cdr list) (car list))

)

(define (rotate-left n list )

(cond

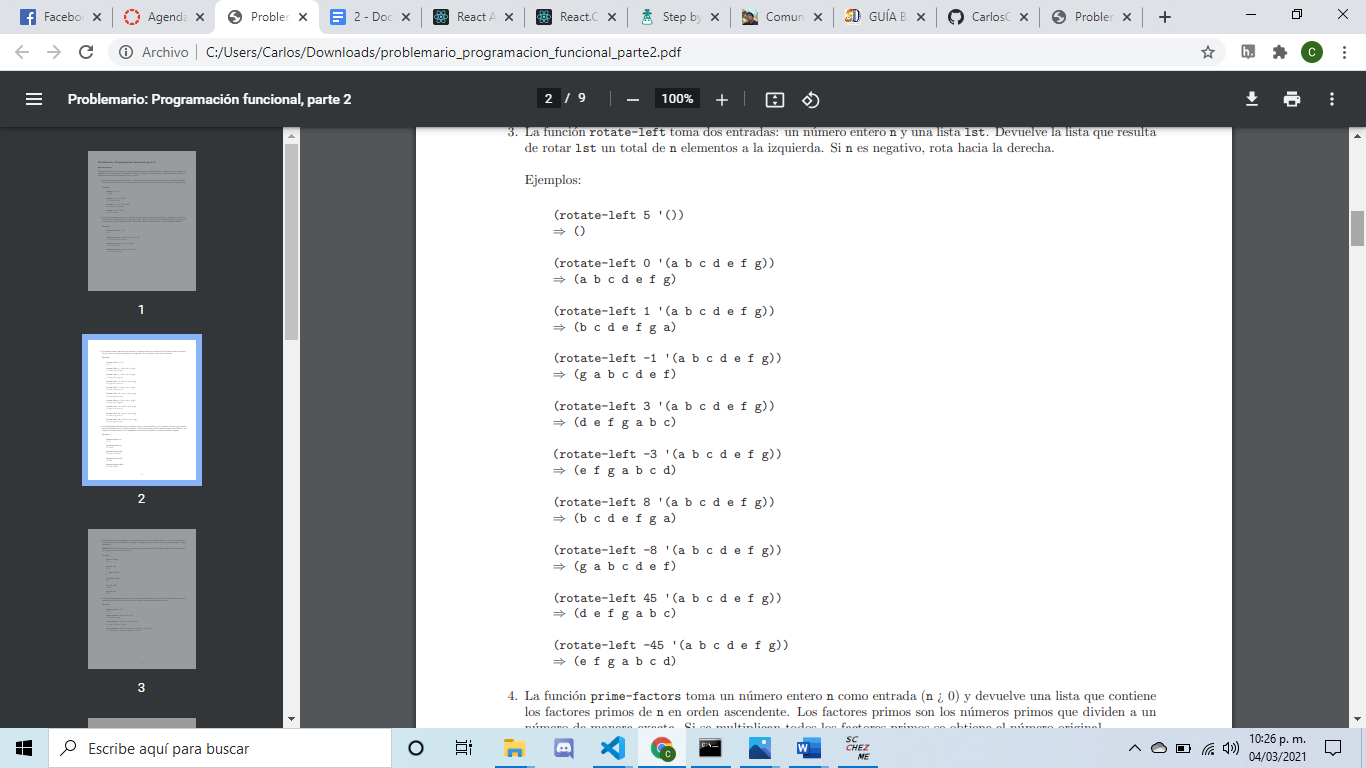
((> n 0) (rotate-left (- n 1) (left list)))

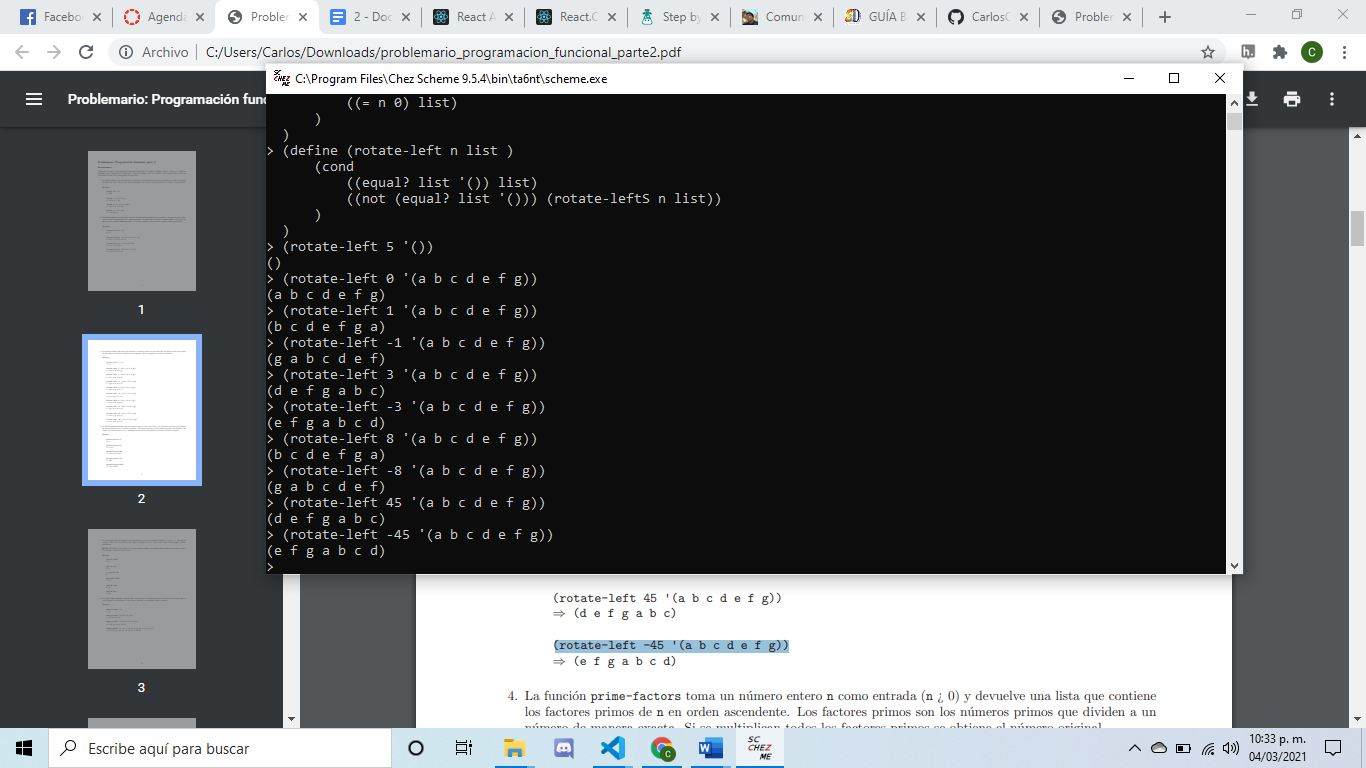
((< n 0) (rotate-left (+ n 1) (right list)))

((= n 0) list)

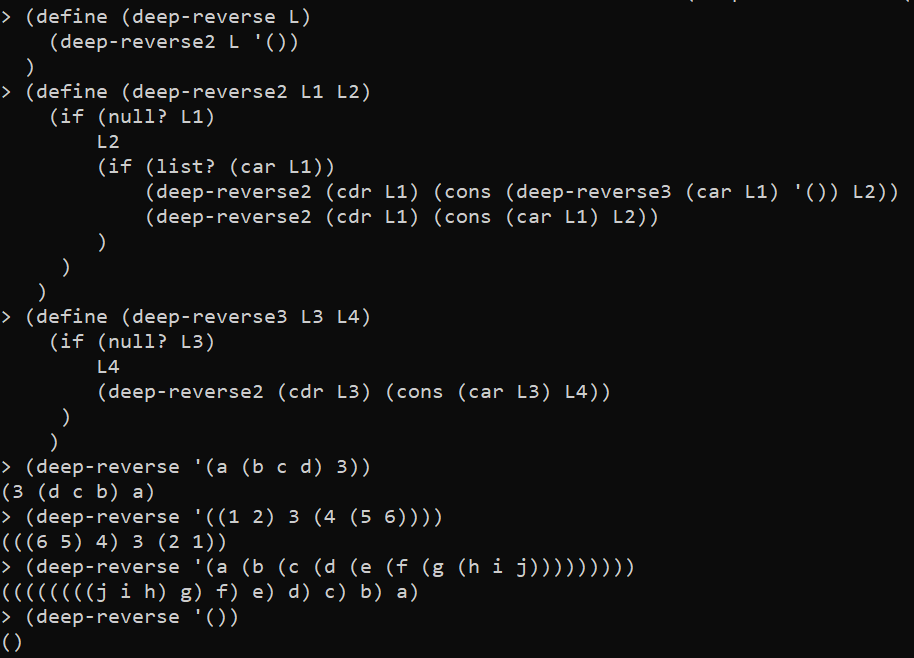
)

)



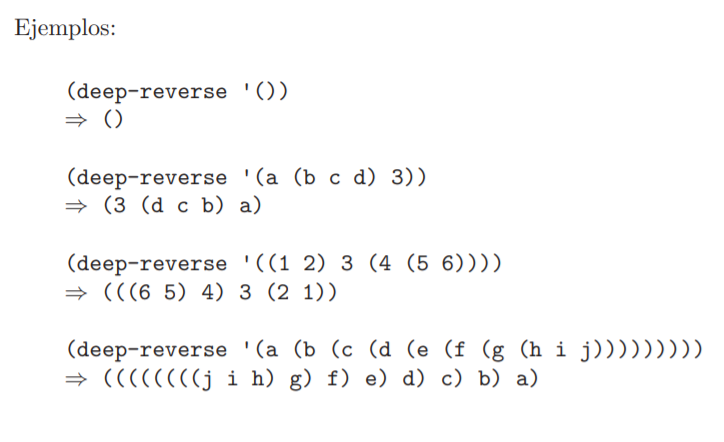


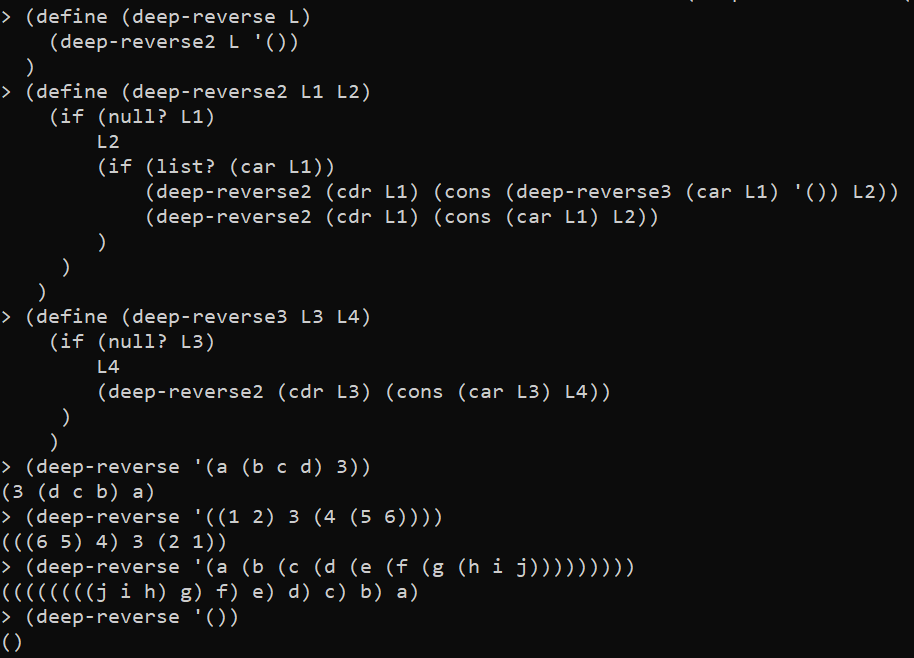
1. La función deep-reverse toma una lista como entrada. Devuelve una lista con los mismos elementos que su entrada pero en orden inverso. Si hay listas anidadas, estas también deben invertirse.

****

> (define (deep-reverse L) (deep-reverse2 L '()) )

> (define (deep-reverse2 L1 L2) (if (null? L1) L2 (if (list? (car L1)) (deep-reverse2 (cdr L1) (cons (deep-reverse3 (car L1) '()) L2)) (deep-reverse2 (cdr L1) (cons (car L1) L2)) ) ) )



****