# Taller # 2
# Corrección de errores y uso de recursividad

**Profesor:** Daniel Wilches Maradei [dmaradei@eisc.univalle.edu.co]
**Monitor:** Christian Alejandro Trujillo [catruro@univalle.edu.co]

**Fecha máxima de entrega:**
Jueves 22 de Marzo, 11:59PM

**Modo de entrega:**
El taller debe enviarse completamente por medio del Campus Virtual de la Universidad del Valle [http://campusvirtual.univalle.edu.co], o a los 2 correos escritos arriba. Si escoge esta última opción queda bajo su responsabilidad si el correo no llega a los destinatarios.

Debe entregar UN solo archivo comprimido con la solución a cada uno de los puntos dentro de él. Incluya su nombre y código dentro del archivo comprimido o en cada uno de los archivos .scm

**La entrega del taller es INDIVIDUAL.**

**Importante:**
Si tiene dudas sobre el desarrollo de este taller puede acercarse a la sala de sistemas # 3 los sábados de 4PM a 6PM donde se encontrará el monitor. O en último caso preguntar por correo electrónico al profesor o al monitor.

1. *Detecte y corrija los errores que tienen las siguientes 5 expresiones Scheme. El resultado esperado después de la corrección es el que aparece después del* ***;****.*
   *Debe indicar a que se debía el error corregido.*

```
   i.  round(54 * 3/PI)                          ; #i52
  ii.  (/ (* 100 5 4)                            ; 125
 iii.  remainder 100 6                           ; 4
  iv.  abs -100 / 10                             ; 10
   v.  (define (fun 1) (* 2 3))                  ;
```

2. *Detecte y corrija los errores que tiene la siguiente función Scheme, si el resultado esperado de aplicar la función es el que aparece en la sección de ejemplos.*
   *Debe indicar a que se debía el error corregido.*

```
(define (operar valor1 valor2)
  (cond
    (= valor1 valor2) (* valor1 valor2)
    [(> valor1 valor2) (+ valor1 valor2)]
    (= valor1 (* 2 valor2)) valor2)
    [else (abs (- valor1 valor2)))
    ))

; Ejemplos:
(operar 1 1)                                    ; 1
(operar 1 2)                                    ; 1
(operar 2 1)                                    ; 3
(operar 4 2)                                    ; 6
```

3. *Diseñe las funciones* ***par?*** *e* ***impar?*** *(con contrato, ejemplos y descripción) que permita resolver el siguiente problema recursivo mediante la función* ***par-o-impar*** *definida así:*

```
(define (par-o-impar numero)
  (cond
    ((par? numero)   "par")
    ((impar? numero) "impar")
    ))
```

   *Las funciones* ***par?*** *e* ***impar?*** *Deben ser creadas de acuerdo a las siguientes definiciones:*

$$par?(N)\begin{cases}0 \text{ si } N = 0\\ \text{impar?}(N-1) \text{ en caso contrario}\end{cases}$$

$$impar?(N)\begin{cases}0 \text{ si } N = 0\\ \text{par?}(N-1) \text{ en caso contrario}\end{cases}$$

*Si tiene problema entendiendo cómo puede funcionar **par-o-impar** basándose en esas 2 funciones puede hacer una prueba de escritorio (a mano).*

4. *Las funciones **par?** e **impar?** tienen un problema con los números negativos, debido a que si son aplicadas a uno de ellos entrarían en recursión infinita.*
   *Corrija las funciones para que den el valor correcto cuando sean aplicadas a números negativos.*

5. *La función **par-o-impar** también tiene un problema si es llamada con el número 0, ya que respondería "par" y el 0 no es considerado un número par*
   *Corrija la función para que responda "cero" cuando sea aplicada al número 0. Así:*

```
(par-o-impar 0) ; Valor esperado: "cero"
```